```diff
diff --git a/.dockerignore b/.dockerignore
deleted file mode 100644
index 6c85e6f6..00000000
--- a/.dockerignore
+++ /dev/null
@@ -1,9 +0,0 @@
-.ci
-.github
-.gitignore
-.golangci.yml
-.idea
-.vscode
-
-LICENSE
-*.md
diff --git a/.github/workflows/pull_ci.yml b/.github/workflows/pull_ci.yml
deleted file mode 100644
index 1ddb8471..00000000
--- a/.github/workflows/pull_ci.yml
+++ /dev/null
@@ -1,52 +0,0 @@
-name: PR CI
-on: [pull_request]
-
-jobs:
-  lint:
-    name: Lint
-    runs-on: ubuntu-18.04
-    steps:
-      - uses: actions/checkout@v2
-      - name: golangci-lint
-        uses: golangci/golangci-lint-action@v2
-        with:
-          version: latest
-          working-directory: .
-          args: --timeout 3m
-  test:
-    name: Golang Unit Tests v${{ matrix.go }} (${{ matrix.os }})
-    runs-on: ${{ matrix.os }}
-    strategy:
-      matrix:
-        go: ['1.16']
-        os: [ubuntu-20.04]
-    steps:
-    - uses: actions/checkout@v2
-    - uses: actions/setup-go@v1
-      with:
-        go-version: ${{ matrix.go }}
-    - run: go mod download
-      shell: bash
-    - run: ./scripts/build.sh evm
-      shell: bash
-    - run: ./scripts/build_test.sh
-      shell: bash
-  e2e:
-    name: Golang E2E Tests v${{ matrix.go }} (${{ matrix.os }})
-    runs-on: ${{ matrix.os }}
-    strategy:
-      matrix:
-        go: [ '1.16' ]
-        os: [ ubuntu-20.04 ]
-    steps:
-    - uses: actions/checkout@v2
-    - uses: actions/setup-go@v1
-      with:
-        go-version: ${{ matrix.go }}
-    - run: .github/workflows/run_e2e_tests.sh --parallelism 1 --client-id $KURTOSIS_CLIENT_ID --client-secret $KURTOSIS_CLIENT_SECRET
-      shell: bash
-      env:
-        DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
-        DOCKER_PASS: ${{ secrets.DOCKER_PASS }}
-        KURTOSIS_CLIENT_ID: ${{ secrets.KURTOSIS_CLIENT_ID }}
-        KURTOSIS_CLIENT_SECRET: ${{ secrets.KURTOSIS_CLIENT_SECRET }}
diff --git a/.github/workflows/push_ci.yml b/.github/workflows/push_ci.yml
deleted file mode 100644
index 1ac73f18..00000000
--- a/.github/workflows/push_ci.yml
+++ /dev/null
@@ -1,52 +0,0 @@
-name: Branch Push CI
-on: [push]
-
-jobs:
-  lint:
-    name: Lint
-    runs-on: ubuntu-18.04
-    steps:
-      - uses: actions/checkout@v2
-      - name: golangci-lint
-        uses: golangci/golangci-lint-action@v2
-        with:
-          version: latest
-          working-directory: .
-          args: --timeout 3m
-  test:
-    name: Golang Unit Tests v${{ matrix.go }} (${{ matrix.os }})
-    runs-on: ${{ matrix.os }}
-    strategy:
-      matrix:
-        go: ['1.16']
-        os: [macos-11.0, ubuntu-18.04, ubuntu-20.04, windows-latest]
-    steps:
-    - uses: actions/checkout@v2
-    - uses: actions/setup-go@v1
-      with:
-        go-version: ${{ matrix.go }}
-    - run: go mod download
-      shell: bash
-    - run: ./scripts/build.sh evm
-      shell: bash
-    - run: ./scripts/build_test.sh
-      shell: bash
-  e2e:
-    name: Golang E2E Tests v${{ matrix.go }} (${{ matrix.os }})
-    runs-on: ${{ matrix.os }}
-    strategy:
-      matrix:
-        go: [ '1.16' ]
-        os: [ ubuntu-20.04 ]
-    steps:
-    - uses: actions/checkout@v2
-    - uses: actions/setup-go@v1
-      with:
-        go-version: ${{ matrix.go }}
-    - run: .github/workflows/run_e2e_tests.sh --parallelism 1 --client-id $KURTOSIS_CLIENT_ID --client-secret $KURTOSIS_CLIENT_SECRET
-      shell: bash
-      env:
-        DOCKER_USERNAME: ${{ secrets.DOCKER_USERNAME }}
-        DOCKER_PASS: ${{ secrets.DOCKER_PASS }}
-        KURTOSIS_CLIENT_ID: ${{ secrets.KURTOSIS_CLIENT_ID }}
-        KURTOSIS_CLIENT_SECRET: ${{ secrets.KURTOSIS_CLIENT_SECRET }}
diff --git a/.github/workflows/run_e2e_tests.sh b/.github/workflows/run_e2e_tests.sh
deleted file mode 100755
```

```
index 04f297fc..00000000
--- a/.github/workflows/run_e2e_tests.sh
+++ /dev/null
@@ -1,86 +0,0 @@
-set -o errexit
-set -o nounset
-set -o pipefail
-
-# If Docker Credentials are not available fail
-if [[ -z ${DOCKER_USERNAME} ]]; then
-    echo "Skipping Tests because Docker Credentials were not present."
-    exit 1
-fi
-
-# Testing specific variables
-avalanche_testing_repo="avaplatform/avalanche-testing"
-avalanchego_repo="avaplatform/avalanchego"
-# Define default avalanche testing version to use
-avalanche_testing_image="${avalanche_testing_repo}:master"
-
-# Avalanche root directory
-CORETH_PATH=$( cd "$( dirname "${BASH_SOURCE[0]}" )"; cd ../.. && pwd )
-
-# Load the versions
-source "$CORETH_PATH"/scripts/versions.sh
-
-# Load the constants
-source "$CORETH_PATH"/scripts/constants.sh
-
-# Login to docker
-echo "$DOCKER_PASS" | docker login --username "$DOCKER_USERNAME" --password-stdin
-
-# Checks available docker tags exist
-function docker_tag_exists() {
-    TOKEN=$(curl -s -H "Content-Type: application/json" -X POST -d '{"username": "'${DOCKER_USERNAME}'", "password": "'${DOCKER_PASS}'"}' https://hub.docker.com/v2/users/login/ | jq -r .token)
-    curl --silent -H "Authorization: JWT ${TOKEN}" -f --head -lL https://hub.docker.com/v2/repositories/$1/tags/$2/ > /dev/null
-}
-
-# Defines the avalanche-testing tag to use
-# Either uses the same tag as the current branch or uses the default
-if docker_tag_exists $avalanche_testing_repo $current_branch; then
-    echo "$avalanche_testing_repo:$current_branch exists; using this image to run e2e tests"
-    avalanche_testing_image="$avalanche_testing_repo:$current_branch"
-else
-    echo "$avalanche_testing_repo $current_branch does NOT exist; using the default image to run e2e tests"
-fi
-
-echo "Using $avalanche_testing_image for e2e tests"
-
-# Defines the avalanchego tag to use
-# Either uses the same tag as the current branch or uses the default
-# Disable matchup in favor of explicit tag
-# TODO re-enable matchup when our workflow better supports it.
-# if docker_tag_exists $avalanchego_repo $current_branch; then
-#     echo "$avalanchego_repo:$current_branch exists; using this avalanchego image to run e2e tests"
-#     AVALANCHE_VERSION=$current_branch
-# else
-#     echo "$avalanchego_repo $current_branch does NOT exist; using the default image to run e2e tests"
-# fi
-
-# pulling the avalanche-testing image
-docker pull $avalanche_testing_image
-
-# Setting the build ID
-git_commit_id=$( git rev-list -1 HEAD )
-
-# Build current avalanchego
-source "$CORETH_PATH"/scripts/build_image.sh
-
-# Target built version to use in avalanche-testing
-avalanche_image="avaplatform/avalanchego:$build_image_id"
-
-echo "Running Avalanche Image: ${avalanche_image}"
-echo "Running Avalanche Testing Image: ${avalanche_testing_image}"
-echo "Git Commit ID : ${git_commit_id}"
-
-
-# >>>>>>>> avalanche-testing custom parameters <<<<<<<<<<<<<
-custom_params_json="{
-    \"isKurtosisCoreDevMode\": false,
-    \"avalanchegoImage\":\"${avalanche_image}\",
-    \"testBatch\":\"avalanchego\"
-}"
-# >>>>>>>> avalanche-testing custom parameters <<<<<<<<<<<<<
-
-bash "$CORETH_PATH/.kurtosis/kurtosis.sh" \
-    --tests "C-Chain Bombard WorkFlow" \
-    --custom-params "${custom_params_json}" \
-    "${avalanche_testing_image}" \
-    $@
diff --git a/.gitignore b/.gitignore
index 87ff040a..84c048a7 100644
--- a/.gitignore
+++ b/.gitignore
@@ -1,46 +1 @@
-./main
-
-*.log
-*~
-.DS_Store
-
-awscpu
-
-# Binaries for programs and plugins
-*.exe
-*.exe~
-*.dll
-*.so
-*.dylib
-*.profile
-
-# Test binary, build with `go test -c`
-*.test
-
-# Output of the go coverage tool, specifically when used with LiteIDE
-*.out
-
-# ignore GoLand metafiles directory
-.idea/
-
-*logs/
-
-.vscode*
-
-*.pb*
-
-*cpu[0-9]*
-*mem[0-9]*
-*lock[0-9]*
-*.profile
-*.swp
-*.aux
-*.fdb*
-*.fls
```

```
-*.gz
-*.pdf
-
-.coverage
-
-bin/
-build/
+/build/
diff --git a/.golangci.yml b/.golangci.yml
deleted file mode 100644
index d2a5e38c..00000000
--- a/.golangci.yml
+++ /dev/null
@@ -1,48 +0,0 @@
-# This file configures github.com/golangci/golangci-lint.
-
-run:
-  timeout: 3m
-  tests: true
-  # default is true. Enables skipping of directories:
-  #   vendor$, third_party$, testdata$, examples$, Godeps$, builtin$
-  skip-dirs-use-default: true
-  skip-files:
-    - core/genesis_alloc.go
-
-linters:
-  disable-all: true
-  enable:
-    - deadcode
-    - goconst
-    - goimports
-    - gosimple
-    - govet
-    - ineffassign
-    - misspell
-    - unconvert
-    - varcheck
-
-linters-settings:
-  gofmt:
-    simplify: true
-  goconst:
-    min-len: 3 # minimum length of string constant
-    min-occurrences: 6 # minimum number of occurrences
-
-issues:
-  exclude-rules:
-    - path: crypto/blake2b/
-      linters:
-        - deadcode
-    - path: crypto/bn256/cloudflare
-      linters:
-        - deadcode
-    - path: p2p/discv5/
-      linters:
-        - deadcode
-    - path: core/vm/instructions_test.go
-      linters:
-        - goconst
-    - path: cmd/faucet/
-      linters:
-        - deadcode
diff --git a/.kurtosis/kurtosis.sh b/.kurtosis/kurtosis.sh
deleted file mode 100755
index a3d1cd85..00000000
--- a/.kurtosis/kurtosis.sh
+++ /dev/null
@@ -1,226 +0,0 @@
-# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-#
-#      Do not modify this file! It will get overwritten when you upgrade Kurtosis!
-#
-# !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!! WARNING !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
-
-set -euo pipefail
-
-
-
-# ============================================================================================
-#                                      Constants
-# ============================================================================================
-# The directory where Kurtosis will store files it uses in between executions, e.g. access tokens
-# Can make this configurable if needed
-KURTOSIS_DIRPATH="${HOME}/.kurtosis"
-
-KURTOSIS_CORE_TAG="1.8"
-KURTOSIS_DOCKERHUB_ORG="kurtosistech"
-INITIALIZER_IMAGE="${KURTOSIS_DOCKERHUB_ORG}/kurtosis-core_initializer:${KURTOSIS_CORE_TAG}"
-API_IMAGE="${KURTOSIS_DOCKERHUB_ORG}/kurtosis-core_api:${KURTOSIS_CORE_TAG}"
-
-POSITIONAL_ARG_DEFINITION_FRAGMENTS=2
-
-
-
-# ============================================================================================
-#                                      Arg Parsing
-# ============================================================================================
-function print_help_and_exit() {
-    echo ""
-    echo "$(basename "${0}") [--custom-params custom_params_json] [--client-id client_id] [--client-secret client_secret] [--help] [--kurtosis-log-level kurtosis_log_level] [--list] [--parallelism paral
-    echo ""
-    echo "   --custom-params custom_params_json            JSON string containing arbitrary data that will be passed as-is to your testsuite, so it can modify its behaviour based on input (default: {})"
-    echo "   --client-id client_id                         An OAuth client ID which is needed for running Kurtosis in CI, and should be left empty when running Kurtosis on a local machine"
-    echo "   --client-secret client_secret                 An OAuth client secret which is needed for running Kurtosis in CI, and should be left empty when running Kurtosis on a local machine"
-    echo "   --help                                        Display this message"
-    echo "   --kurtosis-log-level kurtosis_log_level       The log level that all output generated by the Kurtosis framework itself should log at (panic|fatal|error|warning|info|debug|trace) (default: i
-    echo "   --list                                        Rather than running the tests, lists the tests available to run"
-    echo "   --parallelism parallelism                     The number of texts to execute in parallel (default: 4)"
-    echo "   --tests test_names                            List of test names to run, separated by ',' (default or empty: run all tests)"
-    echo "   --test-suite-log-level test_suite_log_level   A string that will be passed as-is to the test suite container to indicate what log level the test suite container should output at; this strin
-    echo "   test_suite_image                              The Docker image containing the testsuite to execute"
-
-    echo ""
-    exit 1  # Exit with an error code, so that if it gets accidentally called in parent scripts/CI it fails loudly
-}
-
-
-
-# ============================================================================================
-#                                      Arg Parsing
-# ============================================================================================
-client_id=""
-client_secret=""
-custom_params_json="{}"
-do_list="false"
-kurtosis_log_level="info"
-parallelism="4"
-show_help="false"
-test_names=""
-test_suite_image=""
-test_suite_log_level="info"
-
-
-
-POSITIONAL=()
-while [ ${#} -gt 0 ]; do
-    key="${1}"
-    case "${key}" in
-
-        --custom-params)
-
-            custom_params_json="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --client-id)
-
-            client_id="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --client-secret)
-
-            client_secret="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --help)
-            show_help="true"
-            shift   # Shift to clear out the flag
-
-            ;;
-
-        --kurtosis-log-level)
-
-            kurtosis_log_level="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --list)
-            do_list="true"
-            shift   # Shift to clear out the flag
-
-            ;;
-
-        --parallelism)
-
-            parallelism="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --tests)
-
-            test_names="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        --test-suite-log-level)
-
-            test_suite_log_level="${2}"
-            shift   # Shift to clear out the flag
-            shift   # Shift again to clear out the value
-            ;;
-
-        -*)
-            echo "ERROR: Unrecognized flag '${key}'" >&2
-            exit 1
-            ;;
-        *)
-            POSITIONAL+=("${1}")
-            shift
-            ;;
-    esac
-done
-
-if "${show_help}"; then
-    print_help_and_exit
-fi
-
-# Restore positional parameters and assign them to variables
-set -- "${POSITIONAL[@]}"
-test_suite_image="${1:-}"
-
-
-
-
-
-# ==============================================================================================
-#                                    Arg Validation
-# ==============================================================================================
-if [ "${#}" -ne 1 ]; then
-    echo "ERROR: Expected 1 positional variables but got ${#}" >&2
-    print_help_and_exit
-fi
-
-if [ -z "$test_suite_image" ]; then
-    echo "ERROR: Variable 'test_suite_image' cannot be empty" >&2
-    exit 1
-fi
-
-
-
-# ==============================================================================================
-#                                    Main Logic
-# ==============================================================================================# Because Kurtosis X.Y.Z tags are normalized to X.Y so that minor patch updates are transparently
-#  used, we need to pull the latest API & initializer images
-echo "Pulling latest versions of API & initializer image..."
-if ! docker pull "${INITIALIZER_IMAGE}"; then
-    echo "WARN: An error occurred pulling the latest version of the initializer image (${INITIALIZER_IMAGE}); you may be running an out-of-date version" >&2
-else
-    echo "Successfully pulled latest version of initializer image"
-fi
-if ! docker pull "${API_IMAGE}"; then
-    echo "WARN: An error occurred pulling the latest version of the API image (${API_IMAGE}); you may be running an out-of-date version" >&2
-else
-    echo "Successfully pulled latest version of API image"
-fi
-
-# Kurtosis needs a Docker volume to store its execution data in
-# To learn more about volumes, see: https://docs.docker.com/storage/volumes/
-sanitized_image="$(echo "${test_suite_image}" | sed 's/[^a-zA-Z0-9_.-]/_/g')"
-suite_execution_volume="$(date +%Y-%m-%dT%H.%M.%S)_${sanitized_image}"
-if ! docker volume create "${suite_execution_volume}" > /dev/null; then
-    echo "ERROR: Failed to create a Docker volume to store the execution files in" >&2
-    exit 1
-fi
-
-if ! mkdir -p "${KURTOSIS_DIRPATH}"; then
-    echo "ERROR: Failed to create the Kurtosis directory at '${KURTOSIS_DIRPATH}'" >&2
-    exit 1
-fi
-
```

```
-docker run \
-    `# The Kurtosis initializer runs inside a Docker container, but needs to access to the Docker engine; this is how to do it` \
-    `# For more info, see the bottom of: http://jpetazzo.github.io/2015/09/03/do-not-use-docker-in-docker-for-ci/` \
-    --mount "type=bind,source=/var/run/docker.sock,target=/var/run/docker.sock" \
-    \
-    `# Because the Kurtosis initializer runs inside Docker but needs to persist & read files on the host filesystem between execution,` \
-    `#  the container expects the Kurtosis directory to be bind-mounted at the special "/kurtosis" path` \
-    --mount "type=bind,source=${KURTOSIS_DIRPATH},target=/kurtosis" \
-    \
-    `# The Kurtosis initializer image requires the volume for storing suite execution data to be mounted at the special "/suite-execution" path` \
-    --mount "type=volume,source=${suite_execution_volume},target=/suite-execution" \
-    \
-    `# Keep these sorted alphabetically` \
-    --env CLIENT_ID="${client_id}" \
-    --env CLIENT_SECRET="${client_secret}" \
-    --env CUSTOM_PARAMS_JSON="${custom_params_json}" \
-    --env DO_LIST="${do_list}" \
-    --env KURTOSIS_API_IMAGE="${API_IMAGE}" \
-    --env KURTOSIS_LOG_LEVEL="${kurtosis_log_level}" \
-    --env PARALLELISM="${parallelism}" \
-    --env SUITE_EXECUTION_VOLUME="${suite_execution_volume}" \
-    --env TEST_NAMES="${test_names}" \
-    --env TEST_SUITE_IMAGE="${test_suite_image}" \
-    --env TEST_SUITE_LOG_LEVEL="${test_suite_log_level}" \
-    \
-    "${INITIALIZER_IMAGE}"
diff --git a/CHANGES.pdf b/CHANGES.pdf
new file mode 100644
index 00000000..7864125d
Binary files /dev/null and b/CHANGES.pdf differ
diff --git a/Dockerfile b/Dockerfile
deleted file mode 100644
index ec7b1720..00000000
--- a/Dockerfile
+++ /dev/null
@@ -1,30 +0,0 @@
-# syntax=docker/dockerfile:experimental
-
-# ============= Setting up base Stage ================
-# Set required AVALANCHE_VERSION parameter in build image script
-ARG AVALANCHE_VERSION
-
-# ============= Compilation Stage ================
-FROM golang:1.17.1-buster AS builder
-RUN apt-get update && apt-get install -y --no-install-recommends bash=5.0-4 git=1:2.20.1-2+deb10u3 make=4.2.1-1.2 gcc=4:8.3.0-1 musl-dev=1.1.21-2 ca-certificates=20200601~deb10u2 linux-headers-amd64
-
-WORKDIR /build
-# Copy and download avalanche dependencies using go mod
-COPY go.mod .
-COPY go.sum .
-RUN go mod download
-
-# Copy the code into the container
-COPY . .
-
-# Pass in CORETH_COMMIT as an arg to allow the build script to set this externally
-ARG CORETH_COMMIT
-ARG CURRENT_BRANCH
-
-RUN export CORETH_COMMIT=$CORETH_COMMIT && export CURRENT_BRANCH=$CURRENT_BRANCH && ./scripts/build.sh /build/evm
-
-# ============= Cleanup Stage ================
-FROM avaplatform/avalanchego:$AVALANCHE_VERSION AS builtImage
-
-# Copy the evm binary into the correct location in the container
-COPY --from=builder /build/evm /avalanchego/build/plugins/evm
diff --git a/accounts/abi/bind/auth.go b/accounts/abi/bind/auth.go
index 859083c0..b50ac70d 100644
--- a/accounts/abi/bind/auth.go
+++ b/accounts/abi/bind/auth.go
@@ -34,13 +34,23 @@ import (
        "io/ioutil"
        "math/big"
 
-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/accounts/external"
-       "github.com/ava-labs/coreth/accounts/keystore"
-       "github.com/ava-labs/coreth/core/types"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
+
+=======
+>>>>>>> upstream-v0.8.5-rc.2
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/accounts/external"
+       "github.com/flare-foundation/coreth/accounts/keystore"
+       "github.com/flare-foundation/coreth/core/types"
+<<<<<<< HEAD
+=======
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/log"
+>>>>>>> upstream-v0.8.5-rc.2
 )
 
 // ErrNoChainID is returned whenever the user failed to specify a chain id.
diff --git a/accounts/abi/bind/backend.go b/accounts/abi/bind/backend.go
index 29a4b3cb..179a043a 100644
--- a/accounts/abi/bind/backend.go
+++ b/accounts/abi/bind/backend.go
@@ -31,9 +31,16 @@ import (
        "errors"
        "math/big"
 
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+=======
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+       "github.com/ethereum/go-ethereum/common"
+>>>>>>> upstream-v0.8.5-rc.2
 )
 
 var (
diff --git a/accounts/abi/bind/backends/simulated.go b/accounts/abi/bind/backends/simulated.go
index 4860cb34..e4d214e1 100644
--- a/accounts/abi/bind/backends/simulated.go
+++ b/accounts/abi/bind/backends/simulated.go
@@ -34,27 +34,42 @@ import (
        "sync"
        "time"
 
-       "github.com/ava-labs/coreth/eth"
-
-       "github.com/ava-labs/coreth/accounts/abi"
-       "github.com/ava-labs/coreth/accounts/abi/bind"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/bloombits"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/eth/filters"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/interfaces"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+=======
+       "github.com/flare-foundation/coreth/eth"
+>>>>>>> upstream-v0.8.5-rc.2
+
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/bloombits"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+<<<<<<< HEAD
+       "github.com/flare-foundation/coreth/eth"
+=======
+>>>>>>> upstream-v0.8.5-rc.2
+       "github.com/flare-foundation/coreth/eth/filters"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/interfaces"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
+<<<<<<< HEAD
+=======
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/ethereum/go-ethereum/common/math"
+       "github.com/ethereum/go-ethereum/event"
+       "github.com/ethereum/go-ethereum/log"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 // Verify that SimulatedBackend implements required interfaces
@@ -107,7 +122,11 @@ type SimulatedBackend struct {
 func NewSimulatedBackendWithDatabase(database ethdb.Database, alloc core.GenesisAlloc, gasLimit uint64) *SimulatedBackend {
        cpcfg := params.TestChainConfig
        cpcfg.ChainID = big.NewInt(1337)
+<<<<<<< HEAD
+       genesis := core.Genesis{Config: cpcfg, GasLimit: gasLimit, Alloc: alloc, Coinbase: common.HexToAddress("0x0100000000000000000000000000000000000000")}
+=======
        genesis := core.Genesis{Config: cpcfg, GasLimit: gasLimit, Alloc: alloc}
+>>>>>>> upstream-v0.8.5-rc.2
        genesis.MustCommit(database)
        cacheConfig := &core.CacheConfig{}
        blockchain, _ := core.NewBlockChain(database, cacheConfig, genesis.Config, dummy.NewFaker(), vm.Config{}, common.Hash{})
@@ -499,6 +518,12 @@ func (b *SimulatedBackend) AcceptedNonceAt(ctx context.Context, account common.A
 // SuggestGasPrice implements ContractTransactor.SuggestGasPrice. Since the simulated
 // chain doesn't have miners, we just return a gas price of 1 for any call.
 func (b *SimulatedBackend) SuggestGasPrice(ctx context.Context) (*big.Int, error) {
+<<<<<<< HEAD
+=======
+       b.mu.Lock()
+       defer b.mu.Unlock()
+
+>>>>>>> upstream-v0.8.5-rc.2
        if b.acceptedBlock.Header().BaseFee != nil {
                return b.acceptedBlock.Header().BaseFee, nil
        }
diff --git a/accounts/abi/bind/backends/simulated_test.go b/accounts/abi/bind/backends/simulated_test.go
index d979e791..8f3ff124 100644
--- a/accounts/abi/bind/backends/simulated_test.go
+++ b/accounts/abi/bind/backends/simulated_test.go
@@ -37,14 +37,14 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/coreth/accounts/abi"
-       "github.com/ava-labs/coreth/accounts/abi/bind"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+       "github.com/flare-foundation/coreth/params"
 )

 func TestSimulatedBackend(t *testing.T) {
@@ -512,7 +512,7 @@ func TestEstimateGas(t *testing.T) {
                        GasPrice: big.NewInt(0),
                        Value:    nil,
                        Data:     common.Hex2Bytes("b9b046f9"),
-               }, 0, errors.New("invalid opcode: opcode 0xfe not defined"), nil},
+               }, 0, errors.New("invalid opcode: INVALID"), nil},

                {"Valid", interfaces.CallMsg{
                        From:     addr,
diff --git a/accounts/abi/bind/base.go b/accounts/abi/bind/base.go
index 58ac1ea0..3428bb04 100644
--- a/accounts/abi/bind/base.go
+++ b/accounts/abi/bind/base.go
@@ -34,12 +34,28 @@ import (
        "strings"
        "sync"

-       "github.com/ava-labs/coreth/accounts/abi"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/event"
+
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+=======
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/interfaces"
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/event"
+)
+
+var (
+       ErrNilAssetAmount            = errors.New("cannot specify nil asset amount for native asset call")
+       errNativeAssetDeployContract = errors.New("cannot specify native asset params while deploying a contract")
+>>>>>>> upstream-v0.8.5-rc.2
 )

 // SignerFn is a signer function callback when a contract requires a method to
@@ -54,6 +70,15 @@ type CallOpts struct {
        Context     context.Context // Network context to support cancellation and timeouts (nil = no timeout)
 }

+<<<<<<< HEAD
+=======
+// NativeAssetCallOpts contains params for native asset call
+type NativeAssetCallOpts struct {
+       AssetID     common.Hash // Asset ID
+       AssetAmount *big.Int    // Asset amount
+}
+
+>>>>>>> upstream-v0.8.5-rc.2
 // TransactOpts is the collection of authorization data required to create a
 // valid Ethereum transaction.
 type TransactOpts struct {
@@ -70,6 +95,17 @@ type TransactOpts struct {
        Context context.Context // Network context to support cancellation and timeouts (nil = no timeout)

        NoSend bool // Do all transact steps but do not send the transaction
+<<<<<<< HEAD
+=======
+
+       // If set, the transaction is transformed to perform the requested call through the native asset
+       // precompile. This will update the to address of the transaction to that of the native asset precompile
+       // and pack the requested [to] address, [assetID], [assetAmount], and [input] data for the transaction
+       // into the call data of the transaction. When executed within the EVM, the precompile will parse the input
+       // data and attempt to atomically transfer [assetAmount] of [assetID] to the [to] address and invoke the
+       // contract at [to] if present, passing in the original [input] data.
+       NativeAssetCall *NativeAssetCallOpts
+>>>>>>> upstream-v0.8.5-rc.2
 }

 // FilterOpts is the collection of options to fine tune filtering for events
@@ -240,6 +276,42 @@ func (c *BoundContract) Transfer(opts *TransactOpts) (*types.Transaction, error)
        // or not, reject invalid transaction at the first place
        return c.transact(opts, &c.address, nil)
 }
+<<<<<<< HEAD
+=======
+
+// wrapNativeAssetCall preprocesses the arguments to transform the requested call to go through the
+// native asset call precompile if it is specified on [opts].
+func wrapNativeAssetCall(opts *TransactOpts, contract *common.Address, input []byte) (*common.Address, []byte, error) {
+       if opts.NativeAssetCall != nil {
+               // Prevent the user from sending a non-zero value through native asset call precompile as this will
+               // transfer the funds to the precompile address and essentially burn the funds.
+               if opts.Value != nil && opts.Value.Cmp(common.Big0) != 0 {
+                       return nil, nil, fmt.Errorf("value must be 0 when performing native asset call, found %d", opts.Value)
+               }
+               if opts.NativeAssetCall.AssetAmount == nil {
+                       return nil, nil, ErrNilAssetAmount
+               }
+               if opts.NativeAssetCall.AssetAmount.Cmp(common.Big0) < 0 {
+                       return nil, nil, fmt.Errorf("asset value cannot be < 0 when performing native asset call, found %d", opts.NativeAssetCall.AssetAmount)
+               }
+               // Prevent potential panic if [contract] is nil in the case that transact is called through DeployContract.
+               if contract == nil {
+                       return nil, nil, errNativeAssetDeployContract
+               }
+               // wrap input with native asset call params
+               input = vm.PackNativeAssetCallInput(
+                       *contract,
+                       opts.NativeAssetCall.AssetID,
+                       opts.NativeAssetCall.AssetAmount,
+                       input,
+               )
+               // target addr is now precompile
+               contract = &vm.NativeAssetCallAddr
+       }
+       return contract, input, nil
+}
+
+>>>>>>> upstream-v0.8.5-rc.2
 func (c *BoundContract) createDynamicTx(opts *TransactOpts, contract *common.Address, input []byte, head *types.Header) (*types.Transaction, error) {
        // Normalize value
        value := opts.Value
@@ -375,6 +447,14 @@ func (c *BoundContract) transact(opts *TransactOpts, contract *common.Address, i
                rawTx *types.Transaction
                err   error
        )
+<<<<<<< HEAD
+=======
+       // Preprocess native asset call arguments if present
+       contract, input, err = wrapNativeAssetCall(opts, contract, input)
+       if err != nil {
+               return nil, err
+       }
+>>>>>>> upstream-v0.8.5-rc.2
        if opts.GasPrice != nil {
                rawTx, err = c.createLegacyTx(opts, contract, input)
        } else {
diff --git a/accounts/abi/bind/base_test.go b/accounts/abi/bind/base_test.go
index d7988516..6ca2eee8 100644
--- a/accounts/abi/bind/base_test.go
+++ b/accounts/abi/bind/base_test.go
@@ -28,20 +28,38 @@ package bind_test

 import (
        "context"
+<<<<<<< HEAD
+=======
+       "fmt"
+>>>>>>> upstream-v0.8.5-rc.2
        "math/big"
        "reflect"
        "strings"
        "testing"

-       "github.com/ava-labs/coreth/accounts/abi"
-       "github.com/ava-labs/coreth/accounts/abi/bind"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
+<<<<<<< HEAD
+       "github.com/stretchr/testify/assert"
+
+=======
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/interfaces"
+>>>>>>> upstream-v0.8.5-rc.2
```

```
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/rlp"
+<<<<<<< HEAD
+
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+=======
        "github.com/stretchr/testify/assert"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 func mockSign(addr common.Address, tx *types.Transaction) (*types.Transaction, error) { return tx, nil }
@@ -277,6 +295,83 @@ func TestUnpackIndexedBytesTyLogIntoMap(t *testing.T) {
        unpackAndCheck(t, bc, expectedReceivedMap, mockLog)
 }

+<<<<<<< HEAD
+=======
+func TestTransactNativeAssetCallNilAssetAmount(t *testing.T) {
+       assert := assert.New(t)
+       mt := &mockTransactor{}
+       bc := bind.NewBoundContract(common.Address{}, abi.ABI{}, nil, mt, nil)
+       opts := &bind.TransactOpts{
+               Signer: mockSign,
+       }
+       // fails if asset amount is nil
+       opts.NativeAssetCall = &bind.NativeAssetCallOpts{
+               AssetID:     common.Hash{},
+               AssetAmount: nil,
+       }
+       _, err := bc.Transact(opts, "")
+       assert.ErrorIs(err, bind.ErrNilAssetAmount)
+}
+
+func TestTransactNativeAssetCallNonZeroValue(t *testing.T) {
+       assert := assert.New(t)
+       mt := &mockTransactor{}
+       bc := bind.NewBoundContract(common.Address{}, abi.ABI{}, nil, mt, nil)
+       opts := &bind.TransactOpts{
+               Signer: mockSign,
+       }
+       opts.NativeAssetCall = &bind.NativeAssetCallOpts{
+               AssetID:     common.Hash{},
+               AssetAmount: big.NewInt(11),
+       }
+       // fails if value > 0
+       opts.Value = big.NewInt(11)
+       _, err := bc.Transact(opts, "")
+       assert.Equal(err.Error(), fmt.Sprintf("value must be 0 when performing native asset call, found %v", opts.Value))
+       // fails if value < 0
+       opts.Value = big.NewInt(-11)
+       _, err = bc.Transact(opts, "")
+       assert.Equal(err.Error(), fmt.Sprintf("value must be 0 when performing native asset call, found %v", opts.Value))
+}
+
+func TestTransactNativeAssetCall(t *testing.T) {
+       assert := assert.New(t)
+       json := `[{"type":"function","name":"method","inputs":[{"type":"uint256" },{"type":"string"}]}]`
+       parsed, err := abi.JSON(strings.NewReader(json))
+       assert.Nil(err)
+       mt := &mockTransactor{}
+       contractAddr := common.Address{11}
+       bc := bind.NewBoundContract(contractAddr, parsed, nil, mt, nil)
+       opts := &bind.TransactOpts{
+               Signer: mockSign,
+       }
+       // normal call tx
+       methodName := "method"
+       arg1 := big.NewInt(22)
+       arg2 := "33"
+       normalCallTx, err := bc.Transact(opts, methodName, arg1, arg2)
+       assert.Nil(err)
+       // native asset call tx
+       assetID := common.Hash{44}
+       assetAmount := big.NewInt(55)
+       opts.NativeAssetCall = &bind.NativeAssetCallOpts{
+               AssetID:     assetID,
+               AssetAmount: assetAmount,
+       }
+       nativeCallTx, err := bc.Transact(opts, methodName, arg1, arg2)
+       assert.Nil(err)
+       // verify transformations
+       assert.Equal(vm.NativeAssetCallAddr, *nativeCallTx.To())
+       unpackedAddr, unpackedAssetID, unpackedAssetAmount, unpackedData, err := vm.UnpackNativeAssetCallInput(nativeCallTx.Data())
+       assert.Nil(err)
+       assert.NotEmpty(unpackedData)
+       assert.Equal(unpackedData, normalCallTx.Data())
+       assert.Equal(unpackedAddr, contractAddr)
+       assert.Equal(unpackedAssetID, assetID)
+       assert.Equal(unpackedAssetAmount, assetAmount)
+}
+
+>>>>>>> upstream-v0.8.5-rc.2
 func TestTransactGasFee(t *testing.T) {
        assert := assert.New(t)

diff --git a/accounts/abi/bind/bind.go b/accounts/abi/bind/bind.go
index 22e1a8cb..0158d7e3 100644
--- a/accounts/abi/bind/bind.go
+++ b/accounts/abi/bind/bind.go
@@ -40,8 +40,14 @@ import (
        "text/template"
        "unicode"

-       "github.com/ava-labs/coreth/accounts/abi"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/log"
+
+       "github.com/flare-foundation/coreth/accounts/abi"
+=======
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/ethereum/go-ethereum/log"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 // Lang is a target programming language selector to generate bindings for.
@@ -98,6 +104,16 @@ func Bind(types []string, abis []string, bytecodes []string, fsigs []map[string]
                        transactIdentifiers = make(map[string]bool)
                        eventIdentifiers    = make(map[string]bool)
                )
+<<<<<<< HEAD
+=======
+
+               for _, input := range evmABI.Constructor.Inputs {
+                       if hasStruct(input.Type) {
+                               bindStructType[lang](input.Type, structs)
+                       }
+               }
+
+>>>>>>> upstream-v0.8.5-rc.2
```

```
                for _, original := range evmABI.Methods {
                        // Normalize the method for capital cases and non-anonymous inputs/outputs
                        normalized := original
diff --git a/accounts/abi/bind/bind_test.go b/accounts/abi/bind/bind_test.go
index 3cd9f3f1..95103425 100644
--- a/accounts/abi/bind/bind_test.go
+++ b/accounts/abi/bind/bind_test.go
@@ -298,9 +298,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -353,9 +353,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -399,9 +399,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -456,10 +456,10 @@ var bindTests = []struct {
                        "math/big"
                        "reflect"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
                        "github.com/ethereum/go-ethereum/common"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -505,9 +505,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -571,9 +571,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -616,10 +616,10 @@ var bindTests = []struct {
                []string{`6060604052609f8060106000396000f3606060405260e060020a6000350463f97a60058114601a575b005b600060605260c0604052600d60809081527f4920646f6e27742065786973740000000000000000000000000000000
                []string{`[{"constant":true,"inputs":[],"name":"String","outputs":[{"name":"","type":"string"}],"type":"function"}]`},
                `
-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
                        "github.com/ethereum/go-ethereum/common"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/core"
                `,
                `
                        // Create a simulator and wrap a non-deployed contract
@@ -655,10 +655,10 @@ var bindTests = []struct {
                []string{`6080604052348015600f57600080fd5b5060888061001e6000396000f3fe6080604052348015600f57600080fd5b506004361060285760003560e01c8063d5f6622514602d575b600080fd5b6033604c565b6040805192835
                []string{`[{"inputs":[],"name":"Struct","outputs":[{"internalType":"uint256","name":"a","type":"uint256"},{"internalType":"uint256","name":"b","type":"uint256"}],"stateMutability":"pure",
                `
-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
                        "github.com/ethereum/go-ethereum/common"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/core"
                `,
                `
                        // Create a simulator and wrap a non-deployed contract
@@ -703,9 +703,9 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-                       "github.com/ava-labs/coreth/core"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/core"
                        "github.com/ethereum/go-ethereum/crypto"
                `,
                `
@@ -752,10 +752,10 @@ var bindTests = []struct {
                `
                        "math/big"

-                       "github.com/ava-labs/coreth/accounts/abi/bind"
-                       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
+                       "github.com/flare-foundation/coreth/accounts/abi/bind"
```

```diff
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
			"github.com/ethereum/go-ethereum/common"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`
@@ -779,7 +779,11 @@ var bindTests = []struct {
				t.Errorf("Invalid address returned, want: %x, got: %x", (common.Address{}), res)
			}

+<<<<<<< HEAD
+			for _, addr := range []common.Address{common.Address{}, common.Address{2}, common.Address{4}} {
+=======
			for _, addr := range []common.Address{common.Address{}, common.Address{1}, common.Address{2}} {
+>>>>>>> upstream-v0.8.5-rc.2
				if res, err := callfrom.CallFrom(&bind.CallOpts{From: addr}); err != nil {
					t.Fatalf("Failed to call constant function: %v", err)
				} else if res != addr {
@@ -828,9 +832,9 @@ var bindTests = []struct {
			"fmt"
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`
@@ -921,10 +925,10 @@ var bindTests = []struct {
			"math/big"
			"time"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
			"github.com/ethereum/go-ethereum/common"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`
@@ -1112,9 +1116,9 @@ var bindTests = []struct {
		`
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`
@@ -1247,9 +1251,9 @@ var bindTests = []struct {
			"math/big"
			"reflect"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,

@@ -1389,9 +1393,9 @@ var bindTests = []struct {
		`
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`
@@ -1455,11 +1459,11 @@ var bindTests = []struct {
		"math/big"
		"time"

-		"github.com/ava-labs/coreth/accounts/abi/bind"
-		"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-		"github.com/ava-labs/coreth/core"
+		"github.com/flare-foundation/coreth/accounts/abi/bind"
+		"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+		"github.com/flare-foundation/coreth/core"
		"github.com/ethereum/go-ethereum/crypto"
-		"github.com/ava-labs/coreth/params"
+		"github.com/flare-foundation/coreth/params"
		`,
		`
		// Initialize test accounts
@@ -1566,10 +1570,10 @@ var bindTests = []struct {
		`
		"math/big"

-		"github.com/ava-labs/coreth/accounts/abi/bind"
-		"github.com/ava-labs/coreth/accounts/abi/bind/backends"
+		"github.com/flare-foundation/coreth/accounts/abi/bind"
+		"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
		"github.com/ethereum/go-ethereum/crypto"
-		"github.com/ava-labs/coreth/core"
+		"github.com/flare-foundation/coreth/core"
		`,
		`
		// Initialize test accounts
@@ -1629,10 +1633,10 @@ var bindTests = []struct {
		`
		"math/big"

-		"github.com/ava-labs/coreth/accounts/abi/bind"
-		"github.com/ava-labs/coreth/accounts/abi/bind/backends"
+		"github.com/flare-foundation/coreth/accounts/abi/bind"
+		"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
		"github.com/ethereum/go-ethereum/crypto"
-		"github.com/ava-labs/coreth/core"
+		"github.com/flare-foundation/coreth/core"
	`,
		`
		key, _ := crypto.GenerateKey()
@@ -1691,9 +1695,9 @@ var bindTests = []struct {
		`
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
```

```
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
		`,
		`

@@ -1752,9 +1756,9 @@ var bindTests = []struct {
			"bytes"
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
	   `,
		`

@@ -1840,9 +1844,9 @@ var bindTests = []struct {
		`
			"math/big"

-			"github.com/ava-labs/coreth/accounts/abi/bind"
-			"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-			"github.com/ava-labs/coreth/core"
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
			"github.com/ethereum/go-ethereum/crypto"
	   `,
		`

@@ -1909,9 +1913,9 @@ var bindTests = []struct {
		`
				"math/big"

-				"github.com/ava-labs/coreth/accounts/abi/bind"
-				"github.com/ava-labs/coreth/accounts/abi/bind/backends"
-				"github.com/ava-labs/coreth/core"
+				"github.com/flare-foundation/coreth/accounts/abi/bind"
+				"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+				"github.com/flare-foundation/coreth/core"
				"github.com/ethereum/go-ethereum/crypto"
		   `,
		`

@@ -1942,6 +1946,52 @@ var bindTests = []struct {
		nil,
		nil,
	},
+<<<<<<< HEAD
+=======
+	{
+		name: `ConstructorWithStructParam`,
+		contract: `
+		pragma solidity >=0.8.0 <0.9.0;
+		
+		contract ConstructorWithStructParam {
+			struct StructType {
+				uint256 field;
+			}
+		
+			constructor(StructType memory st) {}
+		}
+		`,
+		bytecode: []string{`0x608060405234801561001057600080fd5b506040516101c43803806101c483398181016040528101906100329190610l4a565b50610177565b6000604051905090565b600080fd5b600080fd5b6000601f196
+		abi:      []string{`[{"inputs":[{"components":[{"internalType":"uint256","name":"field","type":"uint256"}],"internalType":"struct ConstructorWithStructParam.StructType","name":"st","type"
+		imports: `
+			"math/big"
+
+			"github.com/flare-foundation/coreth/accounts/abi/bind"
+			"github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+			"github.com/flare-foundation/coreth/core"
+			"github.com/ethereum/go-ethereum/crypto"
+		`,
+		tester: `
+			var (
+				key, _  = crypto.GenerateKey()
+				user, _ = bind.NewKeyedTransactorWithChainID(key, big.NewInt(1337))
+				sim     = backends.NewSimulatedBackend(core.GenesisAlloc{user.From: {Balance: big.NewInt(1000000000000000000)}}, 10000000)
+			)
+			defer sim.Close()
+
+			_, tx, _, err := DeployConstructorWithStructParam(user, sim, ConstructorWithStructParamStructType{Field: big.NewInt(42)})
+			if err != nil {
+				t.Fatalf("DeployConstructorWithStructParam() got err %v; want nil err", err)
+			}
+			sim.Commit(true)
+			
+			if _, err = bind.WaitDeployed(nil, sim, tx); err != nil {
+				t.Logf("Deployment tx: %+v", tx)
+				t.Errorf("bind.WaitDeployed(nil, %T, <deployment tx>) got err %v; want nil err", sim, err)
+			}
+		`,
+	},
+>>>>>>> upstream-v0.8.5-rc.2
 }

 // The binding tests have been modified to run in two separate test
@@ -1975,6 +2025,7 @@ func golangBindings(t *testing.T, overload bool) {
	}
	// Generate the test suite for all the contracts
	for i, tt := range bindTests {
+<<<<<<< HEAD
		// Skip the "Overload" test if [!overload]
		if !overload && tt.name == "Overload" {
			continue
@@ -1999,6 +2050,33 @@ func golangBindings(t *testing.T, overload bool) {
		}
		// Generate the test file with the injected test code
		code := fmt.Sprintf(`
+=======
+		t.Run(tt.name, func(t *testing.T) {
+			// Skip the "Overload" test if [!overload]
+			if !overload && tt.name == "Overload" {
+				return
+			}
+			// Skip all tests except for "Overload" if [overload]
+			if overload && tt.name != "Overload" {
+				return
+			}
+			var types []string
+			if tt.types != nil {
+				types = tt.types
+			} else {
+				types = []string{tt.name}
+			}
+			// Generate the binding and create a Go source file in the workspace
+			bind, err := Bind(types, tt.abi, tt.bytecode, tt.fsigs, "bindtest", LangGo, tt.libs, tt.aliases)
+			if err != nil {
+				t.Fatalf("test %d: failed to generate binding: %v", i, err)
+			}
+			if err = ioutil.WriteFile(filepath.Join(pkg, strings.ToLower(tt.name)+".go"), []byte(bind), 0600); err != nil {
```

```
+                               t.Fatalf("test %d: failed to write binding: %v", i, err)
+                       }
+                       // Generate the test file with the injected test code
+                       code := fmt.Sprintf(`
+>>>>>>> upstream-v0.8.5-rc.2
                        package bindtest

                        import (
@@ -2010,9 +2088,16 @@ func golangBindings(t *testing.T, overload bool) {
                                %s
                        }
                `, tt.imports, tt.name, tt.tester)
+<<<<<<< HEAD
                if err := ioutil.WriteFile(filepath.Join(pkg, strings.ToLower(tt.name)+"_test.go"), []byte(code), 0600); err != nil {
                        t.Fatalf("test %d: failed to write tests: %v", i, err)
                }
+=======
+                       if err := ioutil.WriteFile(filepath.Join(pkg, strings.ToLower(tt.name)+"_test.go"), []byte(code), 0600); err != nil {
+                               t.Fatalf("test %d: failed to write tests: %v", i, err)
+                       }
+               })
+>>>>>>> upstream-v0.8.5-rc.2
        }
        // Convert the package to go modules and use the current source for go-ethereum
        moder := exec.Command(gocmd, "mod", "init", "bindtest")
@@ -2021,7 +2106,7 @@ func golangBindings(t *testing.T, overload bool) {
                t.Fatalf("failed to convert binding test to modules: %v\n%s", err, out)
        }
        pwd, _ := os.Getwd()
-       replacer := exec.Command(gocmd, "mod", "edit", "-x", "-require", "github.com/ava-labs/coreth@v0.0.0", "-replace", "github.com/ava-labs/coreth="+filepath.Join(pwd, "..", "..", "..")) // Repo root
+       replacer := exec.Command(gocmd, "mod", "edit", "-x", "-require", "github.com/flare-foundation/coreth@v0.0.0", "-replace", "github.com/flare-foundation/coreth="+filepath.Join(pwd, "..", "..", ".."
        replacer.Dir = pkg
        if out, err := replacer.CombinedOutput(); err != nil {
                t.Fatalf("failed to replace binding test dependency to current source tree: %v\n%s", err, out)
diff --git a/accounts/abi/bind/template.go b/accounts/abi/bind/template.go
index f66f175c..c88f741f 100644
--- a/accounts/abi/bind/template.go
+++ b/accounts/abi/bind/template.go
@@ -26,7 +26,13 @@

 package bind

-import "github.com/ava-labs/coreth/accounts/abi"
+<<<<<<< HEAD
+import (
+       "github.com/flare-foundation/coreth/accounts/abi"
+)
+=======
+import "github.com/flare-foundation/coreth/accounts/abi"
+>>>>>>> upstream-v0.8.5-rc.2

 // tmplData is the data structure required to fill the binding template.
 type tmplData struct {
@@ -102,12 +108,21 @@ import (
        "strings"
        "errors"

-       "github.com/ava-labs/coreth/accounts/abi"
-       "github.com/ava-labs/coreth/accounts/abi/bind"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
+<<<<<<< HEAD
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/event"
+
+=======
+>>>>>>> upstream-v0.8.5-rc.2
+       "github.com/flare-foundation/coreth/accounts/abi"
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+<<<<<<< HEAD
+=======
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/event"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 // Reference imports to suppress errors if they are not otherwise used.
diff --git a/accounts/abi/bind/util.go b/accounts/abi/bind/util.go
index 3fb6b5c7..315d0626 100644
--- a/accounts/abi/bind/util.go
+++ b/accounts/abi/bind/util.go
@@ -31,9 +31,16 @@ import (
        "errors"
        "time"

-       "github.com/ava-labs/coreth/core/types"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+
+       "github.com/flare-foundation/coreth/core/types"
+=======
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 // WaitMined waits for tx to be mined on the blockchain.
diff --git a/accounts/abi/bind/util_test.go b/accounts/abi/bind/util_test.go
index aab4ae89..37778769 100644
--- a/accounts/abi/bind/util_test.go
+++ b/accounts/abi/bind/util_test.go
@@ -33,12 +33,21 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/coreth/accounts/abi/bind"
-       "github.com/ava-labs/coreth/accounts/abi/bind/backends"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+
+=======
+>>>>>>> upstream-v0.8.5-rc.2
+       "github.com/flare-foundation/coreth/accounts/abi/bind"
+       "github.com/flare-foundation/coreth/accounts/abi/bind/backends"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"
+<<<<<<< HEAD
+=======
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 var testKey, _ = crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291")
diff --git a/accounts/abi/unpack.go b/accounts/abi/unpack.go
index 9bf2c0da..2edaf768 100644
--- a/accounts/abi/unpack.go
```

```
+++ b/accounts/abi/unpack.go
@@ -300,7 +300,7 @@ func tuplePointsTo(index int, output []byte) (start int, err error) {
        offset := big.NewInt(0).SetBytes(output[index : index+32])
        outputLen := big.NewInt(int64(len(output)))

-       if offset.Cmp(big.NewInt(int64(len(output)))) > 0 {
+       if offset.Cmp(outputLen) > 0 {
                return 0, fmt.Errorf("abi: cannot marshal in to go slice: offset %v would go over slice boundary (len=%v)", offset, outputLen)
        }
        if offset.BitLen() > 63 {
diff --git a/accounts/accounts.go b/accounts/accounts.go
index dd7df0be..58d901c2 100644
--- a/accounts/accounts.go
+++ b/accounts/accounts.go
@@ -31,8 +31,8 @@ import (
        "fmt"
        "math/big"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/event"
        "golang.org/x/crypto/sha3"
@@ -186,7 +186,11 @@ type Backend interface {
 // TextHash is a helper function that calculates a hash for the given message that can be
 // safely used to calculate a signature from.
 //
+<<<<<<< HEAD
 // The hash is calulcated as
+=======
+// The hash is calculated as
+>>>>>>> upstream-v0.8.5-rc.2
 //   keccak256("\x19Ethereum Signed Message:\n"${message length}${message}).
 //
 // This gives context to the signed message and prevents signing of transactions.
@@ -198,7 +202,11 @@ func TextHash(data []byte) []byte {
 // TextAndHash is a helper function that calculates a hash for the given message that can be
 // safely used to calculate a signature from.
 //
+<<<<<<< HEAD
 // The hash is calulcated as
+=======
+// The hash is calculated as
+>>>>>>> upstream-v0.8.5-rc.2
 //   keccak256("\x19Ethereum Signed Message:\n"${message length}${message}).
 //
 // This gives context to the signed message and prevents signing of transactions.
diff --git a/accounts/external/backend.go b/accounts/external/backend.go
index 558700fc..1c047667 100644
--- a/accounts/external/backend.go
+++ b/accounts/external/backend.go
@@ -31,15 +31,15 @@ import (
        "math/big"
        "sync"

-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
-       "github.com/ava-labs/coreth/rpc"
-       "github.com/ava-labs/coreth/signer/core/apitypes"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
+       "github.com/flare-foundation/coreth/rpc"
+       "github.com/flare-foundation/coreth/signer/core/apitypes"
 )

 type ExternalBackend struct {
diff --git a/accounts/keystore/account_cache.go b/accounts/keystore/account_cache.go
index 4c35aa74..d35d11ff 100644
--- a/accounts/keystore/account_cache.go
+++ b/accounts/keystore/account_cache.go
@@ -37,10 +37,10 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/coreth/accounts"
        mapset "github.com/deckarep/golang-set"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/accounts"
 )

 // Minimum amount of time between cache reloads. This limit applies if the platform does
diff --git a/accounts/keystore/account_cache_test.go b/accounts/keystore/account_cache_test.go
index ad7ab51f..01671dfd 100644
--- a/accounts/keystore/account_cache_test.go
+++ b/accounts/keystore/account_cache_test.go
@@ -36,10 +36,10 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/coreth/accounts"
        "github.com/cespare/cp"
        "github.com/davecgh/go-spew/spew"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/accounts"
 )

 var (
diff --git a/accounts/keystore/key.go b/accounts/keystore/key.go
index 71402d36..bb7a5436 100644
--- a/accounts/keystore/key.go
+++ b/accounts/keystore/key.go
@@ -39,9 +39,9 @@ import (
        "strings"
        "time"

-       "github.com/ava-labs/coreth/accounts"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/accounts"
        "github.com/google/uuid"
 )

diff --git a/accounts/keystore/keystore.go b/accounts/keystore/keystore.go
index ff82ef88..e63e891f 100644
--- a/accounts/keystore/keystore.go
+++ b/accounts/keystore/keystore.go
@@ -42,11 +42,11 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/event"
```

```diff
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/core/types"
 )
 
 var (
diff --git a/accounts/keystore/keystore_test.go b/accounts/keystore/keystore_test.go
index 651ab709..c04370df 100644
--- a/accounts/keystore/keystore_test.go
+++ b/accounts/keystore/keystore_test.go
@@ -38,10 +38,10 @@ import (
        "testing"
        "time"
 
-       "github.com/ava-labs/coreth/accounts"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/event"
+       "github.com/flare-foundation/coreth/accounts"
 )
 
 var testSigData = make([]byte, 32)
diff --git a/accounts/keystore/passphrase.go b/accounts/keystore/passphrase.go
index 5da41f63..abe81df9 100644
--- a/accounts/keystore/passphrase.go
+++ b/accounts/keystore/passphrase.go
@@ -48,10 +48,10 @@ import (
        "os"
        "path/filepath"
 
-       "github.com/ava-labs/coreth/accounts"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/accounts"
        "github.com/google/uuid"
        "golang.org/x/crypto/pbkdf2"
        "golang.org/x/crypto/scrypt"
diff --git a/accounts/keystore/presale.go b/accounts/keystore/presale.go
index 1dfbd9c2..49cb183b 100644
--- a/accounts/keystore/presale.go
+++ b/accounts/keystore/presale.go
@@ -35,8 +35,8 @@ import (
        "errors"
        "fmt"
 
-       "github.com/ava-labs/coreth/accounts"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/accounts"
        "github.com/google/uuid"
        "golang.org/x/crypto/pbkdf2"
 )
diff --git a/accounts/keystore/wallet.go b/accounts/keystore/wallet.go
index 71935263..084c5190 100644
--- a/accounts/keystore/wallet.go
+++ b/accounts/keystore/wallet.go
@@ -29,10 +29,10 @@ package keystore
 import (
        "math/big"
 
-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
 )
 
 // keystoreWallet implements the accounts.Wallet interface for the original
diff --git a/accounts/scwallet/hub.go b/accounts/scwallet/hub.go
index 7a630fac..c037cebd 100644
--- a/accounts/scwallet/hub.go
+++ b/accounts/scwallet/hub.go
@@ -51,10 +51,10 @@ import (
        "sync"
        "time"
 
-       "github.com/ava-labs/coreth/accounts"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/accounts"
        pcsc "github.com/gballet/go-libpcsclite"
 )
 
diff --git a/accounts/scwallet/wallet.go b/accounts/scwallet/wallet.go
index fcecc10b..bdd7b3e2 100644
--- a/accounts/scwallet/wallet.go
+++ b/accounts/scwallet/wallet.go
@@ -43,12 +43,12 @@ import (
        "sync"
        "time"
 
-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/interfaces"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/interfaces"
        pcsc "github.com/gballet/go-libpcsclite"
        "github.com/status-im/keycard-go/derivationpath"
 )
diff --git a/chain/chain_test.go b/chain/chain_test.go
index 680e91b7..d418cd20 100644
--- a/chain/chain_test.go
+++ b/chain/chain_test.go
@@ -9,20 +9,21 @@ import (
        "math/rand"
        "testing"
 
-       "github.com/ava-labs/coreth/accounts/keystore"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/eth"
-       "github.com/ava-labs/coreth/eth/ethconfig"
-       "github.com/ava-labs/coreth/node"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/accounts/keystore"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/eth"
```

```
+	"github.com/flare-foundation/coreth/eth/ethconfig"
+	"github.com/flare-foundation/coreth/node"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 
 type testChain struct {
@@ -63,6 +64,7 @@ func newTestChain(name string, config *eth.Config,
 			},
 		},
 		common.Hash{},
+		&mockable.Clock{},
 	)
 	if err != nil {
 		t.Fatal(err)
diff --git a/chain/coreth.go b/chain/coreth.go
index 34543a98..d8e64045 100644
--- a/chain/coreth.go
+++ b/chain/coreth.go
@@ -7,15 +7,16 @@ import (
 	"fmt"
 	"time"
 
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/eth"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/node"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/eth"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/node"
+	"github.com/flare-foundation/coreth/rpc"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 
 var (
@@ -34,12 +35,12 @@ type ETHChain struct {
 }
 
 // NewETHChain creates an Ethereum blockchain with the given configs.
-func NewETHChain(config *eth.Config, nodecfg *node.Config, chainDB ethdb.Database, settings eth.Settings, consensusCallbacks *dummy.ConsensusCallbacks, lastAcceptedHash common.Hash) (*ETHChain, error) {
+func NewETHChain(config *eth.Config, nodecfg *node.Config, chainDB ethdb.Database, settings eth.Settings, consensusCallbacks *dummy.ConsensusCallbacks, lastAcceptedHash common.Hash, clock *mockable.Cloc
 	node, err := node.New(nodecfg)
 	if err != nil {
 		return nil, err
 	}
-	backend, err := eth.New(node, config, consensusCallbacks, chainDB, settings, lastAcceptedHash)
+	backend, err := eth.New(node, config, consensusCallbacks, chainDB, settings, lastAcceptedHash, clock)
 	if err != nil {
 		return nil, fmt.Errorf("failed to create backend: %w", err)
 	}
@@ -165,16 +166,33 @@ func (self *ETHChain) NewRPCHandler(maximumDuration time.Duration) *rpc.Server {
 	return rpc.NewServer(maximumDuration)
 }
 
-func (self *ETHChain) AttachEthService(handler *rpc.Server, namespaces []string) {
-	nsmap := make(map[string]bool)
-	for _, ns := range namespaces {
-		nsmap[ns] = true
+// AttachEthService registers the backend RPC services provided by Ethereum
+// to the provided handler under their assigned namespaces.
+func (self *ETHChain) AttachEthService(handler *rpc.Server, names []string) error {
+	enabledServicesSet := make(map[string]struct{})
+	for _, ns := range names {
+		enabledServicesSet[ns] = struct{}{}
 	}
+
+	apiSet := make(map[string]rpc.API)
 	for _, api := range self.backend.APIs() {
-		if nsmap[api.Namespace] {
-			handler.RegisterName(api.Namespace, api.Service)
+		if existingAPI, exists := apiSet[api.Name]; exists {
+			return fmt.Errorf("duplicated API name: %s, namespaces %s and %s", api.Name, api.Namespace, existingAPI.Namespace)
+		}
+		apiSet[api.Name] = api
+	}
+
+	for name := range enabledServicesSet {
+		api, exists := apiSet[name]
+		if !exists {
+			return fmt.Errorf("API service %s not found", name)
+		}
+		if err := handler.RegisterName(api.Namespace, api.Service); err != nil {
+			return err
 		}
 	}
+
+	return nil
 }
 
 func (self *ETHChain) GetTxSubmitCh() <-chan core.NewTxsEvent {
diff --git a/chain/counter_test.go b/chain/counter_test.go
index 7209ac51..af77d569 100644
--- a/chain/counter_test.go
+++ b/chain/counter_test.go
@@ -15,8 +15,8 @@ import (
 
 	"testing"
 
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 
 	"github.com/ethereum/go-ethereum/log"
 )
@@ -30,7 +30,7 @@ func TestCounter(t *testing.T) {
 
 	// NOTE: use precompiled `counter.sol` for portability, do not remove the
 	// following code (for debug purpose)
-	//counterSrc, err := filepath.Abs(gopath + "/src/github.com/ava-labs/coreth/examples/counter/counter.sol")
+	//counterSrc, err := filepath.Abs(gopath + "/src/github.com/flare-foundation/coreth/examples/counter/counter.sol")
 	// if err != nil {
 	// 	t.Fatal(err)
 	// }
diff --git a/chain/multicoin_test.go b/chain/multicoin_test.go
index a2ce4cb5..042c9c6e 100644
--- a/chain/multicoin_test.go
+++ b/chain/multicoin_test.go
@@ -28,19 +28,20 @@ import (
 	"strings"
 	"testing"
 
-	"github.com/ava-labs/coreth/accounts/keystore"
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/rawdb"
```

```
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/eth"
-	"github.com/ava-labs/coreth/eth/ethconfig"
-	"github.com/ava-labs/coreth/node"
 	"github.com/ethereum/go-ethereum/accounts/abi"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/accounts/keystore"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth"
+	"github.com/flare-foundation/coreth/eth/ethconfig"
+	"github.com/flare-foundation/coreth/node"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 
 // TestMulticoin tests multicoin low-level state management and regular
@@ -72,7 +73,7 @@ func TestMulticoin(t *testing.T) {
 	//if gopath == "" {
 	//	gopath = build.Default.GOPATH
 	//}
-	//counterSrc, err := filepath.Abs(gopath + "/src/github.com/ava-labs/coreth/examples/multicoin/mc_test.sol")
+	//counterSrc, err := filepath.Abs(gopath + "/src/github.com/flare-foundation/coreth/examples/multicoin/mc_test.sol")
 	//if err != nil {
 	//	t.Fatal(err)
 	// }
@@ -106,6 +107,7 @@ func TestMulticoin(t *testing.T) {
 		eth.DefaultSettings,
 		new(dummy.ConsensusCallbacks),
 		common.Hash{},
+		&mockable.Clock{},
 	)
 	if err != nil {
 		t.Fatal(err)
diff --git a/chain/payment_test.go b/chain/payment_test.go
index 5d9da4d8..7ae81fcf 100644
--- a/chain/payment_test.go
+++ b/chain/payment_test.go
@@ -7,8 +7,8 @@ import (
 	"math/big"
 	"testing"
 
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 // TestPayment tests basic payment (balance, not multi-coin)
diff --git a/chain/subscribe_accepted_heads_test.go b/chain/subscribe_accepted_heads_test.go
index 0a94bfde..db69c8fa 100644
--- a/chain/subscribe_accepted_heads_test.go
+++ b/chain/subscribe_accepted_heads_test.go
@@ -4,10 +4,10 @@ import (
 	"math/big"
 	"testing"
 
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 func TestAcceptedHeadSubscriptions(t *testing.T) {
diff --git a/chain/subscribe_block_logs_test.go b/chain/subscribe_block_logs_test.go
index 26661dad..186e6b66 100644
--- a/chain/subscribe_block_logs_test.go
+++ b/chain/subscribe_block_logs_test.go
@@ -6,10 +6,10 @@ import (
 	"testing"
 	"time"
 
-	"github.com/ava-labs/coreth/eth/filters"
+	"github.com/flare-foundation/coreth/eth/filters"
 
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 func TestBlockLogsAllowUnfinalized(t *testing.T) {
diff --git a/chain/subscribe_transactions_test.go b/chain/subscribe_transactions_test.go
index aac6db4a..9ed43937 100644
--- a/chain/subscribe_transactions_test.go
+++ b/chain/subscribe_transactions_test.go
@@ -4,10 +4,10 @@ import (
 	"math/big"
 	"testing"
 
-	"github.com/ava-labs/coreth/eth/filters"
+	"github.com/flare-foundation/coreth/eth/filters"
 
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 func TestSubscribeTransactions(t *testing.T) {
diff --git a/chain/test_chain.go b/chain/test_chain.go
index e4420de1..800316f6 100644
--- a/chain/test_chain.go
+++ b/chain/test_chain.go
@@ -8,17 +8,18 @@ import (
 	"math/big"
 	"testing"
 
-	"github.com/ava-labs/coreth/accounts/keystore"
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/eth"
-	"github.com/ava-labs/coreth/eth/ethconfig"
-	"github.com/ava-labs/coreth/node"
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
+	"github.com/flare-foundation/coreth/accounts/keystore"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/eth"
+	"github.com/flare-foundation/coreth/eth/ethconfig"
+	"github.com/flare-foundation/coreth/node"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 var (
@@ -88,6 +89,7 @@ func NewDefaultChain(t *testing.T) (*ETHChain, chan core.NewTxPoolHeadEvent, <-c
 		eth.DefaultSettings,
 		new(dummy.ConsensusCallbacks),
 		common.Hash{},
+		&mockable.Clock{},
 	)
 	if err != nil {
 		t.Fatal(err)
diff --git a/changes.sh b/changes.sh
new file mode 100755
index 00000000..b2b8d301
--- /dev/null
+++ b/changes.sh
@@ -0,0 +1,4 @@
+#!/bin/bash
+# Requires wkhtmltopdf and aha tools
+# Install using: sudo apt install wkhtmltopdf aha
+git diff --color upstream-v0.7.4-rc.1 | aha > CHANGES.html && wkhtmltopdf CHANGES.html CHANGES.pdf && rm CHANGES.html
\ No newline at end of file
diff --git a/cmd/abigen/main.go b/cmd/abigen/main.go
index ebb5b590..a7f2837f 100644
--- a/cmd/abigen/main.go
+++ b/cmd/abigen/main.go
@@ -35,13 +35,13 @@ import (
 	"regexp"
 	"strings"
 
-	"github.com/ava-labs/coreth/accounts/abi"
-	"github.com/ava-labs/coreth/accounts/abi/bind"
-	"github.com/ava-labs/coreth/internal/flags"
 	"github.com/ethereum/go-ethereum/cmd/utils"
 	"github.com/ethereum/go-ethereum/common/compiler"
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/accounts/abi"
+	"github.com/flare-foundation/coreth/accounts/abi/bind"
+	"github.com/flare-foundation/coreth/internal/flags"
 	"gopkg.in/urfave/cli.v1"
 )
 
diff --git a/consensus/consensus.go b/consensus/consensus.go
index 675be10f..cf9e4869 100644
--- a/consensus/consensus.go
+++ b/consensus/consensus.go
@@ -30,11 +30,10 @@ package consensus
 import (
 	"math/big"
 
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
 )
 
 // ChainHeaderReader defines a small collection of methods needed to access the local
@@ -73,8 +72,7 @@ type Engine interface {
 	Author(header *types.Header) (common.Address, error)
 
 	// VerifyHeader checks whether a header conforms to the consensus rules of a
-	// given engine. Verifying the seal may be done optionally here, or explicitly
-	// via the VerifySeal method.
+	// given engine.
 	//
 	// NOTE: VerifyHeader does not validate the correctness of fields that rely
 	// on the contents of the block (as opposed to the current and/or parent
@@ -85,10 +83,6 @@ type Engine interface {
 	// rules of a given engine.
 	VerifyUncles(chain ChainReader, block *types.Block) error
 
-	// VerifySeal checks whether the crypto seal on a header is valid according to
-	// the consensus rules of the given engine.
-	VerifySeal(chain ChainHeaderReader, header *types.Header) error
-
 	// Prepare initializes the consensus fields of a block header according to the
 	// rules of a particular engine. The changes are executed inline.
 	Prepare(chain ChainHeaderReader, header *types.Header) error
@@ -112,9 +106,6 @@ type Engine interface {
 	// that a new block should have.
 	CalcDifficulty(chain ChainHeaderReader, time uint64, parent *types.Header) *big.Int
 
-	// APIs returns the RPC APIs this consensus engine provides.
-	APIs(chain ChainHeaderReader) []rpc.API
-
 	// Close terminates any background threads maintained by the consensus engine.
 	Close() error
 }
diff --git a/consensus/dummy/consensus.go b/consensus/dummy/consensus.go
index 08cc229e..f965a019 100644
--- a/consensus/dummy/consensus.go
+++ b/consensus/dummy/consensus.go
@@ -10,13 +10,13 @@ import (
 	"math/big"
 	"time"
 
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/rpc"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 var (
@@ -31,20 +31,26 @@ var (
 	errExtDataGasUsedTooLarge = errors.New("extDataGasUsed is not uint64")
 )
 
+type Mode uint
+
+const (
+	ModeSkipHeader   Mode = 1 // Skip over header verification
+	ModeSkipBlockFee Mode = 2 // Skip block fee verification
+)
+
 type (
 	OnFinalizeAndAssembleCallbackType = func(header *types.Header, state *state.StateDB, txs []*types.Transaction) (extraData []byte, blockFeeContribution *big.Int, extDataGasUsed *big.Int, err error
 	OnAPIsCallbackType                = func(consensus.ChainHeaderReader) []rpc.API
 	OnExtraStateChangeType            = func(block *types.Block, statedb *state.StateDB) (blockFeeContribution *big.Int, extDataGasUsed *big.Int, err error)
 
 	ConsensusCallbacks struct {
-		OnAPIs                OnAPIsCallbackType
```

```diff
                OnFinalizeAndAssemble OnFinalizeAndAssembleCallbackType
                OnExtraStateChange    OnExtraStateChangeType
        }

        DummyEngine struct {
-               cb       *ConsensusCallbacks
-               ethFaker bool
+               cb            *ConsensusCallbacks
+               consensusMode Mode
        }
 )

@@ -56,15 +62,15 @@ func NewDummyEngine(cb *ConsensusCallbacks) *DummyEngine {

 func NewETHFaker() *DummyEngine {
        return &DummyEngine{
-               cb:       new(ConsensusCallbacks),
-               ethFaker: true,
+               cb:            new(ConsensusCallbacks),
+               consensusMode: ModeSkipBlockFee,
        }
 }

 func NewComplexETHFaker(cb *ConsensusCallbacks) *DummyEngine {
        return &DummyEngine{
-               cb:       cb,
-               ethFaker: true,
+               cb:            cb,
+               consensusMode: ModeSkipBlockFee,
        }
 }

@@ -72,19 +78,29 @@ func NewFaker() *DummyEngine {
        return NewDummyEngine(new(ConsensusCallbacks))
 }

+func NewFullFaker() *DummyEngine {
+       return &DummyEngine{
+               cb:            new(ConsensusCallbacks),
+               consensusMode: ModeSkipHeader,
+       }
+}
+
 func (self *DummyEngine) verifyHeaderGasFields(config *params.ChainConfig, header *types.Header, parent *types.Header) error {
        timestamp := new(big.Int).SetUint64(header.Time)

        // Verify that the gas limit is <= 2^63-1
-       cap := uint64(0x7fffffffffffffff)
-       if header.GasLimit > cap {
-               return fmt.Errorf("invalid gasLimit: have %v, max %v", header.GasLimit, cap)
+       if header.GasLimit > params.MaxGasLimit {
+               return fmt.Errorf("invalid gasLimit: have %v, max %v", header.GasLimit, params.MaxGasLimit)
        }
        // Verify that the gasUsed is <= gasLimit
        if header.GasUsed > header.GasLimit {
                return fmt.Errorf("invalid gasUsed: have %d, gasLimit %d", header.GasUsed, header.GasLimit)
        }
-       if config.IsApricotPhase1(timestamp) {
+       if config.IsApricotPhase5(timestamp) {
+               if header.GasLimit != params.ApricotPhase5GasLimit {
+                       return fmt.Errorf("expected gas limit to be %d, but found %d", params.ApricotPhase5GasLimit, header.GasLimit)
+               }
+       } else if config.IsApricotPhase1(timestamp) {
                if header.GasLimit != params.ApricotPhase1GasLimit {
                        return fmt.Errorf("expected gas limit to be %d, but found %d", params.ApricotPhase1GasLimit, header.GasLimit)
                }
@@ -138,12 +154,16 @@ func (self *DummyEngine) verifyHeaderGasFields(config *params.ChainConfig, heade
                return nil
        }

-       // Enforce Apricot Phase 4 constraints
+       // Enforce BlockGasCost constraints
+       blockGasCostStep := ApricotPhase4BlockGasCostStep
+       if config.IsApricotPhase5(timestamp) {
+               blockGasCostStep = ApricotPhase5BlockGasCostStep
+       }
        expectedBlockGasCost := calcBlockGasCost(
                ApricotPhase4TargetBlockRate,
                ApricotPhase4MinBlockGasCost,
                ApricotPhase4MaxBlockGasCost,
-               ApricotPhase4BlockGasCostStep,
+               blockGasCostStep,
                parent.BlockGasCost,
                parent.Time, header.Time,
        )
@@ -203,8 +223,7 @@ func (self *DummyEngine) verifyHeader(chain consensus.ChainHeaderReader, header
        if diff := new(big.Int).Sub(header.Number, parent.Number); diff.Cmp(big.NewInt(1)) != 0 {
                return consensus.ErrInvalidNumber
        }
-       // Verify the engine specific seal securing the block
-       return self.VerifySeal(chain, header)
+       return nil
 }

 func (self *DummyEngine) Author(header *types.Header) (common.Address, error) {
@@ -212,6 +231,10 @@ func (self *DummyEngine) Author(header *types.Header) (common.Address, error) {
 }

 func (self *DummyEngine) VerifyHeader(chain consensus.ChainHeaderReader, header *types.Header) error {
+       // If we're running a full engine faking, accept any input as valid
+       if self.consensusMode == ModeSkipHeader {
+               return nil
+       }
        // Short circuit if the header is known, or it's parent not
        number := header.Number.Uint64()
        if chain.GetHeader(header.Hash(), number) != nil {
@@ -232,10 +255,6 @@ func (self *DummyEngine) VerifyUncles(chain consensus.ChainReader, block *types.
        return nil
 }

-func (self *DummyEngine) VerifySeal(chain consensus.ChainHeaderReader, header *types.Header) error {
-       return nil
-}
-
 func (self *DummyEngine) Prepare(chain consensus.ChainHeaderReader, header *types.Header) error {
        header.Difficulty = big.NewInt(1)
        return nil
@@ -248,7 +267,7 @@ func (self *DummyEngine) verifyBlockFee(
        receipts []*types.Receipt,
        extraStateChangeContribution *big.Int,
 ) error {
-       if self.ethFaker {
+       if self.consensusMode == ModeSkipBlockFee {
                return nil
        }
        if baseFee == nil || baseFee.Sign() <= 0 {
@@ -324,11 +343,15 @@ func (self *DummyEngine) Finalize(chain consensus.ChainHeaderReader, block *type
                if blockExtDataGasUsed := block.ExtDataGasUsed(); blockExtDataGasUsed == nil || !blockExtDataGasUsed.IsUint64() || blockExtDataGasUsed.Cmp(extDataGasUsed) != 0 {
                        return fmt.Errorf("invalid extDataGasUsed: have %d, want %d", blockExtDataGasUsed, extDataGasUsed)
                }
+               blockGasCostStep := ApricotPhase4BlockGasCostStep
+               if chain.Config().IsApricotPhase5(new(big.Int).SetUint64(block.Time())) {
+                       blockGasCostStep = ApricotPhase5BlockGasCostStep
+               }
```

```diff
		blockGasCost := calcBlockGasCost(
			ApricotPhase4TargetBlockRate,
			ApricotPhase4MinBlockGasCost,
			ApricotPhase4MaxBlockGasCost,
-			ApricotPhase4BlockGasCostStep,
+			blockGasCostStep,
			parent.BlockGasCost,
			parent.Time, block.Time(),
		)
@@ -362,19 +385,20 @@ func (self *DummyEngine) FinalizeAndAssemble(chain consensus.ChainHeaderReader,
			return nil, err
		}
	}
-	if self.ethFaker {
-		extDataGasUsed = new(big.Int).Set(common.Big0)
-	}
	if chain.Config().IsApricotPhase4(new(big.Int).SetUint64(header.Time)) {
		header.ExtDataGasUsed = extDataGasUsed
		if header.ExtDataGasUsed == nil {
			header.ExtDataGasUsed = new(big.Int).Set(common.Big0)
		}
+		blockGasCostStep := ApricotPhase4BlockGasCostStep
+		if chain.Config().IsApricotPhase5(new(big.Int).SetUint64(header.Time)) {
+			blockGasCostStep = ApricotPhase5BlockGasCostStep
+		}
		header.BlockGasCost = calcBlockGasCost(
			ApricotPhase4TargetBlockRate,
			ApricotPhase4MinBlockGasCost,
			ApricotPhase4MaxBlockGasCost,
-			ApricotPhase4BlockGasCostStep,
+			blockGasCostStep,
			parent.BlockGasCost,
			parent.Time, header.Time,
		)
@@ -402,14 +426,6 @@ func (self *DummyEngine) CalcDifficulty(chain consensus.ChainHeaderReader, time
	return big.NewInt(1)
 }
 
-func (self *DummyEngine) APIs(chain consensus.ChainHeaderReader) (res []rpc.API) {
-	res = nil
-	if self.cb.OnAPIs != nil {
-		res = self.cb.OnAPIs(chain)
-	}
-	return
-}
-
 func (self *DummyEngine) Close() error {
	return nil
 }
diff --git a/consensus/dummy/consensus_test.go b/consensus/dummy/consensus_test.go
index 64a84398..5aa7a66a 100644
--- a/consensus/dummy/consensus_test.go
+++ b/consensus/dummy/consensus_test.go
@@ -8,8 +8,8 @@ import (
	"math/big"
	"testing"
 
-	"github.com/ava-labs/coreth/core/types"
	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 func TestVerifyBlockFee(t *testing.T) {
diff --git a/consensus/dummy/dynamic_fees.go b/consensus/dummy/dynamic_fees.go
index e9442487..95365092 100644
--- a/consensus/dummy/dynamic_fees.go
+++ b/consensus/dummy/dynamic_fees.go
@@ -8,24 +8,28 @@ import (
	"fmt"
	"math/big"
 
-	"github.com/ava-labs/avalanchego/utils/wrappers"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/common/math"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/flare/utils/wrappers"
 )
 
 var (
-	ApricotPhase3MinBaseFee              = big.NewInt(params.ApricotPhase3MinBaseFee)
-	ApricotPhase3MaxBaseFee              = big.NewInt(params.ApricotPhase3MaxBaseFee)
-	ApricotPhase4MinBaseFee              = big.NewInt(params.ApricotPhase4MinBaseFee)
-	ApricotPhase4MaxBaseFee              = big.NewInt(params.ApricotPhase4MaxBaseFee)
-	TargetGas                     uint64 = 10_000_000
+	ApricotPhase3MinBaseFee = big.NewInt(params.ApricotPhase3MinBaseFee)
+	ApricotPhase3MaxBaseFee = big.NewInt(params.ApricotPhase3MaxBaseFee)
+	ApricotPhase4MinBaseFee = big.NewInt(params.ApricotPhase4MinBaseFee)
+	ApricotPhase4MaxBaseFee = big.NewInt(params.ApricotPhase4MaxBaseFee)
+
+	ApricotPhase4BaseFeeChangeDenominator = new(big.Int).SetUint64(params.ApricotPhase4BaseFeeChangeDenominator)
+	ApricotPhase5BaseFeeChangeDenominator = new(big.Int).SetUint64(params.ApricotPhase5BaseFeeChangeDenominator)
+
	ApricotPhase3BlockGasFee      uint64 = 1_000_000
	ApricotPhase4MinBlockGasCost         = new(big.Int).Set(common.Big0)
	ApricotPhase4MaxBlockGasCost         = big.NewInt(1_000_000)
	ApricotPhase4BlockGasCostStep        = big.NewInt(50_000)
-	ApricotPhase4TargetBlockRate  uint64 = 2 // in seconds
+	ApricotPhase4TargetBlockRate  uint64 = 1 // in seconds
+	ApricotPhase5BlockGasCostStep        = big.NewInt(200_000)
	rollupWindow                  uint64 = 10
 )
 
@@ -40,6 +44,7 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
	var (
		isApricotPhase3 = config.IsApricotPhase3(bigTimestamp)
		isApricotPhase4 = config.IsApricotPhase4(bigTimestamp)
+		isApricotPhase5 = config.IsApricotPhase5(bigTimestamp)
	)
	if !isApricotPhase3 || parent.Number.Cmp(common.Big0) == 0 {
		initialSlice := make([]byte, params.ApricotPhase3ExtraDataSize)
@@ -62,12 +67,18 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
		return nil, nil, err
	}
 
+	// If AP5, use a less responsive [BaseFeeChangeDenominator] and a higher gas
+	// block limit
	var (
-		parentGasTarget          = TargetGas
-		parentGasTargetBig       = new(big.Int).SetUint64(parentGasTarget)
-		baseFeeChangeDenominator = new(big.Int).SetUint64(params.BaseFeeChangeDenominator)
		baseFee                  = new(big.Int).Set(parent.BaseFee)
+		baseFeeChangeDenominator = ApricotPhase4BaseFeeChangeDenominator
+		parentGasTarget          = params.ApricotPhase3TargetGas
	)
+	if isApricotPhase5 {
+		baseFeeChangeDenominator = ApricotPhase5BaseFeeChangeDenominator
+		parentGasTarget = params.ApricotPhase5TargetGas
+	}
+	parentGasTargetBig := new(big.Int).SetUint64(parentGasTarget)
 
	// Add in the gas used by the parent block in the correct place
	// If the parent consumed gas within the rollup window, add the consumed
@@ -75,7 +86,14 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
	if roll < rollupWindow {
		var blockGasCost, parentExtraStateGasUsed uint64
		switch {
-		// If ApricotPhase4 is enabled, use the updated block fee calculation.
+		case isApricotPhase5:
+			// [blockGasCost] has been removed in AP5, so it is left as 0.
+
+			// At the start of a new network, the parent
+			// may not have a populated [ExtDataGasUsed].
+			if parent.ExtDataGasUsed != nil {
+				parentExtraStateGasUsed = parent.ExtDataGasUsed.Uint64()
+			}
		case isApricotPhase4:
			// The [blockGasCost] is paid by the effective tips in the block using
			// the block's value of [baseFee].
@@ -88,12 +106,11 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
				parent.Time, timestamp,
			).Uint64()

-			// On the boundary of AP3 and AP4, the parent may not have a populated
-			// [ExtDataGasUsed].
+			// On the boundary of AP3 and AP4 or at the start of a new network, the parent
+			// may not have a populated [ExtDataGasUsed].
			if parent.ExtDataGasUsed != nil {
				parentExtraStateGasUsed = parent.ExtDataGasUsed.Uint64()
			}
-		// Otherwise, we must be in ApricotPhase3 and use the constant [ApricotPhase3BlockGasFee].
		default:
			blockGasCost = ApricotPhase3BlockGasFee
		}
@@ -103,10 +120,15 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
		if overflow {
			addedGas = math.MaxUint64
		}
-		addedGas, overflow = math.SafeAdd(addedGas, blockGasCost)
-		if overflow {
-			addedGas = math.MaxUint64
+
+		// Only add the [blockGasCost] to the gas used if it isn't AP5
+		if !isApricotPhase5 {
+			addedGas, overflow = math.SafeAdd(addedGas, blockGasCost)
+			if overflow {
+				addedGas = math.MaxUint64
+			}
		}
+
		slot := rollupWindow - 1 - roll
		start := slot * wrappers.LongLen
		updateLongWindow(newRollupWindow, start, addedGas)
@@ -129,7 +151,6 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
			common.Big1,
		)

-		// Gas price is increasing, so ensure it does not increase past the maximum
		baseFee.Add(baseFee, baseFeeDelta)
	} else {
		// Otherwise if the parent block used less gas than its target, the baseFee should decrease.
@@ -152,7 +173,10 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
		baseFee.Sub(baseFee, baseFeeDelta)
	}

+	// Ensure that the base fee does not increase/decrease outside of the bounds
	switch {
+	case isApricotPhase5:
+		baseFee = selectBigWithinBounds(ApricotPhase4MinBaseFee, baseFee, nil)
	case isApricotPhase4:
		baseFee = selectBigWithinBounds(ApricotPhase4MinBaseFee, baseFee, ApricotPhase4MaxBaseFee)
	default:
@@ -162,14 +186,26 @@ func CalcBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uin
	return newRollupWindow, baseFee, nil
 }

+// EstiamteNextBaseFee attempts to estimate the next base fee based on a block with [parent] being built at
+// [timestamp].
+// If [timestamp] is less than the timestamp of [parent], then it uses the same timestamp as parent.
+// Warning: This function should only be used in estimation and should not be used when calculating the canonical
+// base fee for a subsequent block.
+func EstimateNextBaseFee(config *params.ChainConfig, parent *types.Header, timestamp uint64) ([]byte, *big.Int, error) {
+	if timestamp < parent.Time {
+		timestamp = parent.Time
+	}
+	return CalcBaseFee(config, parent, timestamp)
+}
+
 // selectBigWithinBounds returns [value] if it is within the bounds:
 // lowerBound <= value <= upperBound or the bound at either end if [value]
 // is outside of the defined boundaries.
 func selectBigWithinBounds(lowerBound, value, upperBound *big.Int) *big.Int {
	switch {
-	case value.Cmp(lowerBound) < 0:
+	case lowerBound != nil && value.Cmp(lowerBound) < 0:
		return new(big.Int).Set(lowerBound)
-	case value.Cmp(upperBound) > 0:
+	case upperBound != nil && value.Cmp(upperBound) > 0:
		return new(big.Int).Set(upperBound)
	default:
		return value
diff --git a/consensus/dummy/dynamic_fees_test.go b/consensus/dummy/dynamic_fees_test.go
index 9bbfcc12..a41d9def 100644
--- a/consensus/dummy/dynamic_fees_test.go
+++ b/consensus/dummy/dynamic_fees_test.go
@@ -8,10 +8,10 @@ import (
	"math/big"
	"testing"

-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
	"github.com/ethereum/go-ethereum/common/math"
	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
	"github.com/stretchr/testify/assert"
 )

diff --git a/consensus/misc/dao.go b/consensus/misc/dao.go
index a0ab4029..be7c8df8 100644
--- a/consensus/misc/dao.go
+++ b/consensus/misc/dao.go
@@ -31,9 +31,9 @@ import (
	"errors"
	"math/big"

-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
 )

 var (
diff --git a/core/bench_test.go b/core/bench_test.go
```

```
new file mode 100644
index 00000000..bec5d025
--- /dev/null
+++ b/core/bench_test.go
@@ -0,0 +1,300 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2015 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       "crypto/ecdsa"
+       "io/ioutil"
+       "math/big"
+       "os"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/math"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
+)
+
+func BenchmarkInsertChain_empty_memdb(b *testing.B) {
+       benchInsertChain(b, false, nil)
+}
+func BenchmarkInsertChain_empty_diskdb(b *testing.B) {
+       benchInsertChain(b, true, nil)
+}
+func BenchmarkInsertChain_valueTx_memdb(b *testing.B) {
+       benchInsertChain(b, false, genValueTx(0))
+}
+func BenchmarkInsertChain_valueTx_diskdb(b *testing.B) {
+       benchInsertChain(b, true, genValueTx(0))
+}
+func BenchmarkInsertChain_valueTx_100kB_memdb(b *testing.B) {
+       benchInsertChain(b, false, genValueTx(100*1024))
+}
+func BenchmarkInsertChain_valueTx_100kB_diskdb(b *testing.B) {
+       benchInsertChain(b, true, genValueTx(100*1024))
+}
+
+func BenchmarkInsertChain_ring200_memdb(b *testing.B) {
+       benchInsertChain(b, false, genTxRing(200))
+}
+func BenchmarkInsertChain_ring200_diskdb(b *testing.B) {
+       benchInsertChain(b, true, genTxRing(200))
+}
+func BenchmarkInsertChain_ring1000_memdb(b *testing.B) {
+       benchInsertChain(b, false, genTxRing(1000))
+}
+func BenchmarkInsertChain_ring1000_diskdb(b *testing.B) {
+       benchInsertChain(b, true, genTxRing(1000))
+}
+
+var (
+       // This is the content of the genesis block used by the benchmarks.
+       benchRootKey, _ = crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291")
+       benchRootAddr   = crypto.PubkeyToAddress(benchRootKey.PublicKey)
+       benchRootFunds  = math.BigPow(2, 100)
+)
+
+// genValueTx returns a block generator that includes a single
+// value-transfer transaction with n bytes of extra data in each
+// block.
+func genValueTx(nbytes int) func(int, *BlockGen) {
+       return func(i int, gen *BlockGen) {
+               toaddr := common.Address{}
+               data := make([]byte, nbytes)
+               gas, _ := IntrinsicGas(data, nil, false, false, false)
+               tx, _ := types.SignTx(types.NewTransaction(gen.TxNonce(benchRootAddr), toaddr, big.NewInt(1), gas, big.NewInt(225000000000), data), types.HomesteadSigner{}, benchRootKey)
+               gen.AddTx(tx)
+       }
+}
+
+var (
+       ringKeys  = make([]*ecdsa.PrivateKey, 1000)
+       ringAddrs = make([]common.Address, len(ringKeys))
+)
+
+func init() {
+       ringKeys[0] = benchRootKey
+       ringAddrs[0] = benchRootAddr
+       for i := 1; i < len(ringKeys); i++ {
+               ringKeys[i], _ = crypto.GenerateKey()
+               ringAddrs[i] = crypto.PubkeyToAddress(ringKeys[i].PublicKey)
+       }
+}
+
+// genTxRing returns a block generator that sends ether in a ring
+// among n accounts. This is creates n entries in the state database
+// and fills the blocks with many small transactions.
+func genTxRing(naccounts int) func(int, *BlockGen) {
+       from := 0
+       fee := big.NewInt(0).SetUint64(params.TxGas * 225000000000)
+       amount := big.NewInt(0).Set(benchRootFunds)
+       return func(i int, gen *BlockGen) {
+               block := gen.PrevBlock(i - 1)
+               gas := block.GasLimit()
+               for {
+                       gas -= params.TxGas
+                       if gas < params.TxGas {
+                               break
+                       }
+                       to := (from + 1) % naccounts
+                       tx := types.NewTransaction(
+                               gen.TxNonce(ringAddrs[from]),
+                               ringAddrs[to],
+                               amount.Sub(amount, fee),
+                               params.TxGas,
+                               big.NewInt(225000000000),
+                               nil,
+                       )
+                       tx, _ = types.SignTx(tx, types.HomesteadSigner{}, ringKeys[from])
+                       gen.AddTx(tx)
+                       from = to
+               }
+       }
+}
+
+func benchInsertChain(b *testing.B, disk bool, gen func(int, *BlockGen)) {
+       // Create the database in memory or in a temporary directory.
+       var db ethdb.Database
+       if !disk {
+               db = rawdb.NewMemoryDatabase()
+       } else {
+               dir, err := ioutil.TempDir("", "eth-core-bench")
+               if err != nil {
+                       b.Fatalf("cannot create temporary directory: %v", err)
+               }
+               defer os.RemoveAll(dir)
+               db, err = rawdb.NewLevelDBDatabase(dir, 128, 128, "", false)
+               if err != nil {
+                       b.Fatalf("cannot create temporary database: %v", err)
+               }
+               defer db.Close()
+       }
+
+       // Generate a chain of b.N blocks using the supplied block
+       // generator function.
+       gspec := Genesis{
+               Config: params.TestChainConfig,
+               Alloc:  GenesisAlloc{benchRootAddr: {Balance: benchRootFunds}},
+       }
+       genesis := gspec.MustCommit(db)
+       chain, _, _ := GenerateChain(gspec.Config, genesis, dummy.NewFaker(), db, b.N, 10, gen)
+
+       // Time the insertion of the new chain.
+       // State and blocks are stored in the same DB.
+       chainman, _ := NewBlockChain(db, DefaultCacheConfig, gspec.Config, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       defer chainman.Stop()
+       b.ReportAllocs()
+       b.ResetTimer()
+       if i, err := chainman.InsertChain(chain); err != nil {
+               b.Fatalf("insert error (block %d): %v\n", i, err)
+       }
+}
+
+func BenchmarkChainRead_header_10k(b *testing.B) {
+       benchReadChain(b, false, 10000)
+}
+func BenchmarkChainRead_full_10k(b *testing.B) {
+       benchReadChain(b, true, 10000)
+}
+func BenchmarkChainRead_header_100k(b *testing.B) {
+       benchReadChain(b, false, 100000)
+}
+func BenchmarkChainRead_full_100k(b *testing.B) {
+       benchReadChain(b, true, 100000)
+}
+func BenchmarkChainRead_header_500k(b *testing.B) {
+       benchReadChain(b, false, 500000)
+}
+func BenchmarkChainRead_full_500k(b *testing.B) {
+       benchReadChain(b, true, 500000)
+}
+func BenchmarkChainWrite_header_10k(b *testing.B) {
+       benchWriteChain(b, false, 10000)
+}
+func BenchmarkChainWrite_full_10k(b *testing.B) {
+       benchWriteChain(b, true, 10000)
+}
+func BenchmarkChainWrite_header_100k(b *testing.B) {
+       benchWriteChain(b, false, 100000)
+}
+func BenchmarkChainWrite_full_100k(b *testing.B) {
+       benchWriteChain(b, true, 100000)
+}
+func BenchmarkChainWrite_header_500k(b *testing.B) {
+       benchWriteChain(b, false, 500000)
+}
+func BenchmarkChainWrite_full_500k(b *testing.B) {
+       benchWriteChain(b, true, 500000)
+}
+
+// makeChainForBench writes a given number of headers or empty blocks/receipts
+// into a database.
+func makeChainForBench(db ethdb.Database, full bool, count uint64) {
+       var hash common.Hash
+       for n := uint64(0); n < count; n++ {
+               header := &types.Header{
+                       Coinbase:    common.Address{},
+                       Number:      big.NewInt(int64(n)),
+                       ParentHash:  hash,
+                       Difficulty:  big.NewInt(1),
+                       UncleHash:   types.EmptyUncleHash,
+                       TxHash:      types.EmptyRootHash,
+                       ReceiptHash: types.EmptyRootHash,
+               }
+               hash = header.Hash()
+
+               rawdb.WriteHeader(db, header)
+               rawdb.WriteCanonicalHash(db, hash, n)
+
+               if full || n == 0 {
+                       block := types.NewBlockWithHeader(header)
+                       rawdb.WriteBody(db, hash, n, block.Body())
+                       rawdb.WriteReceipts(db, hash, n, nil)
+               }
+       }
+}
+
+func benchWriteChain(b *testing.B, full bool, count uint64) {
+       for i := 0; i < b.N; i++ {
+               dir, err := ioutil.TempDir("", "eth-chain-bench")
+               if err != nil {
+                       b.Fatalf("cannot create temporary directory: %v", err)
+               }
+               db, err := rawdb.NewLevelDBDatabase(dir, 128, 1024, "", false)
+               if err != nil {
+                       b.Fatalf("error opening database at %v: %v", dir, err)
+               }
+               makeChainForBench(db, full, count)
+               db.Close()
+               os.RemoveAll(dir)
+       }
+}
+
+func benchReadChain(b *testing.B, full bool, count uint64) {
+       dir, err := ioutil.TempDir("", "eth-chain-bench")
```

```diff
+	if err != nil {
+		b.Fatalf("cannot create temporary directory: %v", err)
+	}
+	defer os.RemoveAll(dir)
+
+	db, err := rawdb.NewLevelDBDatabase(dir, 128, 1024, "", false)
+	if err != nil {
+		b.Fatalf("error opening database at %v: %v", dir, err)
+	}
+	makeChainForBench(db, full, count)
+	db.Close()
+
+	b.ReportAllocs()
+	b.ResetTimer()
+
+	for i := 0; i < b.N; i++ {
+		db, err := rawdb.NewLevelDBDatabase(dir, 128, 1024, "", false)
+		if err != nil {
+			b.Fatalf("error opening database at %v: %v", dir, err)
+		}
+		chain, err := NewBlockChain(db, DefaultCacheConfig, params.TestChainConfig, dummy.NewFaker(), vm.Config{}, common.Hash{})
+		if err != nil {
+			b.Fatalf("error creating chain: %v", err)
+		}
+
+		for n := uint64(0); n < count; n++ {
+			header := chain.GetHeaderByNumber(n)
+			if full {
+				hash := header.Hash()
+				rawdb.ReadBody(db, hash, n)
+				rawdb.ReadReceipts(db, hash, n, chain.Config())
+			}
+		}
+		chain.Stop()
+		db.Close()
+	}
+}
diff --git a/core/block_validator.go b/core/block_validator.go
index 287a42fb..59545436 100644
--- a/core/block_validator.go
+++ b/core/block_validator.go
@@ -29,11 +29,11 @@ package core
 import (
 	"fmt"
 
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/trie"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 // BlockValidator is responsible for validating block headers, uncles and
diff --git a/core/block_validator_test.go b/core/block_validator_test.go
new file mode 100644
index 00000000..1ea33274
--- /dev/null
+++ b/core/block_validator_test.go
@@ -0,0 +1,63 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2015 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+	"testing"
+)
+
+func TestCalcGasLimit(t *testing.T) {
+	for i, tc := range []struct {
+		pGasLimit uint64
+		max       uint64
+		min       uint64
+	}{
+		{20000000, 20019530, 19980470},
+		{40000000, 40039061, 39960939},
+	} {
+		// Increase
+		if have, want := CalcGasLimit(0, tc.pGasLimit, 2*tc.pGasLimit, 2*tc.pGasLimit), tc.max; have != want {
+			t.Errorf("test %d: have %d want <%d", i, have, want)
+		}
+		// Decrease
+		if have, want := CalcGasLimit(0, tc.pGasLimit, 0, 0), tc.min; have != want {
+			t.Errorf("test %d: have %d want >%d", i, have, want)
+		}
+		// Small decrease
+		if have, want := CalcGasLimit(0, tc.pGasLimit, tc.pGasLimit-1, tc.pGasLimit-1), tc.pGasLimit-1; have != want {
+			t.Errorf("test %d: have %d want %d", i, have, want)
+		}
+		// Small increase
+		if have, want := CalcGasLimit(0, tc.pGasLimit, tc.pGasLimit+1, tc.pGasLimit+1), tc.pGasLimit+1; have != want {
+			t.Errorf("test %d: have %d want %d", i, have, want)
+		}
+		// No change
+		if have, want := CalcGasLimit(0, tc.pGasLimit, tc.pGasLimit, tc.pGasLimit), tc.pGasLimit; have != want {
+			t.Errorf("test %d: have %d want %d", i, have, want)
+		}
+	}
+}
diff --git a/core/blockchain.go b/core/blockchain.go
index 9e46bcd1..a6179a34 100644
--- a/core/blockchain.go
+++ b/core/blockchain.go
@@ -37,18 +37,18 @@ import (
 	"sync/atomic"
 	"time"
```

```
-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/state/snapshot"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/trie"
        lru "github.com/hashicorp/golang-lru"
 )

@@ -231,11 +231,12 @@ func NewBlockChain(
        var nilBlock *types.Block
        bc.currentBlock.Store(nilBlock)

+       // Create the state manager
+       bc.stateManager = NewTrieWriter(bc.stateCache.TrieDB(), cacheConfig)
+
        if err := bc.loadLastState(lastAcceptedHash); err != nil {
                return nil, err
        }
-       // Create the state manager
-       bc.stateManager = NewTrieWriter(bc.stateCache.TrieDB(), cacheConfig)

        // Make sure the state associated with the block is available
        head := bc.CurrentBlock()
@@ -264,17 +265,6 @@ func (bc *BlockChain) SenderCacher() *TxSenderCacher {
        return bc.senderCacher
 }

-// empty returns an indicator whether the blockchain is empty.
-func (bc *BlockChain) empty() bool {
-       genesis := bc.genesisBlock.Hash()
-       for _, hash := range []common.Hash{rawdb.ReadHeadBlockHash(bc.db), rawdb.ReadHeadHeaderHash(bc.db), rawdb.ReadHeadFastBlockHash(bc.db)} {
-               if hash != genesis {
-                       return false
-               }
-       }
-       return true
-}
-
 // loadLastState loads the last known chain state from the database. This method
 // assumes that the chain manager mutex is held.
 func (bc *BlockChain) loadLastState(lastAcceptedHash common.Hash) error {
@@ -305,11 +295,8 @@ func (bc *BlockChain) loadLastState(lastAcceptedHash common.Hash) error {
        }
        bc.hc.SetCurrentHeader(currentHeader)

-       headerTd := bc.GetTd(currentHeader.Hash(), currentHeader.Number.Uint64())
-       blockTd := bc.GetTd(currentBlock.Hash(), currentBlock.NumberU64())
-
-       log.Info("Loaded most recent local header", "number", currentHeader.Number, "hash", currentHeader.Hash(), "td", headerTd, "age", common.PrettyAge(time.Unix(int64(currentHeader.Time), 0)))
-       log.Info("Loaded most recent local full block", "number", currentBlock.Number(), "hash", currentBlock.Hash(), "td", blockTd, "age", common.PrettyAge(time.Unix(int64(currentBlock.Time()), 0)))
+       log.Info("Loaded most recent local header", "number", currentHeader.Number, "hash", currentHeader.Hash(), "age", common.PrettyAge(time.Unix(int64(currentHeader.Time), 0)))
+       log.Info("Loaded most recent local full block", "number", currentBlock.Number(), "hash", currentBlock.Hash(), "age", common.PrettyAge(time.Unix(int64(currentBlock.Time()), 0)))

        // Otherwise, set the last accepted block and perform a re-org.
        bc.lastAccepted = bc.GetBlockByHash(lastAcceptedHash)
@@ -339,10 +326,10 @@ func (bc *BlockChain) loadLastState(lastAcceptedHash common.Hash) error {
                return fmt.Errorf("failed to set preference to last accepted block while loading last state: %w", err)
        }

-       // reprocessState as necessary to ensure that the last accepted state is
+       // reprocessState is necessary to ensure that the last accepted state is
        // available. The state may not be available if it was not committed due
        // to an unclean shutdown.
-       return bc.reprocessState(bc.lastAccepted, 2*commitInterval, true)
+       return bc.reprocessState(bc.lastAccepted, 2*commitInterval)
 }

 // removeIndices removes all transaction lookup entries for the transactions contained in the canonical chain
@@ -369,7 +356,6 @@ func (bc *BlockChain) removeIndices(from, to uint64) (int, error) {
 func (bc *BlockChain) loadGenesisState() error {
        // Prepare the genesis block and reinitialise the chain
        batch := bc.db.NewBatch()
-       rawdb.WriteTd(batch, bc.genesisBlock.Hash(), bc.genesisBlock.NumberU64(), bc.genesisBlock.Difficulty())
        rawdb.WriteBlock(batch, bc.genesisBlock)
        if err := batch.Write(); err != nil {
                log.Crit("Failed to write genesis block", "err", err)
@@ -418,32 +404,24 @@ func (bc *BlockChain) ExportN(w io.Writer, first uint64, last uint64) error {

 // writeHeadBlock injects a new head block into the current block chain. This method
 // assumes that the block is indeed a true head. It will also reset the head
-// header and the head fast sync block to this very same block if they are older
-// or if they are on a different side chain.
+// header to this very same block if they are older or if they are on a different side chain.
 //
 // Note, this function assumes that the `mu` mutex is held!
 func (bc *BlockChain) writeHeadBlock(block *types.Block) {
        // If the block is on a side chain or an unknown one, force other heads onto it too
-       updateHeads := rawdb.ReadCanonicalHash(bc.db, block.NumberU64()) != block.Hash()
-
        // Add the block to the canonical chain number scheme and mark as the head
        batch := bc.db.NewBatch()
        rawdb.WriteCanonicalHash(batch, block.Hash(), block.NumberU64())
+
        rawdb.WriteHeadBlockHash(batch, block.Hash())
+       rawdb.WriteHeadHeaderHash(batch, block.Hash())

-       // If the block is better than our head or is on a different chain, force update heads
-       if updateHeads {
-               rawdb.WriteHeadHeaderHash(batch, block.Hash())
-               rawdb.WriteHeadFastBlockHash(batch, block.Hash())
-       }
        // Flush the whole batch into the disk, exit the node if failed
        if err := batch.Write(); err != nil {
                log.Crit("Failed to update chain indexes and markers", "err", err)
        }
        // Update all in-memory chain markers in the last step
-       if updateHeads {
-               bc.hc.SetCurrentHeader(block.Header())
-       }
+       bc.hc.SetCurrentHeader(block.Header())
        bc.currentBlock.Store(block)
 }

@@ -561,15 +539,6 @@ func (bc *BlockChain) Stop() {
        log.Info("Blockchain stopped")
 }
```

```diff
-// WriteStatus status of write
-type WriteStatus byte
-
-const (
-       NonStatTy WriteStatus = iota
-       CanonStatTy
-       SideStatTy
-)
-
 // SetPreference attempts to update the head block to be the provided block and
 // emits a ChainHeadEvent if successful. This function will handle all reorg
 // side effects, if necessary.
@@ -607,7 +576,7 @@ func (bc *BlockChain) setPreference(block *types.Block) error {
                return fmt.Errorf("unable to invoke writeKnownBlock: %w", err)
        }
 
-       // Send an ChainHeadEvent if we end up altering
+       // Send a ChainHeadEvent if we end up altering
        // the head block. Many internal aysnc processes rely on
        // receiving these events (i.e. the TxPool).
        bc.chainHeadFeed.Send(ChainHeadEvent{Block: block})
@@ -752,30 +721,38 @@ func (bc *BlockChain) newTip(block *types.Block) bool {
        return block.ParentHash() == bc.CurrentBlock().Hash()
 }
 
+// writeBlockAndSetHead persists the block and associated state to the database
+// and optimistically updates the canonical chain if [block] extends the current
+// canonical chain.
+// writeBlockAndSetHead expects to be the last verification step during InsertBlock
+// since it creates a reference that will only be cleaned up by Accept/Reject.
+func (bc *BlockChain) writeBlockAndSetHead(block *types.Block, receipts []*types.Receipt, logs []*types.Log, state *state.StateDB) error {
+       if err := bc.writeBlockWithState(block, receipts, logs, state); err != nil {
+               return err
+       }
+
+       // If [block] represents a new tip of the canonical chain, we optimistically add it before
+       // setPreference is called. Otherwise, we consider it a side chain block.
+       if bc.newTip(block) {
+               bc.writeCanonicalBlockWithLogs(block, logs)
+       } else {
+               bc.chainSideFeed.Send(ChainSideEvent{Block: block})
+       }
+
+       return nil
+}
+
 // writeBlockWithState writes the block and all associated state to the database,
 // but it expects the chain mutex to be held.
-// writeBlockWithState expects to be the last verification step during InsertBlock
-// since it creates a reference that will only be cleaned up by Accept/Reject.
-func (bc *BlockChain) writeBlockWithState(block *types.Block, receipts []*types.Receipt, logs []*types.Log, state *state.StateDB) (WriteStatus, error) {
+func (bc *BlockChain) writeBlockWithState(block *types.Block, receipts []*types.Receipt, logs []*types.Log, state *state.StateDB) error {
        bc.wg.Add(1)
        defer bc.wg.Done()
 
-       // Calculate the total difficulty of the block
-       ptd := bc.GetTd(block.ParentHash(), block.NumberU64()-1)
-       if ptd == nil {
-               return NonStatTy, consensus.ErrUnknownAncestor
-       }
-       // Make sure no inconsistent state is leaked during insertion
-       // currentBlock := bc.CurrentBlock()
-       // localTd := bc.GetTd(currentBlock.Hash(), currentBlock.NumberU64())
-       externTd := new(big.Int).Add(block.Difficulty(), ptd)
-
        // Irrelevant of the canonical status, write the block itself to the database.
        //
        // Note all the components of block(td, hash->number map, header, body, receipts)
        // should be written atomically. BlockBatch is used for containing all components.
        blockBatch := bc.db.NewBatch()
-       rawdb.WriteTd(blockBatch, block.Hash(), block.NumberU64(), externTd)
        rawdb.WriteBlock(blockBatch, block)
        rawdb.WriteReceipts(blockBatch, block.Hash(), block.NumberU64(), receipts)
        rawdb.WritePreimages(blockBatch, state.Preimages())
@@ -792,7 +769,7 @@ func (bc *BlockChain) writeBlockWithState(block *types.Block, receipts []*types.
                _, err = state.CommitWithSnap(bc.chainConfig.IsEIP158(block.Number()), bc.snaps, block.Hash(), block.ParentHash())
        }
        if err != nil {
-               return NonStatTy, err
+               return err
        }
 
        // Note: if InsertTrie must be the last step in verification that can return an error.
@@ -807,18 +784,10 @@ func (bc *BlockChain) writeBlockWithState(block *types.Block, receipts []*types.
                                log.Debug("failed to discard snapshot after being unable to insert block trie", "block", block.Hash(), "root", block.Root())
                        }
                }
-               return NonStatTy, err
-       }
-
-       // If [block] represents a new tip of the canonical chain, we optimistically add it before
-       // setPreference is called. Otherwise, we consider it a side chain block.
-       if bc.newTip(block) {
-               bc.writeCanonicalBlockWithLogs(block, logs)
-               return CanonStatTy, nil
+               return err
        }
 
-       bc.chainSideFeed.Send(ChainSideEvent{Block: block})
-       return SideStatTy, nil
+       return nil
 }
 
 // InsertChain attempts to insert the given batch of blocks in to the canonical
@@ -992,42 +961,46 @@ func (bc *BlockChain) insertBlock(block *types.Block, writes bool) error {
        }
 
        // Write the block to the chain and get the status.
-       // writeBlockWithState creates a reference that will be cleaned up in Accept/Reject
-       // so we need to ensure an error cannot occur later in verification, since that would
-       // cause the referenced root to never be dereferenced.
-       status, err := bc.writeBlockWithState(block, receipts, logs, statedb)
-       if err != nil {
+       // writeBlockWithState (called within writeBlockAndSethead) creates a reference that
+       // will be cleaned up in Accept/Reject so we need to ensure an error cannot occur
+       // later in verification, since that would cause the referenced root to never be dereferenced.
+       if err := bc.writeBlockAndSetHead(block, receipts, logs, statedb); err != nil {
                return err
        }
+       log.Debug("Inserted new block", "number", block.Number(), "hash", block.Hash(),
+               "parentHash", block.ParentHash(),
+               "uncles", len(block.Uncles()), "txs", len(block.Transactions()), "gas", block.GasUsed(),
+               "elapsed", common.PrettyDuration(time.Since(start)),
+               "root", block.Root(), "baseFeePerGas", block.BaseFee(), "blockGasCost", block.BlockGasCost(),
+       )
 
-       switch status {
-       case CanonStatTy:
-               log.Debug("Inserted new block", "number", block.Number(), "hash", block.Hash(),
-                       "parentHash", block.ParentHash(),
-                       "uncles", len(block.Uncles()), "txs", len(block.Transactions()), "gas", block.GasUsed(),
-                       "elapsed", common.PrettyDuration(time.Since(start)),
-                       "root", block.Root(), "baseFeePerGas", block.BaseFee(), "blockGasCost", block.BlockGasCost(),
-               )
-               // Only count canonical blocks for GC processing time
-       case SideStatTy:
-               log.Debug("Inserted forked block", "number", block.Number(), "hash", block.Hash(),
-                       "parentHash", block.ParentHash(),
-                       "diff", block.Difficulty(), "elapsed", common.PrettyDuration(time.Since(start)),
-                       "txs", len(block.Transactions()), "gas", block.GasUsed(), "uncles", len(block.Uncles()),
-                       "root", block.Root(), "baseFeePerGas", block.BaseFee(), "blockGasCost", block.BlockGasCost(),
-               )
-       default:
-               // This in theory is impossible, but lets be nice to our future selves and leave
-               // a log, instead of trying to track down blocks imports that don't emit logs.
-               log.Warn("Inserted block with unknown status", "number", block.Number(), "hash", block.Hash(),
-                       "parentHash", block.ParentHash(),
-                       "diff", block.Difficulty(), "elapsed", common.PrettyDuration(time.Since(start)),
-                       "txs", len(block.Transactions()), "gas", block.GasUsed(), "uncles", len(block.Uncles()),
-                       "root", block.Root(), "baseFeePerGas", block.BaseFee(), "blockGasCost", block.BlockGasCost(),
-               )
+       return nil
+}
+
+// collectLogs collects the logs that were generated or removed during
+// the processing of the block that corresponds with the given hash.
+// These logs are later announced as deleted or reborn.
+func (bc *BlockChain) collectLogs(hash common.Hash, removed bool) []*types.Log {
+       number := bc.hc.GetBlockNumber(hash)
+       if number == nil {
+               return nil
        }
+       return bc.gatherBlockLogs(hash, *number, removed)
+}

-       return err
+// mergeLogs returns a merged log slice with specified sort order.
+func mergeLogs(logs [][]*types.Log, reverse bool) []*types.Log {
+       var ret []*types.Log
+       if reverse {
+               for i := len(logs) - 1; i >= 0; i-- {
+                       ret = append(ret, logs[i]...)
+               }
+       } else {
+               for i := 0; i < len(logs); i++ {
+                       ret = append(ret, logs[i]...)
+               }
+       }
+       return ret
 }

 // reorg takes two blocks, an old chain and a new chain and will reconstruct the
@@ -1044,45 +1017,17 @@ func (bc *BlockChain) reorg(oldBlock, newBlock *types.Block) error {

                deletedLogs [][]*types.Log
                rebirthLogs [][]*types.Log
-
-               // collectLogs collects the logs that were generated or removed during
-               // the processing of the block that corresponds with the given hash.
-               // These logs are later announced as deleted or reborn
-               collectLogs = func(hash common.Hash, removed bool) {
-                       number := bc.hc.GetBlockNumber(hash)
-                       if number == nil {
-                               return
-                       }
-                       logs := bc.gatherBlockLogs(hash, *number, removed)
-                       if len(logs) > 0 {
-                               if removed {
-                                       deletedLogs = append(deletedLogs, logs)
-                               } else {
-                                       rebirthLogs = append(rebirthLogs, logs)
-                               }
-                       }
-               }
-               // mergeLogs returns a merged log slice with specified sort order.
-               mergeLogs = func(logs [][]*types.Log, reverse bool) []*types.Log {
-                       var ret []*types.Log
-                       if reverse {
-                               for i := len(logs) - 1; i >= 0; i-- {
-                                       ret = append(ret, logs[i]...)
-                               }
-                       } else {
-                               for i := 0; i < len(logs); i++ {
-                                       ret = append(ret, logs[i]...)
-                               }
-                       }
-                       return ret
-               }
        )
        // Reduce the longer chain to the same number as the shorter one
        if oldBlock.NumberU64() > newBlock.NumberU64() {
                // Old chain is longer, gather all transactions and logs as deleted ones
                for ; oldBlock != nil && oldBlock.NumberU64() != newBlock.NumberU64(); oldBlock = bc.GetBlock(oldBlock.ParentHash(), oldBlock.NumberU64()-1) {
                        oldChain = append(oldChain, oldBlock)
-                       collectLogs(oldBlock.Hash(), true)
+                       // Collect deleted logs for notification
+                       logs := bc.collectLogs(oldBlock.Hash(), true)
+                       if len(logs) > 0 {
+                               deletedLogs = append(deletedLogs, logs)
+                       }
                }
        } else {
                // New chain is longer, stash all blocks away for subsequent insertion
@@ -1106,7 +1051,11 @@ func (bc *BlockChain) reorg(oldBlock, newBlock *types.Block) error {
                }
                // Remove an old block as well as stash away a new block
                oldChain = append(oldChain, oldBlock)
-               collectLogs(oldBlock.Hash(), true)
+               // Collect deleted logs for notification
+               logs := bc.collectLogs(oldBlock.Hash(), true)
+               if len(logs) > 0 {
+                       deletedLogs = append(deletedLogs, logs)
+               }

                newChain = append(newChain, newBlock)

@@ -1131,15 +1080,15 @@ func (bc *BlockChain) reorg(oldBlock, newBlock *types.Block) error {
        // Ensure the user sees large reorgs
        if len(oldChain) > 0 && len(newChain) > 0 {
                logFn := log.Info
-               msg := "Chain reorg detected"
+               msg := "Resetting chain preference"
                if len(oldChain) > 63 {
-                       msg = "Large chain reorg detected"
+                       msg = "Large chain preference change detected"
                        logFn = log.Warn
                }
                logFn(msg, "number", commonBlock.Number(), "hash", commonBlock.Hash(),
                        "drop", len(oldChain), "dropfrom", oldChain[0].Hash(), "add", len(newChain), "addfrom", newChain[0].Hash())
        } else {
-               log.Warn("Unlikely reorg (rewind to ancestor) occurred", "oldnum", oldHead.Number(), "oldhash", oldHead.Hash(), "newnum", newHead.Number(), "newhash", newHead.Hash())
+               log.Warn("Unlikely preference change (rewind to ancestor) occurred", "oldnum", oldHead.Number(), "oldhash", oldHead.Hash(), "newnum", newHead.Number(), "newhash", newHead.Hash())
        }
        // Insert the new chain(except the head block(reverse order)),
        // taking care of the proper incremental order.
@@ -1148,7 +1097,10 @@ func (bc *BlockChain) reorg(oldBlock, newBlock *types.Block) error {
                bc.writeHeadBlock(newChain[i])
```

```diff
 		// Collect reborn logs due to chain reorg
-		collectLogs(newChain[i].Hash(), false)
+		logs := bc.collectLogs(newChain[i].Hash(), false)
+		if len(logs) > 0 {
+			rebirthLogs = append(rebirthLogs, logs)
+		}
 	}
 	// Delete any canonical number assignments above the new head
 	indexesBatch := bc.db.NewBatch()
@@ -1255,7 +1207,7 @@ func (bc *BlockChain) RemoveRejectedBlocks(start, end uint64) error {
 // it reaches a block with a state committed to the database. reprocessState does not use
 // snapshots since the disk layer for snapshots will most likely be above the last committed
 // state that reprocessing will start from.
-func (bc *BlockChain) reprocessState(current *types.Block, reexec uint64, report bool) error {
+func (bc *BlockChain) reprocessState(current *types.Block, reexec uint64) error {
 	var (
 		origin = current.NumberU64()
 	)
@@ -1300,7 +1252,7 @@ func (bc *BlockChain) reprocessState(current *types.Block, reexec uint64, report
 	log.Info("Re-executing blocks to generate state for last accepted block", "from", current.NumberU64()+1, "to", origin)
 	for current.NumberU64() < origin {
 		// Print progress logs if long enough time elapsed
-		if time.Since(logged) > 8*time.Second && report {
+		if time.Since(logged) > 8*time.Second {
 			log.Info("Regenerating historical state", "block", current.NumberU64()+1, "target", origin, "remaining", origin-current.NumberU64(), "elapsed", time.Since(start))
 			logged = time.Now()
 		}
@@ -1335,12 +1287,82 @@ func (bc *BlockChain) reprocessState(current *types.Block, reexec uint64, report
 		}
 		previousRoot = root
 	}
-	if report {
-		nodes, imgs := triedb.Size()
-		log.Info("Historical state regenerated", "block", current.NumberU64(), "elapsed", time.Since(start), "nodes", nodes, "preimages", imgs)
-	}
+
+	nodes, imgs := triedb.Size()
+	log.Info("Historical state regenerated", "block", current.NumberU64(), "elapsed", time.Since(start), "nodes", nodes, "preimages", imgs)
 	if previousRoot != (common.Hash{}) {
-		return triedb.Commit(previousRoot, report, nil)
+		return triedb.Commit(previousRoot, true, nil)
+	}
+	return nil
+}
+
+// CleanBlockRootsAboveLastAccepted gathers the blocks that may have previously been in processing above the
+// last accepted block and wipes their block roots from disk to mark their tries as inaccessible.
+// This is used prior to pruning to ensure that all of the tries that may still be in processing are marked
+// as inaccessible and mirrors the handling of middle roots in the geth offline pruning implementation.
+// This is not strictly necessary, but maintains a soft assumption.
+func (bc *BlockChain) CleanBlockRootsAboveLastAccepted() error {
+	targetRoot := bc.LastAcceptedBlock().Root()
+
+	// Clean up any block roots above the last accepted block before we start pruning.
+	// Note: this takes the place of middleRoots in the geth implementation since we do not
+	// track processing block roots via snapshot journals in the same way.
+	processingRoots := bc.gatherBlockRootsAboveLastAccepted()
+	// If there is a block above the last accepted block with an identical state root, we
+	// explicitly remove it from the set to ensure we do not corrupt the last accepted trie.
+	delete(processingRoots, targetRoot)
+	for processingRoot := range processingRoots {
+		// Delete the processing root from disk to mark the trie as inaccessible (no need to handle this in a batch).
+		if err := bc.db.Delete(processingRoot[:]); err != nil {
+			return fmt.Errorf("failed to remove processing root (%s) preparing for offline pruning: %w", processingRoot, err)
+		}
+	}
+
 	return nil
 }
+
+// gatherBlockRootsAboveLastAccepted iterates forward from the last accepted block and returns a list of all block roots
+// for any blocks that were inserted above the last accepted block.
+// Given that we never insert a block into the chain unless all of its ancestors have been inserted, this should gather
+// all of the block roots for blocks inserted above the last accepted block that may have been in processing at some point
+// in the past and are therefore potentially still acceptable.
+// Note: there is an edge case where the node dies while the consensus engine is rejecting a branch of blocks since the
+// consensus engine will reject the lowest ancestor first. In this case, these blocks will not be considered acceptable in
+// the future.
+// Ex.
+//    A
+//  /   \
+// B     C
+// |
+// D
+// |
+// E
+// |
+// F
+//
+// The consensus engine accepts block C and proceeds to reject the other branch in order (B, D, E, F).
+// If the consensus engine dies after rejecting block D, block D will be deleted, such that the forward iteration
+// may not find any blocks at this height and will not reach the previously processing blocks E and F.
+func (bc *BlockChain) gatherBlockRootsAboveLastAccepted() map[common.Hash]struct{} {
+	blockRoots := make(map[common.Hash]struct{})
+	for height := bc.lastAccepted.NumberU64() + 1; ; height++ {
+		blockHashes := rawdb.ReadAllHashes(bc.db, height)
+		// If there are no block hashes at [height], then there should be no further acceptable blocks
+		// past this point.
+		if len(blockHashes) == 0 {
+			break
+		}
+
+		// Fetch the blocks and append their roots.
+		for _, blockHash := range blockHashes {
+			block := bc.GetBlockByHash(blockHash)
+			if block == nil {
+				continue
+			}
+
+			blockRoots[block.Root()] = struct{}{}
+		}
+	}
+
+	return blockRoots
+}
diff --git a/core/blockchain_reader.go b/core/blockchain_reader.go
index 33adaea4..5b2646b8 100644
--- a/core/blockchain_reader.go
+++ b/core/blockchain_reader.go
@@ -27,17 +27,15 @@
 package core
 
 import (
-	"math/big"
-
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/state/snapshot"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
```

```
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )

 // CurrentHeader retrieves the current head header of the canonical chain. The
@@ -208,12 +206,6 @@ func (bc *BlockChain) GetTransactionLookup(hash common.Hash) *rawdb.LegacyTxLook
        return lookup
 }

-// GetTd retrieves a block's total difficulty in the canonical chain from the
-// database by hash and number, caching it if found.
-func (bc *BlockChain) GetTd(hash common.Hash, number uint64) *big.Int {
-       return bc.hc.GetTd(hash, number)
-}
-
 // HasState checks if state trie is fully present in the database or not.
 func (bc *BlockChain) HasState(hash common.Hash) bool {
        _, err := bc.stateCache.OpenTrie(hash)
diff --git a/core/blockchain_repair_test.go b/core/blockchain_repair_test.go
new file mode 100644
index 00000000..a58822a7
--- /dev/null
+++ b/core/blockchain_repair_test.go
@@ -0,0 +1,601 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// Tests that abnormal program termination (i.e.crash) and restart doesn't leave
+// the database in some strange state with gaps in the chain, nor with block data
+// dangling in the future.
+
+package core
+
+import (
+       "io/ioutil"
+       "math/big"
+       "os"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+)
+
+// rewindTest is a test case for chain rollback upon user request.
+type rewindTest struct {
+       canonicalBlocks int    // Number of blocks to generate for the canonical chain (heavier)
+       sidechainBlocks int    // Number of blocks to generate for the side chain (lighter)
+       commitBlock     uint64 // Block number for which to commit the state to disk
+
+       expCanonicalBlocks int    // Number of canonical blocks expected to remain in the database (excl. genesis)
+       expSidechainBlocks int    // Number of sidechain blocks expected to remain in the database (excl. genesis)
+       expHeadBlock       uint64 // Block number of the expected head full block
+}
+
+// Tests a recovery for a short canonical chain where a recent block was already
+// committed to disk and then the process crashed. In this case we expect the full
+// chain to be rolled back to the committed block, but the chain data itself left
+// in the database for replaying.
+func TestShortRepair(t *testing.T)              { testShortRepair(t, false) }
+func TestShortRepairWithSnapshots(t *testing.T) { testShortRepair(t, true) }
+
+func testShortRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    8,
+               sidechainBlocks:    0,
+               commitBlock:        4,
+               expCanonicalBlocks: 8,
+               expSidechainBlocks: 0,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a short canonical chain and a shorter side chain, where a
+// recent block was already committed to disk and then the process crashed. In this
+// test scenario the side chain is below the committed block. In this case we expect
+// the canonical chain to be rolled back to the committed block, but the chain data
+// itself left in the database for replaying.
+func TestShortOldForkedRepair(t *testing.T)              { testShortOldForkedRepair(t, false) }
+func TestShortOldForkedRepairWithSnapshots(t *testing.T) { testShortOldForkedRepair(t, true) }
+
+func testShortOldForkedRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //   └->S1->S2->S3
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //   └->S1->S2->S3
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    8,
+               sidechainBlocks:    3,
+               commitBlock:        4,
+               expCanonicalBlocks: 8,
+               expSidechainBlocks: 3,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a short canonical chain and a shorter side chain, where a
+// recent block was already committed to disk and then the process crashed. In this
+// test scenario the side chain reaches above the committed block. In this case we
+// expect the canonical chain to be rolled back to the committed block, but the
+// chain data itself left in the database for replaying.
+func TestShortNewlyForkedRepair(t *testing.T)              { testShortNewlyForkedRepair(t, false) }
+func TestShortNewlyForkedRepairWithSnapshots(t *testing.T) { testShortNewlyForkedRepair(t, true) }
+
+func testShortNewlyForkedRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //   └->S1->S2->S3->S4->S5->S6
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    8,
+               sidechainBlocks:    6,
+               commitBlock:        4,
+               expCanonicalBlocks: 8,
+               expSidechainBlocks: 6,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a short canonical chain and a longer side chain, where a
+// recent block was already committed to disk and then the process crashed. In this
+// case we expect the canonical chain to be rolled back to the committed block, but
+// the chain data itself left in the database for replaying.
+func TestShortReorgedRepair(t *testing.T)              { testShortReorgedRepair(t, false) }
+func TestShortReorgedRepairWithSnapshots(t *testing.T) { testShortReorgedRepair(t, true) }
+
+func testShortReorgedRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    8,
+               sidechainBlocks:    10,
+               commitBlock:        4,
+               expCanonicalBlocks: 8,
+               expSidechainBlocks: 10,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain where a recent block was already
+// committed to disk and then the process crashed. In this case we expect the chain
+// to be rolled back to the committed block, but the chain data itself left in the
+// database for replaying.
+func TestLongShallowRepair(t *testing.T)              { testLongShallowRepair(t, false) }
+func TestLongShallowRepairWithSnapshots(t *testing.T) { testLongShallowRepair(t, true) }
+
+func testLongShallowRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18 (HEAD)
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    18,
+               sidechainBlocks:    0,
+               commitBlock:        4,
+               expCanonicalBlocks: 18,
+               expSidechainBlocks: 0,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
```

```
+
+// Tests a recovery for a long canonical chain where a recent block was already committed
+// to disk and then the process crashed. In this case we expect the chain to be rolled
+// back to the committed block, but the chain data itself left in the database for replaying.
+func TestLongDeepRepair(t *testing.T)              { testLongDeepRepair(t, false) }
+func TestLongDeepRepairWithSnapshots(t *testing.T) { testLongDeepRepair(t, true) }
+
+func testLongDeepRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24 (HEAD)
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb: none
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    24,
+               sidechainBlocks:    0,
+               commitBlock:        4,
+               expCanonicalBlocks: 24,
+               expSidechainBlocks: 0,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain with a shorter side chain, where a recent
+// block was already committed to disk and then the process crashed. In this test scenario
+// the side chain is below the committed block. In this case we expect the chain to be
+// rolled back to the committed block, but the chain data itself left in the database
+// for replaying.
+func TestLongOldForkedShallowRepair(t *testing.T) {
+       testLongOldForkedShallowRepair(t, false)
+}
+func TestLongOldForkedShallowRepairWithSnapshots(t *testing.T) {
+       testLongOldForkedShallowRepair(t, true)
+}
+
+func testLongOldForkedShallowRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18 (HEAD)
+       //   └->S1->S2->S3
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18
+       //   └->S1->S2->S3
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    18,
+               sidechainBlocks:    3,
+               commitBlock:        4,
+               expCanonicalBlocks: 18,
+               expSidechainBlocks: 3,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain a shorter side chain, where a recent block
+// was already committed to disk and then the process crashed. In this test scenario the side
+// chain is below the committed block. In this case we expect the canonical chain to be
+// rolled back to the committed block, but the chain data itself left in the database for replaying.
+func TestLongOldForkedDeepRepair(t *testing.T)              { testLongOldForkedDeepRepair(t, false) }
+func TestLongOldForkedDeepRepairWithSnapshots(t *testing.T) { testLongOldForkedDeepRepair(t, true) }
+
+func testLongOldForkedDeepRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24 (HEAD)
+       //   └->S1->S2->S3
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24
+       //   └->S1->S2->S3
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    24,
+               sidechainBlocks:    3,
+               commitBlock:        4,
+               expCanonicalBlocks: 24,
+               expSidechainBlocks: 3,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain with a shorter side chain, where a recent
+// block was already committed to disk and then the process crashed. In this test scenario
+// the side chain is above the committed block. In this case we expect the chain to be
+// rolled back to the committed block, but the chain data itself left in the database for replaying.
+func TestLongNewerForkedShallowRepair(t *testing.T) {
+       testLongNewerForkedShallowRepair(t, false)
+}
+func TestLongNewerForkedShallowRepairWithSnapshots(t *testing.T) {
+       testLongNewerForkedShallowRepair(t, true)
+}
+
+func testLongNewerForkedShallowRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    18,
+               sidechainBlocks:    12,
+               commitBlock:        4,
+               expCanonicalBlocks: 18,
+               expSidechainBlocks: 12,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain with a shorter side chain, where a recent block
+// was already committed to disk and then the process crashed. In this test scenario the side
+// chain is above the committed block. In this case we expect the canonical chain to be rolled
+// back to the committed block, but the chain data itself left in the database for replaying.
+func TestLongNewerForkedDeepRepair(t *testing.T)              { testLongNewerForkedDeepRepair(t, false) }
+func TestLongNewerForkedDeepRepairWithSnapshots(t *testing.T) { testLongNewerForkedDeepRepair(t, true) }
+
+func testLongNewerForkedDeepRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    24,
+               sidechainBlocks:    12,
+               commitBlock:        4,
+               expCanonicalBlocks: 24,
+               expSidechainBlocks: 12,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain with a longer side chain, where a recent block
+// was already committed to disk and then the process crashed. In this case we expect the chain to be
+// rolled back to the committed block, but the chain data itself left in the database for replaying.
+func TestLongReorgedShallowRepair(t *testing.T)              { testLongReorgedShallowRepair(t, false) }
+func TestLongReorgedShallowRepairWithSnapshots(t *testing.T) { testLongReorgedShallowRepair(t, true) }
+
+func testLongReorgedShallowRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12->S13->S14->S15->S16->S17->S18->S19->S20->S21->S22->S23->S24->S25->S26
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12->S13->S14->S15->S16->S17->S18->S19->S20->S21->S22->S23->S24->S25->S26
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    18,
+               sidechainBlocks:    26,
+               commitBlock:        4,
+               expCanonicalBlocks: 18,
+               expSidechainBlocks: 26,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+// Tests a recovery for a long canonical chain with a longer side chain, where a recent block
+// was already committed to disk and then the process crashed. In this case we expect the canonical
+// chains to be rolled back to the committed block, but the chain data itself left in the database
+// for replaying.
+func TestLongReorgedDeepRepair(t *testing.T)              { testLongReorgedDeepRepair(t, false) }
+func TestLongReorgedDeepRepairWithSnapshots(t *testing.T) { testLongReorgedDeepRepair(t, true) }
+
+func testLongReorgedDeepRepair(t *testing.T, snapshots bool) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24 (HEAD)
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12->S13->S14->S15->S16->S17->S18->S19->S20->S21->S22->S23->S24->S25->S26
+       //
+       // Commit: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10->C11->C12->C13->C14->C15->C16->C17->C18->C19->C20->C21->C22->C23->C24
+       //   └->S1->S2->S3->S4->S5->S6->S7->S8->S9->S10->S11->S12->S13->S14->S15->S16->S17->S18->S19->S20->S21->S22->S23->S24->S25->S26
+       //
+       // Expected head block     : C4 (C0 with no snapshots)
+       rt := &rewindTest{
+               canonicalBlocks:    24,
+               sidechainBlocks:    26,
+               commitBlock:        4,
+               expCanonicalBlocks: 24,
+               expSidechainBlocks: 26,
+               expHeadBlock:       0,
+       }
+       if snapshots {
+               rt.expHeadBlock = 4
+       }
+       testRepair(t, rt, snapshots)
+}
+
+func testRepair(t *testing.T, tt *rewindTest, snapshots bool) {
+       // It's hard to follow the test case, visualize the input
+       //log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
```

```
+	// fmt.Println(tt.dump(true))
+
+	// Create a temporary persistent database
+	datadir, err := ioutil.TempDir("", "")
+	if err != nil {
+		t.Fatalf("Failed to create temporary datadir: %v", err)
+	}
+	os.RemoveAll(datadir)
+
+	db, err := rawdb.NewLevelDBDatabase(datadir, 0, 0, "", false)
+	if err != nil {
+		t.Fatalf("Failed to create persistent database: %v", err)
+	}
+	defer db.Close() // Might double close, should be fine
+
+	// Initialize a fresh chain
+	var (
+		genesis = (&Genesis{Config: params.TestChainConfig, BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee)}).MustCommit(db)
+		engine  = dummy.NewFullFaker()
+		config  = &CacheConfig{
+			TrieCleanLimit: 256,
+			TrieDirtyLimit: 256,
+			SnapshotLimit:  0, // Disable snapshot by default
+		}
+	)
+	if snapshots {
+		config.SnapshotLimit = 256
+	}
+	chain, err := NewBlockChain(db, config, params.TestChainConfig, engine, vm.Config{}, common.Hash{})
+	if err != nil {
+		t.Fatalf("Failed to create chain: %v", err)
+	}
+	lastAcceptedHash := chain.GetBlockByNumber(0).Hash()
+
+	// If sidechain blocks are needed, make a light chain and import it
+	var sideblocks types.Blocks
+	if tt.sidechainBlocks > 0 {
+		sideblocks, _, _ = GenerateChain(params.TestChainConfig, genesis, engine, rawdb.NewMemoryDatabase(), tt.sidechainBlocks, 10, func(i int, b *BlockGen) {
+			b.SetCoinbase(common.Address{0x01})
+		})
+		if _, err := chain.InsertChain(sideblocks); err != nil {
+			t.Fatalf("Failed to import side chain: %v", err)
+		}
+	}
+	canonblocks, _, _ := GenerateChain(params.TestChainConfig, genesis, engine, rawdb.NewMemoryDatabase(), tt.canonicalBlocks, 10, func(i int, b *BlockGen) {
+		b.SetCoinbase(common.Address{0x02})
+		b.SetDifficulty(big.NewInt(1000000))
+	})
+	if _, err := chain.InsertChain(canonblocks[:tt.commitBlock]); err != nil {
+		t.Fatalf("Failed to import canonical chain start: %v", err)
+	}
+	if tt.commitBlock > 0 {
+		if snapshots {
+			for i := uint64(0); i < tt.commitBlock; i++ {
+				if err := chain.Accept(canonblocks[i]); err != nil {
+					t.Fatalf("Failed to accept block %v: %v", i, err)
+				}
+				lastAcceptedHash = canonblocks[i].Hash()
+			}
+		}
+	}
+	if _, err := chain.InsertChain(canonblocks[tt.commitBlock:]); err != nil {
+		t.Fatalf("Failed to import canonical chain tail: %v", err)
+	}
+
+	// Pull the plug on the database, simulating a hard crash
+	db.Close()
+
+	// Start a new blockchain back up and see where the repait leads us
+	db, err = rawdb.NewLevelDBDatabase(datadir, 0, 0, "", false)
+	if err != nil {
+		t.Fatalf("Failed to reopen persistent database: %v", err)
+	}
+	defer db.Close()
+
+	chain, err = NewBlockChain(db, DefaultCacheConfig, params.TestChainConfig, engine, vm.Config{}, lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	defer chain.Stop()
+
+	// Iterate over all the remaining blocks and ensure there are no gaps
+	verifyNoGaps(t, chain, true, canonblocks)
+	verifyNoGaps(t, chain, false, sideblocks)
+	verifyCutoff(t, chain, true, canonblocks, tt.expCanonicalBlocks)
+	verifyCutoff(t, chain, false, sideblocks, tt.expSidechainBlocks)
+
+	if head := chain.CurrentHeader(); head.Number.Uint64() != tt.expHeadBlock {
+		t.Errorf("Head header mismatch: have %d, want %d", head.Number, tt.expHeadBlock)
+	}
+	if head := chain.CurrentBlock(); head.NumberU64() != tt.expHeadBlock {
+		t.Errorf("Head block mismatch: have %d, want %d", head.NumberU64(), tt.expHeadBlock)
+	}
+}
diff --git a/core/blockchain_sethead_test.go b/core/blockchain_sethead_test.go
new file mode 100644
index 00000000..03cb332d
--- /dev/null
+++ b/core/blockchain_sethead_test.go
@@ -0,0 +1,142 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// Tests that setting the chain head backwards doesn't leave the database in some
+// strange state with gaps in the chain, nor with block data dangling in the future.
+
+package core
+
+import (
+	"testing"
+
+	"github.com/flare-foundation/coreth/core/types"
+)
+
+// verifyNoGaps checks that there are no gaps after the initial set of blocks in
+// the database and errors if found.
+func verifyNoGaps(t *testing.T, chain *BlockChain, canonical bool, inserted types.Blocks) {
+	t.Helper()
+
+	var end uint64
+	for i := uint64(0); i <= uint64(len(inserted)); i++ {
+		header := chain.GetHeaderByNumber(i)
+		if header == nil && end == 0 {
+			end = i
+		}
+		if header != nil && end > 0 {
+			if canonical {
+				t.Errorf("Canonical header gap between #%d-#%d", end, i-1)
+			} else {
+				t.Errorf("Sidechain header gap between #%d-#%d", end, i-1)
+			}
+			end = 0 // Reset for further gap detection
+		}
+	}
+	end = 0
+	for i := uint64(0); i <= uint64(len(inserted)); i++ {
+		block := chain.GetBlockByNumber(i)
+		if block == nil && end == 0 {
+			end = i
+		}
+		if block != nil && end > 0 {
+			if canonical {
+				t.Errorf("Canonical block gap between #%d-#%d", end, i-1)
+			} else {
+				t.Errorf("Sidechain block gap between #%d-#%d", end, i-1)
+			}
+			end = 0 // Reset for further gap detection
+		}
+	}
+	end = 0
+	for i := uint64(1); i <= uint64(len(inserted)); i++ {
+		receipts := chain.GetReceiptsByHash(inserted[i-1].Hash())
+		if receipts == nil && end == 0 {
+			end = i
+		}
+		if receipts != nil && end > 0 {
+			if canonical {
+				t.Errorf("Canonical receipt gap between #%d-#%d", end, i-1)
+			} else {
+				t.Errorf("Sidechain receipt gap between #%d-#%d", end, i-1)
+			}
+			end = 0 // Reset for further gap detection
+		}
+	}
+}
+
+// verifyCutoff checks that there are no chain data available in the chain after
+// the specified limit, but that it is available before.
+func verifyCutoff(t *testing.T, chain *BlockChain, canonical bool, inserted types.Blocks, head int) {
+	t.Helper()
+
+	for i := 1; i <= len(inserted); i++ {
+		if i <= head {
+			if header := chain.GetHeader(inserted[i-1].Hash(), uint64(i)); header == nil {
+				if canonical {
+					t.Errorf("Canonical header   #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain header   #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+			if block := chain.GetBlock(inserted[i-1].Hash(), uint64(i)); block == nil {
+				if canonical {
+					t.Errorf("Canonical block    #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain block    #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+			if receipts := chain.GetReceiptsByHash(inserted[i-1].Hash()); receipts == nil {
+				if canonical {
+					t.Errorf("Canonical receipts #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain receipts #%2d [%x...] missing before cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+		} else {
+			if header := chain.GetHeader(inserted[i-1].Hash(), uint64(i)); header != nil {
+				if canonical {
+					t.Errorf("Canonical header   #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain header   #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+			if block := chain.GetBlock(inserted[i-1].Hash(), uint64(i)); block != nil {
+				if canonical {
+					t.Errorf("Canonical block    #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain block    #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+			if receipts := chain.GetReceiptsByHash(inserted[i-1].Hash()); receipts != nil {
+				if canonical {
+					t.Errorf("Canonical receipts #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				} else {
+					t.Errorf("Sidechain receipts #%2d [%x...] present after cap %d", inserted[i-1].Number(), inserted[i-1].Hash().Bytes()[:3], head)
+				}
+			}
+		}
+	}
+}
diff --git a/core/blockchain_snapshot_test.go b/core/blockchain_snapshot_test.go
new file mode 100644
index 00000000..5a02c123
--- /dev/null
+++ b/core/blockchain_snapshot_test.go
@@ -0,0 +1,613 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// Tests that abnormal program termination (i.e.crash) and restart can recovery
+// the snapshot properly if the snapshot is enabled.
+
+package core
+
+import (
+       "bytes"
+       "fmt"
+       "io/ioutil"
+       "math/big"
+       "os"
+       "strings"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
+)
+
+// snapshotTestBasic wraps the common testing fields in the snapshot tests.
+type snapshotTestBasic struct {
+       chainBlocks   int    // Number of blocks to generate for the canonical chain
+       snapshotBlock uint64 // Block number of the relevant snapshot disk layer
+
+       expCanonicalBlocks int    // Number of canonical blocks expected to remain in the database (excl. genesis)
+       expHeadBlock       uint64 // Block number of the expected head full block
+       expSnapshotBottom  uint64 // The block height corresponding to the snapshot disk layer
+
+       // share fields, set in runtime
+       datadir string
+       db      ethdb.Database
+       gendb   ethdb.Database
+       engine  consensus.Engine
+
+       lastAcceptedHash common.Hash
+}
+
+func (basic *snapshotTestBasic) prepare(t *testing.T) (*BlockChain, []*types.Block) {
+       // Create a temporary persistent database
+       datadir, err := ioutil.TempDir("", "")
+       if err != nil {
+               t.Fatalf("Failed to create temporary datadir: %v", err)
+       }
+       os.RemoveAll(datadir)
+
+       db, err := rawdb.NewLevelDBDatabase(datadir, 0, 0, "", false)
+       if err != nil {
+               t.Fatalf("Failed to create persistent database: %v", err)
+       }
+       // Initialize a fresh chain
+       var (
+               genesis = (&Genesis{Config: params.TestChainConfig, BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee)}).MustCommit(db)
+               engine  = dummy.NewFullFaker()
+               gendb   = rawdb.NewMemoryDatabase()
+
+               // Snapshot is enabled, the first snapshot is created from the Genesis.
+               // The snapshot memory allowance is 256MB, it means no snapshot flush
+               // will happen during the block insertion.
+               cacheConfig = DefaultCacheConfig
+       )
+       chain, err := NewBlockChain(db, cacheConfig, params.TestChainConfig, engine, vm.Config{}, common.Hash{})
+       if err != nil {
+               t.Fatalf("Failed to create chain: %v", err)
+       }
+       blocks, _, _ := GenerateChain(params.TestChainConfig, genesis, engine, gendb, basic.chainBlocks, 10, func(i int, b *BlockGen) {})
+
+       // genesis as last accepted
+       basic.lastAcceptedHash = chain.GetBlockByNumber(0).Hash()
+
+       // Insert the blocks with configured settings.
+       var breakpoints []uint64
+       breakpoints = append(breakpoints, basic.snapshotBlock)
+       var startPoint uint64
+       for _, point := range breakpoints {
+               if _, err := chain.InsertChain(blocks[startPoint:point]); err != nil {
+                       t.Fatalf("Failed to import canonical chain start: %v", err)
+               }
+               startPoint = point
+
+               if basic.snapshotBlock > 0 && basic.snapshotBlock == point {
+                       // Flushing from 0 to snapshotBlock into the disk
+                       for i := uint64(0); i < point; i++ {
+                               if err := chain.Accept(blocks[i]); err != nil {
+                                       t.Fatalf("Failed to accept block %v: %v", i, err)
+                               }
+                               basic.lastAcceptedHash = blocks[i].Hash()
+                       }
+
+                       diskRoot, blockRoot := chain.snaps.DiskRoot(), blocks[point-1].Root()
+                       if !bytes.Equal(diskRoot.Bytes(), blockRoot.Bytes()) {
+                               t.Fatalf("Failed to flush disk layer change, want %x, got %x", blockRoot, diskRoot)
+                       }
+               }
+       }
+       if _, err := chain.InsertChain(blocks[startPoint:]); err != nil {
+               t.Fatalf("Failed to import canonical chain tail: %v", err)
+       }
+
+       // Set runtime fields
+       basic.datadir = datadir
+       basic.db = db
+       basic.gendb = gendb
+       basic.engine = engine
+       return chain, blocks
+}
+
+func (basic *snapshotTestBasic) verify(t *testing.T, chain *BlockChain, blocks []*types.Block) {
+       // Iterate over all the remaining blocks and ensure there are no gaps
+       verifyNoGaps(t, chain, true, blocks)
+       verifyCutoff(t, chain, true, blocks, basic.expCanonicalBlocks)
+
+       if head := chain.CurrentHeader(); head.Number.Uint64() != basic.expHeadBlock {
+               t.Errorf("Head header mismatch: have %d, want %d", head.Number, basic.expHeadBlock)
+       }
+       if head := chain.CurrentBlock(); head.NumberU64() != basic.expHeadBlock {
+               t.Errorf("Head block mismatch: have %d, want %d", head.NumberU64(), basic.expHeadBlock)
+       }
+
+       // Check the disk layer, ensure they are matched
+       block := chain.GetBlockByNumber(basic.expSnapshotBottom)
+       if block == nil {
+               t.Errorf("The correspnding block[%d] of snapshot disk layer is missing", basic.expSnapshotBottom)
+       } else if !bytes.Equal(chain.snaps.DiskRoot().Bytes(), block.Root().Bytes()) {
```

```
+               t.Errorf("The snapshot disk layer root is incorrect, want %x, get %x", block.Root(), chain.snaps.DiskRoot())
+       } else if len(chain.snaps.Snapshots(block.Hash(), -1, false)) != 1 {
+               t.Errorf("The correspnding block[%d] of snapshot disk layer is missing", basic.expSnapshotBottom)
+       }
+
+       // Check the snapshot, ensure it's integrated
+       if err := chain.snaps.Verify(block.Root()); err != nil {
+               t.Errorf("The disk layer is not integrated %v", err)
+       }
+}
+
+func (basic *snapshotTestBasic) dump() string {
+       buffer := new(strings.Builder)
+
+       fmt.Fprint(buffer, "Chain:\n  G")
+       for i := 0; i < basic.chainBlocks; i++ {
+               fmt.Fprintf(buffer, "->C%d", i+1)
+       }
+       fmt.Fprint(buffer, " (HEAD)\n\n")
+
+       fmt.Fprintf(buffer, "Snapshot: G")
+       if basic.snapshotBlock > 0 {
+               fmt.Fprintf(buffer, ", C%d", basic.snapshotBlock)
+       }
+       fmt.Fprint(buffer, "\n")
+
+       //if crash {
+       //      fmt.Fprintf(buffer, "\nCRASH\n\n")
+       //} else {
+       //      fmt.Fprintf(buffer, "\nSetHead(%d)\n\n", basic.setHead)
+       //}
+       fmt.Fprintf(buffer, "------------------------------\n\n")
+
+       fmt.Fprint(buffer, "Expected in leveldb:\n  G")
+       for i := 0; i < basic.expCanonicalBlocks; i++ {
+               fmt.Fprintf(buffer, "->C%d", i+1)
+       }
+       fmt.Fprintf(buffer, "\n\n")
+       fmt.Fprintf(buffer, "Expected head header    : C%d\n", basic.expHeadBlock)
+       if basic.expHeadBlock == 0 {
+               fmt.Fprintf(buffer, "Expected head block     : G\n")
+       } else {
+               fmt.Fprintf(buffer, "Expected head block     : C%d\n", basic.expHeadBlock)
+       }
+       if basic.expSnapshotBottom == 0 {
+               fmt.Fprintf(buffer, "Expected snapshot disk  : G\n")
+       } else {
+               fmt.Fprintf(buffer, "Expected snapshot disk  : C%d\n", basic.expSnapshotBottom)
+       }
+       return buffer.String()
+}
+
+func (basic *snapshotTestBasic) teardown() {
+       basic.db.Close()
+       basic.gendb.Close()
+       os.RemoveAll(basic.datadir)
+}
+
+// snapshotTest is a test case type for normal snapshot recovery.
+// It can be used for testing that restart Geth normally.
+type snapshotTest struct {
+       snapshotTestBasic
+}
+
+func (snaptest *snapshotTest) test(t *testing.T) {
+       // It's hard to follow the test case, visualize the input
+       // log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
+       // fmt.Println(tt.dump())
+       chain, blocks := snaptest.prepare(t)
+
+       // Restart the chain normally
+       chain.Stop()
+       newchain, err := NewBlockChain(snaptest.db, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+       if err != nil {
+               t.Fatalf("Failed to recreate chain: %v", err)
+       }
+       defer newchain.Stop()
+
+       snaptest.verify(t, newchain, blocks)
+}
+
+// crashSnapshotTest is a test case type for innormal snapshot recovery.
+// It can be used for testing that restart Geth after the crash.
+type crashSnapshotTest struct {
+       snapshotTestBasic
+}
+
+func (snaptest *crashSnapshotTest) test(t *testing.T) {
+       // It's hard to follow the test case, visualize the input
+       // log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
+       // fmt.Println(tt.dump())
+       chain, blocks := snaptest.prepare(t)
+
+       // Pull the plug on the database, simulating a hard crash
+       db := chain.db
+       db.Close()
+
+       // Start a new blockchain back up and see where the repair leads us
+       newdb, err := rawdb.NewLevelDBDatabase(snaptest.datadir, 0, 0, "", false)
+       if err != nil {
+               t.Fatalf("Failed to reopen persistent database: %v", err)
+       }
+       defer newdb.Close()
+
+       // The interesting thing is: instead of starting the blockchain after
+       // the crash, we do restart twice here: one after the crash and one
+       // after the normal stop. It's used to ensure the broken snapshot
+       // can be detected all the time.
+       newchain, err := NewBlockChain(newdb, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+       if err != nil {
+               t.Fatalf("Failed to recreate chain: %v", err)
+       }
+       newchain.Stop()
+
+       newchain, err = NewBlockChain(newdb, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+       if err != nil {
+               t.Fatalf("Failed to recreate chain: %v", err)
+       }
+       defer newchain.Stop()
+
+       snaptest.verify(t, newchain, blocks)
+}
+
+// gappedSnapshotTest is a test type used to test this scenario:
+// - have a complete snapshot
+// - restart without enabling the snapshot
+// - insert a few blocks
+// - restart with enabling the snapshot again
+type gappedSnapshotTest struct {
+       snapshotTestBasic
+       gapped int // Number of blocks to insert without enabling snapshot
+}
+
+func (snaptest *gappedSnapshotTest) test(t *testing.T) {
```

```
+	// It's hard to follow the test case, visualize the input
+	// log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
+	// fmt.Println(tt.dump())
+	chain, blocks := snaptest.prepare(t)
+
+	// Insert blocks without enabling snapshot if gapping is required.
+	chain.Stop()
+	gappedBlocks, _, _ := GenerateChain(params.TestChainConfig, blocks[len(blocks)-1], snaptest.engine, snaptest.gendb, snaptest.gapped, 10, func(i int, b *BlockGen) {})
+
+	// Insert a few more blocks without enabling snapshot
+	var cacheConfig = &CacheConfig{
+		TrieCleanLimit: 256,
+		TrieDirtyLimit: 256,
+		SnapshotLimit:  0,
+	}
+	newchain, err := NewBlockChain(snaptest.db, cacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	newchain.InsertChain(gappedBlocks)
+	newchain.Stop()
+
+	// Restart the chain with enabling the snapshot
+	newchain, err = NewBlockChain(snaptest.db, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	defer newchain.Stop()
+
+	snaptest.verify(t, newchain, blocks)
+}
+
+// restartCrashSnapshotTest is the test type used to test this scenario:
+// - have a complete snapshot
+// - restart chain
+// - insert more blocks with enabling the snapshot
+// - commit the snapshot
+// - crash
+// - restart again
+type restartCrashSnapshotTest struct {
+	snapshotTestBasic
+	newBlocks int
+}
+
+func (snaptest *restartCrashSnapshotTest) test(t *testing.T) {
+	// It's hard to follow the test case, visualize the input
+	// log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
+	// fmt.Println(tt.dump())
+	chain, blocks := snaptest.prepare(t)
+
+	// Firstly, stop the chain properly, with all snapshot journal
+	// and state committed.
+	chain.Stop()
+
+	newchain, err := NewBlockChain(snaptest.db, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	newBlocks, _, _ := GenerateChain(params.TestChainConfig, blocks[len(blocks)-1], snaptest.engine, snaptest.gendb, snaptest.newBlocks, 10, func(i int, b *BlockGen) {})
+	newchain.InsertChain(newBlocks)
+
+	// Commit the entire snapshot into the disk if requested. Note only
+	// (a) snapshot root and (b) snapshot generator will be committed,
+	// the diff journal is not.
+	for i := uint64(0); i < uint64(len(newBlocks)); i++ {
+		if err := newchain.Accept(newBlocks[i]); err != nil {
+			t.Fatalf("Failed to accept block %v: %v", i, err)
+		}
+		snaptest.lastAcceptedHash = newBlocks[i].Hash()
+	}
+
+	// Simulate the blockchain crash
+	// Don't call chain.Stop here, so that no snapshot
+	// journal and latest state will be committed
+
+	// Restart the chain after the crash
+	newchain, err = NewBlockChain(snaptest.db, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	defer newchain.Stop()
+
+	snaptest.verify(t, newchain, blocks)
+}
+
+// wipeCrashSnapshotTest is the test type used to test this scenario:
+// - have a complete snapshot
+// - restart, insert more blocks without enabling the snapshot
+// - restart again with enabling the snapshot
+// - crash
+type wipeCrashSnapshotTest struct {
+	snapshotTestBasic
+	newBlocks int
+}
+
+func (snaptest *wipeCrashSnapshotTest) test(t *testing.T) {
+	// It's hard to follow the test case, visualize the input
+	// log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr, log.TerminalFormat(true))))
+	// fmt.Println(tt.dump())
+	chain, blocks := snaptest.prepare(t)
+
+	// Firstly, stop the chain properly, with all snapshot journal
+	// and state committed.
+	chain.Stop()
+
+	config := &CacheConfig{
+		TrieCleanLimit: 256,
+		TrieDirtyLimit: 256,
+		SnapshotLimit:  0,
+	}
+	newchain, err := NewBlockChain(snaptest.db, config, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	newBlocks, _, _ := GenerateChain(params.TestChainConfig, blocks[len(blocks)-1], snaptest.engine, snaptest.gendb, snaptest.newBlocks, 10, func(i int, b *BlockGen) {})
+	newchain.InsertChain(newBlocks)
+	newchain.Stop()
+
+	// Restart the chain, the wiper should starts working
+	config = &CacheConfig{
+		TrieCleanLimit: 256,
+		TrieDirtyLimit: 256,
+		SnapshotLimit:  256,
+	}
+	newchain, err = NewBlockChain(snaptest.db, config, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
+	// Simulate the blockchain crash.
+
+	newchain, err = NewBlockChain(snaptest.db, DefaultCacheConfig, params.TestChainConfig, snaptest.engine, vm.Config{}, snaptest.lastAcceptedHash)
+	if err != nil {
+		t.Fatalf("Failed to recreate chain: %v", err)
+	}
```

```
+       snaptest.verify(t, newchain, blocks)
+}
+
+// Tests a Geth restart with valid snapshot. Before the shutdown, all snapshot
+// journal will be persisted correctly. In this case no snapshot recovery is
+// required.
+func TestRestartWithNewSnapshot(t *testing.T) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //
+       // Snapshot: G
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //
+       // Expected head header    : C8
+       // Expected head block     : C4
+       // Expected snapshot disk  : C4
+       test := &snapshotTest{
+               snapshotTestBasic{
+                       chainBlocks:        8,
+                       snapshotBlock:      4,
+                       expCanonicalBlocks: 8,
+                       expHeadBlock:       4,
+                       expSnapshotBottom:  4, // Initial disk layer built from genesis
+               },
+       }
+       test.test(t)
+       test.teardown()
+}
+
+// Tests a Geth was crashed and restarts with a broken snapshot. In this case the
+// chain head should be rewound to the point with available state. And also the
+// new head should must be lower than disk layer. But there is no committed point
+// so the chain should be rewound to genesis and the disk layer should be left
+// for recovery.
+func TestNoCommitCrashWithNewSnapshot(t *testing.T) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //
+       // Snapshot: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //
+       // Expected head block     : C4
+       // Expected snapshot disk  : C4
+       test := &crashSnapshotTest{
+               snapshotTestBasic{
+                       chainBlocks:        8,
+                       snapshotBlock:      4,
+                       expCanonicalBlocks: 8,
+                       expHeadBlock:       4,
+                       expSnapshotBottom:  4, // Last committed disk layer, wait recovery
+               },
+       }
+       test.test(t)
+       test.teardown()
+}
+
+// Tests a Geth was crashed and restarts with a broken snapshot. In this case the
+// chain head should be rewound to the point with available state. And also the
+// new head should must be lower than disk layer. But there is only a low committed
+// point so the chain should be rewound to committed point and the disk layer
+// should be left for recovery.
+func TestLowCommitCrashWithNewSnapshot(t *testing.T) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //
+       // Snapshot: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //
+       // Expected head block     : C4
+       // Expected snapshot disk  : C4
+       test := &crashSnapshotTest{
+               snapshotTestBasic{
+                       chainBlocks:        8,
+                       snapshotBlock:      4,
+                       expCanonicalBlocks: 8,
+                       expHeadBlock:       4,
+                       expSnapshotBottom:  4, // Last committed disk layer, wait recovery
+               },
+       }
+       test.test(t)
+       test.teardown()
+}
+
+// Tests a Geth was crashed and restarts with a broken snapshot. In this case
+// the chain head should be rewound to the point with available state. And also
+// the new head should must be lower than disk layer. But there is only a high
+// committed point so the chain should be rewound to genesis and the disk layer
+// should be left for recovery.
+func TestHighCommitCrashWithNewSnapshot(t *testing.T) {
+       // Chain:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+       //
+       // Snapshot: G, C4
+       //
+       // CRASH
+       //
+       // ------------------------------
+       //
+       // Expected in leveldb:
+       //   G->C1->C2->C3->C4->C5->C6->C7->C8
+       //
+       // Expected head block     : C4
+       // Expected snapshot disk  : C4
+       test := &crashSnapshotTest{
+               snapshotTestBasic{
+                       chainBlocks:        8,
+                       snapshotBlock:      4,
+                       expCanonicalBlocks: 8,
+                       expHeadBlock:       4,
+                       expSnapshotBottom:  4, // Last committed disk layer, wait recovery
+               },
+       }
+       test.test(t)
+       test.teardown()
+}
+
+// Tests a Geth was running with snapshot enabled. Then restarts without
+// enabling snapshot and after that re-enable the snapshot again. In this
+// case the snapshot should be rebuilt with latest chain head.
+func TestGappedNewSnapshot(t *testing.T) {
+	// Chain:
+	//   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+	//
+	// Snapshot: G
+	//
+	// ------------------------------
+	//
+	// Expected in leveldb:
+	//   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10
+	//
+	// Expected head block     : G
+	// Expected snapshot disk  : G
+	test := &gappedSnapshotTest{
+		snapshotTestBasic: snapshotTestBasic{
+			chainBlocks:        8,
+			snapshotBlock:      0,
+			expCanonicalBlocks: 10,
+			expHeadBlock:       0,
+			expSnapshotBottom:  0, // Rebuilt snapshot from the latest HEAD
+		},
+		gapped: 2,
+	}
+	test.test(t)
+	test.teardown()
+}
+
+// Tests the Geth was running with a complete snapshot and then imports a few
+// more new blocks on top without enabling the snapshot. After the restart,
+// crash happens. Check everything is ok after the restart.
+func TestRecoverSnapshotFromWipingCrash(t *testing.T) {
+	// Chain:
+	//   G->C1->C2->C3->C4->C5->C6->C7->C8 (HEAD)
+	//
+	// Snapshot: G
+	//
+	// ------------------------------
+	//
+	// Expected in leveldb:
+	//   G->C1->C2->C3->C4->C5->C6->C7->C8->C9->C10
+	//
+	// Expected head block     : C4
+	// Expected snapshot disk  : C4
+	test := &wipeCrashSnapshotTest{
+		snapshotTestBasic: snapshotTestBasic{
+			chainBlocks:        8,
+			snapshotBlock:      4,
+			expCanonicalBlocks: 10,
+			expHeadBlock:       4,
+			expSnapshotBottom:  4,
+		},
+		newBlocks: 2,
+	}
+	test.test(t)
+	test.teardown()
+}
diff --git a/core/blockchain_test.go b/core/blockchain_test.go
index 1685e04b..59909458 100644
--- a/core/blockchain_test.go
+++ b/core/blockchain_test.go
@@ -4,17 +4,19 @@
 package core
 
 import (
+	"fmt"
 	"math/big"
 	"testing"
 
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/state/pruner"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/params"
 )
 
 func TestArchiveBlockChain(t *testing.T) {
@@ -323,3 +325,84 @@ func TestCorruptSnapshots(t *testing.T) {
 		})
 	}
 }
+
+func TestBlockChainOfflinePruningUngracefulShutdown(t *testing.T) {
+	create := func(db ethdb.Database, chainConfig *params.ChainConfig, lastAcceptedHash common.Hash) (*BlockChain, error) {
+		// Import the chain. This runs all block validation rules.
+		blockchain, err := NewBlockChain(
+			db,
+			&CacheConfig{
+				TrieCleanLimit: 256,
+				TrieDirtyLimit: 256,
+				Pruning:        true, // Enable pruning
+				SnapshotLimit:  256,
+			},
+			chainConfig,
+			dummy.NewDummyEngine(&dummy.ConsensusCallbacks{
+				OnExtraStateChange: func(block *types.Block, sdb *state.StateDB) (*big.Int, *big.Int, error) {
+					sdb.SetBalanceMultiCoin(common.HexToAddress("0xdeadbeef"), common.HexToHash("0xdeadbeef"), big.NewInt(block.Number().Int64()))
+					return nil, nil, nil
+				},
+				OnFinalizeAndAssemble: func(header *types.Header, sdb *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
+					sdb.SetBalanceMultiCoin(common.HexToAddress("0xdeadbeef"), common.HexToHash("0xdeadbeef"), big.NewInt(header.Number.Int64()))
+					return nil, nil, nil, nil
+				},
+			}),
+			vm.Config{},
+			lastAcceptedHash,
+		)
+		if err != nil {
+			return nil, err
+		}
+
+		// Overwrite state manager, so that Shutdown is not called.
+		// This tests to ensure that the state manager handles an ungraceful shutdown correctly.
+		blockchain.stateManager = &wrappedStateManager{TrieWriter: blockchain.stateManager}
+
+		if lastAcceptedHash == (common.Hash{}) {
+			return blockchain, nil
+		}
+
+		tempDir := t.TempDir()
+		if err := blockchain.CleanBlockRootsAboveLastAccepted(); err != nil {
+			return nil, err
+		}
```

```
+		pruner, err := pruner.NewPruner(db, tempDir, 256)
+		if err != nil {
+			return nil, fmt.Errorf("offline pruning failed (%s, %d): %w", tempDir, 256, err)
+		}
+
+		targetRoot := blockchain.LastAcceptedBlock().Root()
+		if err := pruner.Prune(targetRoot); err != nil {
+			return nil, fmt.Errorf("failed to prune blockchain with target root: %s due to: %w", targetRoot, err)
+		}
+		// Re-initialize the blockchain after pruning
+		return NewBlockChain(
+			db,
+			&CacheConfig{
+				TrieCleanLimit: 256,
+				TrieDirtyLimit: 256,
+				Pruning:        true, // Enable pruning
+				SnapshotLimit:  256,
+			},
+			chainConfig,
+			dummy.NewDummyEngine(&dummy.ConsensusCallbacks{
+				OnExtraStateChange: func(block *types.Block, sdb *state.StateDB) (*big.Int, *big.Int, error) {
+					sdb.SetBalanceMultiCoin(common.HexToAddress("0xdeadbeef"), common.HexToHash("0xdeadbeef"), big.NewInt(block.Number().Int64()))
+					return nil, nil, nil
+				},
+				OnFinalizeAndAssemble: func(header *types.Header, sdb *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
+					sdb.SetBalanceMultiCoin(common.HexToAddress("0xdeadbeef"), common.HexToHash("0xdeadbeef"), big.NewInt(header.Number.Int64()))
+					return nil, nil, nil, nil
+				},
+			}),
+			vm.Config{},
+			lastAcceptedHash,
+		)
+	}
+	for _, tt := range tests {
+		t.Run(tt.Name, func(t *testing.T) {
+			tt.testFunc(t, create)
+		})
+	}
+}
diff --git a/core/bloom_indexer.go b/core/bloom_indexer.go
index 4d8f58fd..9404a818 100644
--- a/core/bloom_indexer.go
+++ b/core/bloom_indexer.go
@@ -20,12 +20,12 @@ import (
 	"context"
 	"time"
 
-	"github.com/ava-labs/coreth/core/bloombits"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/bitutil"
+	"github.com/flare-foundation/coreth/core/bloombits"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 const (
diff --git a/core/bloombits/generator.go b/core/bloombits/generator.go
index c0422caa..32e011a4 100644
--- a/core/bloombits/generator.go
+++ b/core/bloombits/generator.go
@@ -29,7 +29,7 @@ package bloombits
 import (
 	"errors"
 
-	"github.com/ava-labs/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 var (
diff --git a/core/bloombits/generator_test.go b/core/bloombits/generator_test.go
index 067c1db6..345ec4e6 100644
--- a/core/bloombits/generator_test.go
+++ b/core/bloombits/generator_test.go
@@ -31,7 +31,7 @@ import (
 	"math/rand"
 	"testing"
 
-	"github.com/ava-labs/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 // Tests that batched bloom bits are correctly rotated from the input bloom
diff --git a/core/chain_indexer.go b/core/chain_indexer.go
index 975f82b3..7163929c 100644
--- a/core/chain_indexer.go
+++ b/core/chain_indexer.go
@@ -34,12 +34,12 @@ import (
 	"sync/atomic"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // ChainIndexerBackend defines the methods needed to process chain segments in
diff --git a/core/chain_indexer_test.go b/core/chain_indexer_test.go
index 3edf175d..481f4e15 100644
--- a/core/chain_indexer_test.go
+++ b/core/chain_indexer_test.go
@@ -35,9 +35,9 @@ import (
 	"testing"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 // Runs multiple tests with randomized parameters.
diff --git a/core/chain_makers.go b/core/chain_makers.go
index 9830ff71..e65d58ab 100644
--- a/core/chain_makers.go
+++ b/core/chain_makers.go
@@ -30,15 +30,15 @@ import (
 	"fmt"
 	"math/big"
 
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/consensus/misc"
-	"github.com/ava-labs/coreth/core/state"
```

```
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/consensus/misc"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
 )
 
 // BlockGen creates blocks for testing.
@@ -111,7 +111,7 @@ func (b *BlockGen) AddTx(tx *types.Transaction) {
 // the block in chain will be returned.
 func (b *BlockGen) AddTxWithChain(bc *BlockChain, tx *types.Transaction) {
        if b.gasPool == nil {
-               b.SetCoinbase(common.Address{})
+               b.SetCoinbase(common.HexToAddress("0x0100000000000000000000000000000000000000"))
        }
        b.statedb.Prepare(tx.Hash(), len(b.txs))
        receipt, err := ApplyTransaction(b.config, bc, &b.header.Coinbase, b.gasPool, b.statedb, b.header, tx, &b.header.GasUsed, vm.Config{})
@@ -217,11 +217,15 @@ func GenerateChain(config *params.ChainConfig, parent *types.Block, engine conse
                b.header = makeHeader(chainreader, config, parent, gap, statedb, b.engine)
 
                // Mutate the state and block according to any hard-fork specs
-               if daoBlock := config.DAOForkBlock; daoBlock != nil {
-                       limit := new(big.Int).Add(daoBlock, params.DAOForkExtraRange)
-                       if b.header.Number.Cmp(daoBlock) >= 0 && b.header.Number.Cmp(limit) < 0 {
-                               if config.DAOForkSupport {
-                                       b.header.Extra = common.CopyBytes(params.DAOForkBlockExtra)
+               timestamp := new(big.Int).SetUint64(b.header.Time)
+               if !config.IsApricotPhase3(timestamp) {
+                       // avoid dynamic fee extra data override
+                       if daoBlock := config.DAOForkBlock; daoBlock != nil {
+                               limit := new(big.Int).Add(daoBlock, params.DAOForkExtraRange)
+                               if b.header.Number.Cmp(daoBlock) >= 0 && b.header.Number.Cmp(limit) < 0 {
+                                       if config.DAOForkSupport {
+                                               b.header.Extra = common.CopyBytes(params.DAOForkBlockExtra)
+                                       }
                                }
                        }
                }
@@ -277,7 +281,9 @@ func makeHeader(chain consensus.ChainReader, config *params.ChainConfig, parent
 
        timestamp := new(big.Int).SetUint64(time)
        var gasLimit uint64
-       if config.IsApricotPhase1(timestamp) {
+       if config.IsApricotPhase5(timestamp) {
+               gasLimit = params.ApricotPhase5GasLimit
+       } else if config.IsApricotPhase1(timestamp) {
                gasLimit = params.ApricotPhase1GasLimit
        } else {
                gasLimit = CalcGasLimit(parent.GasUsed(), parent.GasLimit(), parent.GasLimit(), parent.GasLimit())
diff --git a/core/chain_makers_test.go b/core/chain_makers_test.go
index cd003932..84a4b7e6 100644
--- a/core/chain_makers_test.go
+++ b/core/chain_makers_test.go
@@ -30,13 +30,13 @@ import (
        "fmt"
        "math/big"
 
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )
 
 func ExampleGenerateChain() {
diff --git a/core/dao_test.go b/core/dao_test.go
new file mode 100644
index 00000000..16cb7de7
--- /dev/null
+++ b/core/dao_test.go
@@ -0,0 +1,197 @@
+// (c) 2021-2022, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2016 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       "math/big"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+)
+
+// Tests that DAO-fork enabled clients can properly filter out fork-commencing
+// blocks based on their extradata fields.
+func TestDAOForkRangeExtradata(t *testing.T) {
+       forkBlock := big.NewInt(32)
+
+       // Generate a common prefix for both pro-forkers and non-forkers
+       db := rawdb.NewMemoryDatabase()
+       gspec := &Genesis{
+               BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee),
+               Config:  params.TestApricotPhase2Config,
+       }
+       genesis := gspec.MustCommit(db)
+       prefix, _, _ := GenerateChain(params.TestApricotPhase2Config, genesis, dummy.NewFaker(), db, int(forkBlock.Int64()-1), 10, func(i int, gen *BlockGen) {})
+
+       // Create the concurrent, conflicting two nodes
+       proDb := rawdb.NewMemoryDatabase()
+       gspec.MustCommit(proDb)
+
+       proConf := *params.TestApricotPhase2Config
+       proConf.DAOForkBlock = forkBlock
+       proConf.DAOForkSupport = true
+
+       proBc, _ := NewBlockChain(proDb, DefaultCacheConfig, &proConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       defer proBc.Stop()
+
+       conDb := rawdb.NewMemoryDatabase()
+       gspec.MustCommit(conDb)
+
+       conConf := *params.TestApricotPhase2Config
+       conConf.DAOForkBlock = forkBlock
+       conConf.DAOForkSupport = false
+
+       conBc, _ := NewBlockChain(conDb, DefaultCacheConfig, &conConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       defer conBc.Stop()
+
+       if _, err := proBc.InsertChain(prefix); err != nil {
+               t.Fatalf("pro-fork: failed to import chain prefix: %v", err)
+       }
+       if _, err := conBc.InsertChain(prefix); err != nil {
+               t.Fatalf("con-fork: failed to import chain prefix: %v", err)
+       }
+       // Try to expand both pro-fork and non-fork chains iteratively with other camp's blocks
+       for i := int64(0); i < params.DAOForkExtraRange.Int64(); i++ {
+               // Create a pro-fork block, and try to feed into the no-fork chain
+               db = rawdb.NewMemoryDatabase()
+               gspec.MustCommit(db)
+               bc, _ := NewBlockChain(db, DefaultCacheConfig, &conConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+               defer bc.Stop()
+
+               blocks := conBc.GetBlocksFromHash(conBc.CurrentBlock().Hash(), int(conBc.CurrentBlock().NumberU64()))
+               for j := 0; j < len(blocks)/2; j++ {
+                       blocks[j], blocks[len(blocks)-1-j] = blocks[len(blocks)-1-j], blocks[j]
+               }
+               if _, err := bc.InsertChain(blocks); err != nil {
+                       t.Fatalf("failed to import contra-fork chain for expansion: %v", err)
+               }
+               if err := bc.stateCache.TrieDB().Commit(bc.CurrentHeader().Root, true, nil); err != nil {
+                       t.Fatalf("failed to commit contra-fork head for expansion: %v", err)
+               }
+               blocks, _, _ = GenerateChain(&proConf, conBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+               if _, err := conBc.InsertChain(blocks); err != nil {
+                       t.Fatalf("contra-fork chain accepted pro-fork block: %v", blocks[0])
+               }
+               // Create a proper no-fork block for the contra-forker
+               blocks, _, _ = GenerateChain(&conConf, conBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+               if _, err := conBc.InsertChain(blocks); err != nil {
+                       t.Fatalf("contra-fork chain didn't accepted no-fork block: %v", err)
+               }
+               // Create a no-fork block, and try to feed into the pro-fork chain
+               db = rawdb.NewMemoryDatabase()
+               gspec.MustCommit(db)
+               bc, _ = NewBlockChain(db, DefaultCacheConfig, &proConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+               defer bc.Stop()
+
+               blocks = proBc.GetBlocksFromHash(proBc.CurrentBlock().Hash(), int(proBc.CurrentBlock().NumberU64()))
+               for j := 0; j < len(blocks)/2; j++ {
+                       blocks[j], blocks[len(blocks)-1-j] = blocks[len(blocks)-1-j], blocks[j]
+               }
+               if _, err := bc.InsertChain(blocks); err != nil {
+                       t.Fatalf("failed to import pro-fork chain for expansion: %v", err)
+               }
+               if err := bc.stateCache.TrieDB().Commit(bc.CurrentHeader().Root, true, nil); err != nil {
+                       t.Fatalf("failed to commit pro-fork head for expansion: %v", err)
+               }
+               blocks, _, _ = GenerateChain(&conConf, proBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+               if _, err := proBc.InsertChain(blocks); err != nil {
+                       t.Fatalf("pro-fork chain accepted contra-fork block: %v", blocks[0])
+               }
+               // Create a proper pro-fork block for the pro-forker
+               blocks, _, _ = GenerateChain(&proConf, proBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+               if _, err := proBc.InsertChain(blocks); err != nil {
+                       t.Fatalf("pro-fork chain didn't accepted pro-fork block: %v", err)
+               }
+       }
+       // Verify that contra-forkers accept pro-fork extra-datas after forking finishes
+       db = rawdb.NewMemoryDatabase()
+       gspec.MustCommit(db)
+       bc, _ := NewBlockChain(db, DefaultCacheConfig, &conConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       defer bc.Stop()
+
+       blocks := conBc.GetBlocksFromHash(conBc.CurrentBlock().Hash(), int(conBc.CurrentBlock().NumberU64()))
+       for j := 0; j < len(blocks)/2; j++ {
+               blocks[j], blocks[len(blocks)-1-j] = blocks[len(blocks)-1-j], blocks[j]
+       }
+       if _, err := bc.InsertChain(blocks); err != nil {
+               t.Fatalf("failed to import contra-fork chain for expansion: %v", err)
+       }
+       if err := bc.stateCache.TrieDB().Commit(bc.CurrentHeader().Root, true, nil); err != nil {
+               t.Fatalf("failed to commit contra-fork head for expansion: %v", err)
+       }
+       blocks, _, _ = GenerateChain(&proConf, conBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+       if _, err := conBc.InsertChain(blocks); err != nil {
+               t.Fatalf("contra-fork chain didn't accept pro-fork block post-fork: %v", err)
+       }
+       // Verify that pro-forkers accept contra-fork extra-datas after forking finishes
+       db = rawdb.NewMemoryDatabase()
+       gspec.MustCommit(db)
+       bc, _ = NewBlockChain(db, DefaultCacheConfig, &proConf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       defer bc.Stop()
+
+       blocks = proBc.GetBlocksFromHash(proBc.CurrentBlock().Hash(), int(proBc.CurrentBlock().NumberU64()))
+       for j := 0; j < len(blocks)/2; j++ {
+               blocks[j], blocks[len(blocks)-1-j] = blocks[len(blocks)-1-j], blocks[j]
+       }
+       if _, err := bc.InsertChain(blocks); err != nil {
+               t.Fatalf("failed to import pro-fork chain for expansion: %v", err)
+       }
+       if err := bc.stateCache.TrieDB().Commit(bc.CurrentHeader().Root, true, nil); err != nil {
+               t.Fatalf("failed to commit pro-fork head for expansion: %v", err)
+       }
+       blocks, _, _ = GenerateChain(&conConf, proBc.CurrentBlock(), dummy.NewFaker(), db, 1, 10, func(i int, gen *BlockGen) {})
+       if _, err := proBc.InsertChain(blocks); err != nil {
+               t.Fatalf("pro-fork chain didn't accept contra-fork block post-fork: %v", err)
+       }
+}
+
+func TestDAOForkSupportPostApricotPhase3(t *testing.T) {
+       forkBlock := big.NewInt(0)
+
+       conf := *params.TestChainConfig
+       conf.DAOForkSupport = true
+       conf.DAOForkBlock = forkBlock
+
+	db := rawdb.NewMemoryDatabase()
+	gspec := &Genesis{
+		BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee),
+		Config:  &conf,
+	}
+	genesis := gspec.MustCommit(db)
+	bc, _ := NewBlockChain(db, DefaultCacheConfig, &conf, dummy.NewFaker(), vm.Config{}, common.Hash{})
+	defer bc.Stop()
+
+	blocks, _, _ := GenerateChain(&conf, genesis, dummy.NewFaker(), db, 32, 10, func(i int, gen *BlockGen) {})
+
+	if _, err := bc.InsertChain(blocks); err != nil {
+		t.Fatalf("failed to import blocks: %v", err)
+	}
+}
diff --git a/core/error.go b/core/error.go
index 02654489..a938eef8 100644
--- a/core/error.go
+++ b/core/error.go
@@ -29,7 +29,7 @@ package core
 import (
 	"errors"
 
-	"github.com/ava-labs/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 var (
@@ -56,6 +56,10 @@ var (
 	// next one expected based on the local chain.
 	ErrNonceTooHigh = errors.New("nonce too high")
 
+	// ErrNonceMax is returned if the nonce of a transaction sender account has
+	// maximum allowed value and would become invalid if incremented.
+	ErrNonceMax = errors.New("nonce has max value")
+
 	// ErrGasLimitReached is returned by the gas pool if the amount of gas required
 	// by a transaction is higher than what's left in the block.
 	ErrGasLimitReached = errors.New("gas limit reached")
diff --git a/core/events.go b/core/events.go
index 4898dbc0..f71c2d40 100644
--- a/core/events.go
+++ b/core/events.go
@@ -27,8 +27,8 @@
 package core
 
 import (
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 // NewTxsEvent is posted when a batch of transactions enter the transaction pool.
diff --git a/core/evm.go b/core/evm.go
index d45f2413..19a85573 100644
--- a/core/evm.go
+++ b/core/evm.go
@@ -29,10 +29,10 @@ package core
 import (
 	"math/big"
 
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
 	//"github.com/ethereum/go-ethereum/log"
 )
 
@@ -125,15 +125,8 @@ func CanTransfer(db vm.StateDB, addr common.Address, amount *big.Int) bool {
 	return db.GetBalance(addr).Cmp(amount) >= 0
 }
 
-func CanTransferMC(db vm.StateDB, addr common.Address, to common.Address, coinID *common.Hash, amount *big.Int) bool {
-	if coinID == nil {
-		return true
-	}
-	if db.GetBalanceMultiCoin(addr, *coinID).Cmp(amount) >= 0 {
-		return true
-	}
-	// insufficient balance
-	return false
+func CanTransferMC(db vm.StateDB, addr common.Address, to common.Address, coinID common.Hash, amount *big.Int) bool {
+	return db.GetBalanceMultiCoin(addr, coinID).Cmp(amount) >= 0
 }
 
 // Transfer subtracts amount from sender and adds amount to recipient using the given Db
@@ -143,10 +136,7 @@ func Transfer(db vm.StateDB, sender, recipient common.Address, amount *big.Int)
 }
 
 // Transfer subtracts amount from sender and adds amount to recipient using the given Db
-func TransferMultiCoin(db vm.StateDB, sender, recipient common.Address, coinID *common.Hash, amount *big.Int) {
-	if coinID == nil {
-		return
-	}
-	db.SubBalanceMultiCoin(sender, *coinID, amount)
-	db.AddBalanceMultiCoin(recipient, *coinID, amount)
+func TransferMultiCoin(db vm.StateDB, sender, recipient common.Address, coinID common.Hash, amount *big.Int) {
+	db.SubBalanceMultiCoin(sender, coinID, amount)
+	db.AddBalanceMultiCoin(recipient, coinID, amount)
 }
diff --git a/core/gen_genesis.go b/core/gen_genesis.go
index a4ec8f54..bc942f96 100644
--- a/core/gen_genesis.go
+++ b/core/gen_genesis.go
@@ -7,7 +7,7 @@ import (
 	"errors"
 	"math/big"
 
-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/common/math"
diff --git a/core/genesis.go b/core/genesis.go
index 7e10249d..ce6b02dd 100644
--- a/core/genesis.go
+++ b/core/genesis.go
@@ -34,16 +34,16 @@ import (
 	"fmt"
 	"math/big"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/common/math"
 	"github.com/ethereum/go-ethereum/log"
```

```
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/trie"
 )

 //go:generate gencodec -type Genesis -field-override genesisSpecMarshaling -out gen_genesis.go
@@ -209,12 +209,14 @@ func SetupGenesisBlock(db ethdb.Database, genesis *Genesis) (*params.ChainConfig

        // Check config compatibility and write the config. Compatibility errors
        // are returned to the caller unless we're already at block zero.
-       height := rawdb.ReadHeaderNumber(db, rawdb.ReadHeadHeaderHash(db))
-       if height == nil {
-               return newcfg, fmt.Errorf("missing block number for head header hash")
+       headBlock := rawdb.ReadHeadBlock(db)
+       if headBlock == nil {
+               return newcfg, fmt.Errorf("missing head block")
        }
-       compatErr := storedcfg.CheckCompatible(newcfg, *height)
-       if compatErr != nil && *height != 0 && compatErr.RewindTo != 0 {
+       height := headBlock.NumberU64()
+       timestamp := headBlock.Time()
+       compatErr := storedcfg.CheckCompatible(newcfg, height, timestamp)
+       if compatErr != nil && height != 0 && compatErr.RewindTo != 0 {
                return newcfg, compatErr
        }
        rawdb.WriteChainConfig(db, stored, newcfg)
@@ -292,12 +294,10 @@ func (g *Genesis) Commit(db ethdb.Database) (*types.Block, error) {
        if err := config.CheckConfigForkOrder(); err != nil {
                return nil, err
        }
-       rawdb.WriteTd(db, block.Hash(), block.NumberU64(), g.Difficulty)
        rawdb.WriteBlock(db, block)
        rawdb.WriteReceipts(db, block.Hash(), block.NumberU64(), nil)
        rawdb.WriteCanonicalHash(db, block.Hash(), block.NumberU64())
        rawdb.WriteHeadBlockHash(db, block.Hash())
-       rawdb.WriteHeadFastBlockHash(db, block.Hash())
        rawdb.WriteHeadHeaderHash(db, block.Hash())
        rawdb.WriteChainConfig(db, block.Hash(), config)
        return block, nil
@@ -316,6 +316,7 @@ func (g *Genesis) MustCommit(db ethdb.Database) *types.Block {
 // GenesisBlockForTesting creates and writes a block in which addr has the given wei balance.
 func GenesisBlockForTesting(db ethdb.Database, addr common.Address, balance *big.Int) *types.Block {
        g := Genesis{
+               Config:  params.TestChainConfig,
                Alloc:   GenesisAlloc{addr: {Balance: balance}},
                BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee),
        }
diff --git a/core/genesis_test.go b/core/genesis_test.go
new file mode 100644
index 00000000..533bed18
--- /dev/null
+++ b/core/genesis_test.go
@@ -0,0 +1,167 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       _ "embed"
+       "math/big"
+       "reflect"
+       "testing"
+
+       "github.com/davecgh/go-spew/spew"
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
+)
+
+func setupGenesisBlock(db ethdb.Database, genesis *Genesis) (*params.ChainConfig, common.Hash, error) {
+       conf, err := SetupGenesisBlock(db, genesis)
+       stored := rawdb.ReadCanonicalHash(db, 0)
+       return conf, stored, err
+}
+
+func TestGenesisBlockForTesting(t *testing.T) {
+       genesisBlockForTestingHash := common.HexToHash("0xb378f22ccd9ad52c6c42f5d46ef2aad6d6866cfcb778ea97a0b6dfde13387330")
+       block := GenesisBlockForTesting(rawdb.NewMemoryDatabase(), common.Address{1}, big.NewInt(1))
+       if block.Hash() != genesisBlockForTestingHash {
+               t.Errorf("wrong testing genesis hash, got %v, want %v", block.Hash(), genesisBlockForTestingHash)
+       }
+}
+
+func TestSetupGenesis(t *testing.T) {
+       apricotPhase1Config := *params.TestApricotPhase1Config
+       apricotPhase1Config.ApricotPhase1BlockTimestamp = big.NewInt(100)
+       var (
+               customghash = common.HexToHash("0x1099a11e9e454bd3ef31d688cf21936671966407bc330f051d754b5ce401e7ed")
+               customg     = Genesis{
+                       Config: &apricotPhase1Config,
+                       Alloc: GenesisAlloc{
+                               {1}: {Balance: big.NewInt(1), Storage: map[common.Hash]common.Hash{{1}: {1}}},
+                       },
+               }
+               oldcustomg = customg
+       )
+
+       rollbackApricotPhase1Config := apricotPhase1Config
+       rollbackApricotPhase1Config.ApricotPhase1BlockTimestamp = big.NewInt(90)
+       oldcustomg.Config = &rollbackApricotPhase1Config
+       tests := []struct {
+               name       string
+               fn         func(ethdb.Database) (*params.ChainConfig, common.Hash, error)
+               wantConfig *params.ChainConfig
+               wantHash   common.Hash
+               wantErr    error
+       }{
+               {
+                       name: "genesis without ChainConfig",
+                       fn: func(db ethdb.Database) (*params.ChainConfig, common.Hash, error) {
+                               return setupGenesisBlock(db, new(Genesis))
+                       },
+                       wantErr:    errGenesisNoConfig,
+                       wantConfig: nil,
+               },
+               {
+                       name: "no block in DB, genesis == nil",
+                       fn: func(db ethdb.Database) (*params.ChainConfig, common.Hash, error) {
+                               return setupGenesisBlock(db, nil)
+                       },
+                       wantErr:    ErrNoGenesis,
+                       wantConfig: nil,
+               },
+               {
+                       name: "custom block in DB, genesis == nil",
+                       fn: func(db ethdb.Database) (*params.ChainConfig, common.Hash, error) {
+                               customg.MustCommit(db)
+                               return setupGenesisBlock(db, nil)
+                       },
+                       wantErr:    ErrNoGenesis,
+                       wantHash:   customghash,
+                       wantConfig: nil,
+               },
+               {
+                       name: "compatible config in DB",
+                       fn: func(db ethdb.Database) (*params.ChainConfig, common.Hash, error) {
+                               oldcustomg.MustCommit(db)
+                               return setupGenesisBlock(db, &customg)
+                       },
+                       wantHash:   customghash,
+                       wantConfig: customg.Config,
+               },
+               {
+                       name: "incompatible config for avalanche fork in DB",
+                       fn: func(db ethdb.Database) (*params.ChainConfig, common.Hash, error) {
+                               // Commit the 'old' genesis block with ApricotPhase1 transition at 90.
+                               // Advance to block #4, past the ApricotPhase1 transition block of customg.
+                               genesis := oldcustomg.MustCommit(db)
+
+                               bc, _ := NewBlockChain(db, DefaultCacheConfig, oldcustomg.Config, dummy.NewFullFaker(), vm.Config{}, common.Hash{})
+                               defer bc.Stop()
+
+                               blocks, _, _ := GenerateChain(oldcustomg.Config, genesis, dummy.NewFullFaker(), db, 4, 25, nil)
+                               bc.InsertChain(blocks)
+                               bc.CurrentBlock()
+                               // This should return a compatibility error.
+                               return setupGenesisBlock(db, &customg)
+                       },
+                       wantHash:   customghash,
+                       wantConfig: customg.Config,
+                       wantErr: &params.ConfigCompatError{
+                               What:         "ApricotPhase1 fork block timestamp",
+                               StoredConfig: big.NewInt(90),
+                               NewConfig:    big.NewInt(100),
+                               RewindTo:     89,
+                       },
+               },
+       }
+
+       for _, test := range tests {
+               t.Run(test.name, func(t *testing.T) {
+                       db := rawdb.NewMemoryDatabase()
+                       config, hash, err := test.fn(db)
+                       // Check the return values.
+                       if !reflect.DeepEqual(err, test.wantErr) {
+                               spew := spew.ConfigState{DisablePointerAddresses: true, DisableCapacities: true}
+                               t.Errorf("returned error %#v, want %#v", spew.NewFormatter(err), spew.NewFormatter(test.wantErr))
+                       }
+                       if !reflect.DeepEqual(config, test.wantConfig) {
+                               t.Errorf("returned %v\nwant     %v", config, test.wantConfig)
+                       }
+                       if hash != test.wantHash {
+                               t.Errorf("returned hash %s, want %s", hash.Hex(), test.wantHash.Hex())
+                       } else if err == nil {
+                               // Check database content.
+                               stored := rawdb.ReadBlock(db, test.wantHash, 0)
+                               if stored.Hash() != test.wantHash {
+                                       t.Errorf("block in DB has hash %s, want %s", stored.Hash(), test.wantHash)
+                               }
+                       }
+               })
+       }
+}
diff --git a/core/headerchain.go b/core/headerchain.go
index 092350a0..48b81e19 100644
--- a/core/headerchain.go
+++ b/core/headerchain.go
@@ -33,12 +33,12 @@ import (
        mrand "math/rand"
        "sync/atomic"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
        lru "github.com/hashicorp/golang-lru"
 )

@@ -55,9 +55,9 @@ const (
 // HeaderChain is responsible for maintaining the header chain including the
 // header query and updating.
 //
-// The components maintained by headerchain includes: (1) total difficult
-// (2) header (3) block hash -> number mapping (4) canonical number -> hash mapping
-// and (5) head header flag.
+// The components maintained by headerchain includes:
+// (1) header (2) block hash -> number mapping (3) canonical number -> hash mapping
+// and (4) head header flag.
 //
 // It is not thread safe either, the encapsulating chain structures should do
 // the necessary mutex locking/unlocking.
@@ -131,22 +131,6 @@ func (hc *HeaderChain) GetBlockNumber(hash common.Hash) *uint64 {
        return number
 }

-// GetTd retrieves a block's total difficulty in the canonical chain from the
-// database by hash and number, caching it if found.
-func (hc *HeaderChain) GetTd(hash common.Hash, number uint64) *big.Int {
-       // Short circuit if the td's already in the cache, retrieve otherwise
-       if cached, ok := hc.tdCache.Get(hash); ok {
-               return cached.(*big.Int)
-       }
-       td := rawdb.ReadTd(hc.chainDb, hash, number)
-       if td == nil {
-               return nil
-       }
-       // Cache the found body for next time and return
-       hc.tdCache.Add(hash, td)
-       return td
-}
-
 // GetHeader retrieves a block header from the database by hash and number,
 // caching it if found.
 func (hc *HeaderChain) GetHeader(hash common.Hash, number uint64) *types.Header {
diff --git a/core/headerchain_test.go b/core/headerchain_test.go
new file mode 100644
index 00000000..552dcc38
--- /dev/null
+++ b/core/headerchain_test.go
@@ -0,0 +1,123 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       "errors"
+       "fmt"
+       "math/big"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+)
+
+func verifyUnbrokenCanonchain(bc *BlockChain) error {
+       h := bc.hc.CurrentHeader()
+       for {
+               canonHash := rawdb.ReadCanonicalHash(bc.hc.chainDb, h.Number.Uint64())
+               if exp := h.Hash(); canonHash != exp {
+                       return fmt.Errorf("Canon hash chain broken, block %d got %x, expected %x",
+                               h.Number, canonHash[:8], exp[:8])
+               }
+               if h.Number.Uint64() == 0 {
+                       break
+               }
+               h = bc.hc.GetHeader(h.ParentHash, h.Number.Uint64()-1)
+       }
+       return nil
+}
+
+func testInsert(t *testing.T, bc *BlockChain, chain []*types.Block, wantErr error) {
+       t.Helper()
+
+       _, err := bc.InsertChain(chain)
+       // Always verify that the header chain is unbroken
+       if err := verifyUnbrokenCanonchain(bc); err != nil {
+               t.Fatal(err)
+       }
+       if !errors.Is(err, wantErr) {
+               t.Fatalf("unexpected error from InsertHeaderChain: %v", err)
+       }
+}
+
+// This test checks status reporting of InsertHeaderChain.
+func TestHeaderInsertion(t *testing.T) {
+       var (
+               db      = rawdb.NewMemoryDatabase()
+               genesis = (&Genesis{
+                       BaseFee: big.NewInt(params.ApricotPhase3InitialBaseFee),
+                       Config:  params.TestChainConfig,
+               }).MustCommit(db)
+       )
+       chain, err := NewBlockChain(db, DefaultCacheConfig, params.TestChainConfig, dummy.NewFaker(), vm.Config{}, common.Hash{})
+       if err != nil {
+               t.Fatal(err)
+       }
+       // chain A: G->A1->A2...A128
+       chainA, _, _ := GenerateChain(params.TestChainConfig, types.NewBlockWithHeader(genesis.Header()), dummy.NewFaker(), db, 128, 10, func(i int, b *BlockGen) {
+               b.SetCoinbase(common.Address{0: byte(10), 19: byte(i)})
+       })
+       // chain B: G->A1->B2...B128
+       chainB, _, _ := GenerateChain(params.TestChainConfig, types.NewBlockWithHeader(chainA[0].Header()), dummy.NewFaker(), db, 128, 10, func(i int, b *BlockGen) {
+               b.SetCoinbase(common.Address{0: byte(10), 19: byte(i)})
+       })
+       log.Root().SetHandler(log.StdoutHandler)
+
+       // Inserting 64 headers on an empty chain
+       testInsert(t, chain, chainA[:64], nil)
+
+       // Inserting 64 identical headers
+       testInsert(t, chain, chainA[:64], nil)
+
+       // Inserting the same some old, some new headers
+       testInsert(t, chain, chainA[32:96], nil)
+
+       // Inserting side blocks, but not overtaking the canon chain
+       testInsert(t, chain, chainB[0:32], nil)
+
+       // Inserting more side blocks, but we don't have the parent
+       testInsert(t, chain, chainB[34:36], consensus.ErrUnknownAncestor)
+
+       // Inserting more sideblocks, overtaking the canon chain
+       testInsert(t, chain, chainB[32:97], nil)
+
+       // Inserting more A-headers, taking back the canonicality
+       testInsert(t, chain, chainA[90:100], nil)
+
+       // And B becomes canon again
+       testInsert(t, chain, chainB[97:107], nil)
+
+       // And B becomes even longer
+       testInsert(t, chain, chainB[107:128], nil)
+}
diff --git a/core/keeper.go b/core/keeper.go
new file mode 100644
index 00000000..a011f45c
--- /dev/null
+++ b/core/keeper.go
@@ -0,0 +1,149 @@
+// (c) 2021, Flare Networks Limited. All rights reserved.
+// Please see the file LICENSE for licensing terms.
+
+package core
+
+import (
+       "fmt"
+       "math/big"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+
+       "github.com/flare-foundation/coreth/core/vm"
+)
+
+// Define errors
+type ErrInvalidKeeperData struct{}
+
+func (e *ErrInvalidKeeperData) Error() string { return "invalid return data from keeper trigger" }
+
+type ErrKeeperDataEmpty struct{}
+
+func (e *ErrKeeperDataEmpty) Error() string { return "return data from keeper trigger empty" }
+
+type ErrMaxMintExceeded struct {
+       mintMax     *big.Int
+       mintRequest *big.Int
+}
+
+func (e *ErrMaxMintExceeded) Error() string {
+       return fmt.Sprintf("mint request of %s exceeded max of %s", e.mintRequest.Text(10), e.mintMax.Text(10))
+}
+
+type ErrMintNegative struct{}
+
+func (e *ErrMintNegative) Error() string { return "mint request cannot be negative" }
+
+// Define interface for dependencies
+type EVMCaller interface {
+       Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error)
+       GetBlockNumber() *big.Int
+       GetGasLimit() uint64
+       AddBalance(addr common.Address, amount *big.Int)
+}
+
+// Define maximums that can change by block height
+func GetKeeperGasMultiplier(blockNumber *big.Int) uint64 {
+       switch {
+       default:
+               return 100
+       }
+}
+
+func GetSystemTriggerContractAddr(blockNumber *big.Int) string {
+       switch {
+       default:
+               return "0x1000000000000000000000000000000000000002"
+       }
+}
+
+func GetSystemTriggerSelector(blockNumber *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0x7f, 0xec, 0x8d, 0x38}
+       }
+}
+
+func GetPrioritisedFTSOContract(blockTime *big.Int) string {
+       switch {
+       default:
+               return "0x1000000000000000000000000000000000000003"
+       }
+}
+
+func GetMaximumMintRequest(blockNumber *big.Int) *big.Int {
+       switch {
+       default:
+               maxRequest, _ := new(big.Int).SetString("50000000000000000000000000", 10)
+               return maxRequest
+       }
+}
+
+func triggerKeeper(evm EVMCaller) (*big.Int, error) {
+       bigZero := big.NewInt(0)
+       // Get the contract to call
+       systemTriggerContract := common.HexToAddress(GetSystemTriggerContractAddr(evm.GetBlockNumber()))
+       // Call the method
+       triggerRet, _, triggerErr := evm.Call(
+               vm.AccountRef(systemTriggerContract),
+               systemTriggerContract,
+               GetSystemTriggerSelector(evm.GetBlockNumber()),
+               GetKeeperGasMultiplier(evm.GetBlockNumber())*evm.GetGasLimit(),
+               bigZero)
+       // If no error and a value came back...
+       if triggerErr == nil && triggerRet != nil {
+               // Did we get one big int?
+               if len(triggerRet) == 32 {
+                       // Convert to big int
+                       // Mint request cannot be less than 0 as SetBytes treats value as unsigned
+                       mintRequest := new(big.Int).SetBytes(triggerRet)
+                       // return the mint request
+                       return mintRequest, nil
+               } else {
+                       // Returned length was not 32 bytes
+                       return bigZero, &ErrInvalidKeeperData{}
+               }
+       } else {
+               if triggerErr != nil {
+                       return bigZero, triggerErr
+               } else {
+                       return bigZero, &ErrKeeperDataEmpty{}
+               }
+       }
+}
+
+func mint(evm EVMCaller, mintRequest *big.Int) error {
+       // If the mint request is greater than zero and less than max
+       max := GetMaximumMintRequest(evm.GetBlockNumber())
+       if mintRequest.Cmp(big.NewInt(0)) > 0 &&
+               mintRequest.Cmp(max) <= 0 {
```

```
+               // Mint the amount asked for on to the keeper contract
+               evm.AddBalance(common.HexToAddress(GetSystemTriggerContractAddr(evm.GetBlockNumber())), mintRequest)
+       } else if mintRequest.Cmp(max) > 0 {
+               // Return error
+               return &ErrMaxMintExceeded{
+                       mintRequest: mintRequest,
+                       mintMax:     max,
+               }
+       } else if mintRequest.Cmp(big.NewInt(0)) < 0 {
+               // Cannot mint negatives
+               return &ErrMintNegative{}
+       }
+       // No error
+       return nil
+}
+
+func triggerKeeperAndMint(evm EVMCaller, log log.Logger) {
+       // Call the keeper
+       mintRequest, triggerErr := triggerKeeper(evm)
+       // If no error...
+       if triggerErr == nil {
+               // time to mint
+               if mintError := mint(evm, mintRequest); mintError != nil {
+                       log.Warn("Error minting inflation request", "error", mintError)
+               }
+       } else {
+               log.Warn("Keeper trigger in error", "error", triggerErr)
+       }
+}
diff --git a/core/keeper_test.go b/core/keeper_test.go
new file mode 100644
index 00000000..ca11ee04
--- /dev/null
+++ b/core/keeper_test.go
@@ -0,0 +1,451 @@
+// (c) 2021, Flare Networks Limited. All rights reserved.
+// Please see the file LICENSE for licensing terms.
+
+package core
+
+import (
+       "errors"
+       "math/big"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+
+       "github.com/flare-foundation/coreth/core/vm"
+)
+
+// Define a mock structure to spy and mock values for keeper calls
+type MockEVMCallerData struct {
+       callCalls            int
+       addBalanceCalls      int
+       blockNumber          big.Int
+       gasLimit             uint64
+       mintRequestReturn    big.Int
+       lastAddBalanceAddr   common.Address
+       lastAddBalanceAmount *big.Int
+}
+
+// Define a mock structure to spy and mock values for logger calls
+type MockLoggerData struct {
+       warnCalls int
+}
+
+// Set up default mock method calls
+func defautCall(e *MockEVMCallerData, caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       e.callCalls++
+
+       buffer := []byte{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
+       return e.mintRequestReturn.FillBytes(buffer), 0, nil
+}
+
+func defaultGetBlockNumber(e *MockEVMCallerData) *big.Int {
+       return &e.blockNumber
+}
+
+func defaultGetGasLimit(e *MockEVMCallerData) uint64 {
+       return e.gasLimit
+}
+
+func defaultAddBalance(e *MockEVMCallerData, addr common.Address, amount *big.Int) {
+       e.addBalanceCalls++
+       e.lastAddBalanceAddr = addr
+       e.lastAddBalanceAmount = amount
+}
+
+// Define the default EVM mock and define default mock receiver functions
+type DefaultEVMMock struct {
+       mockEVMCallerData MockEVMCallerData
+}
+
+func (e *DefaultEVMMock) Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       return defautCall(&e.mockEVMCallerData, caller, addr, input, gas, value)
+}
+
+func (e *DefaultEVMMock) GetBlockNumber() *big.Int {
+       return defaultGetBlockNumber(&e.mockEVMCallerData)
+}
+
+func (e *DefaultEVMMock) GetGasLimit() uint64 {
+       return defaultGetGasLimit(&e.mockEVMCallerData)
+}
+
+func (e *DefaultEVMMock) AddBalance(addr common.Address, amount *big.Int) {
+       defaultAddBalance(&e.mockEVMCallerData, addr, amount)
+}
+
+func TestKeeperTriggerShouldReturnMintRequest(t *testing.T) {
+       mintRequestReturn, _ := new(big.Int).SetString("50000000000000000000000000", 10)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *mintRequestReturn,
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       mintRequest, _ := triggerKeeper(defaultEVMMock)
+
+       if mintRequest.Cmp(mintRequestReturn) != 0 {
+               t.Errorf("got %s want %q", mintRequest.Text(10), "50000000000000000000000000")
+       }
+}
+
+func TestKeeperTriggerShouldNotLetMintRequestOverflow(t *testing.T) {
+       var mintRequestReturn big.Int
+       // TODO: Compact with exponent?
+       buffer := []byte{255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255, 255}
+       mintRequestReturn.SetBytes(buffer)
+
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: mintRequestReturn,
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       mintRequest, mintRequestError := triggerKeeper(defaultEVMMock)
+
+       if mintRequestError != nil {
+               t.Errorf("received unexpected error %s", mintRequestError)
+       }
+
+       if mintRequest.Sign() < 1 {
+               t.Errorf("unexpected negative")
+       }
+}
+
+// Define a bad mint request return size mock
+type BadMintReturnSizeEVMMock struct {
+       mockEVMCallerData MockEVMCallerData
+}
+
+func (e *BadMintReturnSizeEVMMock) Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       e.mockEVMCallerData.callCalls++
+       // Should be size 32 bytes
+       buffer := []byte{0}
+       return e.mockEVMCallerData.mintRequestReturn.FillBytes(buffer), 0, nil
+}
+
+func (e *BadMintReturnSizeEVMMock) GetBlockNumber() *big.Int {
+       return defaultGetBlockNumber(&e.mockEVMCallerData)
+}
+
+func (e *BadMintReturnSizeEVMMock) GetGasLimit() uint64 {
+       return defaultGetGasLimit(&e.mockEVMCallerData)
+}
+
+func (e *BadMintReturnSizeEVMMock) AddBalance(addr common.Address, amount *big.Int) {
+       defaultAddBalance(&e.mockEVMCallerData, addr, amount)
+}
+
+func TestKeeperTriggerValidatesMintRequestReturnValueSize(t *testing.T) {
+       var mintRequestReturn big.Int
+       // TODO: Compact with exponent?
+       buffer := []byte{255}
+       mintRequestReturn.SetBytes(buffer)
+
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: mintRequestReturn,
+       }
+       badMintReturnSizeEVMMock := &BadMintReturnSizeEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+       // Call to return less than 32 bytes
+       _, err := triggerKeeper(badMintReturnSizeEVMMock)
+
+       if err != nil {
+               if err, ok := err.(*ErrInvalidKeeperData); !ok {
+                       want := &ErrInvalidKeeperData{}
+                       t.Errorf("got '%s' want '%s'", err.Error(), want.Error())
+               }
+       } else {
+               t.Errorf("no error returned as expected")
+       }
+}
+
+// Define a mock to simulate keeper trigger returning an error from Call
+type BadTriggerCallEVMMock struct {
+       mockEVMCallerData MockEVMCallerData
+}
+
+func (e *BadTriggerCallEVMMock) Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       e.mockEVMCallerData.callCalls++
+
+       buffer := []byte{0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}
+       return e.mockEVMCallerData.mintRequestReturn.FillBytes(buffer), 0, errors.New("Call error happened")
+}
+
+func (e *BadTriggerCallEVMMock) GetBlockNumber() *big.Int {
+       return defaultGetBlockNumber(&e.mockEVMCallerData)
+}
+
+func (e *BadTriggerCallEVMMock) GetGasLimit() uint64 {
+       return defaultGetGasLimit(&e.mockEVMCallerData)
+}
+
+func (e *BadTriggerCallEVMMock) AddBalance(addr common.Address, amount *big.Int) {
+       defaultAddBalance(&e.mockEVMCallerData, addr, amount)
+}
+
+func TestKeeperTriggerReturnsCallError(t *testing.T) {
+       mockEVMCallerData := &MockEVMCallerData{}
+       badTriggerCallEVMMock := &BadTriggerCallEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+       // Call to return less than 32 bytes
+       _, err := triggerKeeper(badTriggerCallEVMMock)
+
+       if err == nil {
+               t.Errorf("no error received")
+       } else {
+               if err.Error() != "Call error happened" {
+                       t.Errorf("did not get expected error")
+               }
+       }
+}
+
+type LoggerMock struct {
+       mockLoggerData MockLoggerData
+}
+
+func (l *LoggerMock) New(ctx ...interface{}) log.Logger {
+       return nil
+}
+
+func (l *LoggerMock) GetHandler() log.Handler {
+       return nil
+}
+
+func (l *LoggerMock) SetHandler(h log.Handler) {
+}
+
+func (l *LoggerMock) Trace(msg string, ctx ...interface{}) {}
+func (l *LoggerMock) Debug(msg string, ctx ...interface{}) {}
+func (l *LoggerMock) Info(msg string, ctx ...interface{})  {}
+func (l *LoggerMock) Error(msg string, ctx ...interface{}) {}
+func (l *LoggerMock) Crit(msg string, ctx ...interface{})  {}
+
+func (l *LoggerMock) Warn(msg string, ctx ...interface{}) {
```

```
+       l.mockLoggerData.warnCalls++
+}
+
+func TestKeeperTriggerAndMintLogsError(t *testing.T) {
+       // Assemble
+       // Set up mock EVM call to return an error
+       mockEVMCallerData := &MockEVMCallerData{}
+       badTriggerCallEVMMock := &BadTriggerCallEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+       // Set up a mock logger
+       mockLoggerData := &MockLoggerData{}
+       loggerMock := &LoggerMock{
+               mockLoggerData: *mockLoggerData,
+       }
+
+       // Act
+       triggerKeeperAndMint(badTriggerCallEVMMock, loggerMock)
+
+       // Assert
+       if loggerMock.mockLoggerData.warnCalls != 1 {
+               t.Errorf("Logger.Warn not called as expected")
+       }
+}
+
+// Define a mock to simulate keeper trigger returning nil for mint request
+type ReturnNilMintRequestEVMMock struct {
+       mockEVMCallerData MockEVMCallerData
+}
+
+func (e *ReturnNilMintRequestEVMMock) Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       e.mockEVMCallerData.callCalls++
+
+       return nil, 0, nil
+}
+
+func (e *ReturnNilMintRequestEVMMock) GetBlockNumber() *big.Int {
+       return defaultGetBlockNumber(&e.mockEVMCallerData)
+}
+
+func (e *ReturnNilMintRequestEVMMock) GetGasLimit() uint64 {
+       return defaultGetGasLimit(&e.mockEVMCallerData)
+}
+
+func (e *ReturnNilMintRequestEVMMock) AddBalance(addr common.Address, amount *big.Int) {
+       defaultAddBalance(&e.mockEVMCallerData, addr, amount)
+}
+
+func TestKeeperTriggerHandlesNilMintRequest(t *testing.T) {
+       mockEVMCallerData := &MockEVMCallerData{}
+       returnNilMintRequestEVMMock := &ReturnNilMintRequestEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+       // Call to return less than 32 bytes
+       _, err := triggerKeeper(returnNilMintRequestEVMMock)
+
+       if err != nil {
+               if err, ok := err.(*ErrKeeperDataEmpty); !ok {
+                       want := &ErrKeeperDataEmpty{}
+                       t.Errorf("got '%s' want '%s'", err.Error(), want.Error())
+               }
+       } else {
+               t.Errorf("no error returned as expected")
+       }
+}
+
+func TestKeeperTriggerShouldNotMintMoreThanMax(t *testing.T) {
+       mintRequest, _ := new(big.Int).SetString("50000000000000000000000001", 10)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *big.NewInt(0),
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       err := mint(defaultEVMMock, mintRequest)
+
+       if err != nil {
+               if err, ok := err.(*ErrMaxMintExceeded); !ok {
+                       want := &ErrMaxMintExceeded{
+                               mintRequest: mintRequest,
+                               mintMax:     GetMaximumMintRequest(big.NewInt(0)),
+                       }
+                       t.Errorf("got '%s' want '%s'", err.Error(), want.Error())
+               }
+       } else {
+               t.Errorf("no error returned as expected")
+       }
+}
+
+func TestKeeperTriggerShouldNotMintNegative(t *testing.T) {
+       mintRequest := big.NewInt(-1)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *big.NewInt(0),
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       err := mint(defaultEVMMock, mintRequest)
+
+       if err != nil {
+               if err, ok := err.(*ErrMintNegative); !ok {
+                       want := &ErrMintNegative{}
+                       t.Errorf("got '%s' want '%s'", err.Error(), want.Error())
+               }
+       } else {
+               t.Errorf("no error returned as expected")
+       }
+}
+
+func TestKeeperTriggerShouldMint(t *testing.T) {
+       // Assemble
+       mintRequest, _ := new(big.Int).SetString("50000000000000000000000000", 10)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *big.NewInt(0),
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       // Act
+       err := mint(defaultEVMMock, mintRequest)
+
+       // Assert
+       if err == nil {
+               if defaultEVMMock.mockEVMCallerData.addBalanceCalls != 1 {
+                       t.Errorf("AddBalance not called as expected")
+               }
+               if defaultEVMMock.mockEVMCallerData.lastAddBalanceAddr.String() != GetSystemTriggerContractAddr(big.NewInt(0)) {
+                       t.Errorf("wanted addr %s; got addr %s", GetSystemTriggerContractAddr(big.NewInt(0)), defaultEVMMock.mockEVMCallerData.lastAddBalanceAddr)
+               }
+               if defaultEVMMock.mockEVMCallerData.lastAddBalanceAmount.Cmp(mintRequest) != 0 {
+                       t.Errorf("wanted amount %s; got amount %s", mintRequest.Text(10), defaultEVMMock.mockEVMCallerData.lastAddBalanceAmount.Text(10))
+               }
+       } else {
+               t.Errorf("unexpected error returned; was = %s", err.Error())
+       }
+}
+
+func TestKeeperTriggerShouldNotErrorMintingZero(t *testing.T) {
+       // Assemble
+       mintRequest := big.NewInt(0)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *big.NewInt(0),
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       // Act
+       err := mint(defaultEVMMock, mintRequest)
+
+       // Assert
+       if err == nil {
+               if defaultEVMMock.mockEVMCallerData.addBalanceCalls != 0 {
+                       t.Errorf("AddBalance called unexpectedly")
+               }
+       } else {
+               t.Errorf("unexpected error returned; was %s", err.Error())
+       }
+}
+
+func TestKeeperTriggerFiredAndMinted(t *testing.T) {
+       mintRequestReturn, _ := new(big.Int).SetString("5000000000000000000000000", 10)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *mintRequestReturn,
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       log := log.New()
+       triggerKeeperAndMint(defaultEVMMock, log)
+
+       // EVM Call function calling the keeper should have been cqlled
+       if defaultEVMMock.mockEVMCallerData.callCalls != 1 {
+               t.Errorf("EVM Call count not as expected. got %d want 1", defaultEVMMock.mockEVMCallerData.callCalls)
+       }
+       // AddBalance should have been called on the state database, minting the request asked for
+       if defaultEVMMock.mockEVMCallerData.addBalanceCalls != 1 {
+               t.Errorf("Add balance call count not as expected. got %d want 1", defaultEVMMock.mockEVMCallerData.addBalanceCalls)
+       }
+}
+
+func TestKeeperTriggerShouldNotMintMoreThanLimit(t *testing.T) {
+       mintRequestReturn, _ := new(big.Int).SetString("5000000000000000000000001", 10)
+       mockEVMCallerData := &MockEVMCallerData{
+               blockNumber:       *big.NewInt(0),
+               gasLimit:          0,
+               mintRequestReturn: *mintRequestReturn,
+       }
+       defaultEVMMock := &DefaultEVMMock{
+               mockEVMCallerData: *mockEVMCallerData,
+       }
+
+       log := log.New()
+       triggerKeeperAndMint(defaultEVMMock, log)
+
+       // EVM Call function calling the keeper should have been called
+       if defaultEVMMock.mockEVMCallerData.callCalls != 1 {
+               t.Errorf("EVM Call count not as expected. got %d want 1", defaultEVMMock.mockEVMCallerData.callCalls)
+       }
+       // AddBalance should not have been called on the state database, as the mint request was over the limit
+       if defaultEVMMock.mockEVMCallerData.addBalanceCalls != 0 {
+               t.Errorf("Add balance call count not as expected. got %d want 1", defaultEVMMock.mockEVMCallerData.addBalanceCalls)
+       }
+}
diff --git a/core/mkalloc.go b/core/mkalloc.go
index 76978a54..95fcafb9 100644
--- a/core/mkalloc.go
+++ b/core/mkalloc.go
@@ -45,8 +45,8 @@ import (
        "sort"
        "strconv"

-       "github.com/ava-labs/coreth/core"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core"
 )

 type allocItem struct{ Addr, Balance *big.Int }
diff --git a/core/rawdb/accessors_chain.go b/core/rawdb/accessors_chain.go
index 8b134e23..4bc082fc 100644
--- a/core/rawdb/accessors_chain.go
+++ b/core/rawdb/accessors_chain.go
@@ -30,14 +30,13 @@ import (
        "bytes"
        "encoding/binary"
        "errors"
-       "math/big"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
 )

 // ReadCanonicalHash retrieves the hash assigned to a canonical block number.
@@ -202,77 +201,6 @@ func WriteHeadBlockHash(db ethdb.KeyValueWriter, hash common.Hash) {
        }
 }

-// ReadHeadFastBlockHash retrieves the hash of the current fast-sync head block.
-func ReadHeadFastBlockHash(db ethdb.KeyValueReader) common.Hash {
-       data, _ := db.Get(headFastBlockKey)
-       if len(data) == 0 {
-               return common.Hash{}
-       }
-       return common.BytesToHash(data)
-}
-
-// WriteHeadFastBlockHash stores the hash of the current fast-sync head block.
```

```
-func WriteHeadFastBlockHash(db ethdb.KeyValueWriter, hash common.Hash) {
-       if err := db.Put(headFastBlockKey, hash.Bytes()); err != nil {
-               log.Crit("Failed to store last fast block's hash", "err", err)
-       }
-}
-
-// ReadFastTrieProgress retrieves the number of tries nodes fast synced to allow
-// reporting correct numbers across restarts.
-func ReadFastTrieProgress(db ethdb.KeyValueReader) uint64 {
-       data, _ := db.Get(fastTrieProgressKey)
-       if len(data) == 0 {
-               return 0
-       }
-       return new(big.Int).SetBytes(data).Uint64()
-}
-
-// WriteFastTrieProgress stores the fast sync trie process counter to support
-// retrieving it across restarts.
-func WriteFastTrieProgress(db ethdb.KeyValueWriter, count uint64) {
-       if err := db.Put(fastTrieProgressKey, new(big.Int).SetUint64(count).Bytes()); err != nil {
-               log.Crit("Failed to store fast sync trie progress", "err", err)
-       }
-}
-
-// ReadTxIndexTail retrieves the number of oldest indexed block
-// whose transaction indices has been indexed. If the corresponding entry
-// is non-existent in database it means the indexing has been finished.
-func ReadTxIndexTail(db ethdb.KeyValueReader) *uint64 {
-       data, _ := db.Get(txIndexTailKey)
-       if len(data) != 8 {
-               return nil
-       }
-       number := binary.BigEndian.Uint64(data)
-       return &number
-}
-
-// WriteTxIndexTail stores the number of oldest indexed block
-// into database.
-func WriteTxIndexTail(db ethdb.KeyValueWriter, number uint64) {
-       if err := db.Put(txIndexTailKey, encodeBlockNumber(number)); err != nil {
-               log.Crit("Failed to store the transaction index tail", "err", err)
-       }
-}
-
-// ReadFastTxLookupLimit retrieves the tx lookup limit used in fast sync.
-func ReadFastTxLookupLimit(db ethdb.KeyValueReader) *uint64 {
-       data, _ := db.Get(fastTxLookupLimitKey)
-       if len(data) != 8 {
-               return nil
-       }
-       number := binary.BigEndian.Uint64(data)
-       return &number
-}
-
-// WriteFastTxLookupLimit stores the txlookup limit used in fast sync into database.
-func WriteFastTxLookupLimit(db ethdb.KeyValueWriter, number uint64) {
-       if err := db.Put(fastTxLookupLimitKey, encodeBlockNumber(number)); err != nil {
-               log.Crit("Failed to store transaction lookup limit for fast sync", "err", err)
-       }
-}
-
 // ReadHeaderRLP retrieves a block header in its raw RLP database encoding.
 func ReadHeaderRLP(db ethdb.Reader, hash common.Hash, number uint64) rlp.RawValue {
        // Then try to look up the data in leveldb.
@@ -408,48 +336,6 @@ func DeleteBody(db ethdb.KeyValueWriter, hash common.Hash, number uint64) {
        }
 }
 
-// ReadTdRLP retrieves a block's total difficulty corresponding to the hash in RLP encoding.
-func ReadTdRLP(db ethdb.Reader, hash common.Hash, number uint64) rlp.RawValue {
-       // Then try to look up the data in leveldb.
-       data, _ := db.Get(headerTDKey(number, hash))
-       if len(data) > 0 {
-               return data
-       }
-       return nil // Can't find the data anywhere.
-}
-
-// ReadTd retrieves a block's total difficulty corresponding to the hash.
-func ReadTd(db ethdb.Reader, hash common.Hash, number uint64) *big.Int {
-       data := ReadTdRLP(db, hash, number)
-       if len(data) == 0 {
-               return nil
-       }
-       td := new(big.Int)
-       if err := rlp.Decode(bytes.NewReader(data), td); err != nil {
-               log.Error("Invalid block total difficulty RLP", "hash", hash, "err", err)
-               return nil
-       }
-       return td
-}
-
-// WriteTd stores the total difficulty of a block into the database.
-func WriteTd(db ethdb.KeyValueWriter, hash common.Hash, number uint64, td *big.Int) {
-       data, err := rlp.EncodeToBytes(td)
-       if err != nil {
-               log.Crit("Failed to RLP encode block total difficulty", "err", err)
-       }
-       if err := db.Put(headerTDKey(number, hash), data); err != nil {
-               log.Crit("Failed to store block total difficulty", "err", err)
-       }
-}
-
-// DeleteTd removes all block total difficulty data associated with a hash.
-func DeleteTd(db ethdb.KeyValueWriter, hash common.Hash, number uint64) {
-       if err := db.Delete(headerTDKey(number, hash)); err != nil {
-               log.Crit("Failed to delete block total difficulty", "err", err)
-       }
-}
-
 // HasReceipts verifies the existence of all the transaction receipts belonging
 // to a block.
 func HasReceipts(db ethdb.Reader, hash common.Hash, number uint64) bool {
@@ -653,7 +539,6 @@ func DeleteBlock(db ethdb.KeyValueWriter, hash common.Hash, number uint64) {
        DeleteReceipts(db, hash, number)
        DeleteHeader(db, hash, number)
        DeleteBody(db, hash, number)
-       DeleteTd(db, hash, number)
 }
 
 // DeleteBlockWithoutNumber removes all block data associated with a hash, except
@@ -662,7 +547,6 @@ func DeleteBlockWithoutNumber(db ethdb.KeyValueWriter, hash common.Hash, number
        DeleteReceipts(db, hash, number)
        deleteHeaderWithoutNumber(db, hash, number)
        DeleteBody(db, hash, number)
-       DeleteTd(db, hash, number)
 }
 
 // FindCommonAncestor returns the last common ancestor of two block headers
diff --git a/core/rawdb/accessors_chain_test.go b/core/rawdb/accessors_chain_test.go
index 9d35e7ba..4773d72f 100644
--- a/core/rawdb/accessors_chain_test.go
+++ b/core/rawdb/accessors_chain_test.go
@@ -25,10 +25,10 @@ import (
        "reflect"
        "testing"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
        "golang.org/x/crypto/sha3"
 )

@@ -187,29 +187,6 @@ func TestPartialBlockStorage(t *testing.T) {
        }
 }

-// Tests block total difficulty storage and retrieval operations.
-func TestTdStorage(t *testing.T) {
-       db := NewMemoryDatabase()
-
-       // Create a test TD to move around the database and make sure it's really new
-       hash, td := common.Hash{}, big.NewInt(314)
-       if entry := ReadTd(db, hash, 0); entry != nil {
-               t.Fatalf("Non existent TD returned: %v", entry)
-       }
-       // Write and verify the TD in the database
-       WriteTd(db, hash, 0, td)
-       if entry := ReadTd(db, hash, 0); entry == nil {
-               t.Fatalf("Stored TD not found")
-       } else if entry.Cmp(td) != 0 {
-               t.Fatalf("Retrieved TD mismatch: have %v, want %v", entry, td)
-       }
-       // Delete the TD and verify the execution
-       DeleteTd(db, hash, 0)
-       if entry := ReadTd(db, hash, 0); entry != nil {
-               t.Fatalf("Deleted TD returned: %v", entry)
-       }
-}
-
 // Tests that canonical numbers can be mapped to hashes and retrieved.
 func TestCanonicalMappingStorage(t *testing.T) {
        db := NewMemoryDatabase()
@@ -239,7 +216,6 @@ func TestHeadStorage(t *testing.T) {

        blockHead := types.NewBlockWithHeader(&types.Header{Extra: []byte("test block header")})
        blockFull := types.NewBlockWithHeader(&types.Header{Extra: []byte("test block full")})
-       blockFast := types.NewBlockWithHeader(&types.Header{Extra: []byte("test block fast")})

        // Check that no head entries are in a pristine database
        if entry := ReadHeadHeaderHash(db); entry != (common.Hash{}) {
@@ -248,13 +224,9 @@ func TestHeadStorage(t *testing.T) {
        if entry := ReadHeadBlockHash(db); entry != (common.Hash{}) {
                t.Fatalf("Non head block entry returned: %v", entry)
        }
-       if entry := ReadHeadFastBlockHash(db); entry != (common.Hash{}) {
-               t.Fatalf("Non fast head block entry returned: %v", entry)
-       }
        // Assign separate entries for the head header and block
        WriteHeadHeaderHash(db, blockHead.Hash())
        WriteHeadBlockHash(db, blockFull.Hash())
-       WriteHeadFastBlockHash(db, blockFast.Hash())

        // Check that both heads are present, and different (i.e. two heads maintained)
        if entry := ReadHeadHeaderHash(db); entry != blockHead.Hash() {
@@ -263,9 +235,6 @@ func TestHeadStorage(t *testing.T) {
        if entry := ReadHeadBlockHash(db); entry != blockFull.Hash() {
                t.Fatalf("Head block hash mismatch: have %v, want %v", entry, blockFull.Hash())
        }
-       if entry := ReadHeadFastBlockHash(db); entry != blockFast.Hash() {
-               t.Fatalf("Fast head block hash mismatch: have %v, want %v", entry, blockFast.Hash())
-       }
 }

 // Tests that receipts associated with a single block can be stored and retrieved.
@@ -393,7 +362,6 @@ func TestCanonicalHashIteration(t *testing.T) {
        // Fill database with testing data.
        for i := uint64(1); i <= 8; i++ {
                WriteCanonicalHash(db, common.Hash{}, i)
-               WriteTd(db, common.Hash{}, i, big.NewInt(10)) // Write some interferential data
        }
        for i, c := range cases {
                numbers, _ := ReadAllCanonicalHashes(db, c.from, c.to, c.limit)
diff --git a/core/rawdb/accessors_indexes.go b/core/rawdb/accessors_indexes.go
index 0ac33214..a75dc386 100644
--- a/core/rawdb/accessors_indexes.go
+++ b/core/rawdb/accessors_indexes.go
@@ -30,12 +30,12 @@ import (
        "bytes"
        "math/big"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
 )

 // ReadTxLookupEntry retrieves the positional metadata associated with a transaction
diff --git a/core/rawdb/accessors_indexes_test.go b/core/rawdb/accessors_indexes_test.go
index e64818b1..0d2aaf7d 100644
--- a/core/rawdb/accessors_indexes_test.go
+++ b/core/rawdb/accessors_indexes_test.go
@@ -22,10 +22,10 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
        "golang.org/x/crypto/sha3"
 )

diff --git a/core/rawdb/accessors_metadata.go b/core/rawdb/accessors_metadata.go
index 6b1a0f13..6c845dfc 100644
--- a/core/rawdb/accessors_metadata.go
+++ b/core/rawdb/accessors_metadata.go
@@ -28,12 +28,13 @@ package rawdb

 import (
        "encoding/json"
+       "time"

-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
 )

 // ReadDatabaseVersion retrieves the version number of the database.
@@ -89,3 +90,117 @@ func WriteChainConfig(db ethdb.KeyValueWriter, hash common.Hash, cfg *params.Cha
                log.Crit("Failed to store chain config", "err", err)
        }
 }
+
+// crashList is a list of unclean-shutdown-markers, for rlp-encoding to the
+// database
+type crashList struct {
+       Discarded uint64   // how many ucs have we deleted
+       Recent    []uint64 // unix timestamps of 10 latest unclean shutdowns
+}
+
+const crashesToKeep = 10
+
+// PushUncleanShutdownMarker appends a new unclean shutdown marker and returns
+// the previous data
+// - a list of timestamps
+// - a count of how many old unclean-shutdowns have been discarded
+func PushUncleanShutdownMarker(db ethdb.KeyValueStore) ([]uint64, uint64, error) {
+       var uncleanShutdowns crashList
+       // Read old data
+       if data, err := db.Get(uncleanShutdownKey); err != nil {
+               log.Warn("Error reading unclean shutdown markers", "error", err)
+       } else if err := rlp.DecodeBytes(data, &uncleanShutdowns); err != nil {
+               return nil, 0, err
+       }
+       var discarded = uncleanShutdowns.Discarded
+       var previous = make([]uint64, len(uncleanShutdowns.Recent))
+       copy(previous, uncleanShutdowns.Recent)
+       // Add a new (but cap it)
+       uncleanShutdowns.Recent = append(uncleanShutdowns.Recent, uint64(time.Now().Unix()))
+       if count := len(uncleanShutdowns.Recent); count > crashesToKeep+1 {
+               numDel := count - (crashesToKeep + 1)
+               uncleanShutdowns.Recent = uncleanShutdowns.Recent[numDel:]
+               uncleanShutdowns.Discarded += uint64(numDel)
+       }
+       // And save it again
+       data, _ := rlp.EncodeToBytes(uncleanShutdowns)
+       if err := db.Put(uncleanShutdownKey, data); err != nil {
+               log.Warn("Failed to write unclean-shutdown marker", "err", err)
+               return nil, 0, err
+       }
+       return previous, discarded, nil
+}
+
+// PopUncleanShutdownMarker removes the last unclean shutdown marker
+func PopUncleanShutdownMarker(db ethdb.KeyValueStore) {
+       var uncleanShutdowns crashList
+       // Read old data
+       if data, err := db.Get(uncleanShutdownKey); err != nil {
+               log.Warn("Error reading unclean shutdown markers", "error", err)
+       } else if err := rlp.DecodeBytes(data, &uncleanShutdowns); err != nil {
+               log.Error("Error decoding unclean shutdown markers", "error", err) // Should mos def _not_ happen
+       }
+       if l := len(uncleanShutdowns.Recent); l > 0 {
+               uncleanShutdowns.Recent = uncleanShutdowns.Recent[:l-1]
+       }
+       data, _ := rlp.EncodeToBytes(uncleanShutdowns)
+       if err := db.Put(uncleanShutdownKey, data); err != nil {
+               log.Warn("Failed to clear unclean-shutdown marker", "err", err)
+       }
+}
+
+// UpdateUncleanShutdownMarker updates the last marker's timestamp to now.
+func UpdateUncleanShutdownMarker(db ethdb.KeyValueStore) {
+       var uncleanShutdowns crashList
+       // Read old data
+       if data, err := db.Get(uncleanShutdownKey); err != nil {
+               log.Warn("Error reading unclean shutdown markers", "error", err)
+       } else if err := rlp.DecodeBytes(data, &uncleanShutdowns); err != nil {
+               log.Warn("Error decoding unclean shutdown markers", "error", err)
+       }
+       // This shouldn't happen because we push a marker on Backend instantiation
+       count := len(uncleanShutdowns.Recent)
+       if count == 0 {
+               log.Warn("No unclean shutdown marker to update")
+               return
+       }
+       uncleanShutdowns.Recent[count-1] = uint64(time.Now().Unix())
+       data, _ := rlp.EncodeToBytes(uncleanShutdowns)
+       if err := db.Put(uncleanShutdownKey, data); err != nil {
+               log.Warn("Failed to write unclean-shutdown marker", "err", err)
+       }
+}
+
+// WriteOfflinePruning writes a marker of the last attempt to run offline pruning
+// The marker is written when offline pruning completes and is deleted when the node
+// is started successfully with offline pruning disabled. This ensures users must
+// disable offline pruning and start their node successfully between runs of offline
+// pruning.
+func WriteOfflinePruning(db ethdb.KeyValueStore) error {
+       data, err := rlp.EncodeToBytes(uint64(time.Now().Unix()))
+       if err != nil {
+               return err
+       }
+       return db.Put(offlinePruningKey, data)
+}
+
+// ReadOfflinePruning reads to check if there is a marker of the last attempt
+// to run offline pruning.
+func ReadOfflinePruning(db ethdb.KeyValueStore) (uint64, error) {
+       data, err := db.Get(offlinePruningKey)
+       if err != nil {
+               return 0, err
+       }
+
+       var offlinePruningRun uint64
+       if err := rlp.DecodeBytes(data, &offlinePruningRun); err != nil {
+               return 0, err
+       }
+
+       return offlinePruningRun, nil
+}
+
+// DeleteOfflinePruning deletes any marker of the last attempt to run offline pruning.
+func DeleteOfflinePruning(db ethdb.KeyValueStore) error {
+       return db.Delete(offlinePruningKey)
+}
diff --git a/core/rawdb/accessors_snapshot.go b/core/rawdb/accessors_snapshot.go
index 2c7c6c74..0e97a99b 100644
--- a/core/rawdb/accessors_snapshot.go
+++ b/core/rawdb/accessors_snapshot.go
@@ -27,11 +27,9 @@
 package rawdb

 import (
-       "encoding/binary"
-
-       "github.com/ava-labs/coreth/ethdb"
```

```
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // ReadSnapshotRoot retrieves the root of the block whose state is contained in
@@ -158,56 +156,3 @@ func DeleteSnapshotGenerator(db ethdb.KeyValueWriter) {
 		log.Crit("Failed to remove snapshot generator", "err", err)
 	}
 }
-
-// ReadSnapshotRecoveryNumber retrieves the block number of the last persisted
-// snapshot layer.
-func ReadSnapshotRecoveryNumber(db ethdb.KeyValueReader) *uint64 {
-	data, _ := db.Get(snapshotRecoveryKey)
-	if len(data) == 0 {
-		return nil
-	}
-	if len(data) != 8 {
-		return nil
-	}
-	number := binary.BigEndian.Uint64(data)
-	return &number
-}
-
-// WriteSnapshotRecoveryNumber stores the block number of the last persisted
-// snapshot layer.
-func WriteSnapshotRecoveryNumber(db ethdb.KeyValueWriter, number uint64) {
-	var buf [8]byte
-	binary.BigEndian.PutUint64(buf[:], number)
-	if err := db.Put(snapshotRecoveryKey, buf[:]); err != nil {
-		log.Crit("Failed to store snapshot recovery number", "err", err)
-	}
-}
-
-// DeleteSnapshotRecoveryNumber deletes the block number of the last persisted
-// snapshot layer.
-func DeleteSnapshotRecoveryNumber(db ethdb.KeyValueWriter) {
-	if err := db.Delete(snapshotRecoveryKey); err != nil {
-		log.Crit("Failed to remove snapshot recovery number", "err", err)
-	}
-}
-
-// ReadSnapshotSyncStatus retrieves the serialized sync status saved at shutdown.
-func ReadSnapshotSyncStatus(db ethdb.KeyValueReader) []byte {
-	data, _ := db.Get(snapshotSyncStatusKey)
-	return data
-}
-
-// WriteSnapshotSyncStatus stores the serialized sync status to save at shutdown.
-func WriteSnapshotSyncStatus(db ethdb.KeyValueWriter, status []byte) {
-	if err := db.Put(snapshotSyncStatusKey, status); err != nil {
-		log.Crit("Failed to store snapshot sync status", "err", err)
-	}
-}
-
-// DeleteSnapshotSyncStatus deletes the serialized sync status saved at the last
-// shutdown
-func DeleteSnapshotSyncStatus(db ethdb.KeyValueWriter) {
-	if err := db.Delete(snapshotSyncStatusKey); err != nil {
-		log.Crit("Failed to remove snapshot sync status", "err", err)
-	}
-}
diff --git a/core/rawdb/accessors_state.go b/core/rawdb/accessors_state.go
index 38f663cd..f17ac27e 100644
--- a/core/rawdb/accessors_state.go
+++ b/core/rawdb/accessors_state.go
@@ -27,9 +27,9 @@
 package rawdb
 
 import (
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // ReadPreimage retrieves a single preimage of the provided hash.
diff --git a/core/rawdb/database.go b/core/rawdb/database.go
index 55a42d15..abf286c8 100644
--- a/core/rawdb/database.go
+++ b/core/rawdb/database.go
@@ -32,10 +32,11 @@ import (
 	"os"
 	"time"
 
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/ethdb/memorydb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/ethdb/leveldb"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
 	"github.com/olekukonko/tablewriter"
 )
 
@@ -63,6 +64,16 @@ func NewMemoryDatabaseWithCap(size int) ethdb.Database {
 	return NewDatabase(memorydb.NewWithCap(size))
 }
 
+// NewLevelDBDatabase creates a persistent key-value database without a freezer
+// moving immutable chain segments into cold storage.
+func NewLevelDBDatabase(file string, cache int, handles int, namespace string, readonly bool) (ethdb.Database, error) {
+	db, err := leveldb.New(file, cache, handles, namespace, readonly)
+	if err != nil {
+		return nil, err
+	}
+	return NewDatabase(db), nil
+}
+
 type counter uint64
 
 func (c counter) String() string {
@@ -108,7 +119,6 @@ func InspectDatabase(db ethdb.Database, keyPrefix, keyStart []byte) error {
 		headers         stat
 		bodies          stat
 		receipts        stat
-		tds             stat
 		numHashPairings stat
 		hashNumPairings stat
 		tries           stat
@@ -145,8 +155,6 @@ func InspectDatabase(db ethdb.Database, keyPrefix, keyStart []byte) error {
 			bodies.Add(size)
 		case bytes.HasPrefix(key, blockReceiptsPrefix) && len(key) == (len(blockReceiptsPrefix)+8+common.HashLength):
 			receipts.Add(size)
-		case bytes.HasPrefix(key, headerPrefix) && bytes.HasSuffix(key, headerTDSuffix):
-			tds.Add(size)
 		case bytes.HasPrefix(key, headerPrefix) && bytes.HasSuffix(key, headerHashSuffix):
 			numHashPairings.Add(size)
 		case bytes.HasPrefix(key, headerNumberPrefix) && len(key) == (len(headerNumberPrefix)+common.HashLength):
@@ -182,10 +190,8 @@ func InspectDatabase(db ethdb.Database, keyPrefix, keyStart []byte) error {
 		default:
 			var accounted bool
 			for _, meta := range [][]byte{
```

```
-                               databaseVersionKey, headHeaderKey, headBlockKey, headFastBlockKey, lastPivotKey,
-                               fastTrieProgressKey, snapshotDisabledKey, snapshotRootKey, snapshotJournalKey,
-                               snapshotGeneratorKey, snapshotRecoveryKey, txIndexTailKey, fastTxLookupLimitKey,
-                               uncleanShutdownKey, badBlockKey,
+                               databaseVersionKey, headHeaderKey, headBlockKey,
+                               snapshotRootKey, snapshotGeneratorKey, uncleanShutdownKey,
                        } {
                                if bytes.Equal(key, meta) {
                                        metadata.Add(size)
@@ -208,7 +214,6 @@ func InspectDatabase(db ethdb.Database, keyPrefix, keyStart []byte) error {
                {"Key-Value store", "Headers", headers.Size(), headers.Count()},
                {"Key-Value store", "Bodies", bodies.Size(), bodies.Count()},
                {"Key-Value store", "Receipt lists", receipts.Size(), receipts.Count()},
-               {"Key-Value store", "Difficulties", tds.Size(), tds.Count()},
                {"Key-Value store", "Block number->hash", numHashPairings.Size(), numHashPairings.Count()},
                {"Key-Value store", "Block hash->number", hashNumPairings.Size(), hashNumPairings.Count()},
                {"Key-Value store", "Transaction index", txLookups.Size(), txLookups.Count()},
diff --git a/core/rawdb/schema.go b/core/rawdb/schema.go
index 67bdd634..6f6701fc 100644
--- a/core/rawdb/schema.go
+++ b/core/rawdb/schema.go
@@ -46,51 +46,23 @@ var (
        // headBlockKey tracks the latest known full block's hash.
        headBlockKey = []byte("LastBlock")

-       // headFastBlockKey tracks the latest known incomplete block's hash during fast sync.
-       headFastBlockKey = []byte("LastFast")
-
-       // lastPivotKey tracks the last pivot block used by fast sync (to reenable on sethead).
-       lastPivotKey = []byte("LastPivot")
-
-       // fastTrieProgressKey tracks the number of trie entries imported during fast sync.
-       fastTrieProgressKey = []byte("TrieSync")
-
-       // snapshotDisabledKey flags that the snapshot should not be maintained due to initial sync.
-       snapshotDisabledKey = []byte("SnapshotDisabled")
-
        // snapshotRootKey tracks the hash of the last snapshot.
        snapshotRootKey = []byte("SnapshotRoot")

        // snapshotBlockHashKey tracks the block hash of the last snapshot.
        snapshotBlockHashKey = []byte("SnapshotBlockHash")

-       // snapshotJournalKey tracks the in-memory diff layers across restarts.
-       snapshotJournalKey = []byte("SnapshotJournal")
-
        // snapshotGeneratorKey tracks the snapshot generation marker across restarts.
        snapshotGeneratorKey = []byte("SnapshotGenerator")

-       // snapshotRecoveryKey tracks the snapshot recovery marker across restarts.
-       snapshotRecoveryKey = []byte("SnapshotRecovery")
-
-       // snapshotSyncStatusKey tracks the snapshot sync status across restarts.
-       snapshotSyncStatusKey = []byte("SnapshotSyncStatus")
-
-       // txIndexTailKey tracks the oldest block whose transactions have been indexed.
-       txIndexTailKey = []byte("TransactionIndexTail")
-
-       // fastTxLookupLimitKey tracks the transaction lookup limit during fast sync.
-       fastTxLookupLimitKey = []byte("FastTransactionLookupLimit")
-
-       // badBlockKey tracks the list of bad blocks seen by local
-       badBlockKey = []byte("InvalidBlock")
-
        // uncleanShutdownKey tracks the list of local crashes
        uncleanShutdownKey = []byte("unclean-shutdown") // config prefix for the db

+       // offlinePruningKey tracks runs of offline pruning
+       offlinePruningKey = []byte("OfflinePruning")
+
        // Data item prefixes (use single byte to avoid mixing data types, avoid `i`, used for indexes).
        headerPrefix       = []byte("h") // headerPrefix + num (uint64 big endian) + hash -> header
-       headerTDSuffix     = []byte("t") // headerPrefix + num (uint64 big endian) + hash + headerTDSuffix -> td
        headerHashSuffix   = []byte("n") // headerPrefix + num (uint64 big endian) + headerHashSuffix -> hash
        headerNumberPrefix = []byte("H") // headerNumberPrefix + hash -> num (uint64 big endian)

@@ -138,11 +110,6 @@ func headerKey(number uint64, hash common.Hash) []byte {
        return append(append(headerPrefix, encodeBlockNumber(number)...), hash.Bytes()...)
 }

-// headerTDKey = headerPrefix + num (uint64 big endian) + hash + headerTDSuffix
-func headerTDKey(number uint64, hash common.Hash) []byte {
-       return append(headerKey(number, hash), headerTDSuffix...)
-}
-
 // headerHashKey = headerPrefix + num (uint64 big endian) + headerHashSuffix
 func headerHashKey(number uint64) []byte {
        return append(append(headerPrefix, encodeBlockNumber(number)...), headerHashSuffix...)
diff --git a/core/rawdb/table.go b/core/rawdb/table.go
index 179ebf39..3e4f4bbd 100644
--- a/core/rawdb/table.go
+++ b/core/rawdb/table.go
@@ -27,7 +27,7 @@
 package rawdb

 import (
-       "github.com/ava-labs/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 // table is a wrapper around a database that prefixes each key access with a pre-
diff --git a/core/rawdb/table_test.go b/core/rawdb/table_test.go
index c7cac982..2f3bd97f 100644
--- a/core/rawdb/table_test.go
+++ b/core/rawdb/table_test.go
@@ -30,7 +30,7 @@ import (
        "bytes"
        "testing"

-       "github.com/ava-labs/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 func TestTableDatabase(t *testing.T)            { testTableDatabase(t, "prefix") }
diff --git a/core/rlp_test.go b/core/rlp_test.go
new file mode 100644
index 00000000..6815d1ab
--- /dev/null
+++ b/core/rlp_test.go
@@ -0,0 +1,219 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2019 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       "fmt"
+       "math/big"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
+       "golang.org/x/crypto/sha3"
+)
+
+func getBlock(transactions int, uncles int, dataSize int) *types.Block {
+       var (
+               aa = common.HexToAddress("0x000000000000000000000000000000000000aaaa")
+               // Generate a canonical chain to act as the main dataset
+               engine = dummy.NewFaker()
+               db     = rawdb.NewMemoryDatabase()
+               // A sender who makes transactions, has some funds
+               key, _  = crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291")
+               address = crypto.PubkeyToAddress(key.PublicKey)
+               funds   = big.NewInt(50000 * 225000000000 * 200)
+               gspec   = &Genesis{
+                       Config: params.TestChainConfig,
+                       Alloc:  GenesisAlloc{address: {Balance: funds}},
+               }
+               genesis = gspec.MustCommit(db)
+       )
+
+       // We need to generate as many blocks +1 as uncles
+       blocks, _, _ := GenerateChain(params.TestChainConfig, genesis, engine, db, uncles+1, 10,
+               func(n int, b *BlockGen) {
+                       if n == uncles {
+                               // Add transactions and stuff on the last block
+                               for i := 0; i < transactions; i++ {
+                                       tx, _ := types.SignTx(types.NewTransaction(uint64(i), aa,
+                                               big.NewInt(0), 50000, b.header.BaseFee, make([]byte, dataSize)), types.LatestSigner(params.TestChainConfig), key)
+                                       b.AddTx(tx)
+                               }
+                               for i := 0; i < uncles; i++ {
+                                       b.AddUncle(&types.Header{ParentHash: b.PrevBlock(n - 1 - i).Hash(), Number: big.NewInt(int64(n - i))})
+                               }
+                       }
+               })
+       block := blocks[len(blocks)-1]
+       return block
+}
+
+// TestRlpIterator tests that individual transactions can be picked out
+// from blocks without full unmarshalling/marshalling
+func TestRlpIterator(t *testing.T) {
+       for _, tt := range []struct {
+               txs      int
+               uncles   int
+               datasize int
+       }{
+               {0, 0, 0},
+               {0, 2, 0},
+               {10, 0, 0},
+               {10, 2, 0},
+               {10, 2, 50},
+       } {
+               testRlpIterator(t, tt.txs, tt.uncles, tt.datasize)
+       }
+}
+
+func testRlpIterator(t *testing.T, txs, uncles, datasize int) {
+       desc := fmt.Sprintf("%d txs [%d datasize] and %d uncles", txs, datasize, uncles)
+       bodyRlp, _ := rlp.EncodeToBytes(getBlock(txs, uncles, datasize).Body())
+       it, err := rlp.NewListIterator(bodyRlp)
+       if err != nil {
+               t.Fatal(err)
+       }
+       // Check that txs exist
+       if !it.Next() {
+               t.Fatal("expected four elems, got zero")
+       }
+       txdata := it.Value()
+       // Check that uncles exist
+       if !it.Next() {
+               t.Fatal("expected four elems, got one")
+       }
+       // Check that version exist
+       if !it.Next() {
+               t.Fatal("expected four elems, got two")
+       }
+       // Check that extdata exist
+       if !it.Next() {
+               t.Fatal("expected four elems, got three")
+       }
+       // No more after that
+       if it.Next() {
+               t.Fatal("expected only four elems, got more")
+       }
+       txIt, err := rlp.NewListIterator(txdata)
+       if err != nil {
+               t.Fatal(err)
+       }
+       var gotHashes []common.Hash
+       var expHashes []common.Hash
+       for txIt.Next() {
+               gotHashes = append(gotHashes, crypto.Keccak256Hash(txIt.Value()))
+       }
+
+       var expBody types.Body
+       err = rlp.DecodeBytes(bodyRlp, &expBody)
+       if err != nil {
+               t.Fatal(err)
+       }
+       for _, tx := range expBody.Transactions {
+               expHashes = append(expHashes, tx.Hash())
+       }
+       if gotLen, expLen := len(gotHashes), len(expHashes); gotLen != expLen {
+               t.Fatalf("testcase %v: length wrong, got %d exp %d", desc, gotLen, expLen)
+       }
+       // also sanity check against input
+       if gotLen := len(gotHashes); gotLen != txs {
```

```
+		t.Fatalf("testcase %v: length wrong, got %d exp %d", desc, gotLen, txs)
+	}
+	for i, got := range gotHashes {
+		if exp := expHashes[i]; got != exp {
+			t.Errorf("testcase %v: hash wrong, got %x, exp %x", desc, got, exp)
+		}
+	}
+}
+
+// BenchmarkHashing compares the speeds of hashing a rlp raw data directly
+// without the unmarshalling/marshalling step
+func BenchmarkHashing(b *testing.B) {
+	// Make a pretty fat block
+	var (
+		bodyRlp  []byte
+		blockRlp []byte
+	)
+	{
+		block := getBlock(200, 2, 50)
+		bodyRlp, _ = rlp.EncodeToBytes(block.Body())
+		blockRlp, _ = rlp.EncodeToBytes(block)
+	}
+	var got common.Hash
+	var hasher = sha3.NewLegacyKeccak256()
+	b.Run("iteratorhashing", func(b *testing.B) {
+		b.ResetTimer()
+		for i := 0; i < b.N; i++ {
+			var hash common.Hash
+			it, err := rlp.NewListIterator(bodyRlp)
+			if err != nil {
+				b.Fatal(err)
+			}
+			it.Next()
+			txs := it.Value()
+			txIt, err := rlp.NewListIterator(txs)
+			if err != nil {
+				b.Fatal(err)
+			}
+			for txIt.Next() {
+				hasher.Reset()
+				hasher.Write(txIt.Value())
+				hasher.Sum(hash[:0])
+				got = hash
+			}
+		}
+	})
+	var exp common.Hash
+	b.Run("fullbodyhashing", func(b *testing.B) {
+		b.ResetTimer()
+		for i := 0; i < b.N; i++ {
+			var body types.Body
+			rlp.DecodeBytes(bodyRlp, &body)
+			for _, tx := range body.Transactions {
+				exp = tx.Hash()
+			}
+		}
+	})
+	b.Run("fullblockhashing", func(b *testing.B) {
+		b.ResetTimer()
+		for i := 0; i < b.N; i++ {
+			var block types.Block
+			rlp.DecodeBytes(blockRlp, &block)
+			for _, tx := range block.Transactions() {
+				tx.Hash()
+			}
+		}
+	})
+	if got != exp {
+		b.Fatalf("hash wrong, got %x exp %x", got, exp)
+	}
+}
diff --git a/core/state/database.go b/core/state/database.go
index bef0af1e..eb98ad66 100644
--- a/core/state/database.go
+++ b/core/state/database.go
@@ -31,11 +31,11 @@ import (
 	"fmt"
 
 	"github.com/VictoriaMetrics/fastcache"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/trie"
 	lru "github.com/hashicorp/golang-lru"
 )
 
diff --git a/core/state/dump.go b/core/state/dump.go
index 8f30d482..e0cd9f82 100644
--- a/core/state/dump.go
+++ b/core/state/dump.go
@@ -31,12 +31,12 @@ import (
 	"fmt"
 	"time"
 
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 // DumpConfig is a set of options to control what portions of the statewill be
diff --git a/core/state/iterator.go b/core/state/iterator.go
index 2ad4ed93..01ab5d55 100644
--- a/core/state/iterator.go
+++ b/core/state/iterator.go
@@ -30,10 +30,10 @@ import (
 	"bytes"
 	"fmt"
 
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 // NodeIterator is an iterator to traverse the entire state trie post-order,
diff --git a/core/state/pruner/bloom.go b/core/state/pruner/bloom.go
new file mode 100644
index 00000000..13fbebe0
--- /dev/null
+++ b/core/state/pruner/bloom.go
@@ -0,0 +1,142 @@
+// (c) 2019-2020, Ava Labs, Inc.
```

```
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package pruner
+
+import (
+       "encoding/binary"
+       "errors"
+       "os"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       bloomfilter "github.com/holiman/bloomfilter/v2"
+)
+
+// stateBloomHasher is a wrapper around a byte blob to satisfy the interface API
+// requirements of the bloom library used. It's used to convert a trie hash or
+// contract code hash into a 64 bit mini hash.
+type stateBloomHasher []byte
+
+func (f stateBloomHasher) Write(p []byte) (n int, err error) { panic("not implemented") }
+func (f stateBloomHasher) Sum(b []byte) []byte               { panic("not implemented") }
+func (f stateBloomHasher) Reset()                            { panic("not implemented") }
+func (f stateBloomHasher) BlockSize() int                    { panic("not implemented") }
+func (f stateBloomHasher) Size() int                         { return 8 }
+func (f stateBloomHasher) Sum64() uint64                     { return binary.BigEndian.Uint64(f) }
+
+// stateBloom is a bloom filter used during the state convesion(snapshot->state).
+// The keys of all generated entries will be recorded here so that in the pruning
+// stage the entries belong to the specific version can be avoided for deletion.
+//
+// The false-positive is allowed here. The "false-positive" entries means they
+// actually don't belong to the specific version but they are not deleted in the
+// pruning. The downside of the false-positive allowance is we may leave some "dangling"
+// nodes in the disk. But in practice the it's very unlike the dangling node is
+// state root. So in theory this pruned state shouldn't be visited anymore. Another
+// potential issue is for fast sync. If we do another fast sync upon the pruned
+// database, it's problematic which will stop the expansion during the syncing.
+// TODO address it @rjl493456442 @holiman @karalabe.
+//
+// After the entire state is generated, the bloom filter should be persisted into
+// the disk. It indicates the whole generation procedure is finished.
+type stateBloom struct {
+       bloom *bloomfilter.Filter
+}
+
+// newStateBloomWithSize creates a brand new state bloom for state generation.
+// The bloom filter will be created by the passing bloom filter size. According
+// to the https://hur.st/bloomfilter/?n=600000000&p=&m=2048MB&k=4, the parameters
+// are picked so that the false-positive rate for mainnet is low enough.
+func newStateBloomWithSize(size uint64) (*stateBloom, error) {
+       bloom, err := bloomfilter.New(size*1024*1024*8, 4)
+       if err != nil {
+               return nil, err
+       }
+       log.Info("Initialized state bloom", "size", common.StorageSize(float64(bloom.M()/8)))
+       return &stateBloom{bloom: bloom}, nil
+}
+
+// NewStateBloomFromDisk loads the state bloom from the given file.
+// In this case the assumption is held the bloom filter is complete.
+func NewStateBloomFromDisk(filename string) (*stateBloom, error) {
+       bloom, _, err := bloomfilter.ReadFile(filename)
+       if err != nil {
+               return nil, err
+       }
+       return &stateBloom{bloom: bloom}, nil
+}
+
+// Commit flushes the bloom filter content into the disk and marks the bloom
+// as complete.
+func (bloom *stateBloom) Commit(filename, tempname string) error {
+       // Write the bloom out into a temporary file
+       _, err := bloom.bloom.WriteFile(tempname)
+       if err != nil {
+               return err
+       }
+       // Ensure the file is synced to disk
+       f, err := os.OpenFile(tempname, os.O_RDWR, 0666)
+       if err != nil {
+               return err
+       }
+       if err := f.Sync(); err != nil {
+               f.Close()
+               return err
+       }
+       f.Close()
+
+       // Move the teporary file into it's final location
+       return os.Rename(tempname, filename)
+}
+
+// Put implements the KeyValueWriter interface. But here only the key is needed.
+func (bloom *stateBloom) Put(key []byte, value []byte) error {
+       // If the key length is not 32bytes, ensure it's contract code
+       // entry with new scheme.
+       if len(key) != common.HashLength {
+               isCode, codeKey := rawdb.IsCodeKey(key)
+               if !isCode {
+                       return errors.New("invalid entry")
+               }
+               bloom.bloom.Add(stateBloomHasher(codeKey))
+               return nil
+       }
+       bloom.bloom.Add(stateBloomHasher(key))
+       return nil
+}
+
+// Delete removes the key from the key-value data store.
+func (bloom *stateBloom) Delete(key []byte) error { panic("not supported") }
+
+// Contain is the wrapper of the underlying contains function which
+// reports whether the key is contained.
+// - If it says yes, the key may be contained
+// - If it says no, the key is definitely not contained.
+func (bloom *stateBloom) Contain(key []byte) (bool, error) {
+       return bloom.bloom.Contains(stateBloomHasher(key)), nil
+}
diff --git a/core/state/pruner/pruner.go b/core/state/pruner/pruner.go
new file mode 100644
index 00000000..50537b4b
--- /dev/null
+++ b/core/state/pruner/pruner.go
@@ -0,0 +1,417 @@
+// (c) 2019-2020, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package pruner
+
+import (
+       "bytes"
+       "encoding/binary"
+       "errors"
+       "fmt"
+       "math"
+       "os"
+       "path/filepath"
+       "strings"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/trie"
+)
+
+const (
+       // stateBloomFilePrefix is the filename prefix of state bloom filter.
+       stateBloomFilePrefix = "statebloom"
+
+       // stateBloomFilePrefix is the filename suffix of state bloom filter.
+       stateBloomFileSuffix = "bf.gz"
+
+       // stateBloomFileTempSuffix is the filename suffix of state bloom filter
+       // while it is being written out to detect write aborts.
+       stateBloomFileTempSuffix = ".tmp"
+
+       // rangeCompactionThreshold is the minimal deleted entry number for
+       // triggering range compaction. It's a quite arbitrary number but just
+       // to avoid triggering range compaction because of small deletion.
+       rangeCompactionThreshold = 100000
+)
+
+var (
+       // emptyRoot is the known root hash of an empty trie.
+       emptyRoot = common.HexToHash("56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421")
+
+       // emptyCode is the known hash of the empty EVM bytecode.
+       emptyCode = crypto.Keccak256(nil)
+)
+
+// Pruner is an offline tool to prune the stale state with the
+// help of the snapshot. The workflow of pruner is very simple:
+//
+// - iterate the snapshot, reconstruct the relevant state
+// - iterate the database, delete all other state entries which
+//   don't belong to the target state and the genesis state
+//
+// It can take several hours(around 2 hours for mainnet) to finish
+// the whole pruning work. It's recommended to run this offline tool
+// periodically in order to release the disk usage and improve the
+// disk read performance to some extent.
+type Pruner struct {
+       db         ethdb.Database
+       stateBloom *stateBloom
+       datadir    string
+       headHeader *types.Header
+       snaptree   *snapshot.Tree
+}
+
+// NewPruner creates the pruner instance.
+func NewPruner(db ethdb.Database, datadir string, bloomSize uint64) (*Pruner, error) {
+       headBlock := rawdb.ReadHeadBlock(db)
+       if headBlock == nil {
+               return nil, errors.New("Failed to load head block")
+       }
+       // Note: we refuse to start a pruning session unless the snapshot disk layer exists, which should prevent
+       // us from ever needing to enter RecoverPruning in an invalid pruning session (a session where we do not have
+       // the protected trie in the triedb and in the snapshot disk layer).
+       snaptree, err := snapshot.New(db, trie.NewDatabase(db), 256, headBlock.Hash(), headBlock.Root(), false, false, false)
+       if err != nil {
+               return nil, fmt.Errorf("failed to create snapshot for pruning, must restart without offline pruning disabled to recover: %w", err) // The relevant snapshot(s) might not exist
+       }
+       // Sanitize the bloom filter size if it's too small.
+       if bloomSize < 256 {
+               log.Warn("Sanitizing bloomfilter size", "provided(MB)", bloomSize, "updated(MB)", 256)
+               bloomSize = 256
+       }
+       stateBloom, err := newStateBloomWithSize(bloomSize)
+       if err != nil {
+               return nil, err
+       }
+       return &Pruner{
+               db:         db,
+               stateBloom: stateBloom,
+               datadir:    datadir,
+               headHeader: headBlock.Header(),
+		snaptree:   snaptree,
+	}, nil
+}
+
+func prune(maindb ethdb.Database, stateBloom *stateBloom, bloomPath string, start time.Time) error {
+	// Delete all stale trie nodes in the disk. With the help of state bloom
+	// the trie nodes(and codes) belong to the active state will be filtered
+	// out. A very small part of stale tries will also be filtered because of
+	// the false-positive rate of bloom filter. But the assumption is held here
+	// that the false-positive is low enough(~0.05%). The probablity of the
+	// dangling node is the state root is super low. So the dangling nodes in
+	// theory will never ever be visited again.
+	var (
+		count  int
+		size   common.StorageSize
+		pstart = time.Now()
+		logged = time.Now()
+		batch  = maindb.NewBatch()
+		iter   = maindb.NewIterator(nil, nil)
+	)
+	// We wrap iter.Release() in an anonymous function so that the [iter]
+	// value captured is the value of [iter] at the end of the function as opposed
+	// to incorrectly capturing the first iterator immediately.
+	defer func() {
+		iter.Release()
+	}()
+
+	for iter.Next() {
+		key := iter.Key()
+
+		// All state entries don't belong to specific state and genesis are deleted here
+		// - trie node
+		// - legacy contract code
+		// - new-scheme contract code
+		isCode, codeKey := rawdb.IsCodeKey(key)
+		if len(key) == common.HashLength || isCode {
+			checkKey := key
+			if isCode {
+				checkKey = codeKey
+			}
+			if ok, err := stateBloom.Contain(checkKey); err != nil {
+				return err
+			} else if ok {
+				continue
+			}
+			count += 1
+			size += common.StorageSize(len(key) + len(iter.Value()))
+			if err := batch.Delete(key); err != nil {
+				return err
+			}
+
+			var eta time.Duration // Realistically will never remain uninited
+			if done := binary.BigEndian.Uint64(key[:8]); done > 0 {
+				var (
+					left  = math.MaxUint64 - binary.BigEndian.Uint64(key[:8])
+					speed = done/uint64(time.Since(pstart)/time.Millisecond+1) + 1 // +1s to avoid division by zero
+				)
+				eta = time.Duration(left/speed) * time.Millisecond
+			}
+			if time.Since(logged) > 8*time.Second {
+				log.Info("Pruning state data", "nodes", count, "size", size,
+					"elapsed", common.PrettyDuration(time.Since(pstart)), "eta", common.PrettyDuration(eta))
+				logged = time.Now()
+			}
+			// Recreate the iterator after every batch commit in order
+			// to allow the underlying compactor to delete the entries.
+			if batch.ValueSize() >= ethdb.IdealBatchSize {
+				if err := batch.Write(); err != nil {
+					return err
+				}
+				batch.Reset()
+
+				iter.Release()
+				iter = maindb.NewIterator(nil, key)
+			}
+		}
+	}
+	if err := iter.Error(); err != nil {
+		return fmt.Errorf("failed to iterate db during pruning: %w", err)
+	}
+	if batch.ValueSize() > 0 {
+		if err := batch.Write(); err != nil {
+			return err
+		}
+		batch.Reset()
+	}
+	iter.Release()
+	log.Info("Pruned state data", "nodes", count, "size", size, "elapsed", common.PrettyDuration(time.Since(pstart)))
+
+	// Write marker to DB to indicate offline pruning finished successfully. We write before calling os.RemoveAll
+	// to guarantee that if the node dies midway through pruning, then this will run during RecoverPruning.
+	if err := rawdb.WriteOfflinePruning(maindb); err != nil {
+		return fmt.Errorf("failed to write offline pruning success marker: %w", err)
+	}
+
+	// Delete the state bloom, it marks the entire pruning procedure is
+	// finished. If any crashes or manual exit happens before this,
+	// `RecoverPruning` will pick it up in the next restarts to redo all
+	// the things.
+	if err := os.RemoveAll(bloomPath); err != nil {
+		return fmt.Errorf("failed to remove bloom filter from disk: %w", err)
+	}
+
+	// Start compactions, will remove the deleted data from the disk immediately.
+	// Note for small pruning, the compaction is skipped.
+	if count >= rangeCompactionThreshold {
+		cstart := time.Now()
+		for b := 0x00; b <= 0xf0; b += 0x10 {
+			var (
+				start = []byte{byte(b)}
+				end   = []byte{byte(b + 0x10)}
+			)
+			if b == 0xf0 {
+				end = nil
+			}
+			log.Info("Compacting database", "range", fmt.Sprintf("%#x-%#x", start, end), "elapsed", common.PrettyDuration(time.Since(cstart)))
+			if err := maindb.Compact(start, end); err != nil {
+				log.Error("Database compaction failed", "error", err)
+				return err
+			}
+		}
+		log.Info("Database compaction finished", "elapsed", common.PrettyDuration(time.Since(cstart)))
+	}
+	log.Info("State pruning successful", "pruned", size, "elapsed", common.PrettyDuration(time.Since(start)))
+	return nil
+}
+
+// Prune deletes all historical state nodes except the nodes belong to the
+// specified state version. If user doesn't specify the state version, use
+// the bottom-most snapshot diff layer as the target.
+func (p *Pruner) Prune(root common.Hash) error {
+	// If the state bloom filter is already committed previously,
+	// reuse it for pruning instead of generating a new one. It's
+	// mandatory because a part of state may already be deleted,
+	// the recovery procedure is necessary.
+	_, stateBloomRoot, err := findBloomFilter(p.datadir)
+	if err != nil {
+		return err
+	}
+	if stateBloomRoot != (common.Hash{}) {
+		return RecoverPruning(p.datadir, p.db)
+	}
+
+	// If the target state root is not specified, return a fatal error.
+	if root == (common.Hash{}) {
+		return fmt.Errorf("cannot prune with an empty root: %s", root)
+	}
+	// Ensure the root is really present. The weak assumption
+	// is the presence of root can indicate the presence of the
+	// entire trie.
+	if blob := rawdb.ReadTrieNode(p.db, root); len(blob) == 0 {
+		return fmt.Errorf("associated state[%x] is not present", root)
+	} else {
+		log.Info("Selecting last accepted block root as the pruning target", "root", root)
+	}
+
+	// Traverse the target state, re-construct the whole state trie and
+	// commit to the given bloom filter.
+	start := time.Now()
+	if err := snapshot.GenerateTrie(p.snaptree, root, p.db, p.stateBloom); err != nil {
+		return err
+	}
+	// Traverse the genesis, put all genesis state entries into the
+	// bloom filter too.
+	if err := extractGenesis(p.db, p.stateBloom); err != nil {
+		return err
+	}
+	filterName := bloomFilterName(p.datadir, root)
+
+	log.Info("Writing state bloom to disk", "name", filterName)
+	if err := p.stateBloom.Commit(filterName, filterName+stateBloomFileTempSuffix); err != nil {
+		return err
+	}
+	log.Info("State bloom filter committed", "name", filterName)
+	return prune(p.db, p.stateBloom, filterName, start)
+}
+
+// RecoverPruning will resume the pruning procedure during the system restart.
+// This function is used in this case: user tries to prune state data, but the
+// system was interrupted midway because of crash or manual-kill. In this case
+// if the bloom filter for filtering active state is already constructed, the
+// pruning can be resumed. What's more if the bloom filter is constructed, the
+// pruning **has to be resumed**. Otherwise a lot of dangling nodes may be left
+// in the disk.
+func RecoverPruning(datadir string, db ethdb.Database) error {
+	stateBloomPath, stateBloomRoot, err := findBloomFilter(datadir)
+	if err != nil {
+		return err
+	}
+	if stateBloomPath == "" {
+		return nil // nothing to recover
+	}
+	headBlock := rawdb.ReadHeadBlock(db)
+	if headBlock == nil {
+		return errors.New("Failed to load head block")
+	}
+	stateBloom, err := NewStateBloomFromDisk(stateBloomPath)
+	if err != nil {
+		return err
+	}
+	log.Info("Loaded state bloom filter", "path", stateBloomPath)
+
+	// All the state roots of the middle layers should be forcibly pruned,
+	// otherwise the dangling state will be left.
+	if stateBloomRoot != headBlock.Root() {
+		return fmt.Errorf("cannot recover pruning to state bloom root: %s, with head block root: %s", stateBloomRoot, headBlock.Root())
+	}
+
+	return prune(db, stateBloom, stateBloomPath, time.Now())
+}
+
+// extractGenesis loads the genesis state and commits all the state entries
+// into the given bloomfilter.
+func extractGenesis(db ethdb.Database, stateBloom *stateBloom) error {
+	genesisHash := rawdb.ReadCanonicalHash(db, 0)
+	if genesisHash == (common.Hash{}) {
+		return errors.New("missing genesis hash")
+	}
+	genesis := rawdb.ReadBlock(db, genesisHash, 0)
+	if genesis == nil {
+		return errors.New("missing genesis block")
+	}
+	t, err := trie.NewSecure(genesis.Root(), trie.NewDatabase(db))
+	if err != nil {
+		return err
+	}
+	accIter := t.NodeIterator(nil)
+	for accIter.Next(true) {
+		hash := accIter.Hash()
+
+		// Embedded nodes don't have hash.
+		if hash != (common.Hash{}) {
+			stateBloom.Put(hash.Bytes(), nil)
+		}
+		// If it's a leaf node, yes we are touching an account,
+		// dig into the storage trie further.
+		if accIter.Leaf() {
+			var acc types.StateAccount
+			if err := rlp.DecodeBytes(accIter.LeafBlob(), &acc); err != nil {
+				return err
+			}
+			if acc.Root != emptyRoot {
+				storageTrie, err := trie.NewSecure(acc.Root, trie.NewDatabase(db))
+				if err != nil {
+					return err
+				}
+				storageIter := storageTrie.NodeIterator(nil)
+				for storageIter.Next(true) {
+					hash := storageIter.Hash()
+					if hash != (common.Hash{}) {
+						stateBloom.Put(hash.Bytes(), nil)
+					}
+				}
+				if storageIter.Error() != nil {
+					return storageIter.Error()
+				}
+			}
+			if !bytes.Equal(acc.CodeHash, emptyCode) {
+				stateBloom.Put(acc.CodeHash, nil)
+			}
+		}
+	}
+	return accIter.Error()
+}
+
+func bloomFilterName(datadir string, hash common.Hash) string {
+	return filepath.Join(datadir, fmt.Sprintf("%s.%s.%s", stateBloomFilePrefix, hash.Hex(), stateBloomFileSuffix))
+}
```

```
+
+func isBloomFilter(filename string) (bool, common.Hash) {
+       filename = filepath.Base(filename)
+       if strings.HasPrefix(filename, stateBloomFilePrefix) && strings.HasSuffix(filename, stateBloomFileSuffix) {
+               return true, common.HexToHash(filename[len(stateBloomFilePrefix)+1 : len(filename)-len(stateBloomFileSuffix)-1])
+       }
+       return false, common.Hash{}
+}
+
+func findBloomFilter(datadir string) (string, common.Hash, error) {
+       var (
+               stateBloomPath string
+               stateBloomRoot common.Hash
+       )
+       if err := filepath.Walk(datadir, func(path string, info os.FileInfo, err error) error {
+               if info != nil && !info.IsDir() {
+                       ok, root := isBloomFilter(path)
+                       if ok {
+                               stateBloomPath = path
+                               stateBloomRoot = root
+                       }
+               }
+               return nil
+       }); err != nil {
+               return "", common.Hash{}, err
+       }
+       return stateBloomPath, stateBloomRoot, nil
+}
diff --git a/core/state/snapshot/conversion.go b/core/state/snapshot/conversion.go
index bfb157a1..01b3d23d 100644
--- a/core/state/snapshot/conversion.go
+++ b/core/state/snapshot/conversion.go
@@ -36,12 +36,12 @@ import (
        "sync"
        "time"
 
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/trie"
 )
 
 // trieKV represents a trie key-value pair
diff --git a/core/state/snapshot/difflayer_test.go b/core/state/snapshot/difflayer_test.go
index 245acc83..562929f2 100644
--- a/core/state/snapshot/difflayer_test.go
+++ b/core/state/snapshot/difflayer_test.go
@@ -32,9 +32,9 @@ import (
        "testing"
 
        "github.com/VictoriaMetrics/fastcache"
-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )
 
 func copyDestructs(destructs map[common.Hash]struct{}) map[common.Hash]struct{} {
diff --git a/core/state/snapshot/disklayer.go b/core/state/snapshot/disklayer.go
index 07add6be..932722d5 100644
--- a/core/state/snapshot/disklayer.go
+++ b/core/state/snapshot/disklayer.go
@@ -32,11 +32,11 @@ import (
        "time"
 
        "github.com/VictoriaMetrics/fastcache"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/trie"
 )
 
 // diskLayer is a low level persistent snapshot built on top of a key-value store.
diff --git a/core/state/snapshot/disklayer_test.go b/core/state/snapshot/disklayer_test.go
index adf3778e..310a730f 100644
--- a/core/state/snapshot/disklayer_test.go
+++ b/core/state/snapshot/disklayer_test.go
@@ -30,11 +30,11 @@ import (
        "bytes"
        "testing"
 
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )
 
 // reverse reverses the contents of a byte slice. It's used to update random accs
diff --git a/core/state/snapshot/generate.go b/core/state/snapshot/generate.go
index ae655aec..b1031b69 100644
--- a/core/state/snapshot/generate.go
+++ b/core/state/snapshot/generate.go
@@ -34,14 +34,14 @@ import (
        "time"
 
        "github.com/VictoriaMetrics/fastcache"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/trie"
 )
 
 var (
diff --git a/core/state/snapshot/generate_test.go b/core/state/snapshot/generate_test.go
index 8c480e12..ee77b9c8 100644
--- a/core/state/snapshot/generate_test.go
+++ b/core/state/snapshot/generate_test.go
@@ -33,13 +33,13 @@ import (
        "testing"
        "time"
 
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/ethdb/memorydb"
-       "github.com/ava-labs/coreth/trie"
```

```
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/core/rawdb"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
+	"github.com/flare-foundation/coreth/trie"
 	"golang.org/x/crypto/sha3"
 )

diff --git a/core/state/snapshot/iterator.go b/core/state/snapshot/iterator.go
index b07c2938..3bb36a54 100644
--- a/core/state/snapshot/iterator.go
+++ b/core/state/snapshot/iterator.go
@@ -31,9 +31,9 @@ import (
 	"fmt"
 	"sort"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // Iterator is an iterator to step over all the accounts or the specific
diff --git a/core/state/snapshot/iterator_test.go b/core/state/snapshot/iterator_test.go
index 9d1b7440..fb56870f 100644
--- a/core/state/snapshot/iterator_test.go
+++ b/core/state/snapshot/iterator_test.go
@@ -33,8 +33,8 @@ import (
 	"math/rand"
 	"testing"
 
-	"github.com/ava-labs/coreth/core/rawdb"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/rawdb"
 )
 
 // TestAccountIteratorBasics tests some simple single-layer(diff and disk) iteration
diff --git a/core/state/snapshot/journal.go b/core/state/snapshot/journal.go
index 7feba072..43d2a660 100644
--- a/core/state/snapshot/journal.go
+++ b/core/state/snapshot/journal.go
@@ -33,12 +33,12 @@ import (
 	"time"
 
 	"github.com/VictoriaMetrics/fastcache"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 // journalGenerator is a disk layer entry containing the generator progress marker.
diff --git a/core/state/snapshot/snapshot.go b/core/state/snapshot/snapshot.go
index 6c65f938..dad4e98f 100644
--- a/core/state/snapshot/snapshot.go
+++ b/core/state/snapshot/snapshot.go
@@ -36,9 +36,9 @@ import (
 	"time"
 
 	"github.com/VictoriaMetrics/fastcache"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/trie"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/metrics"
@@ -737,10 +737,13 @@ func (t *Tree) Rebuild(blockHash, root common.Hash) {
 	t.lock.Lock()
 	defer t.lock.Unlock()
 
+<<<<<<< HEAD
 	// Firstly delete any recovery flag in the database. Because now we are
 	// building a brand new snapshot. Also reenable the snapshot feature.
 	rawdb.DeleteSnapshotRecoveryNumber(t.diskdb)
 
+=======
+>>>>>>> upstream-v0.8.5-rc.2
 	// Track whether there's a wipe currently running and keep it alive if so
 	var wiper chan struct{}
 
diff --git a/core/state/snapshot/snapshot_test.go b/core/state/snapshot/snapshot_test.go
index 6ccee14d..4f483997 100644
--- a/core/state/snapshot/snapshot_test.go
+++ b/core/state/snapshot/snapshot_test.go
@@ -33,9 +33,9 @@ import (
 	"testing"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/core/rawdb"
 )
 
 // randomHash generates a random blob of data and returns it as a hash.
diff --git a/core/state/snapshot/wipe.go b/core/state/snapshot/wipe.go
index 298a77a1..01f3cec7 100644
--- a/core/state/snapshot/wipe.go
+++ b/core/state/snapshot/wipe.go
@@ -30,10 +30,10 @@ import (
 	"bytes"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // wipeSnapshot starts a goroutine to iterate over the entire key-value database
diff --git a/core/state/snapshot/wipe_test.go b/core/state/snapshot/wipe_test.go
index dbf009ea..5d4329a8 100644
--- a/core/state/snapshot/wipe_test.go
+++ b/core/state/snapshot/wipe_test.go
@@ -30,9 +30,9 @@ import (
 	"math/rand"
 	"testing"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb/memorydb"
 	"github.com/ethereum/go-ethereum/common"
```

```diff
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )
 
 // Tests that given a database with random data content, all parts of a snapshot
@@ -40,57 +40,35 @@ import (
 func TestWipe(t *testing.T) {
        // Create a database with some random snapshot data
        db := memorydb.New()
-
        for i := 0; i < 128; i++ {
-               account := randomHash()
-               rawdb.WriteAccountSnapshot(db, account, randomHash().Bytes())
-               for j := 0; j < 1024; j++ {
-                       rawdb.WriteStorageSnapshot(db, account, randomHash(), randomHash().Bytes())
-               }
+               rawdb.WriteAccountSnapshot(db, randomHash(), randomHash().Bytes())
        }
        rawdb.WriteSnapshotBlockHash(db, randomHash())
        rawdb.WriteSnapshotRoot(db, randomHash())
 
        // Add some random non-snapshot data too to make wiping harder
-       for i := 0; i < 65536; i++ {
-               // Generate a key that's the wrong length for a state snapshot item
-               var keysize int
-               for keysize == 0 || keysize == 32 || keysize == 64 {
-                       keysize = 8 + rand.Intn(64) // +8 to ensure we will "never" randomize duplicates
-               }
-               // Randomize the suffix, dedup and inject it under the snapshot namespace
-               keysuffix := make([]byte, keysize)
+       for i := 0; i < 500; i++ {
+               // Generate keys with wrong length for a state snapshot item
+               keysuffix := make([]byte, 31)
                rand.Read(keysuffix)
-
-               if rand.Int31n(2) == 0 {
-                       db.Put(append(rawdb.SnapshotAccountPrefix, keysuffix...), randomHash().Bytes())
-               } else {
-                       db.Put(append(rawdb.SnapshotStoragePrefix, keysuffix...), randomHash().Bytes())
-               }
-       }
-       // Sanity check that all the keys are present
-       var items int
-
-       it := db.NewIterator(rawdb.SnapshotAccountPrefix, nil)
-       defer it.Release()
-
-       for it.Next() {
-               key := it.Key()
-               if len(key) == len(rawdb.SnapshotAccountPrefix)+common.HashLength {
-                       items++
-               }
+               db.Put(append(rawdb.SnapshotAccountPrefix, keysuffix...), randomHash().Bytes())
+               keysuffix = make([]byte, 33)
+               rand.Read(keysuffix)
+               db.Put(append(rawdb.SnapshotAccountPrefix, keysuffix...), randomHash().Bytes())
        }
-       it = db.NewIterator(rawdb.SnapshotStoragePrefix, nil)
-       defer it.Release()
-
-       for it.Next() {
-               key := it.Key()
-               if len(key) == len(rawdb.SnapshotStoragePrefix)+2*common.HashLength {
-                       items++
+       count := func() (items int) {
+               it := db.NewIterator(rawdb.SnapshotAccountPrefix, nil)
+               defer it.Release()
+               for it.Next() {
+                       if len(it.Key()) == len(rawdb.SnapshotAccountPrefix)+common.HashLength {
+                               items++
+                       }
                }
+               return items
        }
-       if items != 128+128*1024 {
-               t.Fatalf("snapshot size mismatch: have %d, want %d", items, 128+128*1024)
+       // Sanity check that all the keys are present
+       if items := count(); items != 128 {
+               t.Fatalf("snapshot size mismatch: have %d, want %d", items, 128)
        }
        if hash := rawdb.ReadSnapshotBlockHash(db); hash == (common.Hash{}) {
                t.Errorf("snapshot block hash marker mismatch: have %#x, want <not-nil>", hash)
@@ -102,40 +80,24 @@ func TestWipe(t *testing.T) {
        <-wipeSnapshot(db, true)
 
        // Iterate over the database end ensure no snapshot information remains
-       it = db.NewIterator(rawdb.SnapshotAccountPrefix, nil)
-       defer it.Release()
-
-       for it.Next() {
-               key := it.Key()
-               if len(key) == len(rawdb.SnapshotAccountPrefix)+common.HashLength {
-                       t.Errorf("snapshot entry remained after wipe: %x", key)
-               }
+       if items := count(); items != 0 {
+               t.Fatalf("snapshot size mismatch: have %d, want %d", items, 0)
        }
-       it = db.NewIterator(rawdb.SnapshotStoragePrefix, nil)
+       // Iterate over the database and ensure miscellaneous items are present
+       items := 0
+       it := db.NewIterator(nil, nil)
        defer it.Release()
-
        for it.Next() {
-               key := it.Key()
-               if len(key) == len(rawdb.SnapshotStoragePrefix)+2*common.HashLength {
-                       t.Errorf("snapshot entry remained after wipe: %x", key)
-               }
+               items++
        }
+       if items != 1000 {
+               t.Fatalf("misc item count mismatch: have %d, want %d", items, 1000)
+       }
+
        if hash := rawdb.ReadSnapshotBlockHash(db); hash != (common.Hash{}) {
                t.Errorf("snapshot block hash marker remained after wipe: %#x", hash)
        }
        if hash := rawdb.ReadSnapshotRoot(db); hash != (common.Hash{}) {
                t.Errorf("snapshot block root marker remained after wipe: %#x", hash)
        }
-       // Iterate over the database and ensure miscellaneous items are present
-       items = 0
-
-       it = db.NewIterator(nil, nil)
-       defer it.Release()
-
-       for it.Next() {
-               items++
-       }
-       if items != 65536 {
-               t.Fatalf("misc item count mismatch: have %d, want %d", items, 65536)
-       }
 }
diff --git a/core/state/state_object.go b/core/state/state_object.go
```

```
index 0e5be407..8d09e5f3 100644
--- a/core/state/state_object.go
+++ b/core/state/state_object.go
@@ -34,11 +34,11 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/metrics"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
 )

 var emptyCodeHash = crypto.Keccak256(nil)
diff --git a/core/state/state_test.go b/core/state/state_test.go
index e7a05ef2..89c1de80 100644
--- a/core/state/state_test.go
+++ b/core/state/state_test.go
@@ -27,9 +27,9 @@
 package state

 import (
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 type stateTest struct {
diff --git a/core/state/statedb.go b/core/state/statedb.go
index 44a9ee47..2b7f0fe8 100644
--- a/core/state/statedb.go
+++ b/core/state/statedb.go
@@ -34,15 +34,15 @@ import (
        "sort"
        "time"

-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state/snapshot"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/metrics"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/trie"
 )

 type revision struct {
diff --git a/core/state/statedb_test.go b/core/state/statedb_test.go
index 9c295f9e..a4a98878 100644
--- a/core/state/statedb_test.go
+++ b/core/state/statedb_test.go
@@ -39,11 +39,11 @@ import (
        "testing"
        "testing/quick"

-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state/snapshot"
-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // Tests that updating a state trie does not leak any database writes prior to
diff --git a/core/state/trie_prefetcher_test.go b/core/state/trie_prefetcher_test.go
index 563a773d..183baca2 100644
--- a/core/state/trie_prefetcher_test.go
+++ b/core/state/trie_prefetcher_test.go
@@ -31,8 +31,8 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/coreth/core/rawdb"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/rawdb"
 )

 func filledStateDB() *StateDB {
diff --git a/core/state_connector.go b/core/state_connector.go
new file mode 100644
index 00000000..18c9b9af
--- /dev/null
+++ b/core/state_connector.go
@@ -0,0 +1,232 @@
+// (c) 2021, Flare Networks Limited. All rights reserved.
+// Please see the file LICENSE for licensing terms.
+
+package core
+
+import (
+       "encoding/hex"
+       "math/big"
+       "os"
+       "strings"
+
+       "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/coreth/core/vm"
+)
+
+var (
+       flareChainID    = new(big.Int).SetUint64(14) // https://github.com/ethereum-lists/chains/blob/master/_data/chains/eip155-14.json
+       songbirdChainID = new(big.Int).SetUint64(19) // https://github.com/ethereum-lists/chains/blob/master/_data/chains/eip155-19.json
+
+       flareStateConnectorActivationTime    = new(big.Int).SetUint64(1000000000000)
+       songbirdStateConnectorActivationTime = new(big.Int).SetUint64(1000000000000)
+)
+
+type AttestationVotes struct {
+       reachedMajority    bool
+       majorityDecision   string
+       majorityAttestors  []common.Address
+       divergentAttestors []common.Address
+       abstainedAttestors []common.Address
+}
+
+func GetTestingChain(chainID *big.Int) bool {
+       return chainID.Cmp(flareChainID) != 0 && chainID.Cmp(songbirdChainID) != 0
+}
+
+func GetStateConnectorActivated(chainID *big.Int, blockTime *big.Int) bool {
+       if GetTestingChain(chainID) {
+               return true
+       } else if chainID.Cmp(flareChainID) == 0 {
+               return blockTime.Cmp(flareStateConnectorActivationTime) >= 0
+       } else if chainID.Cmp(songbirdChainID) == 0 {
+               return blockTime.Cmp(songbirdStateConnectorActivationTime) >= 0
+       }
+       return false
+}
+
+func GetStateConnectorContract(chainID *big.Int, blockTime *big.Int) common.Address {
+       switch {
+       case GetStateConnectorActivated(chainID, blockTime) && chainID.Cmp(songbirdChainID) == 0:
+               return common.HexToAddress("0x6b5DEa84F71052c1302b5fe652e17FD442D126a9")
+       default:
+               return common.HexToAddress("0x1000000000000000000000000000000000000001")
+       }
+}
+
+func GetStateConnectorCoinbaseSignalAddr(chainID *big.Int, blockTime *big.Int) common.Address {
+       switch {
+       default:
+               return common.HexToAddress("0x000000000000000000000000000000000000dEaD")
+       }
+}
+
+func SubmitAttestationSelector(chainID *big.Int, blockTime *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0xcf, 0xd1, 0xfd, 0xad}
+       }
+}
+
+func GetAttestationSelector(chainID *big.Int, blockTime *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0x29, 0xbe, 0x4d, 0xb2}
+       }
+}
+
+func FinaliseRoundSelector(chainID *big.Int, blockTime *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0xea, 0xeb, 0xf6, 0xd3}
+       }
+}
+
+func GetVoterWhitelisterSelector(chainID *big.Int, blockTime *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0x71, 0xe1, 0xfa, 0xd9}
+       }
+}
+
+func GetFtsoWhitelistedPriceProvidersSelector(chainID *big.Int, blockTime *big.Int) []byte {
+       switch {
+       default:
+               return []byte{0x09, 0xfc, 0xb4, 0x00}
+       }
+}
+
+// The default attestors are the FTSO price providers
+func (st *StateTransition) GetDefaultAttestors(chainID *big.Int, timestamp *big.Int) ([]common.Address, error) {
+       if os.Getenv("TESTING_ATTESTATION_PROVIDERS") != "" && GetTestingChain(chainID) {
+               return GetEnvAttestationProviders("TESTING"), nil
+       } else {
+               // Get VoterWhitelister contract
+               voterWhitelisterContractBytes, _, err := st.evm.Call(
+                       vm.AccountRef(st.msg.From()),
+                       common.HexToAddress(GetPrioritisedFTSOContract(timestamp)),
+                       GetVoterWhitelisterSelector(chainID, timestamp),
+                       GetKeeperGasMultiplier(st.evm.Context.BlockNumber)*st.evm.Context.GasLimit,
+                       big.NewInt(0))
+               if err != nil {
+                       return []common.Address{}, err
+               }
+               // Get FTSO price providers
+               voterWhitelisterContract := common.BytesToAddress(voterWhitelisterContractBytes)
+               priceProvidersBytes, _, err := st.evm.Call(
+                       vm.AccountRef(st.msg.From()),
+                       voterWhitelisterContract,
+                       GetFtsoWhitelistedPriceProvidersSelector(chainID, timestamp),
+                       GetKeeperGasMultiplier(st.evm.Context.BlockNumber)*st.evm.Context.GasLimit,
+                       big.NewInt(0))
+               if err != nil {
+                       return []common.Address{}, err
+               }
+               NUM_ATTESTORS := len(priceProvidersBytes) / 32
+               var attestors []common.Address
+               for i := 0; i < NUM_ATTESTORS; i++ {
+                       attestors = append(attestors, common.BytesToAddress(priceProvidersBytes[i*32:(i+1)*32]))
+               }
+               return attestors, nil
+       }
+}
+
+func GetEnvAttestationProviders(attestorType string) []common.Address {
+       envAttestationProvidersString := os.Getenv(attestorType + "_ATTESTATION_PROVIDERS")
+       if envAttestationProvidersString == "" {
+               return []common.Address{}
+       }
+       envAttestationProviders := strings.Split(envAttestationProvidersString, ",")
+       NUM_ATTESTORS := len(envAttestationProviders)
+       var attestors []common.Address
+       for i := 0; i < NUM_ATTESTORS; i++ {
+               attestors = append(attestors, common.HexToAddress(envAttestationProviders[i]))
+       }
+       return attestors
+}
+
+func (st *StateTransition) GetAttestation(attestor common.Address, instructions []byte) (string, error) {
+       merkleRootHash, _, err := st.evm.Call(vm.AccountRef(attestor), st.to(), instructions, 20000, big.NewInt(0))
+       return hex.EncodeToString(merkleRootHash), err
+}
+
+func (st *StateTransition) CountAttestations(attestors []common.Address, instructions []byte) (AttestationVotes, error) {
+       var attestationVotes AttestationVotes
+       hashFrequencies := make(map[string][]common.Address)
+       for i, a := range attestors {
+               h, err := st.GetAttestation(a, instructions)
+               if err != nil {
+                       attestationVotes.abstainedAttestors = append(attestationVotes.abstainedAttestors, a)
+               }
+               hashFrequencies[h] = append(hashFrequencies[h], attestors[i])
+       }
+       // Find the plurality
+       var pluralityNum int
+       var pluralityKey string
+       for key, val := range hashFrequencies {
+               if len(val) > pluralityNum {
+                       pluralityNum = len(val)
+                       pluralityKey = key
+               }
+       }
+       if pluralityNum > len(attestors)/2 {
+               attestationVotes.reachedMajority = true
+               attestationVotes.majorityDecision = pluralityKey
+               attestationVotes.majorityAttestors = hashFrequencies[pluralityKey]
+       }
+       for key, val := range hashFrequencies {
+               if key != pluralityKey {
+                       attestationVotes.divergentAttestors = append(attestationVotes.divergentAttestors, val...)
+               }
+       }
+       return attestationVotes, nil
+}
+
+func (st *StateTransition) FinalisePreviousRound(chainID *big.Int, timestamp *big.Int, currentRoundNumber []byte) error {
+       getAttestationSelector := GetAttestationSelector(chainID, timestamp)
+       instructions := append(getAttestationSelector[:], currentRoundNumber[:]...)
+       defaultAttestors, err := st.GetDefaultAttestors(chainID, timestamp)
+       if err != nil {
+               return err
+       }
+       defaultAttestationVotes, err := st.CountAttestations(defaultAttestors, instructions)
+       if err != nil {
+               return err
+       }
+       localAttestors := GetEnvAttestationProviders("LOCAL")
+       var finalityReached bool
+       if len(localAttestors) > 0 {
+               localAttestationVotes, err := st.CountAttestations(localAttestors, instructions)
+               if defaultAttestationVotes.reachedMajority && localAttestationVotes.reachedMajority && defaultAttestationVotes.majorityDecision == localAttestationVotes.majorityDecision {
+                       finalityReached = true
+               } else if err != nil || (defaultAttestationVotes.reachedMajority && defaultAttestationVotes.majorityDecision != localAttestationVotes.majorityDecision) {
+                       // Make a back-up of the current state database, because this node is about to branch from the default set
+               }
+       } else if defaultAttestationVotes.reachedMajority {
+               finalityReached = true
+       }
+       if finalityReached {
+               // Finalise defaultAttestationVotes.majorityDecision
+               finaliseRoundSelector := FinaliseRoundSelector(chainID, timestamp)
+               finalisedData := append(finaliseRoundSelector[:], currentRoundNumber[:]...)
+               merkleRootHashBytes, err := hex.DecodeString(defaultAttestationVotes.majorityDecision)
+               if err != nil {
+                       return err
+               }
+               finalisedData = append(finalisedData[:], merkleRootHashBytes[:]...)
+               coinbaseSignal := GetStateConnectorCoinbaseSignalAddr(chainID, timestamp)
+               originalCoinbase := st.evm.Context.Coinbase
+               defer func() {
+                       st.evm.Context.Coinbase = originalCoinbase
+               }()
+               st.evm.Context.Coinbase = coinbaseSignal
+
+               _, _, err = st.evm.Call(vm.AccountRef(coinbaseSignal), st.to(), finalisedData, st.evm.Context.GasLimit, new(big.Int).SetUint64(0))
+               if err != nil {
+                       return err
+               }
+
+               // Issue rewards to defaultAttestationVotes.majorityAttestors here:
+       }
+       return nil
+}
diff --git a/core/state_manager.go b/core/state_manager.go
index ad312d1d..c4d85008 100644
--- a/core/state_manager.go
+++ b/core/state_manager.go
@@ -30,14 +30,14 @@ import (
        "fmt"
        "math/rand"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 const (
        commitInterval = 4096
-       tipBufferSize  = 16
+       tipBufferSize  = 128
 )

 type TrieWriter interface {
diff --git a/core/state_manager_test.go b/core/state_manager_test.go
index 17e2b1d4..9a1e02ad 100644
--- a/core/state_manager_test.go
+++ b/core/state_manager_test.go
@@ -7,7 +7,7 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/types"

        "github.com/ethereum/go-ethereum/common"
        "github.com/stretchr/testify/assert"
diff --git a/core/state_prefetcher.go b/core/state_prefetcher.go
index a1e0cde5..33a9f9d9 100644
--- a/core/state_prefetcher.go
+++ b/core/state_prefetcher.go
@@ -30,11 +30,11 @@ import (
        "math/big"
        "sync/atomic"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )

 // statePrefetcher is a basic Prefetcher, which blindly executes a block on top
diff --git a/core/state_processor.go b/core/state_processor.go
index 636eb20f..c5865b47 100644
--- a/core/state_processor.go
+++ b/core/state_processor.go
@@ -30,14 +30,14 @@ import (
        "fmt"
        "math/big"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/consensus/misc"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/misc"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )

 // StateProcessor is a basic Processor, which takes care of transitioning
diff --git a/core/state_processor_test.go b/core/state_processor_test.go
new file mode 100644
index 00000000..73238af6
--- /dev/null
+++ b/core/state_processor_test.go
@@ -0,0 +1,351 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2020 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package core
+
+import (
+       "math/big"
+       "testing"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/trie"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+       "golang.org/x/crypto/sha3"
+)
+
+// TestStateProcessorErrors tests the output from the 'core' errors
+// as defined in core/error.go. These errors are generated when the
+// blockchain imports bad blocks, meaning blocks which have valid headers but
+// contain invalid transactions
+func TestStateProcessorErrors(t *testing.T) {
+       var (
+               config = &params.ChainConfig{
+                       ChainID:                     big.NewInt(1),
+                       HomesteadBlock:              big.NewInt(0),
+                       EIP150Block:                 big.NewInt(0),
+                       EIP150Hash:                  common.Hash{},
+                       EIP155Block:                 big.NewInt(0),
+                       EIP158Block:                 big.NewInt(0),
+                       ByzantiumBlock:              big.NewInt(0),
+                       ConstantinopleBlock:         big.NewInt(0),
+                       PetersburgBlock:             big.NewInt(0),
+                       IstanbulBlock:               big.NewInt(0),
+                       MuirGlacierBlock:            big.NewInt(0),
+                       ApricotPhase1BlockTimestamp: big.NewInt(0),
+                       ApricotPhase2BlockTimestamp: big.NewInt(0),
+                       ApricotPhase3BlockTimestamp: big.NewInt(0),
+                       ApricotPhase4BlockTimestamp: big.NewInt(0),
+                       ApricotPhase5BlockTimestamp: big.NewInt(0),
+               }
+               signer     = types.LatestSigner(config)
+               testKey, _ = crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291")
+       )
+       var makeTx = func(nonce uint64, to common.Address, amount *big.Int, gasLimit uint64, gasPrice *big.Int, data []byte) *types.Transaction {
+               tx, _ := types.SignTx(types.NewTransaction(nonce, to, amount, gasLimit, gasPrice, data), signer, testKey)
+               return tx
+       }
+       var mkDynamicTx = func(nonce uint64, to common.Address, gasLimit uint64, gasTipCap, gasFeeCap *big.Int) *types.Transaction {
+               tx, _ := types.SignTx(types.NewTx(&types.DynamicFeeTx{
+                       Nonce:     nonce,
+                       GasTipCap: gasTipCap,
+                       GasFeeCap: gasFeeCap,
+                       Gas:       gasLimit,
+                       To:        &to,
+                       Value:     big.NewInt(0),
+               }), signer, testKey)
+               return tx
+       }
+       { // Tests against a 'recent' chain definition
+               var (
+                       db    = rawdb.NewMemoryDatabase()
+                       gspec = &Genesis{
+                               Config: config,
+                               Alloc: GenesisAlloc{
+                                       common.HexToAddress("0x71562b71999873DB5b286dF957af199Ec94617F7"): GenesisAccount{
+                                               Balance: big.NewInt(2000000000000000000), // 2 ether
+                                               Nonce:   0,
+                                       },
+                               },
+                               GasLimit: params.ApricotPhase1GasLimit,
+                       }
+                       genesis       = gspec.MustCommit(db)
+                       blockchain, _ = NewBlockChain(db, DefaultCacheConfig, gspec.Config, dummy.NewFaker(), vm.Config{}, common.Hash{})
+               )
+               defer blockchain.Stop()
+               bigNumber := new(big.Int).SetBytes(common.FromHex("0xffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff"))
+               tooBigNumber := new(big.Int).Set(bigNumber)
+               tooBigNumber.Add(tooBigNumber, common.Big1)
+               for i, tt := range []struct {
+                       txs  []*types.Transaction
+                       want string
+               }{
+                       { // ErrNonceTooLow
+                               txs: []*types.Transaction{
+                                       makeTx(0, common.Address{}, big.NewInt(0), params.TxGas, big.NewInt(225000000000), nil),
+                                       makeTx(0, common.Address{}, big.NewInt(0), params.TxGas, big.NewInt(225000000000), nil),
+                               },
+                               want: "could not apply tx 1 [0x734d821c990099c6ae42d78072aadd3931c35328cf03ef4cf5b2a4ac9c398522]: nonce too low: address 0x71562b71999873DB5b286dF957af199Ec94617F7, tx: 0
+                       },
+                       { // ErrNonceTooHigh
+                               txs: []*types.Transaction{
+                                       makeTx(100, common.Address{}, big.NewInt(0), params.TxGas, big.NewInt(225000000000), nil),
+                               },
+                               want: "could not apply tx 0 [0x0df36254cfbef8ed6961b38fc68aecc777177166144c8a56bc8919e23a559bf4]: nonce too high: address 0x71562b71999873DB5b286dF957af199Ec94617F7, tx: 1
```

```
+			},
+			{ // ErrGasLimitReached
+				txs: []*types.Transaction{
+					makeTx(0, common.Address{}, big.NewInt(0), 8000001, big.NewInt(225000000000), nil),
+				},
+				want: "could not apply tx 0 [0xfbe38b817aaa760c2766b56c019fcdba506560a28fd41c69ae96bdaa4569e317]: gas limit reached",
+			},
+			{ // ErrInsufficientFundsForTransfer
+				txs: []*types.Transaction{
+					makeTx(0, common.Address{}, big.NewInt(2000000000000000000), params.TxGas, big.NewInt(225000000000), nil),
+				},
+				want: "could not apply tx 0 [0xae1601ef55b676ebb824ee7e16a0d14af725b7f9cf5ec79e21f14833c26b5b35]: insufficient funds for gas * price + value: address 0x71562b71999873DB5b2
+			},
+			{ // ErrInsufficientFunds
+				txs: []*types.Transaction{
+					makeTx(0, common.Address{}, big.NewInt(0), params.TxGas, big.NewInt(900000000000000000), nil),
+				},
+				want: "could not apply tx 0 [0x4a69690c4b0cd85e64d0d9ea06302455b01e10a83db964d60281739752003440]: insufficient funds for gas * price + value: address 0x71562b71999873DB5b2
+			},
+			// ErrGasUintOverflow
+			// One missing 'core' error is ErrGasUintOverflow: "gas uint64 overflow",
+			// In order to trigger that one, we'd have to allocate a _huge_ chunk of data, such that the
+			// multiplication len(data) +gas_per_byte overflows uint64. Not testable at the moment
+			{ // ErrIntrinsicGas
+				txs: []*types.Transaction{
+					makeTx(0, common.Address{}, big.NewInt(0), params.TxGas-1000, big.NewInt(225000000000), nil),
+				},
+				want: "could not apply tx 0 [0x2fc3e3b5cc26917d413e26983fe189475f47d4f0757e32aaa5561fcb9c9dc432]: intrinsic gas too low: have 20000, want 21000",
+			},
+			{ // ErrGasLimitReached
+				txs: []*types.Transaction{
+					makeTx(0, common.Address{}, big.NewInt(0), params.TxGas*381, big.NewInt(225000000000), nil),
+				},
+				want: "could not apply tx 0 [0x9ee548e001369418ae53aaa11b5d823f081cc7fa9c9a7ee71a978ae17a2aece0]: gas limit reached",
+			},
+			{ // ErrFeeCapTooLow
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, big.NewInt(0), big.NewInt(0)),
+				},
+				want: "could not apply tx 0 [0xc4ab868fef0c82ae0387b742aee87907f2d0fc528fc6ea0a021459fb0fc4a4a8]: max fee per gas less than block base fee: address 0x71562b71999873DB5b286
+			},
+			{ // ErrTipVeryHigh
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, tooBigNumber, big.NewInt(1)),
+				},
+				want: "could not apply tx 0 [0x15b8391b9981f266b32f3ab7da564bbeb3d6c21628364ea9b32a21139f89f712]: max priority fee per gas higher than 2^256-1: address 0x71562b71999873DB5
+			},
+			{ // ErrFeeCapVeryHigh
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, big.NewInt(1), tooBigNumber),
+				},
+				want: "could not apply tx 0 [0x48bc299b83fdb345c57478f239e89814bb3063eb4e4b49f3b6057a69255c16bd]: max fee per gas higher than 2^256-1: address 0x71562b71999873DB5b286dF957
+			},
+			{ // ErrTipAboveFeeCap
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, big.NewInt(2), big.NewInt(1)),
+				},
+				want: "could not apply tx 0 [0xf987a31ff0c71895780a7612f965a0c8b056deb54e020bb44fa478092f14c9b4]: max priority fee per gas higher than max fee per gas: address 0x71562b719
+			},
+			{ // ErrInsufficientFunds
+				// Available balance:           1000000000000000000
+				// Effective cost:                   18375000021000
+				// FeeCap * gas:                1050000000000000000
+				// This test is designed to have the effective cost be covered by the balance, but
+				// the extended requirement on FeeCap*gas < balance to fail
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, big.NewInt(1), big.NewInt(100000000000000)),
+				},
+				want: "could not apply tx 0 [0x3388378ed60640e75d2edf728d5528a305f599997abc4f23ec46b351b6197499]: insufficient funds for gas * price + value: address 0x71562b71999873DB5b2
+			},
+			{ // Another ErrInsufficientFunds, this one to ensure that feecap/tip of max u256 is allowed
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas, bigNumber, bigNumber),
+				},
+				want: "could not apply tx 0 [0xd82a0c2519acfeac9a948258c47e784acd20651d9d80f9a1c67b4137651c3a24]: insufficient funds for gas * price + value: address 0x71562b71999873DB5b2
+			},
+		} {
+			block := GenerateBadBlock(genesis, dummy.NewFaker(), tt.txs, gspec.Config)
+			_, err := blockchain.InsertChain(types.Blocks{block})
+			if err == nil {
+				t.Fatal("block imported without errors")
+			}
+			if have, want := err.Error(), tt.want; have != want {
+				t.Errorf("test %d:\nhave \"%v\"\nwant \"%v\"\n", i, have, want)
+			}
+		}
+	}
+
+	// ErrTxTypeNotSupported, For this, we need an older chain
+	{
+		var (
+			db    = rawdb.NewMemoryDatabase()
+			gspec = &Genesis{
+				Config: &params.ChainConfig{
+					ChainID:                     big.NewInt(1),
+					HomesteadBlock:              big.NewInt(0),
+					EIP150Block:                 big.NewInt(0),
+					EIP150Hash:                  common.Hash{},
+					EIP155Block:                 big.NewInt(0),
+					EIP158Block:                 big.NewInt(0),
+					ByzantiumBlock:              big.NewInt(0),
+					ConstantinopleBlock:         big.NewInt(0),
+					PetersburgBlock:             big.NewInt(0),
+					IstanbulBlock:               big.NewInt(0),
+					MuirGlacierBlock:            big.NewInt(0),
+					ApricotPhase1BlockTimestamp: big.NewInt(0),
+					ApricotPhase2BlockTimestamp: big.NewInt(0),
+				},
+				Alloc: GenesisAlloc{
+					common.HexToAddress("0x71562b71999873DB5b286dF957af199Ec94617F7"): GenesisAccount{
+						Balance: big.NewInt(1000000000000000000), // 1 ether
+						Nonce:   0,
+					},
+				},
+				GasLimit: params.ApricotPhase1GasLimit,
+			}
+			genesis       = gspec.MustCommit(db)
+			blockchain, _ = NewBlockChain(db, DefaultCacheConfig, gspec.Config, dummy.NewFaker(), vm.Config{}, common.Hash{})
+		)
+		defer blockchain.Stop()
+		for i, tt := range []struct {
+			txs  []*types.Transaction
+			want string
+		}{
+			{ // ErrTxTypeNotSupported
+				txs: []*types.Transaction{
+					mkDynamicTx(0, common.Address{}, params.TxGas-1000, big.NewInt(0), big.NewInt(0)),
+				},
+				want: "could not apply tx 0 [0x88626ac0d53cb65308f2416103c62bb1f18b805573d4f96a3640bbbfff13c14f]: transaction type not supported",
+			},
+		} {
+			block := GenerateBadBlock(genesis, dummy.NewFaker(), tt.txs, gspec.Config)
+			_, err := blockchain.InsertChain(types.Blocks{block})
+			if err == nil {
```

```
+                               t.Fatal("block imported without errors")
+                       }
+                       if have, want := err.Error(), tt.want; have != want {
+                               t.Errorf("test %d:\nhave \"%v\"\nwant \"%v\"\n", i, have, want)
+                       }
+               }
+       }
+
+       // ErrSenderNoEOA, for this we need the sender to have contract code
+       {
+               var (
+                       db    = rawdb.NewMemoryDatabase()
+                       gspec = &Genesis{
+                               Config: config,
+                               Alloc: GenesisAlloc{
+                                       common.HexToAddress("0x71562b71999873DB5b286dF957af199Ec94617F7"): GenesisAccount{
+                                               Balance: big.NewInt(1000000000000000000), // 1 ether
+                                               Nonce:   0,
+                                               Code:    common.FromHex("0xB0B0FACE"),
+                                       },
+                               },
+                               GasLimit: params.ApricotPhase1GasLimit,
+                       }
+                       genesis       = gspec.MustCommit(db)
+                       blockchain, _ = NewBlockChain(db, DefaultCacheConfig, gspec.Config, dummy.NewFaker(), vm.Config{}, common.Hash{})
+               )
+               defer blockchain.Stop()
+               for i, tt := range []struct {
+                       txs  []*types.Transaction
+                       want string
+               }{
+                       { // ErrSenderNoEOA
+                               txs: []*types.Transaction{
+                                       mkDynamicTx(0, common.Address{}, params.TxGas-1000, big.NewInt(0), big.NewInt(0)),
+                               },
+                               want: "could not apply tx 0 [0x88626ac0d53cb65308f2416103c62bb1f18b805573d4f96a3640bbbfff13c14f]: sender not an eoa: address 0x71562b71999873DB5b286dF957af199Ec94617F7, co
+                       },
+               } {
+                       block := GenerateBadBlock(genesis, dummy.NewFaker(), tt.txs, gspec.Config)
+                       _, err := blockchain.InsertChain(types.Blocks{block})
+                       if err == nil {
+                               t.Fatal("block imported without errors")
+                       }
+                       if have, want := err.Error(), tt.want; have != want {
+                               t.Errorf("test %d:\nhave \"%v\"\nwant \"%v\"\n", i, have, want)
+                       }
+               }
+       }
+}
+
+// GenerateBadBlock constructs a "block" which contains the transactions. The transactions are not expected to be
+// valid, and no proper post-state can be made. But from the perspective of the blockchain, the block is sufficiently
+// valid to be considered for import:
+// - valid pow (fake), ancestry, difficulty, gaslimit etc
+func GenerateBadBlock(parent *types.Block, engine consensus.Engine, txs types.Transactions, config *params.ChainConfig) *types.Block {
+       header := &types.Header{
+               ParentHash: parent.Hash(),
+               Coinbase:   parent.Coinbase(),
+               Difficulty: engine.CalcDifficulty(&fakeChainReader{config}, parent.Time()+10, &types.Header{
+                       Number:     parent.Number(),
+                       Time:       parent.Time(),
+                       Difficulty: parent.Difficulty(),
+                       UncleHash:  parent.UncleHash(),
+               }),
+               GasLimit:  parent.GasLimit(),
+               Number:    new(big.Int).Add(parent.Number(), common.Big1),
+               Time:      parent.Time() + 10,
+               UncleHash: types.EmptyUncleHash,
+       }
+       if config.IsApricotPhase3(new(big.Int).SetUint64(header.Time)) {
+               header.Extra, header.BaseFee, _ = dummy.CalcBaseFee(config, parent.Header(), header.Time)
+       }
+       if config.IsApricotPhase4(new(big.Int).SetUint64(header.Time)) {
+               header.BlockGasCost = big.NewInt(0)
+               header.ExtDataGasUsed = big.NewInt(0)
+       }
+       var receipts []*types.Receipt
+       // The post-state result doesn't need to be correct (this is a bad block), but we do need something there
+       // Preferably something unique. So let's use a combo of blocknum + txhash
+       hasher := sha3.NewLegacyKeccak256()
+       hasher.Write(header.Number.Bytes())
+       var cumulativeGas uint64
+       for _, tx := range txs {
+               txh := tx.Hash()
+               hasher.Write(txh[:])
+               receipt := types.NewReceipt(nil, false, cumulativeGas+tx.Gas())
+               receipt.TxHash = tx.Hash()
+               receipt.GasUsed = tx.Gas()
+               receipts = append(receipts, receipt)
+               cumulativeGas += tx.Gas()
+       }
+       header.Root = common.BytesToHash(hasher.Sum(nil))
+       // Assemble and return the final block for sealing
+       return types.NewBlock(header, txs, nil, receipts, trie.NewStackTrie(nil), nil, true)
+}
diff --git a/core/state_transition.go b/core/state_transition.go
index 381327cf..7eba1bca 100644
--- a/core/state_transition.go
+++ b/core/state_transition.go
@@ -33,10 +33,10 @@ import (

        "github.com/ethereum/go-ethereum/crypto"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )

 var emptyCodeHash = crypto.Keccak256Hash(nil)
@@ -115,6 +115,23 @@ func (result *ExecutionResult) Return() []byte {
        return common.CopyBytes(result.ReturnData)
 }

+// Implement the EVMCaller interface on the state transition structure; simply delegate the calls
+func (st *StateTransition) Call(caller vm.ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
+       return st.evm.Call(caller, addr, input, gas, value)
+}
+
+func (st *StateTransition) GetBlockNumber() *big.Int {
+       return st.evm.Context.BlockNumber
+}
+
+func (st *StateTransition) GetGasLimit() uint64 {
+       return st.evm.Context.GasLimit
+}
+
+func (st *StateTransition) AddBalance(addr common.Address, amount *big.Int) {
+       st.state.AddBalance(addr, amount)
+}
+
 // Revert returns the concrete revert reason if the execution is aborted by `REVERT`
 // opcode. Note the reason can be nil if no data supplied with revert opcode.
 func (result *ExecutionResult) Revert() []byte {
@@ -232,12 +249,18 @@ func (st *StateTransition) preCheck() error {
 		} else if stNonce > msgNonce {
 			return fmt.Errorf("%w: address %v, tx: %d state: %d", ErrNonceTooLow,
 				st.msg.From().Hex(), msgNonce, stNonce)
+		} else if stNonce+1 < stNonce {
+			return fmt.Errorf("%w: address %v, nonce: %d", ErrNonceMax,
+				st.msg.From().Hex(), stNonce)
 		}
 		// Make sure the sender is an EOA
 		if codeHash := st.state.GetCodeHash(st.msg.From()); codeHash != emptyCodeHash && codeHash != (common.Hash{}) {
 			return fmt.Errorf("%w: address %v, codehash: %s", ErrSenderNoEOA,
 				st.msg.From().Hex(), codeHash)
 		}
+		if st.msg.From() == st.evm.Context.Coinbase {
+			return fmt.Errorf("%w: address %v", vm.ErrNoSenderBlackhole, st.msg.From())
+		}
 	}
 	// Make sure that transaction gasFeeCap is greater than the baseFee (post london)
 	if st.evm.ChainConfig().IsApricotPhase3(st.evm.Context.Time) {
@@ -321,19 +344,69 @@ func (st *StateTransition) TransitionDb() (*ExecutionResult, error) {
 	if rules := st.evm.ChainConfig().AvalancheRules(st.evm.Context.BlockNumber, st.evm.Context.Time); rules.IsApricotPhase2 {
 		st.state.PrepareAccessList(msg.From(), msg.To(), vm.ActivePrecompiles(rules), msg.AccessList())
 	}
+
 	var (
-		ret   []byte
-		vmerr error // vm errors do not effect consensus and are therefore not assigned to err
+		ret         []byte
+		vmerr       error // vm errors do not affect consensus and are therefore not assigned to err
+		chainID     *big.Int
+		timestamp   *big.Int
+		burnAddress common.Address
 	)
+
+	chainID = st.evm.ChainConfig().ChainID
+	timestamp = st.evm.Context.Time
+	burnAddress = st.evm.Context.Coinbase
+	if burnAddress != common.HexToAddress("0x0100000000000000000000000000000000000000") {
+		return nil, fmt.Errorf("Invalid value for block.coinbase")
+	}
+
 	if contractCreation {
 		ret, _, st.gas, vmerr = st.evm.Create(sender, st.data, st.gas, st.value)
 	} else {
 		// Increment the nonce for the next transaction
 		st.state.SetNonce(msg.From(), st.state.GetNonce(sender.Address())+1)
 		ret, st.gas, vmerr = st.evm.Call(sender, st.to(), st.data, st.gas, st.value)
+		if vmerr == nil && *msg.To() == GetStateConnectorContract(chainID, timestamp) && len(st.data) >= 36 && len(ret) == 32 {
+			if GetStateConnectorActivated(chainID, timestamp) &&
+				bytes.Equal(st.data[0:4], SubmitAttestationSelector(chainID, timestamp)) &&
+				binary.BigEndian.Uint64(ret[24:32]) > 0 {
+				err = st.FinalisePreviousRound(chainID, timestamp, st.data[4:36])
+				if err != nil {
+					log.Warn("Error finalising state connector round", "error", err)
+				}
+			}
+		}
 	}
+
 	st.refundGas(apricotPhase1)
-	st.state.AddBalance(st.evm.Context.Coinbase, new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice))
+	if vmerr == nil && msg.To() != nil && *msg.To() == common.HexToAddress(GetPrioritisedFTSOContract(timestamp)) {
+		nominalGasUsed := uint64(21000)
+		nominalGasPrice := uint64(225_000_000_000)
+		nominalFee := new(big.Int).Mul(new(big.Int).SetUint64(nominalGasUsed), new(big.Int).SetUint64(nominalGasPrice))
+		actualGasUsed := st.gasUsed()
+		actualGasPrice := st.gasPrice
+		actualFee := new(big.Int).Mul(new(big.Int).SetUint64(actualGasUsed), actualGasPrice)
+		if actualFee.Cmp(nominalFee) > 0 {
+			feeRefund := new(big.Int).Sub(actualFee, nominalFee)
+			st.state.AddBalance(st.msg.From(), feeRefund)
+			st.state.AddBalance(burnAddress, nominalFee)
+		} else {
+			st.state.AddBalance(burnAddress, actualFee)
+		}
+	} else {
+		st.state.AddBalance(burnAddress, new(big.Int).Mul(new(big.Int).SetUint64(st.gasUsed()), st.gasPrice))
+	}
+
+	// Call the keeper contract trigger method if there is no vm error
+	if vmerr == nil {
+		// Temporarily disable EVM debugging
+		oldDebug := st.evm.Config.Debug
+		st.evm.Config.Debug = false
+		// Call the keeper contract trigger
+		log := log.Root()
+		triggerKeeperAndMint(st, log)
+		st.evm.Config.Debug = oldDebug
+	}

 	return &ExecutionResult{
 		UsedGas:    st.gasUsed(),
diff --git a/core/test_blockchain.go b/core/test_blockchain.go
index 7b7a1109..1de25016 100644
--- a/core/test_blockchain.go
+++ b/core/test_blockchain.go
@@ -9,14 +9,14 @@ import (
 	"strings"
 	"testing"

-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/params"
 )

 type ChainTest struct {
@@ -1334,7 +1334,7 @@ func TestGenerateChainInvalidBlockFee(t *testing.T, create func(db ethdb.Databas
 	if err == nil {
 		t.Fatal("should not have been able to build a block because of insufficient block fee")
 	}
-	if !strings.Contains(err.Error(), "insufficient gas (0) to cover the block cost (100000)") {
+	if !strings.Contains(err.Error(), "insufficient gas (0) to cover the block cost (400000)") {
 		t.Fatalf("should have gotten insufficient block fee error but got %v instead", err)
 	}
 }
@@ -1404,7 +1404,7 @@ func TestInsertChainInvalidBlockFee(t *testing.T, create func(db ethdb.Database,
 	if err == nil {
 		t.Fatal("should not have been able to build a block because of insufficient block fee")
 	}
-	if !strings.Contains(err.Error(), "insufficient gas (0) to cover the block cost (100000)") {
```

```
+       if !strings.Contains(err.Error(), "insufficient gas (0) to cover the block cost (400000)") {
                t.Fatalf("should have gotten insufficient block fee error but got %v instead", err)
        }
 }
@@ -1440,7 +1440,7 @@ func TestInsertChainValidBlockFee(t *testing.T, create func(db ethdb.Database, c
        signer := types.LatestSigner(params.TestChainConfig)
        // Generate chain of blocks using [genDB] instead of [chainDB] to avoid writing
        // to the BlockChain's database while generating blocks.
-       tip := big.NewInt(2000 * params.GWei)
+       tip := big.NewInt(50000 * params.GWei)
        transfer := big.NewInt(10000)
        chain, _, err := GenerateChain(gspec.Config, genesis, blockchain.engine, genDB, 3, 0, func(i int, gen *BlockGen) {
                feeCap := new(big.Int).Add(gen.BaseFee(), tip)
diff --git a/core/tx_cacher.go b/core/tx_cacher.go
index 19b92806..bd71fbe4 100644
--- a/core/tx_cacher.go
+++ b/core/tx_cacher.go
@@ -27,7 +27,7 @@
 package core

 import (
-       "github.com/ava-labs/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // txSenderCacherRequest is a request for recovering transaction senders with a
diff --git a/core/tx_journal.go b/core/tx_journal.go
index b2bfa538..ee9db7ad 100644
--- a/core/tx_journal.go
+++ b/core/tx_journal.go
@@ -31,10 +31,10 @@ import (
        "io"
        "os"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // errNoActiveJournal is returned if a transaction is attempted to be inserted
diff --git a/core/tx_list.go b/core/tx_list.go
index 4a6999e3..da04b5d3 100644
--- a/core/tx_list.go
+++ b/core/tx_list.go
@@ -35,8 +35,8 @@ import (
        "sync/atomic"
        "time"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // nonceHeap is a heap.Interface implementation over 64bit unsigned integers for
diff --git a/core/tx_list_test.go b/core/tx_list_test.go
index ecfa9154..74209920 100644
--- a/core/tx_list_test.go
+++ b/core/tx_list_test.go
@@ -31,8 +31,8 @@ import (
        "math/rand"
        "testing"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // Tests that transactions can be added to strict lists and list contents and
diff --git a/core/tx_noncer.go b/core/tx_noncer.go
index 0dcd31c4..c1c1e1f6 100644
--- a/core/tx_noncer.go
+++ b/core/tx_noncer.go
@@ -29,8 +29,8 @@ package core
 import (
        "sync"

-       "github.com/ava-labs/coreth/core/state"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/state"
 )

 // txNoncer is a tiny virtual state database to manage the executable nonces of
diff --git a/core/tx_pool.go b/core/tx_pool.go
index abc7f562..7bd7e1e3 100644
--- a/core/tx_pool.go
+++ b/core/tx_pool.go
@@ -36,15 +36,15 @@ import (
        "sync/atomic"
        "time"

-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/prque"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/metrics"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
 )

 const (
@@ -1253,7 +1253,7 @@ func (pool *TxPool) runReorg(done chan struct{}, reset *txpoolResetRequest, dirt
        if reset != nil {
                pool.demoteUnexecutables()
                if reset.newHead != nil && pool.chainconfig.IsApricotPhase3(new(big.Int).SetUint64(reset.newHead.Time)) {
-                       _, baseFeeEstimate, err := dummy.CalcBaseFee(pool.chainconfig, reset.newHead, uint64(time.Now().Unix()))
+                       _, baseFeeEstimate, err := dummy.EstimateNextBaseFee(pool.chainconfig, reset.newHead, uint64(time.Now().Unix()))
                        if err == nil {
                                pool.priced.SetBaseFee(baseFeeEstimate)
                        }
@@ -1696,7 +1696,7 @@ func (pool *TxPool) updateBaseFee() {
        pool.mu.Lock()
        defer pool.mu.Unlock()

-       _, baseFeeEstimate, err := dummy.CalcBaseFee(pool.chainconfig, pool.currentHead, uint64(time.Now().Unix()))
+       _, baseFeeEstimate, err := dummy.EstimateNextBaseFee(pool.chainconfig, pool.currentHead, uint64(time.Now().Unix()))
        if err == nil {
                pool.priced.SetBaseFee(baseFeeEstimate)
        } else {
diff --git a/core/tx_pool_test.go b/core/tx_pool_test.go
index 32725ea0..6ddcb230 100644
--- a/core/tx_pool_test.go
+++ b/core/tx_pool_test.go
@@ -1,3 +1,13 @@
+// (c) 2019-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
 // Copyright 2015 The go-ethereum Authors
 // This file is part of the go-ethereum library.
 //
@@ -29,14 +39,14 @@ import (
 	"testing"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 var (
diff --git a/core/types.go b/core/types.go
index aa8d9873..9ba59a65 100644
--- a/core/types.go
+++ b/core/types.go
@@ -27,9 +27,9 @@
 package core
 
 import (
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
 )
 
 // Validator is an interface which defines the standard for block validation. It
diff --git a/core/types/block.go b/core/types/block.go
index 58d2fc49..9b97bbf5 100644
--- a/core/types/block.go
+++ b/core/types/block.go
@@ -169,10 +169,6 @@ type Block struct {
 	// caches
 	hash atomic.Value
 	size atomic.Value
-
-	// Td is used by package core to store the total difficulty
-	// of the chain up to and including the block.
-	td *big.Int
 }
 
 // "external" block encoding. used for eth protocol, etc.
@@ -195,7 +191,7 @@ func NewBlock(
 	header *Header, txs []*Transaction, uncles []*Header, receipts []*Receipt,
 	hasher TrieHasher, extdata []byte, recalc bool,
 ) *Block {
-	b := &Block{header: CopyHeader(header), td: new(big.Int)}
+	b := &Block{header: CopyHeader(header)}
 
 	// TODO: panic if len(txs) != len(receipts)
 	if len(txs) == 0 {
diff --git a/core/types/block_test.go b/core/types/block_test.go
index 892e5c4c..f00396ad 100644
--- a/core/types/block_test.go
+++ b/core/types/block_test.go
@@ -33,11 +33,11 @@ import (
 	"reflect"
 	"testing"
 
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/math"
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/params"
 	"golang.org/x/crypto/sha3"
 )
 
diff --git a/core/types/dynamic_fee_tx.go b/core/types/dynamic_fee_tx.go
index 7e288210..c4ec28c5 100644
--- a/core/types/dynamic_fee_tx.go
+++ b/core/types/dynamic_fee_tx.go
@@ -35,8 +35,8 @@ import (
 type DynamicFeeTx struct {
 	ChainID    *big.Int
 	Nonce      uint64
-	GasTipCap  *big.Int
-	GasFeeCap  *big.Int
+	GasTipCap  *big.Int // a.k.a. maxPriorityFeePerGas
+	GasFeeCap  *big.Int // a.k.a. maxFeePerGas
 	Gas        uint64
 	To         *common.Address `rlp:"nil"` // nil means contract creation
 	Value      *big.Int
diff --git a/core/types/hashing_test.go b/core/types/hashing_test.go
index 60f9da10..004fa888 100644
--- a/core/types/hashing_test.go
+++ b/core/types/hashing_test.go
@@ -34,12 +34,12 @@ import (
 	mrand "math/rand"
 	"testing"
 
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/trie"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/trie"
 )
 
 func TestDeriveSha(t *testing.T) {
diff --git a/core/types/receipt.go b/core/types/receipt.go
index 176cb109..885722d2 100644
--- a/core/types/receipt.go
+++ b/core/types/receipt.go
@@ -34,11 +34,11 @@ import (
 	"math/big"
 	"unsafe"
 
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/crypto"
```

```
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/params"
 )

 //go:generate gencodec -type Receipt -field-override receiptMarshaling -out gen_receipt_json.go
diff --git a/core/types/receipt_test.go b/core/types/receipt_test.go
index d0c1553e..a52f6555 100644
--- a/core/types/receipt_test.go
+++ b/core/types/receipt_test.go
@@ -33,10 +33,10 @@ import (
        "reflect"
        "testing"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/params"
 )

 var (
diff --git a/core/types/transaction_signing.go b/core/types/transaction_signing.go
index a717749b..eb0e1d8d 100644
--- a/core/types/transaction_signing.go
+++ b/core/types/transaction_signing.go
@@ -32,9 +32,9 @@ import (
        "fmt"
        "math/big"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/params"
 )

 var ErrInvalidChainId = errors.New("invalid chain id for signer")
diff --git a/core/vm/access_list_tracer.go b/core/vm/access_list_tracer.go
index fb67897e..c062a406 100644
--- a/core/vm/access_list_tracer.go
+++ b/core/vm/access_list_tracer.go
@@ -30,8 +30,9 @@ import (
        "math/big"
        "time"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/coreth/core/types"
 )

 // accessList is an accumulator for the set of accounts and storage slots an EVM
diff --git a/core/vm/analysis.go b/core/vm/analysis.go
index 752e27dd..cfbf0e7f 100644
--- a/core/vm/analysis.go
+++ b/core/vm/analysis.go
@@ -27,12 +27,12 @@
 package vm

 const (
-       set2BitsMask = uint16(0b1100_0000_0000_0000)
-       set3BitsMask = uint16(0b1110_0000_0000_0000)
-       set4BitsMask = uint16(0b1111_0000_0000_0000)
-       set5BitsMask = uint16(0b1111_1000_0000_0000)
-       set6BitsMask = uint16(0b1111_1100_0000_0000)
-       set7BitsMask = uint16(0b1111_1110_0000_0000)
+       set2BitsMask = uint16(0b11)
+       set3BitsMask = uint16(0b111)
+       set4BitsMask = uint16(0b1111)
+       set5BitsMask = uint16(0b1_1111)
+       set6BitsMask = uint16(0b11_1111)
+       set7BitsMask = uint16(0b111_1111)
 )

 // bitvec is a bit vector which maps bytes in a program.
@@ -40,32 +40,26 @@ const (
 // it's data (i.e. argument of PUSHxx).
 type bitvec []byte

-var lookup = [8]byte{
-       0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1,
-}
-
 func (bits bitvec) set1(pos uint64) {
-       bits[pos/8] |= lookup[pos%8]
+       bits[pos/8] |= 1 << (pos % 8)
 }

 func (bits bitvec) setN(flag uint16, pos uint64) {
-       a := flag >> (pos % 8)
-       bits[pos/8] |= byte(a >> 8)
-       if b := byte(a); b != 0 {
-               //      If the bit-setting affects the neighbouring byte, we can assign - no need to OR it,
-               //      since it's the first write to that byte
+       a := flag << (pos % 8)
+       bits[pos/8] |= byte(a)
+       if b := byte(a >> 8); b != 0 {
                bits[pos/8+1] = b
        }
 }

 func (bits bitvec) set8(pos uint64) {
-       a := byte(0xFF >> (pos % 8))
+       a := byte(0xFF << (pos % 8))
        bits[pos/8] |= a
        bits[pos/8+1] = ^a
 }

 func (bits bitvec) set16(pos uint64) {
-       a := byte(0xFF >> (pos % 8))
+       a := byte(0xFF << (pos % 8))
        bits[pos/8] |= a
        bits[pos/8+1] = 0xFF
        bits[pos/8+2] = ^a
@@ -73,7 +67,7 @@ func (bits bitvec) set16(pos uint64) {

 // codeSegment checks if the position is in a code segment.
 func (bits *bitvec) codeSegment(pos uint64) bool {
-       return ((*bits)[pos/8] & (0x80 >> (pos % 8))) == 0
+       return (((*bits)[pos/8] >> (pos % 8)) & 1) == 0
 }

 // codeBitmap collects data locations in code.
diff --git a/core/vm/analysis_test.go b/core/vm/analysis_test.go
index 96fa0377..ae6bef9b 100644
--- a/core/vm/analysis_test.go
+++ b/core/vm/analysis_test.go
@@ -27,6 +27,7 @@
 package vm

 import (
+       "math/bits"
        "testing"

        "github.com/ethereum/go-ethereum/crypto"
```

```
@@ -38,24 +39,27 @@ func TestJumpDestAnalysis(t *testing.T) {
                exp   byte
                which int
        }{
-               {[]byte{byte(PUSH1), 0x01, 0x01, 0x01}, 0x40, 0},
-               {[]byte{byte(PUSH1), byte(PUSH1), byte(PUSH1), byte(PUSH1)}, 0x50, 0},
-               {[]byte{byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), 0x01, 0x01, 0x01}, 0x7F, 0},
-               {[]byte{byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x80, 1},
-               {[]byte{0x01, 0x01, 0x01, 0x01, 0x01, byte(PUSH2), byte(PUSH2), byte(PUSH2), 0x01, 0x01, 0x01}, 0x03, 0},
-               {[]byte{0x01, 0x01, 0x01, 0x01, 0x01, byte(PUSH2), 0x01, 0x01, 0x01, 0x01, 0x01}, 0x00, 1},
-               {[]byte{byte(PUSH3), 0x01, 0x01, 0x01, byte(PUSH1), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x74, 0},
-               {[]byte{byte(PUSH3), 0x01, 0x01, 0x01, byte(PUSH1), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x00, 1},
-               {[]byte{0x01, byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x3F, 0},
-               {[]byte{0x01, byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0xC0, 1},
-               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x7F, 0},
-               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0xFF, 1},
-               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0x80, 2},
-               {[]byte{byte(PUSH8), 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, byte(PUSH1), 0x01}, 0x7f, 0},
-               {[]byte{byte(PUSH8), 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, byte(PUSH1), 0x01}, 0xA0, 1},
-               {[]byte{byte(PUSH32)}, 0x7F, 0},
-               {[]byte{byte(PUSH32)}, 0xFF, 1},
-               {[]byte{byte(PUSH32)}, 0xFF, 2},
+               {[]byte{byte(PUSH1), 0x01, 0x01, 0x01}, 0b0000_0010, 0},
+               {[]byte{byte(PUSH1), byte(PUSH1), byte(PUSH1), byte(PUSH1)}, 0b0000_1010, 0},
+               {[]byte{0x00, byte(PUSH1), 0x00, byte(PUSH1), 0x00, byte(PUSH1), 0x00, byte(PUSH1)}, 0b0101_0100, 0},
+               {[]byte{byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), byte(PUSH8), 0x01, 0x01, 0x01}, bits.Reverse8(0x7F), 0},
+               {[]byte{byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0000_0001, 1},
+               {[]byte{0x01, 0x01, 0x01, 0x01, 0x01, byte(PUSH2), byte(PUSH2), byte(PUSH2), 0x01, 0x01, 0x01}, 0b1100_0000, 0},
+               {[]byte{0x01, 0x01, 0x01, 0x01, 0x01, byte(PUSH2), 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0000_0000, 1},
+               {[]byte{byte(PUSH3), 0x01, 0x01, 0x01, byte(PUSH1), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0010_1110, 0},
+               {[]byte{byte(PUSH3), 0x01, 0x01, 0x01, byte(PUSH1), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0000_0000, 1},
+               {[]byte{0x01, byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b1111_1100, 0},
+               {[]byte{0x01, byte(PUSH8), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0000_0011, 1},
+               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b1111_1110, 0},
+               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b1111_1111, 1},
+               {[]byte{byte(PUSH16), 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01, 0x01}, 0b0000_0001, 2},
+               {[]byte{byte(PUSH8), 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, byte(PUSH1), 0x01}, 0b1111_1110, 0},
+               {[]byte{byte(PUSH8), 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07, 0x08, byte(PUSH1), 0x01}, 0b0000_0101, 1},
+               {[]byte{byte(PUSH32)}, 0b1111_1110, 0},
+               {[]byte{byte(PUSH32)}, 0b1111_1111, 1},
+               {[]byte{byte(PUSH32)}, 0b1111_1111, 2},
+               {[]byte{byte(PUSH32)}, 0b1111_1111, 3},
+               {[]byte{byte(PUSH32)}, 0b0000_0001, 4},
        }
        for i, test := range tests {
                ret := codeBitmap(test.code)
diff --git a/core/vm/contract.go b/core/vm/contract.go
index c7e4b53a..78e9fcd8 100644
--- a/core/vm/contract.go
+++ b/core/vm/contract.go
@@ -153,16 +153,11 @@ func (c *Contract) AsDelegate() *Contract {

 // GetOp returns the n'th element in the contract's byte array
 func (c *Contract) GetOp(n uint64) OpCode {
-       return OpCode(c.GetByte(n))
-}
-
-// GetByte returns the n'th byte in the contract's byte array
-func (c *Contract) GetByte(n uint64) byte {
        if n < uint64(len(c.Code)) {
-               return c.Code[n]
+               return OpCode(c.Code[n])
        }

-       return 0
+       return STOP
 }

 // Caller returns the caller of the contract.
diff --git a/core/vm/contracts.go b/core/vm/contracts.go
index cfb26714..40880420 100644
--- a/core/vm/contracts.go
+++ b/core/vm/contracts.go
@@ -32,13 +32,13 @@ import (
        "errors"
        "math/big"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/crypto/blake2b"
        "github.com/ethereum/go-ethereum/crypto/bls12381"
        "github.com/ethereum/go-ethereum/crypto/bn256"
+       "github.com/flare-foundation/coreth/params"

        //lint:ignore SA1019 Needed for precompile
        "golang.org/x/crypto/ripemd160"
@@ -101,8 +101,8 @@ var PrecompiledContractsApricotPhase2 = map[common.Address]StatefulPrecompiledCo
        common.BytesToAddress([]byte{8}): newWrappedPrecompiledContract(&bn256PairingIstanbul{}),
        common.BytesToAddress([]byte{9}): newWrappedPrecompiledContract(&blake2F{}),
        genesisContractAddr:              &deprecatedContract{},
-       nativeAssetBalanceAddr:           &nativeAssetBalance{gasCost: params.AssetBalanceApricot},
-       nativeAssetCallAddr:              &nativeAssetCall{gasCost: params.AssetCallApricot},
+       NativeAssetBalanceAddr:           &nativeAssetBalance{gasCost: params.AssetBalanceApricot},
+       NativeAssetCallAddr:              &nativeAssetCall{gasCost: params.AssetCallApricot},
 }

 var (
diff --git a/core/vm/contracts_stateful.go b/core/vm/contracts_stateful.go
index 981ebd72..bffc33a9 100644
--- a/core/vm/contracts_stateful.go
+++ b/core/vm/contracts_stateful.go
@@ -7,8 +7,8 @@ import (
        "fmt"
        "math/big"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/params"
        "github.com/holiman/uint256"
 )

@@ -20,8 +20,8 @@ import (

 var (
        genesisContractAddr    = common.HexToAddress("0x0100000000000000000000000000000000000000")
-       nativeAssetBalanceAddr = common.HexToAddress("0x0100000000000000000000000000000000000001")
-       nativeAssetCallAddr    = common.HexToAddress("0x0100000000000000000000000000000000000002")
+       NativeAssetBalanceAddr = common.HexToAddress("0x0100000000000000000000000000000000000001")
+       NativeAssetCallAddr    = common.HexToAddress("0x0100000000000000000000000000000000000002")
 )

 // StatefulPrecompiledContract is the interface for executing a precompiled contract
@@ -54,6 +54,8 @@ type nativeAssetBalance struct {
        gasCost uint64
 }

+// PackNativeAssetBalanceInput packs the arguments into the required input data for a transaction to be passed into
+// the native asset balance precompile.
 func PackNativeAssetBalanceInput(address common.Address, assetID common.Hash) []byte {
        input := make([]byte, 52)
        copy(input, address.Bytes())
@@ -61,6 +63,7 @@ func PackNativeAssetBalanceInput(address common.Address, assetID common.Hash) []
```

```
        return input
 }

+// UnpackNativeAssetBalanceInput attempts to unpack [input] into the arguments to the native asset balance precompile
 func UnpackNativeAssetBalanceInput(input []byte) (common.Address, common.Hash, error) {
        if len(input) != 52 {
                return common.Address{}, common.Hash{}, fmt.Errorf("native asset balance input had unexpcted length %d", len(input))
@@ -97,6 +100,9 @@ type nativeAssetCall struct {
        gasCost uint64
 }

+// PackNativeAssetCallInput packs the arguments into the required input data for a transaction to be passed into
+// the native asset precompile.
+// Assumes that [assetAmount] is non-nil.
 func PackNativeAssetCallInput(address common.Address, assetID common.Hash, assetAmount *big.Int, callData []byte) []byte {
        input := make([]byte, 84+len(callData))
        copy(input[0:20], address.Bytes())
@@ -106,13 +112,13 @@ func PackNativeAssetCallInput(address common.Address, assetID common.Hash, asset
        return input
 }

-func UnpackNativeAssetCallInput(input []byte) (common.Address, *common.Hash, *big.Int, []byte, error) {
+// UnpackNativeAssetCallInput attempts to unpack [input] into the arguments to the native asset call precompile
+func UnpackNativeAssetCallInput(input []byte) (common.Address, common.Hash, *big.Int, []byte, error) {
        if len(input) < 84 {
-               return common.Address{}, nil, nil, nil, fmt.Errorf("native asset call input had unexpcted length %d", len(input))
+               return common.Address{}, common.Hash{}, nil, nil, fmt.Errorf("native asset call input had unexpected length %d", len(input))
        }
        to := common.BytesToAddress(input[:20])
-       assetID := new(common.Hash)
-       assetID.SetBytes(input[20:52])
+       assetID := common.BytesToHash(input[20:52])
        assetAmount := new(big.Int).SetBytes(input[52:84])
        callData := input[84:]
        return to, assetID, assetAmount, callData, nil
@@ -135,6 +141,8 @@ func (c *nativeAssetCall) Run(evm *EVM, caller ContractRef, addr common.Address,
                return nil, remainingGas, ErrExecutionReverted
        }

+       // Note: it is not possible for a negative assetAmount to be passed in here due to the fact that decoding a
+       // byte slice into a *big.Int type will always return a positive value.
        if assetAmount.Sign() != 0 && !evm.Context.CanTransferMC(evm.StateDB, caller.Address(), to, assetID, assetAmount) {
                return nil, remainingGas, ErrInsufficientBalance
        }
@@ -174,6 +182,6 @@ func (c *nativeAssetCall) Run(evm *EVM, caller ContractRef, addr common.Address,

 type deprecatedContract struct{}

-func (_ *deprecatedContract) Run(evm *EVM, caller ContractRef, addr common.Address, input []byte, suppliedGas uint64, readOnly bool) (ret []byte, remainingGas uint64, err error) {
+func (*deprecatedContract) Run(evm *EVM, caller ContractRef, addr common.Address, input []byte, suppliedGas uint64, readOnly bool) (ret []byte, remainingGas uint64, err error) {
        return nil, suppliedGas, ErrExecutionReverted
 }
diff --git a/core/vm/contracts_stateful_test.go b/core/vm/contracts_stateful_test.go
index 6126078f..d2d2df10 100644
--- a/core/vm/contracts_stateful_test.go
+++ b/core/vm/contracts_stateful_test.go
@@ -7,11 +7,11 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
        "github.com/stretchr/testify/assert"
 )

@@ -36,16 +36,9 @@ func CanTransfer(db StateDB, addr common.Address, amount *big.Int) bool {
        return db.GetBalance(addr).Cmp(amount) >= 0
 }

-func CanTransferMC(db StateDB, addr common.Address, to common.Address, coinID *common.Hash, amount *big.Int) bool {
+func CanTransferMC(db StateDB, addr common.Address, to common.Address, coinID common.Hash, amount *big.Int) bool {
        log.Info("CanTransferMC", "address", addr, "to", to, "coinID", coinID, "amount", amount)
-       if coinID == nil {
-               return true
-       }
-       if db.GetBalanceMultiCoin(addr, *coinID).Cmp(amount) >= 0 {
-               return true
-       }
-       // insufficient balance
-       return false
+       return db.GetBalanceMultiCoin(addr, coinID).Cmp(amount) >= 0
 }

 // Transfer subtracts amount from sender and adds amount to recipient using the given Db
@@ -55,12 +48,9 @@ func Transfer(db StateDB, sender, recipient common.Address, amount *big.Int) {
 }

 // Transfer subtracts amount from sender and adds amount to recipient using the given Db
-func TransferMultiCoin(db StateDB, sender, recipient common.Address, coinID *common.Hash, amount *big.Int) {
-       if coinID == nil {
-               return
-       }
-       db.SubBalanceMultiCoin(sender, *coinID, amount)
-       db.AddBalanceMultiCoin(recipient, *coinID, amount)
+func TransferMultiCoin(db StateDB, sender, recipient common.Address, coinID common.Hash, amount *big.Int) {
+       db.SubBalanceMultiCoin(sender, coinID, amount)
+       db.AddBalanceMultiCoin(recipient, coinID, amount)
 }

 func TestPackNativeAssetCallInput(t *testing.T) {
@@ -74,7 +64,7 @@ func TestPackNativeAssetCallInput(t *testing.T) {
        unpackedAddr, unpackedAssetID, unpackedAssetAmount, unpackedCallData, err := UnpackNativeAssetCallInput(input)
        assert.NoError(t, err)
        assert.Equal(t, addr, unpackedAddr, "address")
-       assert.Equal(t, &assetID, unpackedAssetID, "assetID")
+       assert.Equal(t, assetID, unpackedAssetID, "assetID")
        assert.Equal(t, assetAmount, unpackedAssetAmount, "assetAmount")
        assert.Equal(t, callData, unpackedCallData, "callData")
 }
@@ -131,7 +121,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                userAddr1,
-                       precompileAddr:      nativeAssetBalanceAddr,
+                       precompileAddr:      NativeAssetBalanceAddr,
                        input:               PackNativeAssetBalanceInput(userAddr1, assetID),
                        value:               big0,
                        gasInput:            params.AssetBalanceApricot,
@@ -157,7 +147,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                userAddr1,
-                       precompileAddr:      nativeAssetBalanceAddr,
+                       precompileAddr:      NativeAssetBalanceAddr,
                        input:               PackNativeAssetBalanceInput(userAddr1, assetID),
                        value:               big0,
                        gasInput:            params.AssetBalanceApricot,
@@ -182,7 +172,7 @@ func TestStatefulPrecompile(t *testing.T) {
```

```
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetBalanceAddr,
+                       precompileAddr:       NativeAssetBalanceAddr,
                        input:                PackNativeAssetBalanceInput(userAddr1, assetID),
                        value:                big0,
                        gasInput:             params.AssetBalanceApricot,
@@ -200,7 +190,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetBalanceAddr,
+                       precompileAddr:       NativeAssetBalanceAddr,
                        input:                nil,
                        value:                big0,
                        gasInput:             params.AssetBalanceApricot,
@@ -218,7 +208,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetBalanceAddr,
+                       precompileAddr:       NativeAssetBalanceAddr,
                        input:                PackNativeAssetBalanceInput(userAddr1, assetID),
                        value:                big0,
                        gasInput:             params.AssetBalanceApricot - 1,
@@ -236,7 +226,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetBalanceAddr,
+                       precompileAddr:       NativeAssetBalanceAddr,
                        input:                PackNativeAssetBalanceInput(userAddr1, assetID),
                        value:                bigHundred,
                        gasInput:             params.AssetBalanceApricot,
@@ -257,7 +247,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(50), nil),
                        value:                big0,
                        gasInput:             params.AssetCallApricot + params.CallNewAccountGas,
@@ -290,7 +280,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(50), nil),
                        value:                big.NewInt(49),
                        gasInput:             params.AssetCallApricot + params.CallNewAccountGas,
@@ -301,7 +291,7 @@ func TestStatefulPrecompile(t *testing.T) {
                        stateDBCheck: func(t *testing.T, stateDB StateDB) {
                                user1Balance := stateDB.GetBalance(userAddr1)
                                user2Balance := stateDB.GetBalance(userAddr2)
-                               nativeAssetCallAddrBalance := stateDB.GetBalance(nativeAssetCallAddr)
+                               nativeAssetCallAddrBalance := stateDB.GetBalance(NativeAssetCallAddr)
                                user1AssetBalance := stateDB.GetBalanceMultiCoin(userAddr1, assetID)
                                user2AssetBalance := stateDB.GetBalanceMultiCoin(userAddr2, assetID)
                                expectedBalance := big.NewInt(50)
@@ -325,7 +315,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(51), nil),
                        value:                big.NewInt(50),
                        gasInput:             params.AssetCallApricot,
@@ -357,7 +347,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(50), nil),
                        value:                big.NewInt(51),
                        gasInput:             params.AssetCallApricot,
@@ -389,7 +379,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(50), nil),
                        value:                big.NewInt(50),
                        gasInput:             params.AssetCallApricot - 1,
@@ -410,7 +400,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                PackNativeAssetCallInput(userAddr2, assetID, big.NewInt(50), nil),
                        value:                big.NewInt(50),
                        gasInput:             params.AssetCallApricot + params.CallNewAccountGas - 1,
@@ -442,7 +432,7 @@ func TestStatefulPrecompile(t *testing.T) {
                                return statedb
                        },
                        from:                 userAddr1,
-                       precompileAddr:       nativeAssetCallAddr,
+                       precompileAddr:       NativeAssetCallAddr,
                        input:                make([]byte, 24),
                        value:                big.NewInt(50),
                        gasInput:             params.AssetCallApricot + params.CallNewAccountGas,
diff --git a/core/vm/eips.go b/core/vm/eips.go
index e525a73c..8d377dcc 100644
--- a/core/vm/eips.go
+++ b/core/vm/eips.go
@@ -30,7 +30,7 @@ import (
        "fmt"
        "sort"
 
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"
        "github.com/holiman/uint256"
 )
 
@@ -166,8 +166,8 @@ func enableAP1(jt *JumpTable) {
 }
 
 func enableAP2(jt *JumpTable) {
-       jt[BALANCEMC] = nil
-       jt[CALLEX] = nil
+       jt[BALANCEMC] = &operation{execute: opUndefined, maxStack: maxStack(0, 0)}
+       jt[CALLEX] = &operation{execute: opUndefined, maxStack: maxStack(0, 0)}
 }
 
 // enable3198 applies EIP-3198 (BASEFEE Opcode)
diff --git a/core/vm/errors.go b/core/vm/errors.go
index 7f777dc3..5c81bb28 100644
--- a/core/vm/errors.go
```

```
+++ b/core/vm/errors.go
@@ -45,6 +45,12 @@ var (
        ErrReturnDataOutOfBounds    = errors.New("return data out of bounds")
        ErrGasUintOverflow          = errors.New("gas uint64 overflow")
        ErrInvalidCode              = errors.New("invalid code: must not begin with 0xef")
+       ErrNonceUintOverflow        = errors.New("nonce uint64 overflow")
+       ErrNoSenderBlackhole        = errors.New("blackhole address cannot be used as sender")
+
+       // errStopToken is an internal token indicating interpreter loop termination,
+       // never returned to outside callers.
+       errStopToken = errors.New("stop token")
 )
 
 // ErrStackUnderflow wraps an evm error when the items on the stack less
diff --git a/core/vm/evm.go b/core/vm/evm.go
index 823f1013..f5a20bbb 100644
--- a/core/vm/evm.go
+++ b/core/vm/evm.go
@@ -31,9 +31,9 @@ import (
        "sync/atomic"
        "time"
 
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/params"
        "github.com/holiman/uint256"
 )
 
@@ -44,10 +44,10 @@ var emptyCodeHash = crypto.Keccak256Hash(nil)
 type (
        // CanTransferFunc is the signature of a transfer guard function
        CanTransferFunc   func(StateDB, common.Address, *big.Int) bool
-       CanTransferMCFunc func(StateDB, common.Address, common.Address, *common.Hash, *big.Int) bool
+       CanTransferMCFunc func(StateDB, common.Address, common.Address, common.Hash, *big.Int) bool
        // TransferFunc is the signature of a transfer function
        TransferFunc   func(StateDB, common.Address, common.Address, *big.Int)
-       TransferMCFunc func(StateDB, common.Address, common.Address, *common.Hash, *big.Int)
+       TransferMCFunc func(StateDB, common.Address, common.Address, common.Hash, *big.Int)
        // GetHashFunc returns the n'th block hash in the blockchain
        // and is used by the BLOCKHASH EVM op code.
        GetHashFunc func(uint64) common.Hash
@@ -182,14 +182,14 @@ func (evm *EVM) Interpreter() *EVMInterpreter {
 // the necessary steps to create accounts and reverses the state in case of an
 // execution error or failed value transfer.
 func (evm *EVM) Call(caller ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, gas, nil
-       }
        // Fail if we're trying to execute above the call depth limit
        if evm.depth > int(params.CallCreateDepth) {
                return nil, gas, ErrDepth
        }
        // Fail if we're trying to transfer more than the available balance
+       // Note: it is not possible for a negative value to be passed in here due to the fact
+       // that [value] will be popped from the stack and decoded to a *big.Int, which will
+       // always yield a positive result.
        if value.Sign() != 0 && !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {
                return nil, gas, ErrInsufficientBalance
        }
@@ -264,16 +264,16 @@ func (evm *EVM) Call(caller ContractRef, addr common.Address, input []byte, gas
 }
 
 // This allows the user transfer balance of a specified coinId in addition to a normal Call().
-func (evm *EVM) CallExpert(caller ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int, coinID *common.Hash, value2 *big.Int) (ret []byte, leftOverGas uint64, err error) {
-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, gas, nil
-       }
+func (evm *EVM) CallExpert(caller ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int, coinID common.Hash, value2 *big.Int) (ret []byte, leftOverGas uint64, err error) {
        // Fail if we're trying to execute above the call depth limit
        if evm.depth > int(params.CallCreateDepth) {
                return nil, gas, ErrDepth
        }
 
        // Fail if we're trying to transfer more than the available balance
+       // Note: it is not possible for a negative value to be passed in here due to the fact
+       // that [value] will be popped from the stack and decoded to a *big.Int, which will
+       // always yield a positive result.
        if value.Sign() != 0 && !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {
                return nil, gas, ErrInsufficientBalance
        }
@@ -348,9 +348,6 @@ func (evm *EVM) CallExpert(caller ContractRef, addr common.Address, input []byte
 // CallCode differs from Call in the sense that it executes the given address'
 // code with the caller as context.
 func (evm *EVM) CallCode(caller ContractRef, addr common.Address, input []byte, gas uint64, value *big.Int) (ret []byte, leftOverGas uint64, err error) {
-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, gas, nil
-       }
        // Fail if we're trying to execute above the call depth limit
        if evm.depth > int(params.CallCreateDepth) {
                return nil, gas, ErrDepth
@@ -359,6 +356,9 @@ func (evm *EVM) CallCode(caller ContractRef, addr common.Address, input []byte,
        // Note although it's noop to transfer X ether to caller itself. But
        // if caller doesn't have enough balance, it would be an error to allow
        // over-charging itself. So the check here is necessary.
+       // Note: it is not possible for a negative value to be passed in here due to the fact
+       // that [value] will be popped from the stack and decoded to a *big.Int, which will
+       // always yield a positive result.
        if !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {
                return nil, gas, ErrInsufficientBalance
        }
@@ -399,9 +399,6 @@ func (evm *EVM) CallCode(caller ContractRef, addr common.Address, input []byte,
 // DelegateCall differs from CallCode in the sense that it executes the given address'
 // code with the caller as context and the caller is set to the caller of the caller.
 func (evm *EVM) DelegateCall(caller ContractRef, addr common.Address, input []byte, gas uint64) (ret []byte, leftOverGas uint64, err error) {
-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, gas, nil
-       }
        // Fail if we're trying to execute above the call depth limit
        if evm.depth > int(params.CallCreateDepth) {
                return nil, gas, ErrDepth
@@ -441,9 +438,6 @@ func (evm *EVM) DelegateCall(caller ContractRef, addr common.Address, input []by
 // Opcodes that attempt to perform such modifications will result in exceptions
 // instead of performing the modifications.
 func (evm *EVM) StaticCall(caller ContractRef, addr common.Address, input []byte, gas uint64) (ret []byte, leftOverGas uint64, err error) {
-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, gas, nil
-       }
        // Fail if we're trying to execute above the call depth limit
        if evm.depth > int(params.CallCreateDepth) {
                return nil, gas, ErrDepth
@@ -514,10 +508,21 @@ func (evm *EVM) create(caller ContractRef, codeAndHash *codeAndHash, gas uint64,
        if evm.depth > int(params.CallCreateDepth) {
                return nil, common.Address{}, gas, ErrDepth
        }
+       // Note: it is not possible for a negative value to be passed in here due to the fact
+       // that [value] will be popped from the stack and decoded to a *big.Int, which will
+       // always yield a positive result.
        if !evm.Context.CanTransfer(evm.StateDB, caller.Address(), value) {
                return nil, common.Address{}, gas, ErrInsufficientBalance
        }
+       // If there is any collision with the Blackhole address, return an error instead
+       // of allowing the contract to be created.
+       if address == evm.Context.Coinbase {
```

```
+               return nil, common.Address{}, gas, ErrNoSenderBlackhole
+       }
        nonce := evm.StateDB.GetNonce(caller.Address())
+       if nonce+1 < nonce {
+               return nil, common.Address{}, gas, ErrNonceUintOverflow
+       }
        evm.StateDB.SetNonce(caller.Address(), nonce+1)
        // We add this to the access list _before_ taking a snapshot. Even if the creation fails,
        // the access-list change should not be rolled back
@@ -542,10 +547,6 @@ func (evm *EVM) create(caller ContractRef, codeAndHash *codeAndHash, gas uint64,
        contract := NewContract(caller, AccountRef(address), value, gas)
        contract.SetCodeOptionalHash(&address, codeAndHash)

-       if evm.Config.NoRecursion && evm.depth > 0 {
-               return nil, address, gas, nil
-       }
-
        if evm.Config.Debug {
                if evm.depth == 0 {
                        evm.Config.Tracer.CaptureStart(evm, caller.Address(), address, true, codeAndHash.code, gas, value)
@@ -609,7 +610,7 @@ func (evm *EVM) Create(caller ContractRef, code []byte, gas uint64, value *big.I

 // Create2 creates a new contract using code as deployment code.
 //
-// The different between Create2 with Create is Create2 uses sha3(0xff ++ msg.sender ++ salt ++ sha3(init_code))[12:]
+// The different between Create2 with Create is Create2 uses keccak256(0xff ++ msg.sender ++ salt ++ keccak256(init_code))[12:]
 // instead of the usual sender-and-nonce-hash as the address where the contract is initialized at.
 func (evm *EVM) Create2(caller ContractRef, code []byte, gas uint64, endowment *big.Int, salt *uint256.Int) (ret []byte, contractAddr common.Address, leftOverGas uint64, err error) {
        codeAndHash := &codeAndHash{code: code}
diff --git a/core/vm/gas_table.go b/core/vm/gas_table.go
index fdabf1eb..0d80c5c0 100644
--- a/core/vm/gas_table.go
+++ b/core/vm/gas_table.go
@@ -29,9 +29,9 @@ package vm
 import (
        "errors"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/math"
+       "github.com/flare-foundation/coreth/params"
 )

 // memoryGasCost calculates the quadratic gas for memory expansion. It does so
@@ -292,7 +292,7 @@ func makeGasLog(n uint64) gasFunc {
        }
 }

-func gasSha3(evm *EVM, contract *Contract, stack *Stack, mem *Memory, memorySize uint64) (uint64, error) {
+func gasKeccak256(evm *EVM, contract *Contract, stack *Stack, mem *Memory, memorySize uint64) (uint64, error) {
        gas, err := memoryGasCost(mem, memorySize)
        if err != nil {
                return 0, err
@@ -301,7 +301,7 @@ func gasSha3(evm *EVM, contract *Contract, stack *Stack, mem *Memory, memorySize
        if overflow {
                return 0, ErrGasUintOverflow
        }
-       if wordGas, overflow = math.SafeMul(toWordSize(wordGas), params.Sha3WordGas); overflow {
+       if wordGas, overflow = math.SafeMul(toWordSize(wordGas), params.Keccak256WordGas); overflow {
                return 0, ErrGasUintOverflow
        }
        if gas, overflow = math.SafeAdd(gas, wordGas); overflow {
@@ -335,7 +335,7 @@ func gasCreate2(evm *EVM, contract *Contract, stack *Stack, mem *Memory, memoryS
        if overflow {
                return 0, ErrGasUintOverflow
        }
-       if wordGas, overflow = math.SafeMul(toWordSize(wordGas), params.Sha3WordGas); overflow {
+       if wordGas, overflow = math.SafeMul(toWordSize(wordGas), params.Keccak256WordGas); overflow {
                return 0, ErrGasUintOverflow
        }
        if gas, overflow = math.SafeAdd(gas, wordGas); overflow {
diff --git a/core/vm/gas_table_test.go b/core/vm/gas_table_test.go
index 92d5d301..fe6a928b 100644
--- a/core/vm/gas_table_test.go
+++ b/core/vm/gas_table_test.go
@@ -31,11 +31,11 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
 )

 func TestMemoryGasCost(t *testing.T) {
diff --git a/core/vm/instructions.go b/core/vm/instructions.go
index 35fb3a86..f30c1d9b 100644
--- a/core/vm/instructions.go
+++ b/core/vm/instructions.go
@@ -28,10 +28,11 @@ package vm

 import (
        "errors"
+       "sync/atomic"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
        "github.com/holiman/uint256"
        "golang.org/x/crypto/sha3"
 )
@@ -243,7 +244,7 @@ func opSAR(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte
        return nil, nil
 }

-func opSha3(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+func opKeccak256(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
        offset, size := scope.Stack.pop(), scope.Stack.peek()
        data := scope.Memory.GetPtr(int64(offset.Uint64()), int64(size.Uint64()))

@@ -537,6 +538,9 @@ func opSload(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]by
 }

 func opSstore(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if interpreter.readOnly {
+               return nil, ErrWriteProtection
+       }
        loc := scope.Stack.pop()
        val := scope.Stack.pop()
        interpreter.evm.StateDB.SetState(scope.Contract.Address(),
@@ -545,23 +549,27 @@ func opSstore(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]b
 }

 func opJump(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if atomic.LoadInt32(&interpreter.evm.abort) != 0 {
+               return nil, errStopToken
+       }
```

```diff
        pos := scope.Stack.pop()
        if !scope.Contract.validJumpdest(&pos) {
                return nil, ErrInvalidJump
        }
-       *pc = pos.Uint64()
+       *pc = pos.Uint64() - 1 // pc will be increased by the interpreter loop
        return nil, nil
 }

 func opJumpi(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if atomic.LoadInt32(&interpreter.evm.abort) != 0 {
+               return nil, errStopToken
+       }
        pos, cond := scope.Stack.pop(), scope.Stack.pop()
        if !cond.IsZero() {
                if !scope.Contract.validJumpdest(&pos) {
                        return nil, ErrInvalidJump
                }
-               *pc = pos.Uint64()
-       } else {
-               *pc++
+               *pc = pos.Uint64() - 1 // pc will be increased by the interpreter loop
        }
        return nil, nil
 }
@@ -586,6 +594,9 @@ func opGas(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte
 }

 func opCreate(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if interpreter.readOnly {
+               return nil, ErrWriteProtection
+       }
        var (
                value        = scope.Stack.pop()
                offset, size = scope.Stack.pop(), scope.Stack.pop()
@@ -621,12 +632,17 @@ func opCreate(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]b
        scope.Contract.Gas += returnGas

        if suberr == ErrExecutionReverted {
+               interpreter.returnData = res // set REVERT data to return data buffer
                return res, nil
        }
+       interpreter.returnData = nil // clear dirty return data buffer
        return nil, nil
 }

 func opCreate2(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if interpreter.readOnly {
+               return nil, ErrWriteProtection
+       }
        var (
                endowment    = scope.Stack.pop()
                offset, size = scope.Stack.pop(), scope.Stack.pop()
@@ -657,8 +673,10 @@ func opCreate2(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]
        scope.Contract.Gas += returnGas

        if suberr == ErrExecutionReverted {
+               interpreter.returnData = res // set REVERT data to return data buffer
                return res, nil
        }
+       interpreter.returnData = nil // clear dirty return data buffer
        return nil, nil
 }

@@ -674,6 +692,9 @@ func opCall(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byt
        // Get the arguments from the memory.
        args := scope.Memory.GetPtr(int64(inOffset.Uint64()), int64(inSize.Uint64()))

+       if interpreter.readOnly && !value.IsZero() {
+               return nil, ErrWriteProtection
+       }
        var bigVal = big0
        //TODO: use uint256.Int instead of converting with toBig()
        // By using big0 here, we save an alloc for the most common case (non-ether-transferring contract calls),
@@ -697,6 +718,7 @@ func opCall(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byt
        }
        scope.Contract.Gas += returnGas

+       interpreter.returnData = ret
        return ret, nil
 }

@@ -713,6 +735,9 @@ func opCallExpert(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext)
        // Get the arguments from the memory.
        args := scope.Memory.GetPtr(int64(inOffset.Uint64()), int64(inSize.Uint64()))

+       if interpreter.readOnly && !value.IsZero() {
+               return nil, ErrWriteProtection
+       }
        var bigVal = big0
        //TODO: use uint256.Int instead of converting with toBig()
        // By using big0 here, we save an alloc for the most common case (non-ether-transferring contract calls),
@@ -730,7 +755,7 @@ func opCallExpert(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext)
                bigVal2 = value2.ToBig()
        }

-       ret, returnGas, err := interpreter.evm.CallExpert(scope.Contract, toAddr, args, gas, bigVal, &coinID, bigVal2)
+       ret, returnGas, err := interpreter.evm.CallExpert(scope.Contract, toAddr, args, gas, bigVal, coinID, bigVal2)

        if err != nil {
                temp.Clear()
@@ -744,6 +769,7 @@ func opCallExpert(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext)
        }
        scope.Contract.Gas += returnGas

+       interpreter.returnData = ret
        return ret, nil
 }
 func opCallCode(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
@@ -778,6 +804,7 @@ func opCallCode(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([
        }
        scope.Contract.Gas += returnGas

+       interpreter.returnData = ret
        return ret, nil
 }

@@ -806,6 +833,7 @@ func opDelegateCall(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext
        }
        scope.Contract.Gas += returnGas

+       interpreter.returnData = ret
        return ret, nil
 }

@@ -834,6 +862,7 @@ func opStaticCall(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext)
        }
        scope.Contract.Gas += returnGas

+       interpreter.returnData = ret
        return ret, nil
 }

@@ -841,21 +870,29 @@ func opReturn(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]b
        offset, size := scope.Stack.pop(), scope.Stack.pop()
        ret := scope.Memory.GetPtr(int64(offset.Uint64()), int64(size.Uint64()))

-       return ret, nil
+       return ret, errStopToken
 }

 func opRevert(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
        offset, size := scope.Stack.pop(), scope.Stack.pop()
        ret := scope.Memory.GetPtr(int64(offset.Uint64()), int64(size.Uint64()))

-       return ret, nil
+       interpreter.returnData = ret
+       return ret, ErrExecutionReverted
+}
+
+func opUndefined(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       return nil, &ErrInvalidOpCode{opcode: OpCode(scope.Contract.Code[*pc])}
 }

 func opStop(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
-       return nil, nil
+       return nil, errStopToken
 }

-func opSuicide(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+func opSelfdestruct(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+       if interpreter.readOnly {
+               return nil, ErrWriteProtection
+       }
        beneficiary := scope.Stack.pop()
        balance := interpreter.evm.StateDB.GetBalance(scope.Contract.Address())
        interpreter.evm.StateDB.AddBalance(beneficiary.Bytes20(), balance)
@@ -864,7 +901,7 @@ func opSuicide(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]
                interpreter.cfg.Tracer.CaptureEnter(SELFDESTRUCT, scope.Contract.Address(), beneficiary.Bytes20(), []byte{}, 0, balance)
                interpreter.cfg.Tracer.CaptureExit([]byte{}, 0, nil)
        }
-       return nil, nil
+       return nil, errStopToken
 }

 // following functions are used by the instruction jump  table
@@ -872,6 +909,9 @@ func opSuicide(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]
 // make log instruction function
 func makeLog(size int) executionFunc {
        return func(pc *uint64, interpreter *EVMInterpreter, scope *ScopeContext) ([]byte, error) {
+               if interpreter.readOnly {
+                       return nil, ErrWriteProtection
+               }
                topics := make([]common.Hash, size)
                stack := scope.Stack
                mStart, mSize := stack.pop(), stack.pop()
diff --git a/core/vm/instructions_test.go b/core/vm/instructions_test.go
index 1570b36f..5cde24ea 100644
--- a/core/vm/instructions_test.go
+++ b/core/vm/instructions_test.go
@@ -33,9 +33,9 @@ import (
        "io/ioutil"
        "testing"

-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/params"
        "github.com/holiman/uint256"
 )

@@ -535,12 +535,14 @@ func TestOpMstore(t *testing.T) {
        mem.Resize(64)
        pc := uint64(0)
        v := "abcdef00000000000000abba000000000deaf000000c0de00100000000133700"
-       stack.pushN(*new(uint256.Int).SetBytes(common.Hex2Bytes(v)), *new(uint256.Int))
+       stack.push(new(uint256.Int).SetBytes(common.Hex2Bytes(v)))
+       stack.push(new(uint256.Int))
        opMstore(&pc, evmInterpreter, &ScopeContext{mem, stack, nil})
        if got := common.Bytes2Hex(mem.GetCopy(0, 32)); got != v {
                t.Fatalf("Mstore fail, got %v, expected %v", got, v)
        }
-       stack.pushN(*new(uint256.Int).SetUint64(0x1), *new(uint256.Int))
+       stack.push(new(uint256.Int).SetUint64(0x1))
+       stack.push(new(uint256.Int))
        opMstore(&pc, evmInterpreter, &ScopeContext{mem, stack, nil})
        if common.Bytes2Hex(mem.GetCopy(0, 32)) != "0000000000000000000000000000000000000000000000000000000000000001" {
                t.Fatalf("Mstore failed to overwrite previous value")
@@ -563,12 +565,13 @@ func BenchmarkOpMstore(bench *testing.B) {

        bench.ResetTimer()
        for i := 0; i < bench.N; i++ {
-               stack.pushN(*value, *memStart)
+               stack.push(value)
+               stack.push(memStart)
                opMstore(&pc, evmInterpreter, &ScopeContext{mem, stack, nil})
        }
 }

-func BenchmarkOpSHA3(bench *testing.B) {
+func BenchmarkOpKeccak256(bench *testing.B) {
        var (
                env            = NewEVM(BlockContext{}, TxContext{}, nil, params.TestChainConfig, Config{})
                stack          = newstack()
@@ -582,8 +585,9 @@ func BenchmarkOpSHA3(bench *testing.B) {

        bench.ResetTimer()
        for i := 0; i < bench.N; i++ {
-               stack.pushN(*uint256.NewInt(32), *start)
-               opSha3(&pc, evmInterpreter, &ScopeContext{mem, stack, nil})
+               stack.push(uint256.NewInt(32))
+               stack.push(start)
+               opKeccak256(&pc, evmInterpreter, &ScopeContext{mem, stack, nil})
        }
 }

diff --git a/core/vm/interface.go b/core/vm/interface.go
index bde4b08e..e5acbfbf 100644
--- a/core/vm/interface.go
+++ b/core/vm/interface.go
@@ -29,8 +29,8 @@ package vm
 import (
        "math/big"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // StateDB is an EVM database for full state querying.
diff --git a/core/vm/interpreter.go b/core/vm/interpreter.go
index 805aea61..653d64ca 100644
--- a/core/vm/interpreter.go
+++ b/core/vm/interpreter.go
@@ -28,7 +28,6 @@ package vm

 import (
        "hash"
```

```
-       "sync/atomic"

        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/math"
@@ -46,11 +45,10 @@ var (
 type Config struct {
        Debug                   bool      // Enables debugging
        Tracer                  EVMLogger // Opcode logger
-       NoRecursion             bool      // Disables call, callcode, delegate call and create
        NoBaseFee               bool      // Forces the EIP-1559 baseFee to 0 (needed for 0 price calls)
        EnablePreimageRecording bool      // Enables recording of SHA3/keccak preimages

-       JumpTable [256]*operation // EVM instruction table, automatically populated if unset
+       JumpTable *JumpTable // EVM instruction table, automatically populated if unset

        ExtraEips []int // Additional EIPS that are to be enabled

@@ -88,41 +86,39 @@ type EVMInterpreter struct {

 // NewEVMInterpreter returns a new instance of the Interpreter.
 func NewEVMInterpreter(evm *EVM, cfg Config) *EVMInterpreter {
-       // We use the STOP instruction whether to see
-       // the jump table was initialised. If it was not
-       // we'll set the default jump table.
-       if cfg.JumpTable[STOP] == nil {
-               var jt JumpTable
+       // If jump table was not initialised we set the default one.
+       if cfg.JumpTable == nil {
                switch {
                case evm.chainRules.IsApricotPhase3:
-                       jt = apricotPhase3InstructionSet
+                       cfg.JumpTable = &apricotPhase3InstructionSet
                case evm.chainRules.IsApricotPhase2:
-                       jt = apricotPhase2InstructionSet
+                       cfg.JumpTable = &apricotPhase2InstructionSet
                case evm.chainRules.IsApricotPhase1:
-                       jt = apricotPhase1InstructionSet
+                       cfg.JumpTable = &apricotPhase1InstructionSet
                case evm.chainRules.IsIstanbul:
-                       jt = istanbulInstructionSet
+                       cfg.JumpTable = &istanbulInstructionSet
                case evm.chainRules.IsConstantinople:
-                       jt = constantinopleInstructionSet
+                       cfg.JumpTable = &constantinopleInstructionSet
                case evm.chainRules.IsByzantium:
-                       jt = byzantiumInstructionSet
+                       cfg.JumpTable = &byzantiumInstructionSet
                case evm.chainRules.IsEIP158:
-                       jt = spuriousDragonInstructionSet
+                       cfg.JumpTable = &spuriousDragonInstructionSet
                case evm.chainRules.IsEIP150:
-                       jt = tangerineWhistleInstructionSet
+                       cfg.JumpTable = &tangerineWhistleInstructionSet
                case evm.chainRules.IsHomestead:
-                       jt = homesteadInstructionSet
+                       cfg.JumpTable = &homesteadInstructionSet
                default:
-                       jt = frontierInstructionSet
+                       cfg.JumpTable = &frontierInstructionSet
                }
                for i, eip := range cfg.ExtraEips {
-                       if err := EnableEIP(eip, &jt); err != nil {
+                       copy := *cfg.JumpTable
+                       if err := EnableEIP(eip, &copy); err != nil {
                                // Disable it, so caller can check if it's activated or not
                                cfg.ExtraEips = append(cfg.ExtraEips[:i], cfg.ExtraEips[i+1:]...)
                                log.Error("EIP activation failed", "eip", eip, "error", err)
                        }
+                       cfg.JumpTable = &copy
                }
-               cfg.JumpTable = jt
        }

        return &EVMInterpreter{
@@ -138,7 +134,11 @@ func NewEVMInterpreter(evm *EVM, cfg Config) *EVMInterpreter {
 // considered a revert-and-consume-all-gas operation except for
 // ErrExecutionReverted which means revert-and-keep-gas-left.
 func (in *EVMInterpreter) Run(contract *Contract, input []byte, readOnly bool) (ret []byte, err error) {
-       if contract.Address() == BuiltinAddr {
+       // Deprecate special handling of [BuiltinAddr] as of ApricotPhase2.
+       // In ApricotPhase2, the contract deployed in the genesis is overridden by a deprecated precompiled
+       // contract which will return an error immediately if its ever called. Therefore, this function should
+       // never be called after ApricotPhase2 with [BuiltinAddr] as the contract address.
+       if !in.evm.chainRules.IsApricotPhase2 && contract.Address() == BuiltinAddr {
                self := AccountRef(contract.Caller())
                if _, ok := contract.caller.(*Contract); ok {
                        contract = contract.AsDelegate()
@@ -201,9 +201,9 @@ func (in *EVMInterpreter) Run(contract *Contract, input []byte, readOnly bool) (
                defer func() {
                        if err != nil {
                                if !logged {
-                                       in.cfg.Tracer.CaptureState(in.evm, pcCopy, op, gasCopy, cost, callContext, in.returnData, in.evm.depth, err)
+                                       in.cfg.Tracer.CaptureState(pcCopy, op, gasCopy, cost, callContext, in.returnData, in.evm.depth, err)
                                } else {
-                                       in.cfg.Tracer.CaptureFault(in.evm, pcCopy, op, gasCopy, cost, callContext, in.evm.depth, err)
+                                       in.cfg.Tracer.CaptureFault(pcCopy, op, gasCopy, cost, callContext, in.evm.depth, err)
                                }
                        }
                }()
@@ -212,101 +212,71 @@ func (in *EVMInterpreter) Run(contract *Contract, input []byte, readOnly bool) (
        // explicit STOP, RETURN or SELFDESTRUCT is executed, an error occurred during
        // the execution of one of the operations or until the done flag is set by the
        // parent context.
-       steps := 0
        for {
-               steps++
-               if steps%1000 == 0 && atomic.LoadInt32(&in.evm.abort) != 0 {
-                       break
-               }
                if in.cfg.Debug {
                        // Capture pre-execution values for tracing.
                        logged, pcCopy, gasCopy = false, pc, contract.Gas
                }
-
                // Get the operation from the jump table and validate the stack to ensure there are
                // enough stack items available to perform the operation.
                op = contract.GetOp(pc)
                operation := in.cfg.JumpTable[op]
-               if operation == nil {
-                       return nil, &ErrInvalidOpCode{opcode: op}
-               }
+               cost = operation.constantGas // For tracing
                // Validate stack
                if sLen := stack.len(); sLen < operation.minStack {
                        return nil, &ErrStackUnderflow{stackLen: sLen, required: operation.minStack}
                } else if sLen > operation.maxStack {
                        return nil, &ErrStackOverflow{stackLen: sLen, limit: operation.maxStack}
                }
-               // If the operation is valid, enforce write restrictions
-               if in.readOnly && in.evm.chainRules.IsByzantium {
-                       // If the interpreter is operating in readonly mode, make sure no
-                       // state-modifying operation is performed. The 3rd stack item
-                       // for a call operation is the value. Transferring value from one
-                       // account to the others means the state is modified and should also
-                       // return with an error.
-                       if operation.writes || ((op == CALL || op == CALLEX) && stack.Back(2).Sign() != 0) {
-                               return nil, ErrWriteProtection
-                       }
-               }
-               // Static portion of gas
-               cost = operation.constantGas // For tracing
-               if !contract.UseGas(operation.constantGas) {
+               if !contract.UseGas(cost) {
                        return nil, ErrOutOfGas
                }

-               var memorySize uint64
-               // calculate the new memory size and expand the memory to fit
-               // the operation
-               // Memory check needs to be done prior to evaluating the dynamic gas portion,
-               // to detect calculation overflows
-               if operation.memorySize != nil {
-                       memSize, overflow := operation.memorySize(stack)
-                       if overflow {
-                               return nil, ErrGasUintOverflow
-                       }
-                       // memory is expanded in words of 32 bytes. Gas
-                       // is also calculated in words.
-                       if memorySize, overflow = math.SafeMul(toWordSize(memSize), 32); overflow {
-                               return nil, ErrGasUintOverflow
-                       }
-               }
-               // Dynamic portion of gas
-               // consume the gas and return an error if not enough gas is available.
-               // cost is explicitly set so that the capture state defer method can get the proper cost
                if operation.dynamicGas != nil {
+                       // All ops with a dynamic memory usage also has a dynamic gas cost.
+                       var memorySize uint64
+                       // calculate the new memory size and expand the memory to fit
+                       // the operation
+                       // Memory check needs to be done prior to evaluating the dynamic gas portion,
+                       // to detect calculation overflows
+                       if operation.memorySize != nil {
+                               memSize, overflow := operation.memorySize(stack)
+                               if overflow {
+                                       return nil, ErrGasUintOverflow
+                               }
+                               // memory is expanded in words of 32 bytes. Gas
+                               // is also calculated in words.
+                               if memorySize, overflow = math.SafeMul(toWordSize(memSize), 32); overflow {
+                                       return nil, ErrGasUintOverflow
+                               }
+                       }
+                       // Consume the gas and return an error if not enough gas is available.
+                       // cost is explicitly set so that the capture state defer method can get the proper cost
                        var dynamicCost uint64
                        dynamicCost, err = operation.dynamicGas(in.evm, contract, stack, mem, memorySize)
-                       cost += dynamicCost // total cost, for debug tracing
+                       cost += dynamicCost // for tracing
                        if err != nil || !contract.UseGas(dynamicCost) {
                                return nil, ErrOutOfGas
                        }
+                       if memorySize > 0 {
+                               mem.Resize(memorySize)
+                       }
                }
-               if memorySize > 0 {
-                       mem.Resize(memorySize)
-               }
-
                if in.cfg.Debug {
-                       in.cfg.Tracer.CaptureState(in.evm, pc, op, gasCopy, cost, callContext, in.returnData, in.evm.depth, err)
+                       in.cfg.Tracer.CaptureState(pc, op, gasCopy, cost, callContext, in.returnData, in.evm.depth, err)
                        logged = true
                }

                // execute the operation
                res, err = operation.execute(&pc, in, callContext)
-               // if the operation clears the return data (e.g. it has returning data)
-               // set the last return to the result of the operation.
-               if operation.returns {
-                       in.returnData = res
-               }
-
-               switch {
-               case err != nil:
-                       return nil, err
-               case operation.reverts:
-                       return res, ErrExecutionReverted
-               case operation.halts:
-                       return res, nil
-               case !operation.jumps:
-                       pc++
+               if err != nil {
+                       break
                }
+               pc++
        }
-       return nil, nil
+       if err == errStopToken {
+               err = nil // clear stop token error
+       }
+
+       return res, err
 }
diff --git a/core/vm/interpreter_test.go b/core/vm/interpreter_test.go
new file mode 100644
index 00000000..770f3f67
--- /dev/null
+++ b/core/vm/interpreter_test.go
@@ -0,0 +1,87 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package vm
```

```diff
+
+import (
+       "math/big"
+       "testing"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/math"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
+)
+
+var loopInterruptTests = []string{
+       // infinite loop using JUMP: push(2) jumpdest dup1 jump
+       "60025b8056",
+       // infinite loop using JUMPI: push(1) push(4) jumpdest dup2 dup2 jumpi
+       "600160045b818157",
+}
+
+func TestLoopInterrupt(t *testing.T) {
+       address := common.BytesToAddress([]byte("contract"))
+       vmctx := BlockContext{
+               Transfer: func(StateDB, common.Address, common.Address, *big.Int) {},
+       }
+
+       for i, tt := range loopInterruptTests {
+               statedb, _ := state.New(common.Hash{}, state.NewDatabase(rawdb.NewMemoryDatabase()), nil)
+               statedb.CreateAccount(address)
+               statedb.SetCode(address, common.Hex2Bytes(tt))
+               statedb.Finalise(true)
+
+               evm := NewEVM(vmctx, TxContext{}, statedb, params.TestChainConfig, Config{})
+
+               errChannel := make(chan error)
+               timeout := make(chan bool)
+
+               go func(evm *EVM) {
+                       _, _, err := evm.Call(AccountRef(common.Address{}), address, nil, math.MaxUint64, new(big.Int))
+                       errChannel <- err
+               }(evm)
+
+               go func() {
+                       <-time.After(time.Second)
+                       timeout <- true
+               }()
+
+               evm.Cancel()
+
+               select {
+               case <-timeout:
+                       t.Errorf("test %d timed out", i)
+               case err := <-errChannel:
+                       if err != nil {
+                               t.Errorf("test %d failure: %v", i, err)
+                       }
+               }
+       }
+
+}
diff --git a/core/vm/jump_table.go b/core/vm/jump_table.go
index d833e523..2d250301 100644
--- a/core/vm/jump_table.go
+++ b/core/vm/jump_table.go
@@ -27,7 +27,9 @@
 package vm
 
 import (
-       "github.com/ava-labs/coreth/params"
+       "fmt"
+
+       "github.com/flare-foundation/coreth/params"
 )
 
 type (
@@ -50,12 +52,6 @@ type operation struct {
 
        // memorySize returns the memory size required for the operation
        memorySize memorySizeFunc
-
-       halts   bool // indicates whether the operation should halt further execution
-       jumps   bool // indicates whether the program counter should not increment
-       writes  bool // determines whether this a state modifying operation
-       reverts bool // determines whether the operation reverts state (implicitly halts)
-       returns bool // determines whether the operations sets the return data content
 }
 
 var (
@@ -74,12 +70,30 @@ var (
 // JumpTable contains the EVM opcodes supported at a given fork.
 type JumpTable [256]*operation
 
+func validate(jt JumpTable) JumpTable {
+       for i, op := range jt {
+               if op == nil {
+                       panic(fmt.Sprintf("op 0x%x is not set", i))
+               }
+               // The interpreter has an assumption that if the memorySize function is
+               // set, then the dynamicGas function is also set. This is a somewhat
+               // arbitrary assumption, and can be removed if we need to -- but it
+               // allows us to avoid a condition check. As long as we have that assumption
+               // in there, this little sanity check prevents us from merging in a
+               // change which violates it.
+               if op.memorySize != nil && op.dynamicGas == nil {
+                       panic(fmt.Sprintf("op %v has dynamic memory but not dynamic gas", OpCode(i).String()))
+               }
+       }
+       return jt
+}
+
 // newApricotPhase3InstructionSet returns the frontier, homestead, byzantium,
 // contantinople, istanbul, petersburg, apricotPhase1, 2, and 3 instructions.
 func newApricotPhase3InstructionSet() JumpTable {
        instructionSet := newApricotPhase2InstructionSet()
        enable3198(&instructionSet) // Base fee opcode https://eips.ethereum.org/EIPS/eip-3198
-       return instructionSet
+       return validate(instructionSet)
 }
 
 // newApricotPhase1InstructionSet returns the frontier,
@@ -91,7 +105,7 @@ func newApricotPhase2InstructionSet() JumpTable {
        enable2929(&instructionSet)
        enableAP2(&instructionSet)
 
-       return instructionSet
+       return validate(instructionSet)
 }
 
 // newApricotPhase1InstructionSet returns the frontier,
@@ -102,7 +116,7 @@ func newApricotPhase1InstructionSet() JumpTable {
 
        enableAP1(&instructionSet)
 
-       return instructionSet
```

```
+       return validate(instructionSet)
 }

 // newIstanbulInstructionSet returns the frontier,
@@ -114,7 +128,7 @@ func newIstanbulInstructionSet() JumpTable {
        enable1884(&instructionSet) // Reprice reader opcodes - https://eips.ethereum.org/EIPS/eip-1884
        enable2200(&instructionSet) // Net metered SSTORE - https://eips.ethereum.org/EIPS/eip-2200

-       return instructionSet
+       return validate(instructionSet)
 }

 // newConstantinopleInstructionSet returns the frontier, homestead,
@@ -152,10 +166,8 @@ func newConstantinopleInstructionSet() JumpTable {
                minStack:    minStack(4, 1),
                maxStack:    maxStack(4, 1),
                memorySize:  memoryCreate2,
-               writes:      true,
-               returns:     true,
        }
-       return instructionSet
+       return validate(instructionSet)
 }

 // newByzantiumInstructionSet returns the frontier, homestead and
@@ -169,7 +181,6 @@ func newByzantiumInstructionSet() JumpTable {
                minStack:    minStack(6, 1),
                maxStack:    maxStack(6, 1),
                memorySize:  memoryStaticCall,
-               returns:     true,
        }
        instructionSet[RETURNDATASIZE] = &operation{
                execute:     opReturnDataSize,
@@ -191,17 +202,15 @@ func newByzantiumInstructionSet() JumpTable {
                minStack:   minStack(2, 0),
                maxStack:   maxStack(2, 0),
                memorySize: memoryRevert,
-               reverts:    true,
-               returns:    true,
        }
-       return instructionSet
+       return validate(instructionSet)
 }

 // EIP 158 a.k.a Spurious Dragon
 func newSpuriousDragonInstructionSet() JumpTable {
        instructionSet := newTangerineWhistleInstructionSet()
        instructionSet[EXP].dynamicGas = gasExpEIP158
-       return instructionSet
+       return validate(instructionSet)

 }

@@ -216,7 +225,7 @@ func newTangerineWhistleInstructionSet() JumpTable {
        instructionSet[CALLEX].constantGas = params.CallGasEIP150
        instructionSet[CALLCODE].constantGas = params.CallGasEIP150
        instructionSet[DELEGATECALL].constantGas = params.CallGasEIP150
-       return instructionSet
+       return validate(instructionSet)
 }

 // newHomesteadInstructionSet returns the frontier and homestead
@@ -230,21 +239,19 @@ func newHomesteadInstructionSet() JumpTable {
                minStack:    minStack(6, 1),
                maxStack:    maxStack(6, 1),
                memorySize:  memoryDelegateCall,
-               returns:     true,
        }
-       return instructionSet
+       return validate(instructionSet)
 }

 // newFrontierInstructionSet returns the frontier instructions
 // that can be executed during the frontier phase.
 func newFrontierInstructionSet() JumpTable {
-       return JumpTable{
+       tbl := JumpTable{
                STOP: {
                        execute:     opStop,
                        constantGas: 0,
                        minStack:    minStack(0, 0),
                        maxStack:    maxStack(0, 0),
-                       halts:       true,
                },
                ADD: {
                        execute:     opAdd,
@@ -378,13 +385,13 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:    minStack(2, 1),
                        maxStack:    maxStack(2, 1),
                },
-               SHA3: {
-                       execute:     opSha3,
-                       constantGas: params.Sha3Gas,
-                       dynamicGas:  gasSha3,
+               KECCAK256: {
+                       execute:     opKeccak256,
+                       constantGas: params.Keccak256Gas,
+                       dynamicGas:  gasKeccak256,
                        minStack:    minStack(2, 1),
                        maxStack:    maxStack(2, 1),
-                       memorySize:  memorySha3,
+                       memorySize:  memoryKeccak256,
                },
                ADDRESS: {
                        execute:     opAddress,
@@ -553,21 +560,18 @@ func newFrontierInstructionSet() JumpTable {
                        dynamicGas: gasSStore,
                        minStack:   minStack(2, 0),
                        maxStack:   maxStack(2, 0),
-                       writes:     true,
                },
                JUMP: {
                        execute:     opJump,
                        constantGas: GasMidStep,
                        minStack:    minStack(1, 0),
                        maxStack:    maxStack(1, 0),
-                       jumps:       true,
                },
                JUMPI: {
                        execute:     opJumpi,
                        constantGas: GasSlowStep,
                        minStack:    minStack(2, 0),
                        maxStack:    maxStack(2, 0),
-                       jumps:       true,
                },
                PC: {
                        execute:     opPc,
@@ -983,7 +987,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(2, 0),
                        maxStack:   maxStack(2, 0),
                        memorySize: memoryLog,
-                       writes:     true,
                },
                LOG1: {
                        execute:    makeLog(1),
@@ -991,7 +994,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(3, 0),
                        maxStack:   maxStack(3, 0),
                        memorySize: memoryLog,
-                       writes:     true,
                },
                LOG2: {
                        execute:    makeLog(2),
@@ -999,7 +1001,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(4, 0),
                        maxStack:   maxStack(4, 0),
                        memorySize: memoryLog,
-                       writes:     true,
                },
                LOG3: {
                        execute:    makeLog(3),
@@ -1007,7 +1008,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(5, 0),
                        maxStack:   maxStack(5, 0),
                        memorySize: memoryLog,
-                       writes:     true,
                },
                LOG4: {
                        execute:    makeLog(4),
@@ -1015,7 +1015,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(6, 0),
                        maxStack:   maxStack(6, 0),
                        memorySize: memoryLog,
-                       writes:     true,
                },
                CREATE: {
                        execute:     opCreate,
@@ -1024,8 +1023,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:    minStack(3, 1),
                        maxStack:    maxStack(3, 1),
                        memorySize:  memoryCreate,
-                       writes:      true,
-                       returns:     true,
                },
                CALL: {
                        execute:     opCall,
@@ -1034,7 +1031,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:    minStack(7, 1),
                        maxStack:    maxStack(7, 1),
                        memorySize:  memoryCall,
-                       returns:     true,
                },
                CALLEX: {
                        execute:     opCallExpert,
@@ -1043,7 +1039,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:    minStack(9, 1),
                        maxStack:    maxStack(9, 1),
                        memorySize:  memoryCallExpert,
-                       returns:     true,
                },
                CALLCODE: {
                        execute:     opCallCode,
@@ -1052,7 +1047,6 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:    minStack(7, 1),
                        maxStack:    maxStack(7, 1),
                        memorySize:  memoryCall,
-                       returns:     true,
                },
                RETURN: {
                        execute:    opReturn,
@@ -1060,15 +1054,21 @@ func newFrontierInstructionSet() JumpTable {
                        minStack:   minStack(2, 0),
                        maxStack:   maxStack(2, 0),
                        memorySize: memoryReturn,
-                       halts:      true,
                },
                SELFDESTRUCT: {
-                       execute:    opSuicide,
+                       execute:    opSelfdestruct,
                        dynamicGas: gasSelfdestruct,
                        minStack:   minStack(1, 0),
                        maxStack:   maxStack(1, 0),
-                       halts:      true,
-                       writes:     true,
                },
        }
+
+       // Fill all unassigned slots with opUndefined.
+       for i, entry := range tbl {
+               if entry == nil {
+                       tbl[i] = &operation{execute: opUndefined, maxStack: maxStack(0, 0)}
+               }
+       }
+
+       return validate(tbl)
 }
diff --git a/core/vm/logger.go b/core/vm/logger.go
index ff11a6f5..ba962f77 100644
--- a/core/vm/logger.go
+++ b/core/vm/logger.go
@@ -27,87 +27,12 @@
 package vm

 import (
-       "encoding/hex"
-       "fmt"
-       "io"
        "math/big"
-       "strings"
        "time"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
-       "github.com/ethereum/go-ethereum/common/hexutil"
-       "github.com/ethereum/go-ethereum/common/math"
-       "github.com/holiman/uint256"
 )

-// Storage represents a contract's storage.
-type Storage map[common.Hash]common.Hash
-
-// Copy duplicates the current storage.
-func (s Storage) Copy() Storage {
-       cpy := make(Storage)
-       for key, value := range s {
-               cpy[key] = value
-       }
-       return cpy
-}
-
-// LogConfig are the configuration options for structured logger the EVM
-type LogConfig struct {
-       EnableMemory     bool // enable memory capture
-       DisableStack     bool // disable stack capture
-       DisableStorage   bool // disable storage capture
-       EnableReturnData bool // enable return data capture
-       Debug            bool // print output during capture end
```

```
-	Limit            int  // maximum length of output, but zero means unlimited
-	// Chain overrides, can be used to execute a trace using future fork rules
-	Overrides *params.ChainConfig `json:"overrides,omitempty"`
-}
-
-//go:generate gencodec -type StructLog -field-override structLogMarshaling -out gen_structlog.go
-
-// StructLog is emitted to the EVM each cycle and lists information about the current internal state
-// prior to the execution of the statement.
-type StructLog struct {
-	Pc            uint64                      `json:"pc"`
-	Op            OpCode                      `json:"op"`
-	Gas           uint64                      `json:"gas"`
-	GasCost       uint64                      `json:"gasCost"`
-	Memory        []byte                      `json:"memory"`
-	MemorySize    int                         `json:"memSize"`
-	Stack         []uint256.Int               `json:"stack"`
-	ReturnData    []byte                      `json:"returnData"`
-	Storage       map[common.Hash]common.Hash `json:"-"`
-	Depth         int                         `json:"depth"`
-	RefundCounter uint64                      `json:"refund"`
-	Err           error                       `json:"-"`
-}
-
-// overrides for gencodec
-type structLogMarshaling struct {
-	Gas         math.HexOrDecimal64
-	GasCost     math.HexOrDecimal64
-	Memory      hexutil.Bytes
-	ReturnData  hexutil.Bytes
-	OpName      string `json:"opName"` // adds call to OpName() in MarshalJSON
-	ErrorString string `json:"error"`  // adds call to ErrorString() in MarshalJSON
-}
-
-// OpName formats the operand name in a human-readable format.
-func (s *StructLog) OpName() string {
-	return s.Op.String()
-}
-
-// ErrorString formats the log's error as a string.
-func (s *StructLog) ErrorString() string {
-	if s.Err != nil {
-		return s.Err.Error()
-	}
-	return ""
-}
-
 // EVMLogger is used to collect execution traces from an EVM transaction
 // execution. CaptureState is called for each step of the VM with the
 // current VM state.
@@ -115,252 +40,9 @@ func (s *StructLog) ErrorString() string {
 // if you need to retain them beyond the current call.
 type EVMLogger interface {
 	CaptureStart(env *EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int)
-	CaptureState(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, rData []byte, depth int, err error)
+	CaptureState(pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, rData []byte, depth int, err error)
 	CaptureEnter(typ OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int)
 	CaptureExit(output []byte, gasUsed uint64, err error)
-	CaptureFault(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, depth int, err error)
+	CaptureFault(pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, depth int, err error)
 	CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error)
 }
-
-// StructLogger is an EVM state logger and implements EVMLogger.
-//
-// StructLogger can capture state based on the given Log configuration and also keeps
-// a track record of modified storage which is used in reporting snapshots of the
-// contract their storage.
-type StructLogger struct {
-	cfg LogConfig
-
-	storage map[common.Address]Storage
-	logs    []StructLog
-	output  []byte
-	err     error
-}
-
-// NewStructLogger returns a new logger
-func NewStructLogger(cfg *LogConfig) *StructLogger {
-	logger := &StructLogger{
-		storage: make(map[common.Address]Storage),
-	}
-	if cfg != nil {
-		logger.cfg = *cfg
-	}
-	return logger
-}
-
-// Reset clears the data held by the logger.
-func (l *StructLogger) Reset() {
-	l.storage = make(map[common.Address]Storage)
-	l.output = make([]byte, 0)
-	l.logs = l.logs[:0]
-	l.err = nil
-}
-
-// CaptureStart implements the EVMLogger interface to initialize the tracing operation.
-func (l *StructLogger) CaptureStart(env *EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
-}
-
-// CaptureState logs a new structured log message and pushes it out to the environment
-//
-// CaptureState also tracks SLOAD/SSTORE ops to track storage change.
-func (l *StructLogger) CaptureState(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, rData []byte, depth int, err error) {
-	memory := scope.Memory
-	stack := scope.Stack
-	contract := scope.Contract
-	// check if already accumulated the specified number of logs
-	if l.cfg.Limit != 0 && l.cfg.Limit <= len(l.logs) {
-		return
-	}
-	// Copy a snapshot of the current memory state to a new buffer
-	var mem []byte
-	if l.cfg.EnableMemory {
-		mem = make([]byte, len(memory.Data()))
-		copy(mem, memory.Data())
-	}
-	// Copy a snapshot of the current stack state to a new buffer
-	var stck []uint256.Int
-	if !l.cfg.DisableStack {
-		stck = make([]uint256.Int, len(stack.Data()))
-		for i, item := range stack.Data() {
-			stck[i] = item
-		}
-	}
-	// Copy a snapshot of the current storage to a new container
-	var storage Storage
-	if !l.cfg.DisableStorage && (op == SLOAD || op == SSTORE) {
-		// initialise new changed values storage container for this contract
-		// if not present.
-		if l.storage[contract.Address()] == nil {
-			l.storage[contract.Address()] = make(Storage)
-		}
-		// capture SLOAD opcodes and record the read entry in the local storage
```

```
-               if op == SLOAD && stack.len() >= 1 {
-                       var (
-                               address = common.Hash(stack.data[stack.len()-1].Bytes32())
-                               value   = env.StateDB.GetState(contract.Address(), address)
-                       )
-                       l.storage[contract.Address()][address] = value
-                       storage = l.storage[contract.Address()].Copy()
-               } else if op == SSTORE && stack.len() >= 2 {
-                       // capture SSTORE opcodes and record the written entry in the local storage.
-                       var (
-                               value   = common.Hash(stack.data[stack.len()-2].Bytes32())
-                               address = common.Hash(stack.data[stack.len()-1].Bytes32())
-                       )
-                       l.storage[contract.Address()][address] = value
-                       storage = l.storage[contract.Address()].Copy()
-               }
-       }
-       var rdata []byte
-       if l.cfg.EnableReturnData {
-               rdata = make([]byte, len(rData))
-               copy(rdata, rData)
-       }
-       // create a new snapshot of the EVM.
-       log := StructLog{pc, op, gas, cost, mem, memory.Len(), stck, rdata, storage, depth, env.StateDB.GetRefund(), err}
-       l.logs = append(l.logs, log)
-}
-
-// CaptureFault implements the EVMLogger interface to trace an execution fault
-// while running an opcode.
-func (l *StructLogger) CaptureFault(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, depth int, err error) {
-}
-
-// CaptureEnd is called after the call finishes to finalize the tracing.
-func (l *StructLogger) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {
-       l.output = output
-       l.err = err
-       if l.cfg.Debug {
-               fmt.Printf("0x%x\n", output)
-               if err != nil {
-                       fmt.Printf(" error: %v\n", err)
-               }
-       }
-}
-
-func (l *StructLogger) CaptureEnter(typ OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
-}
-
-func (l *StructLogger) CaptureExit(output []byte, gasUsed uint64, err error) {}
-
-// StructLogs returns the captured log entries.
-func (l *StructLogger) StructLogs() []StructLog { return l.logs }
-
-// Error returns the VM error captured by the trace.
-func (l *StructLogger) Error() error { return l.err }
-
-// Output returns the VM return value captured by the trace.
-func (l *StructLogger) Output() []byte { return l.output }
-
-// WriteTrace writes a formatted trace to the given writer
-func WriteTrace(writer io.Writer, logs []StructLog) {
-       for _, log := range logs {
-               fmt.Fprintf(writer, "%-16spc=%08d gas=%v cost=%v", log.Op, log.Pc, log.Gas, log.GasCost)
-               if log.Err != nil {
-                       fmt.Fprintf(writer, " ERROR: %v", log.Err)
-               }
-               fmt.Fprintln(writer)
-
-               if len(log.Stack) > 0 {
-                       fmt.Fprintln(writer, "Stack:")
-                       for i := len(log.Stack) - 1; i >= 0; i-- {
-                               fmt.Fprintf(writer, "%08d  %s\n", len(log.Stack)-i-1, log.Stack[i].Hex())
-                       }
-               }
-               if len(log.Memory) > 0 {
-                       fmt.Fprintln(writer, "Memory:")
-                       fmt.Fprint(writer, hex.Dump(log.Memory))
-               }
-               if len(log.Storage) > 0 {
-                       fmt.Fprintln(writer, "Storage:")
-                       for h, item := range log.Storage {
-                               fmt.Fprintf(writer, "%x: %x\n", h, item)
-                       }
-               }
-               if len(log.ReturnData) > 0 {
-                       fmt.Fprintln(writer, "ReturnData:")
-                       fmt.Fprint(writer, hex.Dump(log.ReturnData))
-               }
-               fmt.Fprintln(writer)
-       }
-}
-
-// WriteLogs writes vm logs in a readable format to the given writer
-func WriteLogs(writer io.Writer, logs []*types.Log) {
-       for _, log := range logs {
-               fmt.Fprintf(writer, "LOG%d: %x bn=%d txi=%x\n", len(log.Topics), log.Address, log.BlockNumber, log.TxIndex)
-
-               for i, topic := range log.Topics {
-                       fmt.Fprintf(writer, "%08d  %x\n", i, topic)
-               }
-
-               fmt.Fprint(writer, hex.Dump(log.Data))
-               fmt.Fprintln(writer)
-       }
-}
-
-type mdLogger struct {
-       out io.Writer
-       cfg *LogConfig
-}
-
-// NewMarkdownLogger creates a logger which outputs information in a format adapted
-// for human readability, and is also a valid markdown table
-func NewMarkdownLogger(cfg *LogConfig, writer io.Writer) *mdLogger {
-       l := &mdLogger{writer, cfg}
-       if l.cfg == nil {
-               l.cfg = &LogConfig{}
-       }
-       return l
-}
-
-func (t *mdLogger) CaptureStart(env *EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
-       if !create {
-               fmt.Fprintf(t.out, "From: `%v`\nTo: `%v`\nData: `0x%x`\nGas: `%d`\nValue `%v` wei\n",
-                       from.String(), to.String(),
-                       input, gas, value)
-       } else {
-               fmt.Fprintf(t.out, "From: `%v`\nCreate at: `%v`\nData: `0x%x`\nGas: `%d`\nValue `%v` wei\n",
-                       from.String(), to.String(),
-                       input, gas, value)
-       }
-
-       fmt.Fprintf(t.out, `
-|  Pc   |      Op     | Cost |   Stack   |   RStack  |  Refund |
-|-------|-------------|------|-----------|-----------|---------|
-`)
-}
-
-// CaptureState also tracks SLOAD/SSTORE ops to track storage change.
-func (t *mdLogger) CaptureState(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, rData []byte, depth int, err error) {
-       stack := scope.Stack
-       fmt.Fprintf(t.out, "| %4d  | %10v  |  %3d |", pc, op, cost)
-
-       if !t.cfg.DisableStack {
-               // format stack
-               var a []string
-               for _, elem := range stack.data {
-                       a = append(a, elem.Hex())
-               }
-               b := fmt.Sprintf("[%v]", strings.Join(a, ","))
-               fmt.Fprintf(t.out, "%10v |", b)
-       }
-       fmt.Fprintf(t.out, "%10v |", env.StateDB.GetRefund())
-       fmt.Fprintln(t.out, "")
-       if err != nil {
-               fmt.Fprintf(t.out, "Error: %v\n", err)
-       }
-}
-
-func (t *mdLogger) CaptureFault(env *EVM, pc uint64, op OpCode, gas, cost uint64, scope *ScopeContext, depth int, err error) {
-       fmt.Fprintf(t.out, "\nError: at pc=%d, op=%v: %v\n", pc, op, err)
-}
-
-func (t *mdLogger) CaptureEnd(output []byte, gasUsed uint64, tm time.Duration, err error) {
-       fmt.Fprintf(t.out, "\nOutput: `0x%x`\nConsumed gas: `%d`\nError: `%v`\n",
-               output, gasUsed, err)
-}
-
-func (t *mdLogger) CaptureEnter(typ OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
-}
-
-func (t *mdLogger) CaptureExit(output []byte, gasUsed uint64, err error) {}
diff --git a/core/vm/logger_test.go b/core/vm/logger_test.go
index 1d87e46a..4fb66b02 100644
--- a/core/vm/logger_test.go
+++ b/core/vm/logger_test.go
@@ -30,10 +30,12 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ethereum/go-ethereum/common"
        "github.com/holiman/uint256"
+
+       "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
 )

 type dummyContractRef struct {
diff --git a/core/vm/memory_table.go b/core/vm/memory_table.go
index 45dd99bb..4af8c93c 100644
--- a/core/vm/memory_table.go
+++ b/core/vm/memory_table.go
@@ -26,7 +26,7 @@

 package vm

-func memorySha3(stack *Stack) (uint64, bool) {
+func memoryKeccak256(stack *Stack) (uint64, bool) {
        return calcMemSize64(stack.Back(0), stack.Back(1))
 }

diff --git a/core/vm/opcodes.go b/core/vm/opcodes.go
index 55f86ffc..c8739d5b 100644
--- a/core/vm/opcodes.go
+++ b/core/vm/opcodes.go
@@ -42,75 +42,70 @@ func (op OpCode) IsPush() bool {
        return false
 }

-// IsStaticJump specifies if an opcode is JUMP.
-func (op OpCode) IsStaticJump() bool {
-       return op == JUMP
-}
-
 // 0x0 range - arithmetic ops.
 const (
-       STOP OpCode = iota
-       ADD
-       MUL
-       SUB
-       DIV
-       SDIV
-       MOD
-       SMOD
-       ADDMOD
-       MULMOD
-       EXP
-       SIGNEXTEND
+       STOP       OpCode = 0x0
+       ADD        OpCode = 0x1
+       MUL        OpCode = 0x2
+       SUB        OpCode = 0x3
+       DIV        OpCode = 0x4
+       SDIV       OpCode = 0x5
+       MOD        OpCode = 0x6
+       SMOD       OpCode = 0x7
+       ADDMOD     OpCode = 0x8
+       MULMOD     OpCode = 0x9
+       EXP        OpCode = 0xa
+       SIGNEXTEND OpCode = 0xb
 )

 // 0x10 range - comparison ops.
 const (
-       LT OpCode = iota + 0x10
-       GT
-       SLT
-       SGT
-       EQ
-       ISZERO
-       AND
-       OR
-       XOR
-       NOT
-       BYTE
-       SHL
-       SHR
-       SAR
+       LT     OpCode = 0x10
+       GT     OpCode = 0x11
+       SLT    OpCode = 0x12
+       SGT    OpCode = 0x13
+       EQ     OpCode = 0x14
+       ISZERO OpCode = 0x15
+       AND    OpCode = 0x16
+       OR     OpCode = 0x17
+       XOR    OpCode = 0x18
+       NOT    OpCode = 0x19
+       BYTE   OpCode = 0x1a
+       SHL    OpCode = 0x1b
+       SHR    OpCode = 0x1c
+       SAR    OpCode = 0x1d

-       SHA3 OpCode = 0x20
+       KECCAK256 OpCode = 0x20
 )

 // 0x30 range - closure state.
 const (
-       ADDRESS OpCode = 0x30 + iota
-       BALANCE
-       ORIGIN
-       CALLER
-       CALLVALUE
-       CALLDATALOAD
-       CALLDATASIZE
-       CALLDATACOPY
-       CODESIZE
-       CODECOPY
-       GASPRICE
-       EXTCODESIZE
-       EXTCODECOPY
-       RETURNDATASIZE
-       RETURNDATACOPY
-       EXTCODEHASH
+       ADDRESS        OpCode = 0x30
+       BALANCE        OpCode = 0x31
+       ORIGIN         OpCode = 0x32
+       CALLER         OpCode = 0x33
+       CALLVALUE      OpCode = 0x34
+       CALLDATALOAD   OpCode = 0x35
+       CALLDATASIZE   OpCode = 0x36
+       CALLDATACOPY   OpCode = 0x37
+       CODESIZE       OpCode = 0x38
+       CODECOPY       OpCode = 0x39
+       GASPRICE       OpCode = 0x3a
+       EXTCODESIZE    OpCode = 0x3b
+       EXTCODECOPY    OpCode = 0x3c
+       RETURNDATASIZE OpCode = 0x3d
+       RETURNDATACOPY OpCode = 0x3e
+       EXTCODEHASH    OpCode = 0x3f
 )

 // 0x40 range - block operations.
 const (
-       BLOCKHASH OpCode = 0x40 + iota
-       COINBASE
-       TIMESTAMP
-       NUMBER
-       DIFFICULTY
-       GASLIMIT
+       BLOCKHASH   OpCode = 0x40
+       COINBASE    OpCode = 0x41
+       TIMESTAMP   OpCode = 0x42
+       NUMBER      OpCode = 0x43
+       DIFFICULTY  OpCode = 0x44
+       GASLIMIT    OpCode = 0x45
        CHAINID     OpCode = 0x46
        SELFBALANCE OpCode = 0x47
        BASEFEE     OpCode = 0x48
@@ -132,7 +127,7 @@ const (
        JUMPDEST OpCode = 0x5b
 )

-// 0x60 range.
+// 0x60 range - pushes.
 const (
        PUSH1 OpCode = 0x60 + iota
        PUSH2
@@ -166,7 +161,11 @@ const (
        PUSH30
        PUSH31
        PUSH32
-       DUP1
+)
+
+// 0x80 range - dups.
+const (
+       DUP1 = 0x80 + iota
        DUP2
        DUP3
        DUP4
@@ -182,7 +181,11 @@ const (
        DUP14
        DUP15
        DUP16
-       SWAP1
+)
+
+// 0x90 range - swaps.
+const (
+       SWAP1 = 0x90 + iota
        SWAP2
        SWAP3
        SWAP4
@@ -209,13 +212,6 @@ const (
        LOG4
 )

-// unofficial opcodes used for parsing.
-const (
-       PUSH OpCode = 0xb0 + iota
-       DUP
-       SWAP
-)
-
 const (
        BALANCEMC = 0xcd
        CALLEX    = 0xcf
@@ -223,14 +219,16 @@ const (

 // 0xf0 range - closures.
 const (
-       CREATE OpCode = 0xf0 + iota
-       CALL
-       CALLCODE
-       RETURN
-       DELEGATECALL
-       CREATE2
+       CREATE       OpCode = 0xf0
+       CALL         OpCode = 0xf1
+       CALLCODE     OpCode = 0xf2
+       RETURN       OpCode = 0xf3
+       DELEGATECALL OpCode = 0xf4
+       CREATE2      OpCode = 0xf5
+
        STATICCALL   OpCode = 0xfa
        REVERT       OpCode = 0xfd
+       INVALID      OpCode = 0xfe
```

```
	SELFDESTRUCT OpCode = 0xff
 )

@@ -267,7 +265,7 @@ var opCodeToString = map[OpCode]string{
	MULMOD: "MULMOD",

	// 0x20 range - crypto.
-	SHA3: "SHA3",
+	KECCAK256: "KECCAK256",

	// 0x30 range - closure state.
	ADDRESS:        "ADDRESS",
@@ -398,11 +396,8 @@ var opCodeToString = map[OpCode]string{
	CREATE2:      "CREATE2",
	STATICCALL:   "STATICCALL",
	REVERT:       "REVERT",
+	INVALID:      "INVALID",
	SELFDESTRUCT: "SELFDESTRUCT",
-
-	PUSH: "PUSH",
-	DUP:  "DUP",
-	SWAP: "SWAP",
 }

 func (op OpCode) String() string {
@@ -441,7 +436,7 @@ var stringToOp = map[string]OpCode{
	"SAR":            SAR,
	"ADDMOD":         ADDMOD,
	"MULMOD":         MULMOD,
-	"SHA3":           SHA3,
+	"KECCAK256":      KECCAK256,
	"ADDRESS":        ADDRESS,
	"BALANCE":        BALANCE,
	"BALANCEMC":      BALANCEMC,
@@ -558,6 +553,7 @@ var stringToOp = map[string]OpCode{
	"RETURN":         RETURN,
	"CALLCODE":       CALLCODE,
	"REVERT":         REVERT,
+	"INVALID":        INVALID,
	"SELFDESTRUCT":   SELFDESTRUCT,
 }

diff --git a/core/vm/operations_acl.go b/core/vm/operations_acl.go
index adfe7729..31c012cb 100644
--- a/core/vm/operations_acl.go
+++ b/core/vm/operations_acl.go
@@ -29,9 +29,9 @@ package vm
 import (
	"errors"

-	"github.com/ava-labs/coreth/params"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/common/math"
+	"github.com/flare-foundation/coreth/params"
 )

 // gasSStoreEIP2929 implements gas cost for SSTORE according to EIP-2929
diff --git a/core/vm/runtime/env.go b/core/vm/runtime/env.go
index 5293c408..00718ebe 100644
--- a/core/vm/runtime/env.go
+++ b/core/vm/runtime/env.go
@@ -27,8 +27,8 @@
 package runtime

 import (
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/vm"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/vm"
 )

 func NewEnv(cfg *Config) *vm.EVM {
diff --git a/core/vm/runtime/runtime.go b/core/vm/runtime/runtime.go
index 74d5499f..ac87b5e0 100644
--- a/core/vm/runtime/runtime.go
+++ b/core/vm/runtime/runtime.go
@@ -31,12 +31,12 @@ import (
	"math/big"
	"time"

-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/params"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/params"
 )

 // Config is a basic type specifying certain configuration flags for running
diff --git a/core/vm/runtime/runtime_example_test.go b/core/vm/runtime/runtime_example_test.go
index 9850e283..7d9c1352 100644
--- a/core/vm/runtime/runtime_example_test.go
+++ b/core/vm/runtime/runtime_example_test.go
@@ -29,8 +29,8 @@ package runtime_test
 import (
	"fmt"

-	"github.com/ava-labs/coreth/core/vm/runtime"
	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/vm/runtime"
 )

 func ExampleExecute() {
diff --git a/core/vm/runtime/runtime_test.go b/core/vm/runtime/runtime_test.go
index 456ae781..b3271145 100644
--- a/core/vm/runtime/runtime_test.go
+++ b/core/vm/runtime/runtime_test.go
@@ -34,17 +34,21 @@ import (
	"testing"
	"time"

-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/eth/tracers"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ethereum/go-ethereum/accounts/abi"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/core/asm"
+	"github.com/flare-foundation/coreth/accounts/abi"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/tracers"
```

```
+       "github.com/flare-foundation/coreth/eth/tracers/logger"
+       "github.com/flare-foundation/coreth/params"
+
+       // force-load js tracers to trigger registration
+       _ "github.com/flare-foundation/coreth/eth/tracers/js"
 )

 func TestDefaults(t *testing.T) {
@@ -333,19 +337,19 @@ func TestBlockhash(t *testing.T) {
 }

 type stepCounter struct {
-       inner *vm.JSONLogger
+       inner *logger.JSONLogger
        steps int
 }

 func (s *stepCounter) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
 }

-func (s *stepCounter) CaptureFault(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+func (s *stepCounter) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
 }

 func (s *stepCounter) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {}

-func (s *stepCounter) CaptureState(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+func (s *stepCounter) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
        s.steps++
        // Enable this for more output
        //s.inner.CaptureState(env, pc, op, gas, cost, memory, stack, rStack, contract, depth, err)
@@ -500,7 +504,7 @@ func BenchmarkSimpleLoop(b *testing.B) {
                byte(vm.JUMP),
        }

-       //tracer := vm.NewJSONLogger(nil, os.Stdout)
+       //tracer := logger.NewJSONLogger(nil, os.Stdout)
        //Execute(loopingCode, nil, &Config{
        //      EVMConfig: vm.Config{
        //              Debug:  true,
@@ -521,7 +525,7 @@ func BenchmarkSimpleLoop(b *testing.B) {
 // TestEip2929Cases contains various testcases that are used for
 // EIP-2929 about gas repricings
 func TestEip2929Cases(t *testing.T) {
-
+       t.Skip("Test only useful for generating documentation")
        id := 1
        prettyPrint := func(comment string, code []byte) {

@@ -543,7 +547,7 @@ func TestEip2929Cases(t *testing.T) {
                Execute(code, nil, &Config{
                        EVMConfig: vm.Config{
                                Debug:     true,
-                               Tracer:    vm.NewMarkdownLogger(nil, os.Stdout),
+                               Tracer:    logger.NewMarkdownLogger(nil, os.Stdout),
                                ExtraEips: []int{2929},
                        },
                })
@@ -693,7 +697,7 @@ func TestColdAccountAccessCost(t *testing.T) {
                        want: 7600,
                },
        } {
-               tracer := vm.NewStructLogger(nil)
+               tracer := logger.NewStructLogger(nil)
                Execute(tc.code, nil, &Config{
                        EVMConfig: vm.Config{
                                Debug:  true,
diff --git a/core/vm/stack.go b/core/vm/stack.go
index ebfeeef1..7ff708c6 100644
--- a/core/vm/stack.go
+++ b/core/vm/stack.go
@@ -64,10 +64,6 @@ func (st *Stack) push(d *uint256.Int) {
        // NOTE push limit (1024) is checked in baseCheck
        st.data = append(st.data, *d)
 }
-func (st *Stack) pushN(ds ...uint256.Int) {
-       // FIXME: Is there a way to pass args by pointers.
-       st.data = append(st.data, ds...)
-}

 func (st *Stack) pop() (ret uint256.Int) {
        ret = st.data[len(st.data)-1]
diff --git a/core/vm/stack_table.go b/core/vm/stack_table.go
index 487acaef..42197e64 100644
--- a/core/vm/stack_table.go
+++ b/core/vm/stack_table.go
@@ -27,7 +27,7 @@
 package vm

 import (
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"
 )

 func minSwapStack(n int) int {
diff --git a/eth/api.go b/eth/api.go
index e58e1545..d00efb5a 100644
--- a/eth/api.go
+++ b/eth/api.go
@@ -36,17 +36,17 @@ import (
        "strings"
        "time"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/internal/ethapi"
-       "github.com/ava-labs/coreth/rpc"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/internal/ethapi"
+       "github.com/flare-foundation/coreth/rpc"
+       "github.com/flare-foundation/coreth/trie"
 )

 // PublicEthereumAPI provides an API to access Ethereum full node-related
@@ -93,7 +93,7 @@ func (api *PrivateAdminAPI) ExportChain(file string, first *uint64, last *uint64
                last = &head
        }
        if _, err := os.Stat(file); err == nil {
-               // File already exists. Allowing overwrite could be a DoS vecotor,
+               // File already exists. Allowing overwrite could be a DoS vector,
                // since the 'file' may point to arbitrary paths on the drive
                return false, errors.New("location would overwrite an existing file")
        }
diff --git a/eth/api_backend.go b/eth/api_backend.go
```

```diff
index 3cc182b3..c2189509 100644
--- a/eth/api_backend.go
+++ b/eth/api_backend.go
@@ -32,21 +32,21 @@ import (
 	"math/big"
 	"time"
 
-	"github.com/ava-labs/coreth/accounts"
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/bloombits"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/eth/gasprice"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/accounts"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/bloombits"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/gasprice"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/rpc"
 )
 
 var (
@@ -242,13 +242,6 @@ func (b *EthAPIBackend) GetLogs(ctx context.Context, hash common.Hash) ([][]*typ
 	return logs, nil
 }
 
-func (b *EthAPIBackend) GetTd(ctx context.Context, hash common.Hash) *big.Int {
-	if header := b.eth.blockchain.GetHeaderByHash(hash); header != nil {
-		return b.eth.blockchain.GetTd(hash, header.Number.Uint64())
-	}
-	return nil
-}
-
 func (b *EthAPIBackend) GetEVM(ctx context.Context, msg core.Message, state *state.StateDB, header *types.Header, vmConfig *vm.Config) (*vm.EVM, func() error, error) {
 	vmError := func() error { return nil }
 	if vmConfig == nil {
@@ -428,7 +421,7 @@ func (b *EthAPIBackend) GetMaxBlocksPerRequest() int64 {
 }
 
 func (b *EthAPIBackend) StateAtBlock(ctx context.Context, block *types.Block, reexec uint64, base *state.StateDB, checkLive bool, preferDisk bool) (*state.StateDB, error) {
-	return b.eth.stateAtBlock(block, reexec, base, checkLive, preferDisk)
+	return b.eth.StateAtBlock(block, reexec, base, checkLive, preferDisk)
 }
 
 func (b *EthAPIBackend) StateAtTransaction(ctx context.Context, block *types.Block, txIndex int, reexec uint64) (core.Message, vm.BlockContext, *state.StateDB, error) {
diff --git a/eth/backend.go b/eth/backend.go
index 8b76075f..32cfbdf1 100644
--- a/eth/backend.go
+++ b/eth/backend.go
@@ -33,27 +33,30 @@ import (
 	"sync"
 	"time"
 
-	"github.com/ava-labs/coreth/accounts"
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/consensus/dummy"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/bloombits"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/eth/ethconfig"
-	"github.com/ava-labs/coreth/eth/filters"
-	"github.com/ava-labs/coreth/eth/gasprice"
-	"github.com/ava-labs/coreth/eth/tracers"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/internal/ethapi"
-	"github.com/ava-labs/coreth/miner"
-	"github.com/ava-labs/coreth/node"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/accounts"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/consensus/dummy"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/bloombits"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/state/pruner"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/ethconfig"
+	"github.com/flare-foundation/coreth/eth/filters"
+	"github.com/flare-foundation/coreth/eth/gasprice"
+	"github.com/flare-foundation/coreth/eth/tracers"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/internal/ethapi"
+	"github.com/flare-foundation/coreth/internal/shutdowncheck"
+	"github.com/flare-foundation/coreth/miner"
+	"github.com/flare-foundation/coreth/node"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/rpc"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 
 // Config contains the configuration options of the ETH protocol.
@@ -97,16 +100,23 @@ type Ethereum struct {
 
 	lock sync.RWMutex // Protects the variadic fields (e.g. gas price and etherbase)
 
+	shutdownTracker *shutdowncheck.ShutdownTracker // Tracks if and when the node has shutdown ungracefully
+
+	stackRPCs []rpc.API
+
 	settings Settings // Settings for Ethereum API
 }
 
 // New creates a new Ethereum object (including the
 // initialisation of the common Ethereum object)
-func New(stack *node.Node, config *Config,
+func New(
+	stack *node.Node,
+	config *Config,
 	cb *dummy.ConsensusCallbacks,
 	chainDb ethdb.Database,
 	settings Settings,
        lastAcceptedHash common.Hash,
+       clock *mockable.Clock,
 ) (*Ethereum, error) {
        if chainDb == nil {
                return nil, errors.New("chainDb cannot be nil")
@@ -131,14 +141,19 @@ func New(stack *node.Node, config *Config,
        }
        log.Info("Initialised chain configuration", "config", chainConfig)

-       // FIXME RecoverPruning once that package is migrated over
-       // if err := pruner.RecoverPruning(stack.ResolvePath(""), chainDb, stack.ResolvePath(config.TrieCleanCacheJournal)); err != nil {
-       //              log.Error("Failed to recover state", "error", err)
-       // }
+       // Note: RecoverPruning must be called to handle the case that we are midway through offline pruning.
+       // If the data directory is changed in between runs preventing RecoverPruning from performing its job correctly,
+       // it may cause DB corruption.
+       // Since RecoverPruning will only continue a pruning run that already began, we do not need to ensure that
+       // reprocessState has already been called and completed successfully. To ensure this, we must maintain
+       // that Prune is only run after reprocessState has finished successfully.
+       if err := pruner.RecoverPruning(config.OfflinePruningDataDirectory, chainDb); err != nil {
+               log.Error("Failed to recover state", "error", err)
+       }
        eth := &Ethereum{
                config:            config,
                chainDb:           chainDb,
-               eventMux:          stack.EventMux(),
+               eventMux:          new(event.TypeMux),
                accountManager:    stack.AccountManager(),
                engine:            dummy.NewDummyEngine(cb),
                closeBloomHandler: make(chan struct{}),
@@ -147,6 +162,7 @@ func New(stack *node.Node, config *Config,
                bloomRequests:     make(chan chan *bloombits.Retrieval),
                bloomIndexer:      core.NewBloomIndexer(chainDb, params.BloomBitsBlocks, params.BloomConfirms),
                settings:          settings,
+               shutdownTracker:   shutdowncheck.NewShutdownTracker(chainDb),
        }

        bcVersion := rawdb.ReadDatabaseVersion(chainDb)
@@ -184,16 +200,17 @@ func New(stack *node.Node, config *Config,
        if err != nil {
                return nil, err
        }
+
+       if err := eth.handleOfflinePruning(cacheConfig, chainConfig, vmConfig, lastAcceptedHash); err != nil {
+               return nil, err
+       }
+
        eth.bloomIndexer.Start(eth.blockchain)

-       // Original code (requires disk):
-       // if config.TxPool.Journal != "" {
-       //      config.TxPool.Journal = stack.ResolvePath(config.TxPool.Journal)
-       // }
        config.TxPool.Journal = ""
        eth.txPool = core.NewTxPool(config.TxPool, chainConfig, eth.blockchain)

-       eth.miner = miner.New(eth, &config.Miner, chainConfig, eth.EventMux(), eth.engine)
+       eth.miner = miner.New(eth, &config.Miner, chainConfig, eth.EventMux(), eth.engine, clock)

        eth.APIBackend = &EthAPIBackend{
                extRPCEnabled:       stack.Config().ExtRPCEnabled(),
@@ -213,8 +230,10 @@ func New(stack *node.Node, config *Config,
        // Start the RPC service
        eth.netRPCService = ethapi.NewPublicNetAPI(eth.NetVersion())

-       // Register the backend on the node
-       stack.RegisterAPIs(eth.APIs())
+       eth.stackRPCs = stack.APIs()
+
+       // Successful startup; push a marker and check previous unclean shutdowns.
+       eth.shutdownTracker.MarkStartup()

        return eth, nil
 }
@@ -227,8 +246,8 @@ func (s *Ethereum) APIs() []rpc.API {
        // Append tracing APIs
        apis = append(apis, tracers.APIs(s.APIBackend)...)

-       // Append any APIs exposed explicitly by the consensus engine
-       apis = append(apis, s.engine.APIs(s.BlockChain())...)
+       // Add the APIs from the node
+       apis = append(apis, s.stackRPCs...)

        // Append all the local APIs and return
        return append(apis, []rpc.API{
@@ -237,29 +256,35 @@ func (s *Ethereum) APIs() []rpc.API {
                        Version:   "1.0",
                        Service:   NewPublicEthereumAPI(s),
                        Public:    true,
+                       Name:      "public-eth",
                }, {
                        Namespace: "eth",
                        Version:   "1.0",
                        Service:   filters.NewPublicFilterAPI(s.APIBackend, false, 5*time.Minute),
                        Public:    true,
+                       Name:      "public-eth-filter",
                }, {
                        Namespace: "admin",
                        Version:   "1.0",
                        Service:   NewPrivateAdminAPI(s),
+                       Name:      "private-admin",
                }, {
                        Namespace: "debug",
                        Version:   "1.0",
                        Service:   NewPublicDebugAPI(s),
                        Public:    true,
+                       Name:      "public-debug",
                }, {
                        Namespace: "debug",
                        Version:   "1.0",
                        Service:   NewPrivateDebugAPI(s),
+                       Name:      "private-debug",
                }, {
                        Namespace: "net",
                        Version:   "1.0",
                        Service:   s.netRPCService,
                        Public:    true,
+                       Name:      "net",
                },
        }...)
 }
@@ -314,6 +339,9 @@ func (s *Ethereum) BloomIndexer() *core.ChainIndexer { return s.bloomIndexer }
 func (s *Ethereum) Start() {
        // Start the bloom bits servicing goroutines
        s.startBloomHandlers(params.BloomBitsBlocks)
+
+       // Regularly update shutdown marker
+       s.shutdownTracker.Start()
 }

 // Stop implements node.Lifecycle, terminating all internal goroutines used by the
@@ -325,6 +353,10 @@ func (s *Ethereum) Stop() error {
        s.txPool.Stop()
        s.blockchain.Stop()
        s.engine.Close()
+
+       // Clean shutdown marker as the last thing before closing db
+       s.shutdownTracker.Stop()
+
        s.chainDb.Close()
        s.eventMux.Stop()
        return nil
@@ -333,3 +365,46 @@ func (s *Ethereum) Stop() error {
 func (s *Ethereum) LastAcceptedBlock() *types.Block {
        return s.blockchain.LastAcceptedBlock()
 }
+
+func (s *Ethereum) handleOfflinePruning(cacheConfig *core.CacheConfig, chainConfig *params.ChainConfig, vmConfig vm.Config, lastAcceptedHash common.Hash) error {
+       if !s.config.OfflinePruning {
+               // Delete the offline pruning marker to indicate that the node started with offline pruning disabled.
+               if err := rawdb.DeleteOfflinePruning(s.chainDb); err != nil {
+                       return fmt.Errorf("failed to write offline pruning disabled marker: %w", err)
+               }
+               return nil
+       }
+
+       // Perform offline pruning after NewBlockChain has been called to ensure that we have rolled back the chain
+       // to the last accepted block before pruning begins.
+       // If offline pruning marker is on disk, then we force the node to be started with offline pruning disabled
+       // before allowing another run of offline pruning.
+       if _, err := rawdb.ReadOfflinePruning(s.chainDb); err == nil {
+               log.Error("Offline pruning is not meant to be left enabled permanently. Please disable offline pruning and allow your node to start successfully before running offline pruning again.")
+               return errors.New("cannot start chain with offline pruning enabled on consecutive starts")
+       }
+
+       // Clean up middle roots
+       if err := s.blockchain.CleanBlockRootsAboveLastAccepted(); err != nil {
+               return err
+       }
+       targetRoot := s.blockchain.LastAcceptedBlock().Root()
+
+       // Allow the blockchain to be garbage collected immediately, since we will shut down the chain after offline pruning completes.
+       s.blockchain.Stop()
+       s.blockchain = nil
+       log.Info("Starting offline pruning", "dataDir", s.config.OfflinePruningDataDirectory, "bloomFilterSize", s.config.OfflinePruningBloomFilterSize)
+       pruner, err := pruner.NewPruner(s.chainDb, s.config.OfflinePruningDataDirectory, s.config.OfflinePruningBloomFilterSize)
+       if err != nil {
+               return fmt.Errorf("failed to create new pruner with data directory: %s, size: %d, due to: %w", s.config.OfflinePruningDataDirectory, s.config.OfflinePruningBloomFilterSize, err)
+       }
+       if err := pruner.Prune(targetRoot); err != nil {
+               return fmt.Errorf("failed to prune blockchain with target root: %s due to: %w", targetRoot, err)
+       }
+       s.blockchain, err = core.NewBlockChain(s.chainDb, cacheConfig, chainConfig, s.engine, vmConfig, lastAcceptedHash)
+       if err != nil {
+               return fmt.Errorf("failed to re-initialize blockchain after offline pruning: %w", err)
+       }
+
+       return nil
+}
diff --git a/eth/bloombits.go b/eth/bloombits.go
index ecc0aaf1..1a0d5d46 100644
--- a/eth/bloombits.go
+++ b/eth/bloombits.go
@@ -29,8 +29,8 @@ package eth
 import (
        "time"

-       "github.com/ava-labs/coreth/core/rawdb"
        "github.com/ethereum/go-ethereum/common/bitutil"
+       "github.com/flare-foundation/coreth/core/rawdb"
 )

 const (
diff --git a/eth/ethconfig/config.go b/eth/ethconfig/config.go
index d2d76292..a0ab38ae 100644
--- a/eth/ethconfig/config.go
+++ b/eth/ethconfig/config.go
@@ -29,15 +29,15 @@ package ethconfig
 import (
        "time"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/eth/gasprice"
-       "github.com/ava-labs/coreth/miner"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/eth/gasprice"
+       "github.com/flare-foundation/coreth/miner"
 )

 // DefaultFullGPOConfig contains default gasprice oracle settings for full node.
 var DefaultFullGPOConfig = gasprice.Config{
-       Blocks:           20,
+       Blocks:           40,
        Percentile:       60,
        MaxHeaderHistory: 1024,
        MaxBlockHistory:  1024,
@@ -51,22 +51,19 @@ var DefaultConfig = NewDefaultConfig()

 func NewDefaultConfig() Config {
        return Config{
-               NetworkId:               1,
-               LightPeers:              100,
-               UltraLightFraction:      75,
-               DatabaseCache:           512,
-               TrieCleanCache:          75,
-               TrieCleanCacheJournal:   "triecache",
-               TrieCleanCacheRejournal: 60 * time.Minute,
-               TrieDirtyCache:          256,
-               TrieTimeout:             60 * time.Minute,
-               SnapshotCache:           128,
-               Miner:                   miner.Config{},
-               TxPool:                  core.DefaultTxPoolConfig,
-               RPCGasCap:               25000000,
-               RPCEVMTimeout:           5 * time.Second,
-               GPO:                     DefaultFullGPOConfig,
-               RPCTxFeeCap:             1, // 1 AVAX
+               NetworkId:          1,
+               LightPeers:         100,
+               UltraLightFraction: 75,
+               DatabaseCache:      512,
+               TrieCleanCache:     75,
+               TrieDirtyCache:     256,
+               SnapshotCache:      128,
+               Miner:              miner.Config{},
+               TxPool:             core.DefaultTxPoolConfig,
+               RPCGasCap:          25000000,
+               RPCEVMTimeout:      5 * time.Second,
+               GPO:                DefaultFullGPOConfig,
+               RPCTxFeeCap:        1, // 1 AVAX
        }
 }

@@ -110,13 +107,10 @@ type Config struct {
        DatabaseCache      int
        // DatabaseFreezer    string

-       TrieCleanCache          int
```

```
-	TrieCleanCacheJournal   string        `toml:",omitempty"` // Disk journal directory for trie cache to survive node restarts
-	TrieCleanCacheRejournal time.Duration `toml:",omitempty"` // Time interval to regenerate the journal for clean cache
-	TrieDirtyCache          int
-	TrieTimeout             time.Duration
-	SnapshotCache           int
-	Preimages               bool
+	TrieCleanCache int
+	TrieDirtyCache int
+	SnapshotCache  int
+	Preimages      bool

 	// Mining options
 	Miner miner.Config
@@ -150,4 +144,10 @@ type Config struct {
 	// Unprotected transactions are transactions that are signed without EIP-155
 	// replay protection.
 	AllowUnprotectedTxs bool
+
+	// OfflinePruning enables offline pruning on startup of the node. If a node is started
+	// with this configuration option, it must finish pruning before resuming normal operation.
+	OfflinePruning                bool
+	OfflinePruningBloomFilterSize uint64
+	OfflinePruningDataDirectory   string
 }
diff --git a/eth/filters/api.go b/eth/filters/api.go
index f269156c..9d70fa0b 100644
--- a/eth/filters/api.go
+++ b/eth/filters/api.go
@@ -35,13 +35,12 @@ import (
 	"sync"
 	"time"

-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/interfaces"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/interfaces"
+	"github.com/flare-foundation/coreth/rpc"
 )

 // filter is a helper struct that holds meta information over the filter type
@@ -61,7 +60,6 @@ type PublicFilterAPI struct {
 	backend   Backend
 	mux       *event.TypeMux
 	quit      chan struct{}
-	chainDb   ethdb.Database
 	events    *EventSystem
 	filtersMu sync.Mutex
 	filters   map[rpc.ID]*filter
@@ -72,7 +70,6 @@ type PublicFilterAPI struct {
 func NewPublicFilterAPI(backend Backend, lightMode bool, timeout time.Duration) *PublicFilterAPI {
 	api := &PublicFilterAPI{
 		backend: backend,
-		chainDb: backend.ChainDb(),
 		events:  NewEventSystem(backend, lightMode),
 		filters: make(map[rpc.ID]*filter),
 		timeout: timeout,
diff --git a/eth/filters/api_test.go b/eth/filters/api_test.go
index 6f356395..041bd46b 100644
--- a/eth/filters/api_test.go
+++ b/eth/filters/api_test.go
@@ -21,8 +21,8 @@ import (
 	"fmt"
 	"testing"

-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/rpc"
 )

 func TestUnmarshalJSONNewFilterArgs(t *testing.T) {
diff --git a/eth/filters/filter.go b/eth/filters/filter.go
index 227ccd64..b90bca8b 100644
--- a/eth/filters/filter.go
+++ b/eth/filters/filter.go
@@ -32,15 +32,15 @@ import (
 	"fmt"
 	"math/big"

-	"github.com/ava-labs/coreth/core/vm"
+	"github.com/flare-foundation/coreth/core/vm"

-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/bloombits"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/bloombits"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/rpc"
 )

 type Backend interface {
diff --git a/eth/filters/filter_system.go b/eth/filters/filter_system.go
index 86f5a0bd..67829974 100644
--- a/eth/filters/filter_system.go
+++ b/eth/filters/filter_system.go
@@ -34,14 +34,14 @@ import (
 	"sync"
 	"time"

-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/interfaces"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/interfaces"
+	"github.com/flare-foundation/coreth/rpc"
 )

 // Type determines the kind of filter and is used to put the filter in to
diff --git a/eth/gasprice/feehistory.go b/eth/gasprice/feehistory.go
index 081f6c48..59b3b08d 100644
--- a/eth/gasprice/feehistory.go
+++ b/eth/gasprice/feehistory.go
@@ -36,11 +36,11 @@ import (
 	"sort"
 	"sync/atomic"

-	_ "github.com/ava-labs/coreth/consensus/misc"
```

```
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       _ "github.com/flare-foundation/coreth/consensus/misc"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/rpc"
 )

 var (
diff --git a/eth/gasprice/feehistory_test.go b/eth/gasprice/feehistory_test.go
index 146c2814..e60c4d9c 100644
--- a/eth/gasprice/feehistory_test.go
+++ b/eth/gasprice/feehistory_test.go
@@ -32,12 +32,12 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"

-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
 )

 func TestFeeHistory(t *testing.T) {
diff --git a/eth/gasprice/gasprice.go b/eth/gasprice/gasprice.go
index 210bc701..ce67d395 100644
--- a/eth/gasprice/gasprice.go
+++ b/eth/gasprice/gasprice.go
@@ -32,22 +32,24 @@ import (
        "sort"
        "sync"

-       "github.com/ava-labs/avalanchego/utils/timer/mockable"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
+       "github.com/flare-foundation/flare/utils/timer/mockable"
        lru "github.com/hashicorp/golang-lru"
 )

 var (
        DefaultMaxPrice   = big.NewInt(150 * params.GWei)
        DefaultMinPrice   = big.NewInt(0 * params.GWei)
-       DefaultMinGasUsed = big.NewInt(2_000_000) // block gas limit is 8,000,000
+       DefaultMinBaseFee = big.NewInt(params.ApricotPhase3InitialBaseFee)
+       DefaultMinGasUsed = big.NewInt(12_000_000) // block gas target is 15,000,000
 )

 type Config struct {
@@ -74,9 +76,10 @@ type OracleBackend interface {
 // Oracle recommends gas prices based on the content of recent
 // blocks. Suitable for both light and full clients.
 type Oracle struct {
-       backend   OracleBackend
-       lastHead  common.Hash
-       lastPrice *big.Int
+       backend     OracleBackend
+       lastHead    common.Hash
+       lastPrice   *big.Int
+       lastBaseFee *big.Int
        // [minPrice] ensures we don't get into a positive feedback loop where tips
        // sink to 0 during a period of slow block production, such that nobody's
        // transactions will be included until the full block fee duration has
@@ -109,8 +112,7 @@ func NewOracle(backend OracleBackend, config Config) *Oracle {
        if percent < 0 {
                percent = 0
                log.Warn("Sanitizing invalid gasprice oracle sample percentile", "provided", config.Percentile, "updated", percent)
-       }
-       if percent > 100 {
+       } else if percent > 100 {
                percent = 100
                log.Warn("Sanitizing invalid gasprice oracle sample percentile", "provided", config.Percentile, "updated", percent)
        }
@@ -129,6 +131,16 @@ func NewOracle(backend OracleBackend, config Config) *Oracle {
                minGasUsed = DefaultMinGasUsed
                log.Warn("Sanitizing invalid gasprice oracle min gas used", "provided", config.MinGasUsed, "updated", minGasUsed)
        }
+       maxHeaderHistory := config.MaxHeaderHistory
+       if maxHeaderHistory < 1 {
+               maxHeaderHistory = 1
+               log.Warn("Sanitizing invalid gasprice oracle max header history", "provided", config.MaxHeaderHistory, "updated", maxHeaderHistory)
+       }
+       maxBlockHistory := config.MaxBlockHistory
+       if maxBlockHistory < 1 {
+               maxBlockHistory = 1
+               log.Warn("Sanitizing invalid gasprice oracle max block history", "provided", config.MaxBlockHistory, "updated", maxBlockHistory)
+       }

        cache, _ := lru.New(2048)
        headEvent := make(chan core.ChainHeadEvent, 1)
@@ -146,13 +158,14 @@ func NewOracle(backend OracleBackend, config Config) *Oracle {
        return &Oracle{
                backend:          backend,
                lastPrice:        minPrice,
+               lastBaseFee:      DefaultMinBaseFee,
                minPrice:         minPrice,
                maxPrice:         maxPrice,
                minGasUsed:       minGasUsed,
                checkBlocks:      blocks,
                percentile:       percent,
-               maxHeaderHistory: config.MaxHeaderHistory,
-               maxBlockHistory:  config.MaxBlockHistory,
+               maxHeaderHistory: maxHeaderHistory,
+               maxBlockHistory:  maxBlockHistory,
                historyCache:     cache,
        }
 }
@@ -161,6 +174,34 @@ func NewOracle(backend OracleBackend, config Config) *Oracle {
 // produced at the current time. If ApricotPhase3 has not been activated, it may
 // return a nil value and a nil error.
 func (oracle *Oracle) EstimateBaseFee(ctx context.Context) (*big.Int, error) {
+       _, baseFee, err := oracle.suggestDynamicFees(ctx)
+       if err != nil {
+               return nil, err
+       }
+
```

```diff
+       // We calculate the [nextBaseFee] if a block were to be produced immediately.
+       // If [nextBaseFee] is lower than the estimate from sampling, then we return it
+       // to prevent returning an incorrectly high fee when the network is quiescent.
+       nextBaseFee, err := oracle.estimateNextBaseFee(ctx)
+       if err != nil {
+               log.Warn("failed to estimate next base fee", "err", err)
+               return baseFee, nil
+       }
+       // If base fees have not been enabled, return a nil value.
+       if nextBaseFee == nil {
+               return nil, nil
+       }
+
+       baseFee = math.BigMin(baseFee, nextBaseFee)
+       return baseFee, nil
+}
+
+// estimateNextBaseFee calculates what the base fee should be on the next block if it
+// were produced immediately. If the current time is less than the timestamp of the latest
+// block, this esimtate uses the timestamp of the latest block instead.
+// If the latest block has a nil base fee, this function will return nil as the base fee
+// of the next block.
+func (oracle *Oracle) estimateNextBaseFee(ctx context.Context) (*big.Int, error) {
        // Fetch the most recent block by number
        block, err := oracle.backend.BlockByNumber(ctx, rpc.LatestBlockNumber)
        if err != nil {
@@ -171,37 +212,35 @@ func (oracle *Oracle) EstimateBaseFee(ctx context.Context) (*big.Int, error) {
                return nil, nil
        }

-       // If the current time is prior to the parent timestamp, then we use the parent
-       // timestamp instead.
-       header := block.Header()
-       timestamp := oracle.clock.Unix()
-       if timestamp < header.Time {
-               timestamp = header.Time
-       }
        // If the block does have a baseFee, calculate the next base fee
        // based on the current time and add it to the tip to estimate the
        // total gas price estimate.
-       _, nextBaseFee, err := dummy.CalcBaseFee(oracle.backend.ChainConfig(), header, timestamp)
+       _, nextBaseFee, err := dummy.EstimateNextBaseFee(oracle.backend.ChainConfig(), block.Header(), oracle.clock.Unix())
        return nextBaseFee, err
 }

 // SuggestPrice returns an estimated price for legacy transactions.
 func (oracle *Oracle) SuggestPrice(ctx context.Context) (*big.Int, error) {
        // Estimate the effective tip based on recent blocks.
-       tip, err := oracle.suggestTipCap(ctx)
+       tip, baseFee, err := oracle.suggestDynamicFees(ctx)
        if err != nil {
                return nil, err
        }
-       nextBaseFee, err := oracle.EstimateBaseFee(ctx)
+
+       // We calculate the [nextBaseFee] if a block were to be produced immediately.
+       // If [nextBaseFee] is lower than the estimate from sampling, then we return it
+       // to prevent returning an incorrectly high fee when the network is quiescent.
+       nextBaseFee, err := oracle.estimateNextBaseFee(ctx)
        if err != nil {
-               return nil, err
+               log.Warn("failed to estimate next base fee", "err", err)
        }
-       // If [nextBaseFee] is nil, return [tip] without modification.
-       if nextBaseFee == nil {
-               return tip, nil
+       // Separately from checking the error value, check that [nextBaseFee] is non-nil
+       // before attempting to take the minimum.
+       if nextBaseFee != nil {
+               baseFee = math.BigMin(baseFee, nextBaseFee)
        }

-       return new(big.Int).Add(tip, nextBaseFee), nil
+       return new(big.Int).Add(tip, baseFee), nil
 }

 // SuggestTipCap returns a tip cap so that newly created transaction can have a
@@ -211,54 +250,43 @@ func (oracle *Oracle) SuggestPrice(ctx context.Context) (*big.Int, error) {
 // necessary to add the basefee to the returned number to fall back to the legacy
 // behavior.
 func (oracle *Oracle) SuggestTipCap(ctx context.Context) (*big.Int, error) {
-       return oracle.suggestTipCap(ctx)
+       tip, _, err := oracle.suggestDynamicFees(ctx)
+       return tip, err
 }

-// sugggestTipCap checks the clock to estimate what network rules will be applied to
-// new transactions and then suggests a gas tip cap based on the response.
-func (oracle *Oracle) suggestTipCap(ctx context.Context) (*big.Int, error) {
-       bigTimestamp := big.NewInt(oracle.clock.Time().Unix())
-
-       switch {
-       case oracle.backend.ChainConfig().IsApricotPhase4(bigTimestamp):
-               return oracle.suggestDynamicTipCap(ctx)
-       case oracle.backend.ChainConfig().IsApricotPhase3(bigTimestamp):
-               return new(big.Int).Set(common.Big0), nil
-       case oracle.backend.ChainConfig().IsApricotPhase1(bigTimestamp):
-               return big.NewInt(params.ApricotPhase1MinGasPrice), nil
-       default:
-               return big.NewInt(params.LaunchMinGasPrice), nil
+// suggestDynamicFees estimates the gas tip and base fee based on a simple sampling method
+func (oracle *Oracle) suggestDynamicFees(ctx context.Context) (*big.Int, *big.Int, error) {
+       head, err := oracle.backend.HeaderByNumber(ctx, rpc.LatestBlockNumber)
+       if err != nil {
+               return nil, nil, err
        }
-}

-// suggestDynamicTipCap estimates the gas tip based on a simple sampling method
-func (oracle *Oracle) suggestDynamicTipCap(ctx context.Context) (*big.Int, error) {
-       head, _ := oracle.backend.HeaderByNumber(ctx, rpc.LatestBlockNumber)
        headHash := head.Hash()

        // If the latest gasprice is still available, return it.
        oracle.cacheLock.RLock()
-       lastHead, lastPrice := oracle.lastHead, oracle.lastPrice
+       lastHead, lastPrice, lastBaseFee := oracle.lastHead, oracle.lastPrice, oracle.lastBaseFee
        oracle.cacheLock.RUnlock()
        if headHash == lastHead {
-               return new(big.Int).Set(lastPrice), nil
+               return new(big.Int).Set(lastPrice), new(big.Int).Set(lastBaseFee), nil
        }
        oracle.fetchLock.Lock()
        defer oracle.fetchLock.Unlock()

        // Try checking the cache again, maybe the last fetch fetched what we need
        oracle.cacheLock.RLock()
-       lastHead, lastPrice = oracle.lastHead, oracle.lastPrice
+       lastHead, lastPrice, lastBaseFee = oracle.lastHead, oracle.lastPrice, oracle.lastBaseFee
        oracle.cacheLock.RUnlock()
        if headHash == lastHead {
-               return new(big.Int).Set(lastPrice), nil
+               return new(big.Int).Set(lastPrice), new(big.Int).Set(lastBaseFee), nil
        }
        var (
-               sent, exp int
-               number    = head.Number.Uint64()
-               result    = make(chan results, oracle.checkBlocks)
-               quit      = make(chan struct{})
-               results   []*big.Int
+               sent, exp      int
+               number         = head.Number.Uint64()
+               result         = make(chan results, oracle.checkBlocks)
+               quit           = make(chan struct{})
+               tipResults     []*big.Int
+               baseFeeResults []*big.Int
        )
        for sent < oracle.checkBlocks && number > 0 {
                go oracle.getBlockTips(ctx, number, result, quit)
@@ -270,19 +298,31 @@ func (oracle *Oracle) suggestDynamicTipCap(ctx context.Context) (*big.Int, error
                res := <-result
                if res.err != nil {
                        close(quit)
-                       return new(big.Int).Set(lastPrice), res.err
+                       return new(big.Int).Set(lastPrice), new(big.Int).Set(lastBaseFee), res.err
                }
                exp--
-               if res.value != nil {
-                       results = append(results, res.value)
+               if res.tip != nil {
+                       tipResults = append(tipResults, res.tip)
+               } else {
+                       tipResults = append(tipResults, new(big.Int).Set(common.Big0))
+               }
+
+               if res.baseFee != nil {
+                       baseFeeResults = append(baseFeeResults, res.baseFee)
                } else {
-                       results = append(results, new(big.Int).Set(common.Big0))
+                       baseFeeResults = append(baseFeeResults, new(big.Int).Set(common.Big0))
                }
        }
        price := lastPrice
-       if len(results) > 0 {
-               sort.Sort(bigIntArray(results))
-               price = results[(len(results)-1)*oracle.percentile/100]
+       baseFee := lastBaseFee
+       if len(tipResults) > 0 {
+               sort.Sort(bigIntArray(tipResults))
+               price = tipResults[(len(tipResults)-1)*oracle.percentile/100]
+       }
+
+       if len(baseFeeResults) > 0 {
+               sort.Sort(bigIntArray(baseFeeResults))
+               baseFee = baseFeeResults[(len(baseFeeResults)-1)*oracle.percentile/100]
        }
        if price.Cmp(oracle.maxPrice) > 0 {
                price = new(big.Int).Set(oracle.maxPrice)
@@ -293,14 +333,16 @@ func (oracle *Oracle) suggestDynamicTipCap(ctx context.Context) (*big.Int, error
        oracle.cacheLock.Lock()
        oracle.lastHead = headHash
        oracle.lastPrice = price
+       oracle.lastBaseFee = baseFee
        oracle.cacheLock.Unlock()

-       return new(big.Int).Set(price), nil
+       return new(big.Int).Set(price), new(big.Int).Set(baseFee), nil
 }

 type results struct {
-       value *big.Int
-       err   error
+       tip     *big.Int
+       baseFee *big.Int
+       err     error
 }

 // getBlockTips calculates the minimum required tip to be included in a given
@@ -309,7 +351,7 @@ func (oracle *Oracle) getBlockTips(ctx context.Context, blockNum uint64, result
        header, err := oracle.backend.HeaderByNumber(ctx, rpc.BlockNumber(blockNum))
        if header == nil {
                select {
-               case result <- results{nil, err}:
+               case result <- results{nil, nil, err}:
                case <-quit:
                }
                return
@@ -319,7 +361,7 @@ func (oracle *Oracle) getBlockTips(ctx context.Context, blockNum uint64, result
        // expedite block production.
        if header.GasUsed < oracle.minGasUsed.Uint64() {
                select {
-               case result <- results{nil, nil}:
+               case result <- results{nil, header.BaseFee, nil}:
                case <-quit:
                }
                return
@@ -335,7 +377,7 @@ func (oracle *Oracle) getBlockTips(ctx context.Context, blockNum uint64, result
        // delay in transaction inclusion.
        minTip, err := oracle.backend.MinRequiredTip(ctx, header)
        select {
-       case result <- results{minTip, err}:
+       case result <- results{minTip, header.BaseFee, err}:
        case <-quit:
        }
 }
diff --git a/eth/gasprice/gasprice_test.go b/eth/gasprice/gasprice_test.go
index 4a52849b..fce0cfaa 100644
--- a/eth/gasprice/gasprice_test.go
+++ b/eth/gasprice/gasprice_test.go
@@ -31,17 +31,17 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/event"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
 )

 const testHead = 32
@@ -195,37 +195,6 @@ func applyGasPriceTest(t *testing.T, test suggestTipCapTest) {
        }
 }
-func TestSuggestTipCapNetworkUpgrades(t *testing.T) {
-       tests := map[string]suggestTipCapTest{
-               "launch": {
-                       chainConfig: params.TestLaunchConfig,
-                       expectedTip: big.NewInt(params.LaunchMinGasPrice),
-               },
-               "apricot phase 1": {
-                       chainConfig: params.TestApricotPhase1Config,
-                       expectedTip: big.NewInt(params.ApricotPhase1MinGasPrice),
-               },
-               "apricot phase 2": {
-                       chainConfig: params.TestApricotPhase2Config,
-                       expectedTip: big.NewInt(params.ApricotPhase1MinGasPrice),
-               },
-               "apricot phase 3": {
-                       chainConfig: params.TestApricotPhase3Config,
-                       expectedTip: big.NewInt(0),
-               },
-               "apricot phase 4": {
-                       chainConfig: params.TestApricotPhase4Config,
-                       expectedTip: DefaultMinPrice,
-               },
-       }
-
-       for name, test := range tests {
-               t.Run(name, func(t *testing.T) {
-                       applyGasPriceTest(t, test)
-               })
-       }
-}
-
 func TestSuggestTipCapEmptyExtDataGasUsage(t *testing.T) {
        txTip := big.NewInt(55 * params.GWei)
        applyGasPriceTest(t, suggestTipCapTest{
@@ -255,7 +224,7 @@ func TestSuggestTipCapEmptyExtDataGasUsage(t *testing.T) {
                                b.AddTx(tx)
                        }
                },
-               expectedTip: big.NewInt(2_844_353_281),
+               expectedTip: big.NewInt(11_427_927_927),
        })
 }

@@ -288,7 +257,7 @@ func TestSuggestTipCapSimple(t *testing.T) {
                                b.AddTx(tx)
                        }
                },
-               expectedTip: big.NewInt(2_844_353_281),
+               expectedTip: big.NewInt(11_427_927_927),
        })
 }

@@ -369,7 +338,7 @@ func TestSuggestTipCapSmallTips(t *testing.T) {
                        }
                },
                // NOTE: small tips do not bias estimate
-               expectedTip: big.NewInt(2_844_353_281),
+               expectedTip: big.NewInt(11_427_927_927),
        })
 }

@@ -402,12 +371,12 @@ func TestSuggestTipCapExtDataUsage(t *testing.T) {
                                b.AddTx(tx)
                        }
                },
-               expectedTip: big.NewInt(2_840_938_303),
+               expectedTip: big.NewInt(11_413_453_299),
        })
 }

 func TestSuggestTipCapMinGas(t *testing.T) {
-       txTip := big.NewInt(55 * params.GWei)
+       txTip := big.NewInt(500 * params.GWei)
        applyGasPriceTest(t, suggestTipCapTest{
                chainConfig:     params.TestChainConfig,
                numBlocks:       3,
@@ -438,3 +407,40 @@ func TestSuggestTipCapMinGas(t *testing.T) {
                expectedTip: big.NewInt(0),
        })
 }
+
+// Regression test to ensure that SuggestPrice does not panic prior to activation of ApricotPhase3
+// Note: support for gas estimation without activated hard forks has been deprecated, but we still
+// ensure that the call does not panic.
+func TestSuggestGasPricePreAP3(t *testing.T) {
+       config := Config{
+               Blocks:     20,
+               Percentile: 60,
+       }
+
+       backend := newTestBackend(t, params.TestApricotPhase2Config, 3, nil, func(i int, b *core.BlockGen) {
+               b.SetCoinbase(common.Address{1})
+
+               signer := types.LatestSigner(params.TestApricotPhase2Config)
+               gasPrice := big.NewInt(params.ApricotPhase1MinGasPrice)
+               for j := 0; j < 50; j++ {
+                       tx := types.NewTx(&types.LegacyTx{
+                               Nonce:    b.TxNonce(addr),
+                               To:       &common.Address{},
+                               Gas:      params.TxGas,
+                               GasPrice: gasPrice,
+                               Data:     []byte{},
+                       })
+                       tx, err := types.SignTx(tx, signer, key)
+                       if err != nil {
+                               t.Fatalf("failed to create tx: %s", err)
+                       }
+                       b.AddTx(tx)
+               }
+       })
+       oracle := NewOracle(backend, config)
+
+       _, err := oracle.SuggestPrice(context.Background())
+       if err != nil {
+               t.Fatal(err)
+       }
+}
diff --git a/eth/state_accessor.go b/eth/state_accessor.go
index 1ef40509..a1efb07b 100644
--- a/eth/state_accessor.go
+++ b/eth/state_accessor.go
@@ -32,15 +32,20 @@ import (
        "math/big"
        "time"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/trie"
 )

+// StateAtBlock retrieves the state database associated with a certain block.
+// If no state is locally available for the given block, a number of blocks
+// are attempted to be reexecuted to generate the desired state. The optional
+// base layer statedb can be passed then it's regarded as the statedb of the
+// parent block.
 // Parameters:
 // - block: The block for which we want the state (== state at the stateRoot of the parent)
 // - reexec: The maximum number of blocks to reprocess trying to obtain the desired state
@@ -51,7 +56,7 @@ import (
 //        storing trash persistently
 // - preferDisk: this arg can be used by the caller to signal that even though the 'base' is provided,
 //        it would be preferrable to start from a fresh state, if we have it on disk.
-func (eth *Ethereum) stateAtBlock(block *types.Block, reexec uint64, base *state.StateDB, checkLive bool, preferDisk bool) (statedb *state.StateDB, err error) {
+func (eth *Ethereum) StateAtBlock(block *types.Block, reexec uint64, base *state.StateDB, checkLive bool, preferDisk bool) (statedb *state.StateDB, err error) {
        var (
                current  *types.Block
                database state.Database
@@ -178,7 +183,7 @@ func (eth *Ethereum) stateAtTransaction(block *types.Block, txIndex int, reexec
        }
        // Lookup the statedb of parent block from the live database,
        // otherwise regenerate it on the flight.
-       statedb, err := eth.stateAtBlock(parent, reexec, nil, true, false)
+       statedb, err := eth.StateAtBlock(parent, reexec, nil, true, false)
        if err != nil {
                return nil, vm.BlockContext{}, nil, err
        }
diff --git a/eth/tracers/api.go b/eth/tracers/api.go
index fcff10f0..8ada419a 100644
--- a/eth/tracers/api.go
+++ b/eth/tracers/api.go
@@ -36,19 +36,20 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/internal/ethapi"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/eth/tracers/logger"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/internal/ethapi"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
 )

 const (
@@ -170,7 +171,7 @@ func (api *API) blockByNumberAndHash(ctx context.Context, number rpc.BlockNumber

 // TraceConfig holds extra parameters to trace functions.
 type TraceConfig struct {
-       *vm.LogConfig
+       *logger.Config
        Tracer  *string
        Timeout *string
        Reexec  *uint64
@@ -179,7 +180,7 @@ type TraceConfig struct {
 // TraceCallConfig is the config for traceCall API. It holds one more
 // field to override the state for tracing.
 type TraceCallConfig struct {
-       *vm.LogConfig
+       *logger.Config
        Tracer         *string
        Timeout        *string
        Reexec         *uint64
@@ -188,7 +189,7 @@ type TraceCallConfig struct {

 // StdTraceConfig holds extra parameters to standard-json trace functions.
 type StdTraceConfig struct {
-       vm.LogConfig
+       logger.Config
        Reexec *uint64
        TxHash common.Hash
 }
@@ -570,12 +571,13 @@ func (api *API) traceBlock(ctx context.Context, block *types.Block, config *Trac
        if threads > len(txs) {
                threads = len(txs)
        }
-       blockCtx := core.NewEVMBlockContext(block.Header(), api.chainContext(ctx), nil)
        blockHash := block.Hash()
        for th := 0; th < threads; th++ {
                pend.Add(1)
                go func() {
                        defer pend.Done()
+
+                       blockCtx := core.NewEVMBlockContext(block.Header(), api.chainContext(ctx), nil)
                        // Fetch and execute the next transaction trace tasks
                        for task := range jobs {
                                msg, _ := txs[task.index].AsMessage(signer, block.BaseFee())
@@ -595,6 +597,7 @@ func (api *API) traceBlock(ctx context.Context, block *types.Block, config *Trac
        }
        // Feed the transactions into the tracers and return
        var failed error
+       blockCtx := core.NewEVMBlockContext(block.Header(), api.chainContext(ctx), nil)
        for i, tx := range txs {
                // Send the trace task over for execution
                jobs <- &txTraceTask{statedb: statedb.Copy(), index: i}
@@ -697,10 +700,10 @@ func (api *API) TraceCall(ctx context.Context, args ethapi.TransactionArgs, bloc
        var traceConfig *TraceConfig
        if config != nil {
                traceConfig = &TraceConfig{
-                       LogConfig: config.LogConfig,
-                       Tracer:    config.Tracer,
-                       Timeout:   config.Timeout,
-                       Reexec:    config.Reexec,
+                       Config:  config.Config,
+                       Tracer:  config.Tracer,
+                       Timeout: config.Timeout,
+                       Reexec:  config.Reexec,
                }
        }
        return api.traceTx(ctx, msg, new(Context), vmctx, statedb, traceConfig)
@@ -718,7 +721,7 @@ func (api *API) traceTx(ctx context.Context, message core.Message, txctx *Contex
        )
        switch {
```

```
        case config == nil:
-               tracer = vm.NewStructLogger(nil)
+               tracer = logger.NewStructLogger(nil)
        case config.Tracer != nil:
                // Define a meaningful timeout of a single transaction trace
                timeout := defaultTraceTimeout
@@ -742,7 +745,7 @@ func (api *API) traceTx(ctx context.Context, message core.Message, txctx *Contex
                }

        default:
-               tracer = vm.NewStructLogger(config.LogConfig)
+               tracer = logger.NewStructLogger(config.Config)
        }
        // Run the transaction with tracing enabled.
        vmenv := vm.NewEVM(vmctx, txContext, statedb, api.backend.ChainConfig(), vm.Config{Debug: true, Tracer: tracer, NoBaseFee: true})
@@ -757,7 +760,7 @@ func (api *API) traceTx(ctx context.Context, message core.Message, txctx *Contex

        // Depending on the tracer type, format and return the output.
        switch tracer := tracer.(type) {
-       case *vm.StructLogger:
+       case *logger.StructLogger:
                // If the result contains a revert reason, return it.
                returnVal := fmt.Sprintf("%x", result.Return())
                if len(result.Revert()) > 0 {
@@ -787,6 +790,7 @@ func APIs(backend Backend) []rpc.API {
                        Version:   "1.0",
                        Service:   NewAPI(backend),
                        Public:    false,
+                       Name:      "debug-tracer",
                },
        }
 }
diff --git a/eth/tracers/api_test.go b/eth/tracers/api_test.go
index a76dc10d..713e0c70 100644
--- a/eth/tracers/api_test.go
+++ b/eth/tracers/api_test.go
@@ -30,7 +30,6 @@ import (
        "bytes"
        "context"
        "crypto/ecdsa"
-       "encoding/json"
        "errors"
        "fmt"
        "math/big"
@@ -38,20 +37,20 @@ import (
        "sort"
        "testing"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/internal/ethapi"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/internal/ethapi"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
 )

 var (
@@ -324,147 +323,6 @@ func TestTraceCall(t *testing.T) {
        }
 }

-func TestOverriddenTraceCall(t *testing.T) {
-       t.Parallel()
-
-       // Initialize test accounts
-       accounts := newAccounts(3)
-       genesis := &core.Genesis{Alloc: core.GenesisAlloc{
-               accounts[0].addr: {Balance: big.NewInt(params.Ether)},
-               accounts[1].addr: {Balance: big.NewInt(params.Ether)},
-               accounts[2].addr: {Balance: big.NewInt(params.Ether)},
-       }}
-       genBlocks := 10
-       signer := types.HomesteadSigner{}
-       api := NewAPI(newTestBackend(t, genBlocks, genesis, func(i int, b *core.BlockGen) {
-               // Transfer from account[0] to account[1]
-               //    value: 1000 wei
-               //    fee:   0 wei
-               tx, _ := types.SignTx(types.NewTransaction(uint64(i), accounts[1].addr, big.NewInt(1000), params.TxGas, new(big.Int).Add(b.BaseFee(), big.NewInt(int64(500*params.GWei))), nil), signer, ac
-               b.AddTx(tx)
-       }))
-       randomAccounts, tracer := newAccounts(3), "callTracerJs"
-
-       var testSuite = []struct {
-               blockNumber rpc.BlockNumber
-               call        ethapi.TransactionArgs
-               config      *TraceCallConfig
-               expectErr   error
-               expect      *callTrace
-       }{
-               // Succcessful call with state overriding
-               {
-                       blockNumber: rpc.PendingBlockNumber,
-                       call: ethapi.TransactionArgs{
-                               From:  &randomAccounts[0].addr,
-                               To:    &randomAccounts[1].addr,
-                               Value: (*hexutil.Big)(big.NewInt(1000)),
-                       },
-                       config: &TraceCallConfig{
-                               Tracer: &tracer,
-                               StateOverrides: &ethapi.StateOverride{
-                                       randomAccounts[0].addr: ethapi.OverrideAccount{Balance: newRPCBalance(new(big.Int).Mul(big.NewInt(1), big.NewInt(params.Ether)))},
-                               },
-                       },
-                       expectErr: nil,
-                       expect: &callTrace{
-                               Type:    "CALL",
-                               From:    randomAccounts[0].addr,
-                               To:      randomAccounts[1].addr,
-                               Gas:     newRPCUint64(24979000),
-                               GasUsed: newRPCUint64(0),
-                               Value:   (*hexutil.Big)(big.NewInt(1000)),
-                       },
-               },
-               // Invalid call without state overriding
-               {
```

```
-			blockNumber: rpc.PendingBlockNumber,
-			call: ethapi.TransactionArgs{
-				From:  &randomAccounts[0].addr,
-				To:    &randomAccounts[1].addr,
-				Value: (*hexutil.Big)(big.NewInt(1000)),
-			},
-			config: &TraceCallConfig{
-				Tracer: &tracer,
-			},
-			expectErr: core.ErrInsufficientFunds,
-			expect:    nil,
-		},
-		// Successful simple contract call
-		//
-		// // SPDX-License-Identifier: GPL-3.0
-		//
-		//  pragma solidity >=0.7.0 <0.8.0;
-		//
-		//  /**
-		//   * @title Storage
-		//   * @dev Store & retrieve value in a variable
-		//   */
-		//  contract Storage {
-		//      uint256 public number;
-		//      constructor() {
-		//          number = block.number;
-		//      }
-		//  }
-		{
-			blockNumber: rpc.PendingBlockNumber,
-			call: ethapi.TransactionArgs{
-				From: &randomAccounts[0].addr,
-				To:   &randomAccounts[2].addr,
-				Data: newRPCBytes(common.Hex2Bytes("8381f58a")), // call number()
-			},
-			config: &TraceCallConfig{
-				Tracer: &tracer,
-				StateOverrides: &ethapi.StateOverride{
-					randomAccounts[2].addr: ethapi.OverrideAccount{
-						Code:      newRPCBytes(common.Hex2Bytes("6080604052348015600f57600080fd5b506004361060285760003560e01c80638381f58a14602d575b600080fd5b60336049565b6040518082
-						StateDiff: newStates([]common.Hash{{}}, []common.Hash{common.BigToHash(big.NewInt(123))}),
-					},
-				},
-			},
-			expectErr: nil,
-			expect: &callTrace{
-				Type:    "CALL",
-				From:    randomAccounts[0].addr,
-				To:      randomAccounts[2].addr,
-				Input:   hexutil.Bytes(common.Hex2Bytes("8381f58a")),
-				Output:  hexutil.Bytes(common.BigToHash(big.NewInt(123)).Bytes()),
-				Gas:     newRPCUint64(24978936),
-				GasUsed: newRPCUint64(2283),
-				Value:   (*hexutil.Big)(big.NewInt(0)),
-			},
-		},
-	}
-	for i, testspec := range testSuite {
-		result, err := api.TraceCall(context.Background(), testspec.call, rpc.BlockNumberOrHash{BlockNumber: &testspec.blockNumber}, testspec.config)
-		if testspec.expectErr != nil {
-			if err == nil {
-				t.Errorf("test %d: want error %v, have nothing", i, testspec.expectErr)
-				continue
-			}
-			if !errors.Is(err, testspec.expectErr) {
-				t.Errorf("test %d: error mismatch, want %v, have %v", i, testspec.expectErr, err)
-			}
-		} else {
-			if err != nil {
-				t.Errorf("test %d: want no error, have %v", i, err)
-				continue
-			}
-			ret := new(callTrace)
-			if err := json.Unmarshal(result.(json.RawMessage), ret); err != nil {
-				t.Fatalf("test %d: failed to unmarshal trace result: %v", i, err)
-			}
-			if !jsonEqual(ret, testspec.expect) {
-				// uncomment this for easier debugging
-				//have, _ := json.MarshalIndent(ret, "", " ")
-				//want, _ := json.MarshalIndent(testspec.expect, "", " ")
-				//t.Fatalf("trace mismatch: \nhave %+v\nwant %+v", string(have), string(want))
-				t.Fatalf("trace mismatch: \nhave %+v\nwant %+v", ret, testspec.expect)
-			}
-		}
-	}
-}
-
 func TestTraceTransaction(t *testing.T) {
 	t.Parallel()
 
@@ -629,29 +487,3 @@ func newAccounts(n int) (accounts Accounts) {
 	sort.Sort(accounts)
 	return accounts
 }
-
-func newRPCBalance(balance *big.Int) **hexutil.Big {
-	rpcBalance := (*hexutil.Big)(balance)
-	return &rpcBalance
-}
-
-func newRPCUint64(number uint64) *hexutil.Uint64 {
-	rpcUint64 := hexutil.Uint64(number)
-	return &rpcUint64
-}
-
-func newRPCBytes(bytes []byte) *hexutil.Bytes {
-	rpcBytes := hexutil.Bytes(bytes)
-	return &rpcBytes
-}
-
-func newStates(keys []common.Hash, vals []common.Hash) *map[common.Hash]common.Hash {
-	if len(keys) != len(vals) {
-		panic("invalid input")
-	}
-	m := make(map[common.Hash]common.Hash)
-	for i := 0; i < len(keys); i++ {
-		m[keys[i]] = vals[i]
-	}
-	return &m
-}
diff --git a/eth/tracers/internal/tracetest/calltrace_test.go b/eth/tracers/internal/tracetest/calltrace_test.go
new file mode 100644
index 00000000..2e628832
--- /dev/null
+++ b/eth/tracers/internal/tracetest/calltrace_test.go
@@ -0,0 +1,404 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
```

```
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package tracetest
+
+import (
+       "encoding/json"
+       "io/ioutil"
+       "math/big"
+       "path/filepath"
+       "reflect"
+       "strings"
+       "testing"
+       "unicode"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/ethereum/go-ethereum/common/math"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/eth/tracers"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/tests"
+
+       // Force-load native and js pacakges, to trigger registration
+       _ "github.com/flare-foundation/coreth/eth/tracers/js"
+       _ "github.com/flare-foundation/coreth/eth/tracers/native"
+)
+
+// To generate a new callTracer test, copy paste the makeTest method below into
+// a Geth console and call it with a transaction hash you which to export.
+
+/*
+// makeTest generates a callTracer test by running a prestate reassembled and a
+// call trace run, assembling all the gathered information into a test case.
+var makeTest = function(tx, rewind) {
+  // Generate the genesis block from the block, transaction and prestate data
+  var block   = eth.getBlock(eth.getTransaction(tx).blockHash);
+  var genesis = eth.getBlock(block.parentHash);
+
+  delete genesis.gasUsed;
+  delete genesis.logsBloom;
+  delete genesis.parentHash;
+  delete genesis.receiptsRoot;
+  delete genesis.sha3Uncles;
+  delete genesis.size;
+  delete genesis.transactions;
+  delete genesis.transactionsRoot;
+  delete genesis.uncles;
+
+  genesis.gasLimit  = genesis.gasLimit.toString();
+  genesis.number    = genesis.number.toString();
+  genesis.timestamp = genesis.timestamp.toString();
+
+  genesis.alloc = debug.traceTransaction(tx, {tracer: "prestateTracer", rewind: rewind});
+  for (var key in genesis.alloc) {
+    genesis.alloc[key].nonce = genesis.alloc[key].nonce.toString();
+  }
+  genesis.config = admin.nodeInfo.protocols.eth.config;
+
+  // Generate the call trace and produce the test input
+  var result = debug.traceTransaction(tx, {tracer: "callTracer", rewind: rewind});
+  delete result.time;
+
+  console.log(JSON.stringify({
+    genesis: genesis,
+    context: {
+      number:     block.number.toString(),
+      difficulty: block.difficulty,
+      timestamp:  block.timestamp.toString(),
+      gasLimit:   block.gasLimit.toString(),
+      miner:      block.miner,
+    },
+    input:  eth.getRawTransaction(tx),
+    result: result,
+  }, null, 2));
+}
+*/
+
+type callContext struct {
+       Number     math.HexOrDecimal64   `json:"number"`
+       Difficulty *math.HexOrDecimal256 `json:"difficulty"`
+       Time       math.HexOrDecimal64   `json:"timestamp"`
+       GasLimit   math.HexOrDecimal64   `json:"gasLimit"`
+       Miner      common.Address        `json:"miner"`
+}
+
+// callTrace is the result of a callTracer run.
+type callTrace struct {
+       Type    string          `json:"type"`
+       From    common.Address  `json:"from"`
+       To      common.Address  `json:"to"`
+       Input   hexutil.Bytes   `json:"input"`
+       Output  hexutil.Bytes   `json:"output"`
+       Gas     *hexutil.Uint64 `json:"gas,omitempty"`
+       GasUsed *hexutil.Uint64 `json:"gasUsed,omitempty"`
+       Value   *hexutil.Big    `json:"value,omitempty"`
+       Error   string          `json:"error,omitempty"`
+       Calls   []callTrace     `json:"calls,omitempty"`
+}
+
+// callTracerTest defines a single test to check the call tracer against.
+type callTracerTest struct {
+       Genesis *core.Genesis `json:"genesis"`
+       Context *callContext  `json:"context"`
+       Input   string        `json:"input"`
+       Result  *callTrace    `json:"result"`
+}
+
+// Iterates over all the input-output datasets in the tracer test harness and
+// runs the JavaScript tracers against them.
+func TestCallTracerLegacy(t *testing.T) {
+       testCallTracer("callTracerLegacy", "call_tracer_legacy", t)
+}
+
+func TestCallTracerJs(t *testing.T) {
+       testCallTracer("callTracerJs", "call_tracer", t)
+}
+
+func TestCallTracerNative(t *testing.T) {
+       testCallTracer("callTracer", "call_tracer", t)
+}
+
+func testCallTracer(tracerName string, dirPath string, t *testing.T) {
+       files, err := ioutil.ReadDir(filepath.Join("testdata", dirPath))
+       if err != nil {
+               t.Fatalf("failed to retrieve tracer test suite: %v", err)
+       }
+       for _, file := range files {
+               if !strings.HasSuffix(file.Name(), ".json") {
+                       continue
+               }
+               file := file // capture range variable
+               t.Run(camel(strings.TrimSuffix(file.Name(), ".json")), func(t *testing.T) {
+                       t.Parallel()
+
+                       var (
+                               test = new(callTracerTest)
+                               tx   = new(types.Transaction)
+                       )
+                       // Call tracer test found, read if from disk
+                       if blob, err := ioutil.ReadFile(filepath.Join("testdata", dirPath, file.Name())); err != nil {
+                               t.Fatalf("failed to read testcase: %v", err)
+                       } else if err := json.Unmarshal(blob, test); err != nil {
+                               t.Fatalf("failed to parse testcase: %v", err)
+                       }
+                       if err := rlp.DecodeBytes(common.FromHex(test.Input), tx); err != nil {
+                               t.Fatalf("failed to parse testcase input: %v", err)
+                       }
+                       // Configure a blockchain with the given prestate
+                       var (
+                               signer    = types.MakeSigner(test.Genesis.Config, new(big.Int).SetUint64(uint64(test.Context.Number)), new(big.Int).SetUint64(uint64(test.Context.Time)))
+                               origin, _ = signer.Sender(tx)
+                               txContext = vm.TxContext{
+                                       Origin:   origin,
+                                       GasPrice: tx.GasPrice(),
+                               }
+                               context = vm.BlockContext{
+                                       CanTransfer: core.CanTransfer,
+                                       Transfer:    core.Transfer,
+                                       Coinbase:    test.Context.Miner,
+                                       BlockNumber: new(big.Int).SetUint64(uint64(test.Context.Number)),
+                                       Time:        new(big.Int).SetUint64(uint64(test.Context.Time)),
+                                       Difficulty:  (*big.Int)(test.Context.Difficulty),
+                                       GasLimit:    uint64(test.Context.GasLimit),
+                               }
+                               _, statedb = tests.MakePreState(rawdb.NewMemoryDatabase(), test.Genesis.Alloc, false)
+                       )
+                       tracer, err := tracers.New(tracerName, new(tracers.Context))
+                       if err != nil {
+                               t.Fatalf("failed to create call tracer: %v", err)
+                       }
+                       evm := vm.NewEVM(context, txContext, statedb, test.Genesis.Config, vm.Config{Debug: true, Tracer: tracer})
+                       msg, err := tx.AsMessage(signer, nil)
+                       if err != nil {
+                               t.Fatalf("failed to prepare transaction for tracing: %v", err)
+                       }
+                       st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
+                       if _, err = st.TransitionDb(); err != nil {
+                               t.Fatalf("failed to execute transaction: %v", err)
+                       }
+                       // Retrieve the trace result and compare against the etalon
+                       res, err := tracer.GetResult()
+                       if err != nil {
+                               t.Fatalf("failed to retrieve trace result: %v", err)
+                       }
+                       ret := new(callTrace)
+                       if err := json.Unmarshal(res, ret); err != nil {
+                               t.Fatalf("failed to unmarshal trace result: %v", err)
+                       }
+
+                       if !jsonEqual(ret, test.Result) {
+                               // uncomment this for easier debugging
+                               //have, _ := json.MarshalIndent(ret, "", " ")
+                               //want, _ := json.MarshalIndent(test.Result, "", " ")
+                               //t.Fatalf("trace mismatch: \nhave %+v\nwant %+v", string(have), string(want))
+                               t.Fatalf("trace mismatch: \nhave %+v\nwant %+v", ret, test.Result)
+                       }
+               })
+       }
+}
+
+// jsonEqual is similar to reflect.DeepEqual, but does a 'bounce' via json prior to
+// comparison
+func jsonEqual(x, y interface{}) bool {
+       xTrace := new(callTrace)
+       yTrace := new(callTrace)
+       if xj, err := json.Marshal(x); err == nil {
+               json.Unmarshal(xj, xTrace)
+       } else {
+               return false
+       }
+       if yj, err := json.Marshal(y); err == nil {
+               json.Unmarshal(yj, yTrace)
+       } else {
+               return false
+       }
+       return reflect.DeepEqual(xTrace, yTrace)
+}
+
+// camel converts a snake cased input string into a camel cased output.
+func camel(str string) string {
+       pieces := strings.Split(str, "_")
+       for i := 1; i < len(pieces); i++ {
+               pieces[i] = string(unicode.ToUpper(rune(pieces[i][0]))) + pieces[i][1:]
+       }
+       return strings.Join(pieces, "")
+}
+func BenchmarkTracers(b *testing.B) {
+       files, err := ioutil.ReadDir(filepath.Join("testdata", "call_tracer"))
+       if err != nil {
+               b.Fatalf("failed to retrieve tracer test suite: %v", err)
+       }
+       for _, file := range files {
+               if !strings.HasSuffix(file.Name(), ".json") {
+                       continue
+               }
+               file := file // capture range variable
+               b.Run(camel(strings.TrimSuffix(file.Name(), ".json")), func(b *testing.B) {
+                       blob, err := ioutil.ReadFile(filepath.Join("testdata", "call_tracer", file.Name()))
+                       if err != nil {
+                               b.Fatalf("failed to read testcase: %v", err)
+                       }
+                       test := new(callTracerTest)
+                       if err := json.Unmarshal(blob, test); err != nil {
+                               b.Fatalf("failed to parse testcase: %v", err)
+                       }
+                       benchTracer("callTracerNative", test, b)
+               })
+       }
+}
+
+func benchTracer(tracerName string, test *callTracerTest, b *testing.B) {
+       // Configure a blockchain with the given prestate
+       tx := new(types.Transaction)
+       if err := rlp.DecodeBytes(common.FromHex(test.Input), tx); err != nil {
+               b.Fatalf("failed to parse testcase input: %v", err)
+       }
+       signer := types.MakeSigner(test.Genesis.Config, new(big.Int).SetUint64(uint64(test.Context.Number)), new(big.Int).SetUint64(uint64(test.Context.Time)))
+       msg, err := tx.AsMessage(signer, nil)
+       if err != nil {
+               b.Fatalf("failed to prepare transaction for tracing: %v", err)
+       }
+       origin, _ := signer.Sender(tx)
+       txContext := vm.TxContext{
+               Origin:   origin,
+               GasPrice: tx.GasPrice(),
+       }
+       context := vm.BlockContext{
+               CanTransfer: core.CanTransfer,
+               Transfer:    core.Transfer,
+               Coinbase:    test.Context.Miner,
+               BlockNumber: new(big.Int).SetUint64(uint64(test.Context.Number)),
+               Time:        new(big.Int).SetUint64(uint64(test.Context.Time)),
+               Difficulty:  (*big.Int)(test.Context.Difficulty),
+               GasLimit:    uint64(test.Context.GasLimit),
+       }
+       _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), test.Genesis.Alloc, false)
+
+       b.ReportAllocs()
+       b.ResetTimer()
+       for i := 0; i < b.N; i++ {
+               tracer, err := tracers.New(tracerName, new(tracers.Context))
+               if err != nil {
+                       b.Fatalf("failed to create call tracer: %v", err)
+               }
+               evm := vm.NewEVM(context, txContext, statedb, test.Genesis.Config, vm.Config{Debug: true, Tracer: tracer})
+               snap := statedb.Snapshot()
+               st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
+               if _, err = st.TransitionDb(); err != nil {
+                       b.Fatalf("failed to execute transaction: %v", err)
+               }
+               if _, err = tracer.GetResult(); err != nil {
+                       b.Fatal(err)
+               }
+               statedb.RevertToSnapshot(snap)
+       }
+}
+
+// TestZeroValueToNotExitCall tests the calltracer(s) on the following:
+// Tx to A, A calls B with zero value. B does not already exist.
+// Expected: that enter/exit is invoked and the inner call is shown in the result
+func TestZeroValueToNotExitCall(t *testing.T) {
+       var to = common.HexToAddress("0x00000000000000000000000000000000deadbeef")
+       privkey, err := crypto.HexToECDSA("0000000000000000deadbeef00000000000000000000000000000000deadbeef")
+       if err != nil {
+               t.Fatalf("err %v", err)
+       }
+       signer := types.NewEIP155Signer(big.NewInt(1))
+       tx, err := types.SignNewTx(privkey, signer, &types.LegacyTx{
+               GasPrice: big.NewInt(0),
+               Gas:      50000,
+               To:       &to,
+       })
+       if err != nil {
+               t.Fatalf("err %v", err)
+       }
+       origin, _ := signer.Sender(tx)
+       txContext := vm.TxContext{
+               Origin:   origin,
+               GasPrice: big.NewInt(1),
+       }
+       context := vm.BlockContext{
+               CanTransfer: core.CanTransfer,
+               Transfer:    core.Transfer,
+               Coinbase:    common.Address{},
+               BlockNumber: new(big.Int).SetUint64(8000000),
+               Time:        new(big.Int).SetUint64(5),
+               Difficulty:  big.NewInt(0x30000),
+               GasLimit:    uint64(6000000),
+       }
+       var code = []byte{
+               byte(vm.PUSH1), 0x0, byte(vm.DUP1), byte(vm.DUP1), byte(vm.DUP1), // in and outs zero
+               byte(vm.DUP1), byte(vm.PUSH1), 0xff, byte(vm.GAS), // value=0,address=0xff, gas=GAS
+               byte(vm.CALL),
+       }
+       var alloc = core.GenesisAlloc{
+               to: core.GenesisAccount{
+                       Nonce: 1,
+                       Code:  code,
+               },
+               origin: core.GenesisAccount{
+                       Nonce:   0,
+                       Balance: big.NewInt(500000000000000),
+               },
+       }
+       _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), alloc, false)
+       // Create the tracer, the EVM environment and run it
+       tracer, err := tracers.New("callTracer", nil)
+       if err != nil {
+               t.Fatalf("failed to create call tracer: %v", err)
+       }
+       evm := vm.NewEVM(context, txContext, statedb, params.AvalancheMainnetChainConfig, vm.Config{Debug: true, Tracer: tracer})
+       msg, err := tx.AsMessage(signer, nil)
+       if err != nil {
+               t.Fatalf("failed to prepare transaction for tracing: %v", err)
+       }
+       st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
+       if _, err = st.TransitionDb(); err != nil {
+               t.Fatalf("failed to execute transaction: %v", err)
+       }
+       // Retrieve the trace result and compare against the etalon
+       res, err := tracer.GetResult()
+       if err != nil {
+               t.Fatalf("failed to retrieve trace result: %v", err)
+       }
+       have := new(callTrace)
+       if err := json.Unmarshal(res, have); err != nil {
+               t.Fatalf("failed to unmarshal trace result: %v", err)
+       }
+       wantStr := `{"type":"CALL","from":"0x682a80a6f560eec50d54e63cbeda1c324c5f8d1b","to":"0x00000000000000000000000000000000deadbeef","value":"0x0","gas":"0x7148","gasUsed":"0x2d0","input":"0x","outpu
+       want := new(callTrace)
+       json.Unmarshal([]byte(wantStr), want)
+       if !jsonEqual(have, want) {
+               t.Error("have != want")
+       }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/create.json b/eth/tracers/internal/tracetest/testdata/call_tracer/create.json
new file mode 100644
index 00000000..8699bf3e
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/create.json
@@ -0,0 +1,58 @@
```

```
+{
+  "context": {
+    "difficulty": "3755480783",
+    "gasLimit": "5401723",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "number": "2294702",
+    "timestamp": "1513676146"
+  },
+  "genesis": {
+    "alloc": {
+      "0x13e4acefe6a6700604929946e70e6443e4e73447": {
+        "balance": "0xcf3e0938579f000",
+        "code": "0x",
+        "nonce": "9",
+        "storage": {}
+      },
+      "0x7dc9c9730689ff0b0fd506c67db815f12d90a448": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3757315409",
+    "extraData": "0x566961425443",
+    "gasLimit": "5406414",
+    "hash": "0xae107f592eebdd9ff8d6ba00363676096e6afb0e1007a7d3d0af88173077378d",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "mixHash": "0xc927aa05a38bc3de864e95c33b3ae559d3f39c4ccd51cef6f113f9c50ba0caf1",
+    "nonce": "0x93363bbd2c95f410",
+    "number": "2294701",
+    "stateRoot": "0x6b6737d5bde8058990483e915866bd1578014baeff57bd5e4ed228a2bfad635c",
+    "timestamp": "1513676127",
+    "totalDifficulty": "7160808139332585"
+  },
+  "input": "0xf907ef098504e3b29200830897be8080b9079c606060405260405160208061077c833981016040528080519060200190919050506001600090549061010000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffff
+  "result": {
+    "from": "0x13e4acefe6a6700604929946e70e6443e4e73447",
+    "gas": "0x5e106",
+    "gasUsed": "0x5e106",
+    "input": "0x606060405260405160208061077c833981016040528080519060200190919050506001600090549061010000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ff
+    "output": "0x60606040526004361061008357600035 7c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806305e4382a146100855780631c02708d146100ae5780632e1a7d4d146100c35780635114cb5
+    "to": "0x7dc9c9730689ff0b0fd506c67db815f12d90a448",
+    "type": "CREATE",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/deep_calls.json b/eth/tracers/internal/tracetest/testdata/call_tracer/deep_calls.json
new file mode 100644
index 00000000..0353d4cf
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/deep_calls.json
@@ -0,0 +1,415 @@
+{
+  "context": {
+    "difficulty": "117066904",
+    "gasLimit": "4712384",
+    "miner": "0x1977c248e1014cc103929dd7f154199c916e39ec",
+    "number": "25001",
+    "timestamp": "1479891545"
+  },
+  "genesis": {
+    "alloc": {
+      "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38": {
+        "balance": "0x0",
+        "code": "0x606060405236156100825760e060020a600035046302d05d3f811461008a5780630accce061461009c5780631ab9075a146100c757806331ed274614610102578063645a3b72146101335780637 72fdae314610155578063a7f4377
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0x2cccf5e0538493c235d1c5ef6580f77d99e91396": {
+        "balance": "0x0",
+        "code": "0x606060405236156100775760e060020a600035046302d05d3f811461007f57806313bc6d4b146100915780633688a877146100b95780635188f9961461012f5780637eadc976146101545780638ad79680146101d3578063a43e04d
+        "nonce": "1",
+        "storage": {
+          "0x0684ac65a9fa32414dda56996f4183597d695987fdb82b145d722743891a6fe8": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+          "0x1cd76f78169a420d99346e3501dd3e541622c38a226f9b63e01cfebc69879dc7": "0x000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+          "0x8e54a4494fe5da016bfc01363f4f6cdc91013bb5434bd2a4a3359f13a23afa2f": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+          "0x94edf7f600ba56655fd65fca1f1424334ce369326c1dc3e53151dcd1ad06bc13": "0x0000000000000000000000000000000000000000000000000000000000000001",
+          "0xbbee47108b275f55f98482c6800f6372165e88b0330d3f5dae6419df4734366c": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+          "0xd38c0c4e84de118cfdcc775130155d83b8bbaaf23dc7f3c83a626b10473213bd": "0x0000000000000000000000000000000000000000000000000000000000000001",
+          "0xfb3aa5c655c2ec9d40609401f88d505d1da61afaa550e36ef5da0509ada257ba": "0x0000000000000000000000007986bad81f4cbd9317f5a46861437dae58d69113"
+        }
+      },
+      "0x3e9286eafa2db8101246c2131c09b49080d00690": {
+        "balance": "0x0",
+        "code": "0x606060405236156100cf5760e060020a600035046302d05d3f81146100d7578063056d4470146100e957806316c66cc61461010c5780631ab9075a146101935780633ae1005c146101ce57806358541662146101fe5780635ed61af
+        "nonce": "16",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0x70c9217d814985faef62b124420f8dfbddd96433": {
+        "balance": "0x4ef436dcbda6cd4a",
+        "code": "0x",
+        "nonce": "1634",
+        "storage": {}
+      },
+      "0x7986bad81f4cbd9317f5a46861437dae58d69113": {
+        "balance": "0x0",
+        "code": "0x6060604052361561008d5760e060020a600035046302d05d3f811461009557806316c66cc6146100a75780631ab9075a146100d7578063213fe2b7146101125780639859387b1461013f578063988db79c1461015e578063a7f4377
+        "nonce": "7",
+        "storage": {
+          "0xffc4df2d4f3d2cffad590bed6296406ab7926ca9e74784f74a95191fa069a174": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433"
+        }
+      },
+      "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f": {
+        "balance": "0x0",
+        "code": "0x606060405236156100ae5760e060020a600035046302d05d3f81146100b65780631ab9075a146100c85780632b68bb2d146101035780634cc927d7146101c557806351a34eb81461028e57806356ccb6f0146103545780635928d37
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0xc212e03b9e060e36facad5fd8f4435412ca22e6b": {
+        "balance": "0x0",
+        "code": "0x606060405236156101745760e060020a600035046302d05d3f811461017c57806304a7fdbc1461018e5780630e90f957146101fb5780630fb5a6b41461021257806314baa1b61461021b57806317fc45e21461023a5780632b09692
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000000000000000000000000000000000000000006195",
+          "0x0000000000000000000000000000000000000000000000000000000000000004": "0x5842545553440000000000000000000000000000000000000000000000000000",
+          "0x0000000000000000000000000000000000000000000000000000000000000005": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433",
+          "0x0000000000000000000000000000000000000000000000000000000000000006": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+          "0x000000000000000000000000000000000000000000000000000000000000000b": "0x0000000000000000000000000000000000000000000000283c7b9181eca20000"
+        }
+      },
+      "0xcf00ffd997ad14939736f026006498e3f099baaf": {
+        "balance": "0x0",
+        "code": "0x606060405236156100cf5760e060020a600035046302d05d3f81146100d7578063031e7f5d146100e95780631ab9075a1461010b5780632243118a1461014657806327aad68a1461016557806338a699a4146101da5780635188f99
+        "nonce": "3",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x3571d73f14f31a1463bd0a2f92f7fde1653d4e1ead7aedf4b0a5df02f16092ab": "0x0000000000000000000000000000000000000000000007d634e4c55188be0000",
+          "0x4e64fe2d1b72d95a0a31945cc6e4f4e524ac5ad56d6bd44a85ec7bc9cc0462c0": "0x000000000000000000000000000000000000000000000002b5e3af16b1880000"
+        }
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "117124093",
+    "extraData": "0xd5830105008650617269747986312e31322e31826d61",
+    "gasLimit": "4707788",
+    "hash": "0xad325e4c49145fb7a4058a68ac741cc8607a71114e23fc88083c7e881dd653e7",
+    "miner": "0x00714b9ac97fd6bd9325a059a70c9b9fa94ce050",
+    "mixHash": "0x0af918f65cb4af04b608fc1f14a849707696986a0e7049e97ef3981808bcc65f",
+    "nonce": "0x38dee147326a8d40",
+    "number": "25000",
+    "stateRoot": "0xc5d6bbcd46236fcdcc80b332ffaaa5476b980b01608f9708408cfef01b58bd5b",
+    "timestamp": "1479891517",
+    "totalDifficulty": "1895410389427"
+  },
+  "input": "0xf88b8206628504a817c8008303d09094c212e03b9e060e36facad5fd8f4435412ca22e6b80a451a34eb80000000000000000000000000000000000000000000000280faf689c35ac00002aa0a7ee5b7877811bf671d121b40569462e7226
+  "result": {
+    "calls": [
+      {
+        "from": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+        "gas": "0x31217",
+        "gasUsed": "0x334",
+        "input": "0xe16c7d98636f6e747261637461706900000000000000000000000000000000000000000000",
+        "output": "0x000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+        "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+        "type": "CALL",
+        "value": "0x0"
+      },
+      {
+        "calls": [
+          {
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x2a68d",
+            "gasUsed": "0x334",
+            "input": "0xe16c7d98636f6e747261637463746c000000000000000000000000000000000000000000",
+            "output": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+            "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "calls": [
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x23ac9",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d98636f6e7472616374646200000000000000000000000000000000000000000000",
+                "output": "0x0000000000000000000000007986bad81f4cbd9317f5a46861437dae58d69113",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x23366",
+                "gasUsed": "0x273",
+                "input": "0x16c66cc6000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                "to": "0x7986bad81f4cbd9317f5a46861437dae58d69113",
+                "type": "CALL",
+                "value": "0x0"
+              }
+            ],
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x29f35",
+            "gasUsed": "0xf8d",
+            "input": "0x16c66cc6000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b",
+            "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+            "to": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x28a9e",
+            "gasUsed": "0x334",
+            "input": "0xe16c7d98636f6e747261637463746c000000000000000000000000000000000000000000",
+            "output": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+            "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "calls": [
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x21d79",
+                "gasUsed": "0x24d",
+                "input": "0x13bc6d4b000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+                "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x2165b",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d986d61726b6574646200000000000000000000000000000000000000000000000000",
+                "output": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "calls": [
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x1a8e8",
+                    "gasUsed": "0x24d",
+                     "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                     "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                     "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                     "type": "CALL",
+                     "value": "0x0"
+                   },
+                   {
+                     "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                     "gas": "0x1a2c6",
+                     "gasUsed": "0x3cb",
+                     "input": "0xc9503fe2",
+                     "output": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+                     "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                     "type": "CALL",
+                     "value": "0x0"
+                   },
+                   {
+                     "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                     "gas": "0x19b72",
+                     "gasUsed": "0x3cb",
+                     "input": "0xc9503fe2",
+                     "output": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+                     "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                     "type": "CALL",
+                     "value": "0x0"
+                   },
+                   {
+                     "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                     "gas": "0x19428",
+                     "gasUsed": "0x305",
+                     "input": "0x6f265b93",
+                     "output": "0x0000000000000000000000000000000000000000000000283c7b9181eca20000",
+                     "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                     "type": "CALL",
+                     "value": "0x0"
+                   },
+                   {
+                     "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                     "gas": "0x18d45",
+                     "gasUsed": "0x229",
+                     "input": "0x2e94420f",
+                     "output": "0x5842455553440000000000000000000000000000000000000000000000000000",
+                     "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                     "type": "CALL",
+                     "value": "0x0"
+                   },
+                   {
+                     "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                     "gas": "0x1734e",
+                     "gasUsed": "0x229",
+                     "input": "0x2e94420f",
+                     "output": "0x5842455553440000000000000000000000000000000000000000000000000000",
+                     "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                     "type": "CALL",
+                     "value": "0x0"
+                   }
+                 ],
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x20ee1",
+                 "gasUsed": "0x5374",
+                 "input": "0x581d5d60000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b0000000000000000000000000000000000000000000000280faf689c35ac0000",
+                 "output": "0x",
+                 "to": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x1b6c1",
+                 "gasUsed": "0x334",
+                 "input": "0xe16c7d986c6f676d677200000000000000000000000000000000000000000000000000000",
+                 "output": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                 "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x1af69",
+                 "gasUsed": "0x229",
+                 "input": "0x2e94420f",
+                 "output": "0x5842455553440000000000000000000000000000000000000000000000000000",
+                 "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "calls": [
+                   {
+                     "from": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                     "gas": "0x143a5",
+                     "gasUsed": "0x24d",
+                     "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                     "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                     "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                     "type": "CALL",
+                     "value": "0x0"
+                   }
+                 ],
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x1a91d",
+                 "gasUsed": "0x12fa",
+                 "input": "0x0accce0600000000000000000000000000000000000000000000000000000000000000025842455553440000000000000000000000000000000000000000000000000000000000000000000000000000c212e03b9e060e
+                 "output": "0x",
+                 "to": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x19177",
+                 "gasUsed": "0x334",
+                 "input": "0xe16c7d986c6f676d677200000000000000000000000000000000000000000000000000000",
+                 "output": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                 "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x18a22",
+                 "gasUsed": "0x229",
+                 "input": "0x2e94420f",
+                 "output": "0x5842455553440000000000000000000000000000000000000000000000000000",
+                 "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                 "type": "CALL",
+                 "value": "0x0"
+               },
+               {
+                 "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                 "gas": "0x18341",
+                 "gasUsed": "0x334",
+                 "input": "0xe16c7d986d61726b657464000000000000000000000000000000000000000000000000000",
+                 "output": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x17bec",
+                "gasUsed": "0x229",
+                "input": "0x2e94420f",
+                "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x1764e",
+                "gasUsed": "0x45c",
+                "input": "0xf92eb7745842545553440000000000000000000000000000000000000000000000000000",
+                "output": "0x00000000000000000000000000000000000000000000002816d180e30c390000",
+                "to": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "calls": [
+                  {
+                    "from": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                    "gas": "0x108ba",
+                    "gasUsed": "0x24d",
+                    "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                    "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                    "type": "CALL",
+                    "value": "0x0"
+                  }
+                ],
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x16e62",
+                "gasUsed": "0xebb",
+                "input": "0x645a3b72584254555344000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002816d180e30c390000",
+                "output": "0x",
+                "to": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                "type": "CALL",
+                "value": "0x0"
+              }
+            ],
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x283b9",
+            "gasUsed": "0xc51c",
+            "input": "0x949ae479000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b0000000000000000000000000000000000000000000000280faf689c35ac0000",
+            "output": "0x",
+            "to": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+            "type": "CALL",
+            "value": "0x0"
+          }
+        ],
+        "from": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+        "gas": "0x30b4a",
+        "gasUsed": "0xedb7",
+        "input": "0x51a34eb80000000000000000000000000000000000000000000000280faf689c35ac0000",
+        "output": "0x",
+        "to": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "from": "0x70c9217d814985faef62b124420f8dfbddd96433",
+    "gas": "0x37b38",
+    "gasUsed": "0x12bb3",
+    "input": "0x51a34eb80000000000000000000000000000000000000000000000280faf689c35ac0000",
+    "output": "0x",
+    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/delegatecall.json b/eth/tracers/internal/tracetest/testdata/call_tracer/delegatecall.json
new file mode 100644
index 00000000..f7ad6df5
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/delegatecall.json
@@ -0,0 +1,97 @@
+{
+  "context": {
+    "difficulty": "31927752",
+    "gasLimit": "4707788",
+    "miner": "0x5659922ce141eedbc2733678f9806c77b4eebee8",
+    "number": "11495",
+    "timestamp": "1479735917"
+  },
+  "genesis": {
+    "alloc": {
+      "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff": {
+        "balance": "0x0",
+        "code": "0x606060405236156100825760e060020a60003504630a0313a981146100875780630a3b0a4f146101095780630cd40fea1461021257806329092d0e1461021f5780634cd06a5f146103295780635dbe47e8146103395780637a9e541
+        "nonce": "1",
+        "storage": {
+          "0x4d140b25abf3c71052885c66f73ce07cff141c1afabffdaf5cba04d625b7ebcc": "0x0000000000000000000000000000000000000000000000000000000000000001"
+        }
+      },
+      "0x269296dddce321a6bcbaa2f0181127593d732cba": {
+        "balance": "0x0",
+        "code": "0x606060405236156101275760e060020a60003504630cd40fea811461012c578063173825d9146101395780631849cb5a146101c7578063285791371461030f5780632a58b3301461033f5780632cb0d48a146103565780632f54bf6
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x000113204f5d64c28326fd7bd05fd4ea855302d7f2ff00000000000000000000"
+        }
+      },
+      "0x42b02b5deeb78f34cd5ac896473b63e6c99a71a2": {
+        "balance": "0x0",
+        "code": "0x6504032353da7150606060405236156100695760e060020a60003504631bf7509d811461006e57806321ce24d4146100815780633335556e84146100ec578063685a1f3c146101035780637d65837a1461011757806389489a8714610
+        "nonce": "1",
+        "storage": {}
+      },
+      "0xa529806c67cc6486d4d62024471772f47f6fd672": {
+        "balance": "0x67820e39ac8fe9800",
+        "code": "0x",
+        "nonce": "68",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "31912170",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712388",
+    "hash": "0x0855914bdc581bccdc62591fd438498386ffb59ea4d5361ed5c3702e26e2c72f",
+    "miner": "0x334391aa808257952a462d1475562ee2106a6c90",
+    "mixHash": "0x64bb70b8ca883cadb8fbbda2c70a861612407864089ed87b98e5de20acceada6",
+    "nonce": "0x684129f283aaef18",
+    "number": "11494",
+    "stateRoot": "0x7057f31fe3dab1d620771adad35224aae43eb70e94861208bc84c557ff5b9d10",
+    "timestamp": "1479735912",
+    "totalDifficulty": "90744064339"
+  },
+  "input": "0xf889448504a817c800832dc6c094269296dddce321a6bcbaa2f0181127593d732cba80a47065cb48000000000000000000000000001523e55a1ca4efbae03355775ae89f8d7699ad9e29a080ed81e4c5e9971a730efab4885566e2c868cd80
+  "result": {
+    "calls": [
+      {
+        "calls": [
+          {
+            "from": "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff",
+            "gas": "0x2bf459",
+            "gasUsed": "0x2aa",
+            "input": "0x7d65837a0000000000000000000000000000000000000000000000000000000000000000000000000000000a529806c67cc6486d4d62024471772f47f6fd672",
+            "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+            "to": "0x42b02b5deeb78f34cd5ac896473b63e6c99a71a2",
+            "type": "DELEGATECALL"
+          }
+        ],
+        "from": "0x269296dddce321a6bcbaa2f0181127593d732cba",
+        "gas": "0x2cae73",
+        "gasUsed": "0xa9d",
+        "input": "0x5dbe47e8000000000000000000000000a529806c67cc6486d4d62024471772f47f6fd672",
+        "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+        "to": "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "from": "0xa529806c67cc6486d4d62024471772f47f6fd672",
+    "gas": "0x2d6e28",
+    "gasUsed": "0x64bd",
+    "input": "0x7065cb48000000000000000000000000001523e55a1ca4efbae03355775ae89f8d7699ad9e",
+    "output": "0x",
+    "to": "0x269296dddce321a6bcbaa2f0181127593d732cba",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/inner_create_oog_outer_throw.json b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_create_oog_outer_throw.json
new file mode 100644
index 00000000..9395eb40
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_create_oog_outer_throw.json
@@ -0,0 +1,77 @@
+{
+  "context": {
+    "difficulty": "3451177886",
+    "gasLimit": "4709286",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2290744",
+    "timestamp": "1513616439"
+  },
+  "genesis": {
+    "alloc": {
+      "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a": {
+        "balance": "0x0",
+        "code": "0x606060405263ffffffff60e060020a6000350416633b91f506811461005057806355bb478081461007157806355f51fca01461008c578063bc7647a9146100ad578063f1bd0d7a146100c8575b610000565b346100005761006f600160
+        "nonce": "789",
+        "storage": {
+          "0xfe9ec0542a1c009be8b1f3acf43af97100ffff42eb736850fb038fa1151ad4d9": "0x000000000000000000000000e4a13bc304682a903e9472f469c33801dd18d9e8"
+        }
+      },
+      "0x5cb4a6b902fcb21588c86c3517e797b07cdaadb9": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      },
+      "0xe4a13bc304682a903e9472f469c33801dd18d9e8": {
+        "balance": "0x33c763c929f62c4f",
+        "code": "0x",
+        "nonce": "14",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3451177886",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4713874",
+    "hash": "0x5d52a672417cd1269bf4f7095e25dcbf837747bba908cd5ef809dc1bd06144b5",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x01a12845ed546b94a038a7a03e8df8d7952024ed41ccb3db7a7ade4abc290ce1",
+    "nonce": "0x28c446f1cb9748c1",
+    "number": "2290743",
+    "stateRoot": "0x4898aceede76739daef76448a367d10015a2c022c9e7909b99a10fbf6fb16708",
+    "timestamp": "1513616414",
+    "totalDifficulty": "7146523769022564"
+  },
+  "input": "0xf8aa0e8509502f9000830493e0941d3ddf7caf024f253487e18bc4a15b1a360c170a80b8443b91f506000000000000000000000000a14bdd7e5666d784dcce98ad24d383a6b1cd4182000000000000000000000000e4a13bc304682a903e
+  "result": {
+    "calls": [
+      {
+        "error": "contract creation code storage out of gas",
+        "from": "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a",
+        "gas": "0x39ff0",
+        "gasUsed": "0x39ff0",
+        "input": "0x606060405234620000057604051602080620001fd283398101604052515b805b600a8054600160a060020a031916600160a060020a0383161790555b506001600d819055600e81905560408051808201909152600c8082527f566f
+        "type": "CREATE",
+        "value": "0x0"
+      }
+    ],
+    "error": "invalid jump destination",
+    "from": "0xe4a13bc304682a903e9472f469c33801dd18d9e8",
+    "gas": "0x435c8",
+    "gasUsed": "0x435c8",
+    "input": "0x3b91f506000000000000000000000000a14bdd7e5666d784dcce98ad24d383a6b1cd4182000000000000000000000000e4a13bc304682a903e9472f469c33801dd18d9e8",
+    "to": "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/inner_instafail.json b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_instafail.json
new file mode 100644
index 00000000..6e221b3c
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_instafail.json
```

```
@@ -0,0 +1,63 @@
+{
+  "genesis": {
+    "difficulty": "117067574",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712380",
+    "hash": "0xe05db05eeb3f288041ecb10a787df121c0ed69499355716e17c307de313a4486",
+    "miner": "0x0c062b329265c965deef1eede55183b3acb8f611",
+    "mixHash": "0xb669ae39118a53d2c65fd3b1e1d3850dd3f8c6842030698ed846a2762d68b61d",
+    "nonce": "0x2b469722b8e28c45",
+    "number": "24973",
+    "stateRoot": "0x532a5c3f75453a696428db078e32ae283c85cb97e4d8560dbdf022adac6df369",
+    "timestamp": "1479891145",
+    "totalDifficulty": "1892250259406",
+    "alloc": {
+      "0x6c06b16512b332e6cd8293a2974872674716ce18": {
+        "balance": "0x0",
+        "nonce": "1",
+        "code": "0x60606040526000357c0100000000000000000000000000000000000000000000000000000000900480632e1a7d4d146036575b6000565b34600057604e60048080359060200190919050506050565b005b3373ffffffffffffffff
+        "storage": {}
+      },
+      "0x66fdfd05e46126a07465ad24e40cc0597bc1ef31": {
+        "balance": "0x229ebbb36c3e0f20",
+        "nonce": "3",
+        "code": "0x",
+        "storage": {}
+      }
+    },
+    "config": {
+      "chainId": 3,
+      "homesteadBlock": 0,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "byzantiumBlock": 1700000,
+      "constantinopleBlock": 4230000,
+      "petersburgBlock": 4939394,
+      "istanbulBlock": 6485846,
+      "muirGlacierBlock": 7117117,
+      "ethash": {}
+    }
+  },
+  "context": {
+    "number": "24974",
+    "difficulty": "117067574",
+    "timestamp": "1479891162",
+    "gasLimit": "4712388",
+    "miner": "0xc822ef32e6d26e170b70cf761e204c1806265914"
+  },
+  "input": "0xf889038504a81557008301f97e946c06b16512b332e6cd8293a2974872674716ce1880a42e1a7d4d00000000000000000000000000000000000000000000000000014d1120d7b1600002aa0e2a6558040c5d72bc59f2fb62a38993a314c849c
+  "result": {
+    "type": "CALL",
+    "from": "0x66fdfd05e46126a07465ad24e40cc0597bc1ef31",
+    "to": "0x6c06b16512b332e6cd8293a2974872674716ce18",
+    "value": "0x0",
+    "gas": "0x1a466",
+    "gasUsed": "0x1dc6",
+    "input": "0x2e1a7d4d00000000000000000000000000000000000000000000000000014d1120d7b160000",
+    "output": "0x",
+    "calls": []
+  }
+}
```
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/inner_throw_outer_revert.json b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_throw_outer_revert.json
new file mode 100644
index 00000000..ec2ceb42
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/inner_throw_outer_revert.json
```
@@ -0,0 +1,81 @@
+{
+  "context": {
+    "difficulty": "3956606365",
+    "gasLimit": "5413248",
+    "miner": "0x00d8ae40d9a06d0e7a2877b62e32eb959afbe16d",
+    "number": "2295104",
+    "timestamp": "1513681256"
+  },
+  "genesis": {
+    "alloc": {
+      "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76": {
+        "balance": "0x0",
+        "code": "0x606060405260043610610015e576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff1680625b4487146101a257806311df9995146101cb578063278ecde11461022057806330adce0
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000000": "0x0000000000000000000000008d69d00910d0b2afb2a99ed6c16c8129fa8e1751",
+          "0x0000000000000000000000000000000000000000000000000000000000000003": "0x000000000000000000000000e819f024b41358d2c08e3a868a5c5dd0566078d4",
+          "0x0000000000000000000000000000000000000000000000000000000000000007": "0x000000000000000000000000000000000000000000000000000000005a388981",
+          "0x0000000000000000000000000000000000000000000000000000000000000008": "0x000000000000000000000000000000000000000000000000000000005a3b38e6"
+        }
+      },
+      "0xd4fcab9f0a6dc0493af47c864f6f17a8a5e2e826": {
+        "balance": "0x2a2dd979a35cf000",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      },
+      "0xe819f024b41358d2c08e3a868a5c5dd0566078d4": {
+        "balance": "0x0",
+        "code": "0x6060604052600436106100ba576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806306fdde03146100bf578063095ea7b31461014d57806318160ddd146101a757806323b87
+        "nonce": "1",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3956606365",
+    "extraData": "0x566961425443",
+    "gasLimit": "5418523",
+    "hash": "0x6f37eb930a25da673ea1bb80fd9e32ddac19cdf7cd4bb2eac62cc13598624077",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "mixHash": "0x10971cde68c587c750c23b8589ae868ce82c2c646636b97e7d9856470c5297c7",
+    "nonce": "0x810f923ff4b450a1",
+    "number": "2295103",
+    "stateRoot": "0xff403612573d76dfdaf4fea2429b77dbe9764021ae0e38dc8ac79a3cf551179e",
+    "timestamp": "1513681246",
+    "totalDifficulty": "7162347056825919"
+  },
+  "input": "0xf86d808504e3b292008307dfa69433056b5dcac09a9b4becad0e1dcf92c19bd0af76880e92596fd62900008029a0e5f27bb66431f7081bb7f1f242003056d7f3f35414c352cd3d1848b52716dac2a07d0be78980edb0bd2a0678fc53aa90
+  "result": {
+    "calls": [
+      {
+        "error": "invalid opcode: INVALID",
+        "from": "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76",
+        "gas": "0x75fe3",
+        "gasUsed": "0x75fe3",
+        "input": "0xa9059cbb000000000000000000000000d4fcab9f0a6dc0493af47c864f6f17a8a5e2e82600000000000000000000000000000000000000000000000000000000000002f4",
+        "to": "0xe819f024b41358d2c08e3a868a5c5dd0566078d4",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "error": "execution reverted",
+    "from": "0xd4fcab9f0a6dc0493af47c864f6f17a8a5e2e826",
+    "gas": "0x78d9e",
+    "gasUsed": "0x76fc0",
+    "input": "0x",
+    "to": "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76",
+    "type": "CALL",
+    "value": "0xe92596fd6290000"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/oog.json b/eth/tracers/internal/tracetest/testdata/call_tracer/oog.json
new file mode 100644
index 00000000..de4fed6a
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/oog.json
@@ -0,0 +1,60 @@
+{
+  "context": {
+    "difficulty": "3699098917",
+    "gasLimit": "5258985",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "number": "2294631",
+    "timestamp": "1513675366"
+  },
+  "genesis": {
+    "alloc": {
+      "0x43064693d3d38ad6a7cb579e0d6d9718c8aa6b62": {
+        "balance": "0x0",
+        "code": "0x6060604052600436106100ba576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806306fdde03146100bf578063095ea7b31461014d57806318160ddd146101a757806323b87
+        "nonce": "1",
+        "storage": {
+          "0x296b66049cc4f9c8bf3d4f14752add261d1a980b39bdd194a7897baf39ac7579": "0x0000000000000000000000000000000000000000033b2e3c9fc9653f9e72b1e0"
+        }
+      },
+      "0x94194bc2aaf494501d7880b61274a169f6502a54": {
+        "balance": "0xea8c39a876d19888d",
+        "code": "0x",
+        "nonce": "265",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3699098917",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "5263953",
+    "hash": "0x03a0f62a8106793dafcfae7b75fd2654322062d585a19cea568314d7205790dc",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x15482cc64b7c00a947f5bf015dfc010db1a6a668c74df61974d6a7848c174408",
+    "nonce": "0xd1bdb150f6fd170e",
+    "number": "2294630",
+    "stateRoot": "0x1ab1a534e84cc787cda1db21e0d5920ab06017948075b759166cfea7274657a1",
+    "timestamp": "1513675347",
+    "totalDifficulty": "7160543502214733"
+  },
+  "input": "0xf8ab820109855d21dba00082ca1d9443064693d3d38ad6a7cb579e0d6d9718c8aa6b6280b844a9059cbb000000000000000000000000e77b1ac803616503510bed0086e3a7be2627a6990000000000000000000000000000000000000000000000
+  "result": {
+    "error": "out of gas",
+    "from": "0x94194bc2aaf494501d7880b61274a169f6502a54",
+    "gas": "0x7045",
+    "gasUsed": "0x7045",
+    "input": "0xa9059cbb000000000000000000000000e77b1ac803616503510bed0086e3a7be2627a69900000000000000000000000000000000000000000000000000000009502f9000",
+    "to": "0x43064693d3d38ad6a7cb579e0d6d9718c8aa6b62",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/revert.json b/eth/tracers/internal/tracetest/testdata/call_tracer/revert.json
new file mode 100644
index 00000000..059040a1
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/revert.json
@@ -0,0 +1,58 @@
+{
+  "context": {
+    "difficulty": "3665057456",
+    "gasLimit": "5232723",
+    "miner": "0xf4d8e706cfb25c0decbbdd4d2e2cc10c66376a3f",
+    "number": "2294501",
+    "timestamp": "1513673601"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0f6cef2b7fbb504782e35aa82a2207e816a2b7a9": {
+        "balance": "0x2a3fc32bcc019283",
+        "code": "0x",
+        "nonce": "10",
+        "storage": {}
+      },
+      "0xabbcd5b340c80b5f1c0545c04c987b87310296ae": {
+        "balance": "0x0",
+        "code": "0x606060405236156100755763ffffffff7c01000000000000000000000000000000000000000000000000000000006000350416632d0335ab811461007a578063548db174146100ab5780637f649783146100fc578063b092145e146
+        "nonce": "1",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3672229776",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "5227619",
+    "hash": "0xa07b3d6c6bf63f5f981016db9f2d1d93033833f2c17e8bf7209e85f1faf08076",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x806e151ce2817be922e93e8d5921fa0f0d0fd213d6b2b9a3fa17458e74a163d0",
+    "nonce": "0xbc5d43adc2c30c7d",
+    "number": "2294500",
+    "stateRoot": "0xca645b335888352ef9d8b1ef083e9019648180b259026572e3139717270de97d",
+    "timestamp": "1513673552",
+    "totalDifficulty": "7160066586979149"
+  },
+  "input": "0xf9018b0a8505d21dba00832dc6c094abbcd5b340c80b5f1c0545c04c987b87310296ae80b9012473b40a5c000000000000000000000000400de2e016bda6577407dfc379faba9899bc73ef0000000000000000000000002cc31912b2b0f3
+  "result": {
+    "error": "execution reverted",
+    "from": "0x0f6cef2b7fbb504782e35aa82a2207e816a2b7a9",
+    "gas": "0x2d55e8",
+    "gasUsed": "0xc3",
+    "input": "0x73b40a5c000000000000000000000000400de2e016bda6577407dfc379faba9899bc73ef0000000000000000000000002cc31912b2b0f3075a87b3640923d45a26cef3ee00000000000000000000000000000000000000000000000000
+    "to": "0xabbcd5b340c80b5f1c0545c04c987b87310296ae",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/revert_reason.json b/eth/tracers/internal/tracetest/testdata/call_tracer/revert_reason.json
new file mode 100644
index 00000000..094b0446
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/revert_reason.json
@@ -0,0 +1,64 @@
+{
+  "context": {
+    "difficulty": "2",
+    "gasLimit": "8000000",
+    "miner": "0x0000000000000000000000000000000000000000",
+    "number": "3212651",
+    "timestamp": "1597246515"
+  },
+  "genesis": {
+    "alloc": {
+      "0xf58833cf0c791881b494eb79d461e08a1f043f52": {
+        "balance": "0x0",
+        "code": "0x608060405234801561001057600080fd5b50600436106100a5576000357c010000000000000000000000000000000000000000000000000000000090048063609ff1bd11610078578063609ff1bd146101af5780639e7b8d6114610
+        "nonce": "1",
+        "storage": {
+          "0x6200beec95762de01ce05f2a0e58ce3299dbb53c68c9f3254a242121223cdf58": "0x0000000000000000000000000000000000000000000000000000000000000000"
+        }
+      },
+      "0xf7579c3d8a669c89d5ed246a22eb6db8f6fedbf1": {
+        "balance": "0x57af9d6b3df812900",
+        "code": "0x",
+        "nonce": "6",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 0,
+      "constantinopleBlock": 0,
+      "petersburgBlock": 0,
+      "IstanbulBlock":1561651,
+      "chainId": 5,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf888068449504f80832dc6c094f58833cf0c791881b494eb79d461e08a1f043f5280a45c19a95c000000000000000000000000f7579c3d8a669c89d5ed246a22eb6db8f6fedbf12da0264664db3e71fae1dbdaf2f53954be149ad3b7ba8a
+  "result": {
+    "error": "execution reverted",
+    "from": "0xf7579c3d8a669c89d5ed246a22eb6db8f6fedbf1",
+    "gas": "0x2d7308",
+    "gasUsed": "0x588",
+    "input": "0x5c19a95c000000000000000000000000f7579c3d8a669c89d5ed246a22eb6db8f6fedbf1",
+    "to": "0xf58833cf0c791881b494eb79d461e08a1f043f52",
+    "type": "CALL",
+    "value": "0x0",
+    "output": "0x08c379a00000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000001e53656c662d64656c65676174696f6e20697320646973616c6
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/selfdestruct.json b/eth/tracers/internal/tracetest/testdata/call_tracer/selfdestruct.json
new file mode 100644
index 00000000..dd717906
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/selfdestruct.json
@@ -0,0 +1,75 @@
+{
+  "context": {
+    "difficulty": "3502894804",
+    "gasLimit": "4722976",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2289806",
+    "timestamp": "1513601314"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "22",
+        "storage": {}
+      },
+      "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe": {
+        "balance": "0x4d87094125a369d9bd5",
+        "code": "0x61deadff",
+        "nonce": "1",
+        "storage": {}
+      },
+      "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb": {
+        "balance": "0x1780d77678137ac1b775",
+        "code": "0x",
+        "nonce": "29072",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf88b8271908506fc23ac0083015f90943b873a919aa0512d5a0f09e6dcceaa4a6727fafe80a463e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c52aa0bdce0b59e8761854e857fe64015f06dd08a4
+  "result": {
+    "calls": [
+      {
+        "from": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+        "gas": "0x0",
+        "gasUsed": "0x0",
+        "input": "0x",
+        "to": "0x000000000000000000000000000000000000dEaD",
+        "type": "SELFDESTRUCT",
+        "value": "0x4d87094125a369d9bd5"
+      }
+    ],
+    "from": "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb",
+    "gas": "0x10738",
+    "gasUsed": "0x7533",
+    "input": "0x63e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+    "output": "0x",
+    "to": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/simple.json b/eth/tracers/internal/tracetest/testdata/call_tracer/simple.json
new file mode 100644
index 00000000..08cb7b2d
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/simple.json
@@ -0,0 +1,80 @@
+{
+  "context": {
+    "difficulty": "3502894804",
+    "gasLimit": "4722976",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2289806",
+    "timestamp": "1513601314"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "22",
+        "storage": {}
+      },
+      "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe": {
+        "balance": "0x4d87094125a369d9bd5",
+        "code": "0x606060405236156100935763ffffffff60e060020a60003504166311ee8382811461009c57806313af4035146100be5780631f5e8f4c146100ee57806324daddc5146101125780634921a91a1461013b57806363e4bff4146101575
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000000": "0x000000000000000000000001b436ba50d378d4bbc8660d312a13df6af6e89dfb",
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x000000000000000000000000000000000000000000000000006f05b59d3b20000",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x000000000000000000000000000000000000000000000000000000000000003c",
+          "0x0000000000000000000000000000000000000000000000000000000000000003": "0x000000000000000000000000000000000000000000000000000000005a37b834"
+        }
+      },
+      "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb": {
+        "balance": "0x1780d77678137ac1b775",
+        "code": "0x",
+        "nonce": "29072",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf88b8271908506fc23ac0083015f90943b873a919aa0512d5a0f09e6dcceaa4a6727fafe80a463e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c52aa0bdce0b59e8761854e857fe64015f06dd08a4
+  "result": {
+    "calls": [
+      {
+        "from": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+        "gas": "0x6d05",
+        "gasUsed": "0x0",
+        "input": "0x",
+        "to": "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+        "type": "CALL",
+        "value": "0x6f05b59d3b20000"
+      }
+    ],
+    "from": "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb",
+    "gas": "0x10738",
+    "gasUsed": "0x3ef9",
+    "input": "0x63e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+    "to": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer/throw.json b/eth/tracers/internal/tracetest/testdata/call_tracer/throw.json
new file mode 100644
index 00000000..09cf4497
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer/throw.json
@@ -0,0 +1,62 @@
+{
+  "context": {
+    "difficulty": "117009631",
+    "gasLimit": "4712388",
+    "miner": "0x294e5d6c39a36ce38af1dca70c1060f78dee8070",
+    "number": "25009",
+    "timestamp": "1479891666"
+  },
+  "genesis": {
+    "alloc": {
+      "0x70c9217d814985faef62b124420f8dfbddd96433": {
+        "balance": "0x4ecd70668f5d854a",
+        "code": "0x",
+        "nonce": "1638",
+        "storage": {}
+      },
+      "0xc212e03b9e060e36facad5fd8f4435412ca22e6b": {
+        "balance": "0x0",
+        "code": "0x606060405236156101745760e060020a600035046302d05d3f811461017c57806304a7fdbc1461018e5780630e90f957146101fb5780630fb5a6b41461021257806314baa1b61461021b57806317fc45e21461023a5780632b09692
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x00000000000000000000000000000000000000000000000000000000000061a9",
+          "0x0000000000000000000000000000000000000000000000000000000000000005": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433"
+        }
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "117066792",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712388",
+    "hash": "0xe23e8d4562a1045b70cbc99fefb20c101a8f0fc8559a80d65fea8896e2f1d46e",
+    "miner": "0x71842f946b98800fe6feb49f0ae4e253259031c9",
+    "mixHash": "0x0aada9d6e93dd4db0d09c0488dc0a048fca2ccdc1f3fc7b83ba2a8d393a3a4ff",
+    "nonce": "0x70849d5838dee2e9",
+    "number": "25008",
+    "stateRoot": "0x1e01d2161794768c5b917069e73d86e8dca80cd7f3168c0597de420ab93a3b7b",
+    "timestamp": "1479891641",
+    "totalDifficulty": "1896347038589"
+  },
+  "input": "0xf88b8206668504a817c8008303d09094c212e03b9e060e36facad5fd8f4435412ca22e6b80a451a34eb80000000000000000000000000000000000000000000000027fad02094277c000029a0692a3b4e7b2842f8dd7832e712c21e09f451
+  "result": {
+    "error": "invalid jump destination",
+    "from": "0x70c9217d814985faef62b124420f8dfbddd96433",
+    "gas": "0x37b38",
+    "gasUsed": "0x37b38",
+    "input": "0x51a34eb80000000000000000000000000000000000000000000000027fad02094277c0000",
+    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/create.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/create.json
new file mode 100644
index 00000000..8699bf3e
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/create.json
@@ -0,0 +1,58 @@
+{
+  "context": {
+    "difficulty": "3755480783",
+    "gasLimit": "5401723",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "number": "2294702",
+    "timestamp": "1513676146"
+  },
+  "genesis": {
+    "alloc": {
+      "0x13e4acefe6a6700604929946e70e6443e4e73447": {
+        "balance": "0xcf3e0938579f000",
+        "code": "0x",
+        "nonce": "9",
+        "storage": {}
+      },
+      "0x7dc9c9730689ff0b0fd506c67db815f12d90a448": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3757315409",
+    "extraData": "0x566961425443",
+    "gasLimit": "5406414",
+    "hash": "0xae107f592eebdd9ff8d6ba00363676096e6afb0e1007a7d3d0af88173077378d",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "mixHash": "0xc927aa05a38bc3de864e95c33b3ae559d3f39c4ccd51cef6f113f9c50ba0caf1",
+    "nonce": "0x93363bbd2c95f410",
+    "number": "2294701",
+    "stateRoot": "0x6b6737d5bde8058990483e915866bd1578014baeff57bd5e4ed228a2bfad635c",
+    "timestamp": "1513676127",
+    "totalDifficulty": "7160808139332585"
+  },
+  "input": "0xf907ef098504e3b29200830897be8080b9079c606060405260405160208061077c83398101604052808051906020019091905050600160009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffff
+  "result": {
+    "from": "0x13e4acefe6a6700604929946e70e6443e4e73447",
+    "gas": "0x5e106",
+    "gasUsed": "0x5e106",
+    "input": "0x606060405260405160208061077c83398101604052808051906020019091905050600160009054906101000a900473ffffffffffffffffffffffffffffffffffffffff1673ffffffffffffffffffffffffffffffffffffffff163373ff
+    "output": "0x60606040526004361061008357600035 7c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806305e4382a146100855780631c02708d146100ae5780632e1a7d4d146100c35780635114cb5
+    "to": "0x7dc9c9730689ff0b0fd506c67db815f12d90a448",
+    "type": "CREATE",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/deep_calls.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/deep_calls.json
new file mode 100644
index 00000000..0353d4cf
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/deep_calls.json
@@ -0,0 +1,415 @@
+{
+  "context": {
+    "difficulty": "117066904",
+    "gasLimit": "4712384",
+    "miner": "0x1977c248e1014cc103929dd7f154199c916e39ec",
+    "number": "25001",
+    "timestamp": "1479891545"
+  },
+  "genesis": {
+    "alloc": {
+      "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38": {
+        "balance": "0x0",
+        "code": "0x606060405236156100825760e060020a600035046302d05d3f811461008a5780630accce061461009c5780631ab9075a146100c757806331ed274614610102578063645a3b72146101335780637720fdae3146101555780 63a7f4377
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0x2cccf5e0538493c235d1c5ef6580f77d99e91396": {
+        "balance": "0x0",
+        "code": "0x606060405236156100775760e060020a600035046302d05d3f811461007f57806313bc6d4b146100915780633688a877146100b95780635188f9961461012f5780637eadc976146101545780638ad79680146101d3578063a43e04d
+        "nonce": "1",
+        "storage": {
+          "0x0684ac65a9fa32414dda56996f4183597d695987fdb82b145d722743891a6fe8": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+          "0x1cd76f78169a420d99346e3501dd3e541622c38a226f9b63e01cfebc69879dc7": "0x000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+          "0x8e54a4494fe5da016bfc01363f4f6cdc91013bb5434bd2a4a3359f13a23afa2f": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+          "0x94edf7f600ba56655fd65fca1f1424334ce369326c1dc3e53151dcd1ad06bc13": "0x0000000000000000000000000000000000000000000000000000000000000001",
+          "0xbbee47108b275f55f98482c6800f6372165e88b0330d3f5dae6419df4734366c": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+          "0xd38c0c4e84de118cfdcc775130155d83b8bbaaf23dc7f3c83a626b10473213bd": "0x0000000000000000000000000000000000000000000000000000000000000001",
+          "0xfb3aa5c655c2ec9d40609401f88d505d1da61afaa550e36ef5da0509ada257ba": "0x0000000000000000000000007986bad81f4cbd9317f5a46861437dae58d69113"
+        }
+      },
+      "0x3e9286eafa2db8101246c2131c09b49080d00690": {
+        "balance": "0x0",
+        "code": "0x606060405236156100cf5760e060020a600035046302d05d3f81146100d7578063056d4470146100e957806316c66cc61461010c5780631ab9075a146101935780633ae1005c146101ce57806358541662146101fe5780635ed61af
+        "nonce": "16",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0x70c9217d814985faef62b124420f8dfbddd96433": {
+        "balance": "0x4ef436dcbda6cd4a",
+        "code": "0x",
+        "nonce": "1634",
+        "storage": {}
+      },
+      "0x7986bad81f4cbd9317f5a46861437dae58d69113": {
+        "balance": "0x0",
+        "code": "0x6060604052361561008d5760e060020a600035046302d05d3f811461009557806316c66cc6146100a75780631ab9075a146100d7578063213fe2b7146101125780639859387b1461013f578063988db79c1461015e578063a7f4377
+        "nonce": "7",
+        "storage": {
+          "0xffc4df2d4f3d2cffad590bed6296406ab7926ca9e74784f74a95191fa069a174": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433"
+        }
+      },
+      "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f": {
+        "balance": "0x0",
+        "code": "0x606060405236156100ae5760e060020a600035046302d05d3f81146100b65780631ab9075a146100c85780632b68bb2d146101035780634cc927d7146101c557806351a34eb81461028e57806356ccb6f0146103545780635928d37
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396"
+        }
+      },
+      "0xc212e03b9e060e36facad5fd8f4435412ca22e6b": {
+        "balance": "0x0",
+        "code": "0x606060405236156101745760e060020a600035046302d05d3f811461017c57806304a7fdbc1461018e5780630e90f957146101fb5780630fb5a6b41461021257806314baa1b61461021b57806317fc45e21461023a5780632b0969
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000000000000000000000000000000000000000006195",
+          "0x0000000000000000000000000000000000000000000000000000000000000004": "0x5842545553440000000000000000000000000000000000000000000000000000",
+          "0x0000000000000000000000000000000000000000000000000000000000000005": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433",
+          "0x0000000000000000000000000000000000000000000000000000000000000006": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+          "0x000000000000000000000000000000000000000000000000000000000000000b": "0x0000000000000000000000000000000000000000000000283c7b9181eca20000"
+        }
+      },
+      "0xcf00ffd997ad14939736f026006498e3f099baaf": {
+        "balance": "0x0",
+        "code": "0x606060405236156100cf5760e060020a600035046302d05d3f81146100d7578063031e7f5d146100e95780631ab9075a1461010b5780632243118a1461014657806327aad68a1461016557806338a699a4146101da5780635188f99
+        "nonce": "3",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x3571d73f14f31a1463bd0a2f92f7fde1653d4e1ead7aedf4b0a5df02f16092ab": "0x0000000000000000000000000000000000000000000007d634e4c55188be0000",
+          "0x4e64fe2d1b72d95a0a31945cc6e4f4e524ac5ad56d6bd44a85ec7bc9cc0462c0": "0x000000000000000000000000000000000000000000000002b5e3af16b1880000"
+        }
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "117124093",
+    "extraData": "0xd5830105008650617269747986312e31322e31826d61",
+    "gasLimit": "4707788",
+    "hash": "0xad325e4c49145fb7a4058a68ac741cc8607a71114e23fc88083c7e881dd653e7",
+    "miner": "0x00714b9ac97fd6bd9325a059a70c9b9fa94ce050",
+    "mixHash": "0x0af918f65cb4af04b608fc1f14a849707696986a0e7049e97ef3981808bcc65f",
+    "nonce": "0x38dee147326a8d40",
+    "number": "25000",
+    "stateRoot": "0xc5d6bbcd46236fcdcc80b332ffaaa5476b980b01608f9708408cfef01b58bd5b",
+    "timestamp": "1479891517",
+    "totalDifficulty": "1895410389427"
+  },
+  "input": "0xf88b8206628504a817c8008303d09094c212e03b9e060e36facad5fd8f4435412ca22e6b80a451a34eb80000000000000000000000000000000000000000000000280faf689c35ac00002aa0a7ee5b7877811bf671d121b40569462e7226
+  "result": {
+    "calls": [
+      {
+        "from": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+        "gas": "0x31217",
+        "gasUsed": "0x334",
+        "input": "0xe16c7d98636f6e7472616374617070690000000000000000000000000000000000000000",
+        "output": "0x000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+        "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+        "type": "CALL",
+        "value": "0x0"
+      },
+      {
+        "calls": [
+          {
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x2a68d",
+            "gasUsed": "0x334",
+            "input": "0xe16c7d98636f6e747261637463746c000000000000000000000000000000000000000000",
+            "output": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+            "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "calls": [
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x23ac9",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d98636f6e7472616374646462000000000000000000000000000000000000000000",
+                "output": "0x0000000000000000000000007986bad81f4cbd9317f5a46861437dae58d69113",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x23366",
+                "gasUsed": "0x273",
+                "input": "0x16c66cc6000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                "to": "0x7986bad81f4cbd9317f5a46861437dae58d69113",
+                "type": "CALL",
+                "value": "0x0"
+              }
+            ],
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x29f35",
+            "gasUsed": "0xf8d",
+            "input": "0x16c66cc6000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b",
+            "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+            "to": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x28a9e",
+            "gasUsed": "0x334",
+            "input": "0xe16c7d98636f6e747261637463746c000000000000000000000000000000000000000000",
+            "output": "0x0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+            "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+            "type": "CALL",
+            "value": "0x0"
+          },
+          {
+            "calls": [
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x21d79",
+                "gasUsed": "0x24d",
+                "input": "0x13bc6d4b000000000000000000000000b4fe7aa695b326c9d219158d2ca50db77b39f99f",
+                "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x2165b",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d986d61726b657464620000000000000000000000000000000000000000000000000",
+                "output": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "calls": [
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x1a8e8",
+                    "gasUsed": "0x24d",
+                    "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                    "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                    "type": "CALL",
+                    "value": "0x0"
+                  },
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x1a2c6",
+                    "gasUsed": "0x3cb",
+                    "input": "0xc9503fe2",
+                    "output": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+                    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                    "type": "CALL",
+                    "value": "0x0"
+                  },
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x19b72",
+                    "gasUsed": "0x3cb",
+                    "input": "0xc9503fe2",
+                    "output": "0x0000000000000000000000000000000000000000000000008ac7230489e80000",
+                    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                    "type": "CALL",
+                    "value": "0x0"
+                  },
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x19428",
+                    "gasUsed": "0x305",
+                    "input": "0x6f265b93",
+                    "output": "0x0000000000000000000000000000000000000000000000283c7b9181eca20000",
+                    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                    "type": "CALL",
+                    "value": "0x0"
+                  },
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x18d45",
+                    "gasUsed": "0x229",
+                    "input": "0x2e94420f",
+                    "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                    "type": "CALL",
+                    "value": "0x0"
+                  },
+                  {
+                    "from": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                    "gas": "0x1734e",
+                    "gasUsed": "0x229",
+                    "input": "0x2e94420f",
+                    "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                    "type": "CALL",
+                    "value": "0x0"
+                  }
+                ],
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x20ee1",
+                "gasUsed": "0x5374",
+                "input": "0x581d5d60000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b0000000000000000000000000000000000000000000000280faf689c35ac0000",
+                "output": "0x",
+                "to": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x1b6c1",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d986c6f676d6772000000000000000000000000000000000000000000000000000000",
+                "output": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x1af69",
+                "gasUsed": "0x229",
+                "input": "0x2e94420f",
+                "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "calls": [
+                  {
+                    "from": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                    "gas": "0x143a5",
+                    "gasUsed": "0x24d",
+                    "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                    "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                    "type": "CALL",
+                    "value": "0x0"
+                  }
+                ],
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x1a91d",
+                "gasUsed": "0x12fa",
+                "input": "0x0accce06000000000000000000000000000000000000000000000000000000000000000258425455534400000000000000000000000000000000000000000000000000000000000000000000000000000000c212e03b9e060e
+                "output": "0x",
+                "to": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x19177",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d986c6f676d67720000000000000000000000000000000000000000000000000000",
+                "output": "0x0000000000000000000000002a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x18a22",
+                "gasUsed": "0x229",
+                "input": "0x2e94420f",
+                "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x18341",
+                "gasUsed": "0x334",
+                "input": "0xe16c7d986d61726b6574646200000000000000000000000000000000000000000000000000",
+                "output": "0x000000000000000000000000cf00ffd997ad14939736f026006498e3f099baaf",
+                "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x17bec",
+                "gasUsed": "0x229",
+                "input": "0x2e94420f",
+                "output": "0x5842545553440000000000000000000000000000000000000000000000000000",
+                "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x1764e",
+                "gasUsed": "0x45c",
+                "input": "0xf92eb7745842545553440000000000000000000000000000000000000000000000000000",
+                "output": "0x00000000000000000000000000000000000000000000002816d180e30c390000",
+                "to": "0xcf00ffd997ad14939736f026006498e3f099baaf",
+                "type": "CALL",
+                "value": "0x0"
+              },
+              {
+                "calls": [
+                  {
+                    "from": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                    "gas": "0x108ba",
+                    "gasUsed": "0x24d",
+                    "input": "0x13bc6d4b0000000000000000000000003e9286eafa2db8101246c2131c09b49080d00690",
+                    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+                    "to": "0x2cccf5e0538493c235d1c5ef6580f77d99e91396",
+                    "type": "CALL",
+                    "value": "0x0"
+                  }
+                ],
+                "from": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+                "gas": "0x16e62",
+                "gasUsed": "0xebb",
+                "input": "0x645a3b72584254555344000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000002816d180e30c390000",
+                "output": "0x",
+                "to": "0x2a98c5f40bfa3dee83431103c535f6fae9a8ad38",
+                "type": "CALL",
+                "value": "0x0"
+              }
+            ],
+            "from": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+            "gas": "0x283b9",
+            "gasUsed": "0xc51c",
+            "input": "0x949ae479000000000000000000000000c212e03b9e060e36facad5fd8f4435412ca22e6b0000000000000000000000000000000000000000000000280faf689c35ac0000",
+            "output": "0x",
+            "to": "0x3e9286eafa2db8101246c2131c09b49080d00690",
+            "type": "CALL",
+            "value": "0x0"
+          }
+        ],
+        "from": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+        "gas": "0x30b4a",
+        "gasUsed": "0xedb7",
+        "input": "0x51a34eb80000000000000000000000000000000000000000000000280faf689c35ac0000",
+        "output": "0x",
+        "to": "0xb4fe7aa695b326c9d219158d2ca50db77b39f99f",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "from": "0x70c9217d814985faef62b124420f8dfbddd96433",
+    "gas": "0x37b38",
+    "gasUsed": "0x12bb3",
+    "input": "0x51a34eb80000000000000000000000000000000000000000000000280faf689c35ac0000",
+    "output": "0x",
+    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
```

diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/delegatecall.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/delegatecall.json

```
new file mode 100644
index 00000000..f7ad6df5
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/delegatecall.json
@@ -0,0 +1,97 @@
+{
+  "context": {
+    "difficulty": "31927752",
+    "gasLimit": "4707788",
+    "miner": "0x5659922ce141eedbc2733678f9806c77b4eebee8",
+    "number": "11495",
+    "timestamp": "1479735917"
+  },
+  "genesis": {
+    "alloc": {
+      "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff": {
+        "balance": "0x0",
+        "code": "0x606060405236156100825760e060020a60003504630a0313a981146100875780630a3b0a4f146101095780630cd40fea1461021257806329092d0e1461021f5780634cd06a5f146103295780635dbe47e8146103395780637a9e541
+        "nonce": "1",
+        "storage": {
+          "0x4d140b25abf3c71052885c66f73ce07cff141c1afabffdaf5cba04d625b7ebcc": "0x0000000000000000000000000000000000000000000000000000000000000001"
+        }
+      },
+      "0x269296dddce321a6bcbaa2f0181127593d732cba": {
+        "balance": "0x0",
+        "code": "0x606060405236156101275760e060020a60003504630cd40fea811461012c578063173825d9146101395780631849cb5a146101c7578063285791371461030f5780632a58b3301461033f5780632cb0d48a146103565780632f54bf6
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x000113204f5d64c28326fd7bd05fd4ea855302d7f2ff00000000000000000000"
+        }
+      },
+      "0x42b02b5deeb78f34cd5ac896473b63e6c99a71a2": {
+        "balance": "0x0",
+        "code": "0x6504032353da7150606060405236156100695760e060020a60003504631bf7509d811461006e57806321ce24d4146100815780633556e84146100ec578063685a1f3c146101035780637d65837a1461011757806389489a8714610
+        "nonce": "1",
+        "storage": {}
+      },
+      "0xa529806c67cc6486d4d62024471772f47f6fd672": {
+        "balance": "0x67820e39ac8fe9800",
+        "code": "0x",
+        "nonce": "68",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "31912170",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712388",
+    "hash": "0x0855914bdc581bccdc62591fd438498386ffb59ea4d5361ed5c3702e26e2c72f",
+    "miner": "0x334391aa808257952a462d1475562ee2106a6c90",
+    "mixHash": "0x64bb70b8ca883cadb8fbbda2c70a861612407864089ed87b98e5de20acceada6",
+    "nonce": "0x684129f283aaef18",
+    "number": "11494",
+    "stateRoot": "0x7057f31fe3dab1d620771adad35224aae43eb70e94861208bc84c557ff5b9d10",
+    "timestamp": "1479735912",
+    "totalDifficulty": "90744064339"
+  },
+  "input": "0xf889448504a817c800832dc6c094269296dddce321a6bcbaa2f0181127593d732cba80a47065cb48000000000000000000000000001523e55a1ca4efbae03355775ae89f8d7699ad9e29a080ed81e4c5e9971a730efab4885566e2c868cd80
+  "result": {
+    "calls": [
+      {
+        "calls": [
+          {
+            "from": "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff",
+            "gas": "0x2bf459",
+            "gasUsed": "0x2aa",
+            "input": "0x7d65837a000000000000000000000000a529806c67cc6486d4d62024471772f47f6fd672",
+            "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+            "to": "0x42b02b5deeb78f34cd5ac896473b63e6c99a71a2",
+            "type": "DELEGATECALL"
+          }
+        ],
+        "from": "0x269296dddce321a6bcbaa2f0181127593d732cba",
+        "gas": "0x2cae73",
+        "gasUsed": "0xa9d",
+        "input": "0x5dbe47e8000000000000000000000000a529806c67cc6486d4d62024471772f47f6fd672",
+        "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+        "to": "0x13204f5d64c28326fd7bd05fd4ea855302d7f2ff",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "from": "0xa529806c67cc6486d4d62024471772f47f6fd672",
+    "gas": "0x2d6e28",
+    "gasUsed": "0x64bd",
+    "input": "0x7065cb480000000000000000000000001523e55a1ca4efbae03355775ae89f8d7699ad9e",
+    "output": "0x",
+    "to": "0x269296dddce321a6bcbaa2f0181127593d732cba",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_create_oog_outer_throw.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_create_oog_outer_throw.json
new file mode 100644
index 00000000..72152e27
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_create_oog_outer_throw.json
@@ -0,0 +1,77 @@
+{
+  "context": {
+    "difficulty": "3451177886",
+    "gasLimit": "4709286",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2290744",
+    "timestamp": "1513616439"
+  },
+  "genesis": {
+    "alloc": {
+      "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a": {
+        "balance": "0x0",
+        "code": "0x606060405263ffffffff60e060020a6000350416633b91f506811461005057806335bb478081146100715780635f51fca01461008c578063bc7647a9146100ad578063f1bd0d7a146100c8575b610000565b346100005761006f60016
+        "nonce": "789",
+        "storage": {
+          "0xfe9ec0542a1c009be8b1f3acf43af97100ffff42eb736850fb038fa1151ad4d9": "0x000000000000000000000000e4a13bc304682a903e9472f469c33801dd18d9e8"
+        }
+      },
+      "0x5cb4a6b902fcb21588c86c3517e797b07cdaadb9": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      },
+      "0xe4a13bc304682a903e9472f469c33801dd18d9e8": {
+        "balance": "0x33c763c929f62c4f",
+        "code": "0x",
+        "nonce": "14",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3451177886",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4713874",
+    "hash": "0x5d52a672417cd1269bf4f7095e25dcbf837747bba908cd5ef809dc1bd06144b5",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x01a12845ed546b94a038a7a03e8df8d7952024ed41ccb3db7a7ade4abc290ce1",
+    "nonce": "0x28c446f1cb9748c1",
+    "number": "2290743",
+    "stateRoot": "0x4898aceede76739daef76448a367d10015a2c022c9e7909b99a10fbf6fb16708",
+    "timestamp": "1513616414",
+    "totalDifficulty": "7146523769022564"
+  },
+  "input": "0xf8aa0e8509502f9000830493e0941d3ddf7caf024f253487e18bc4a15b1a360c170a80b8443b91f506000000000000000000000000a14bdd7e5666d784dcce98ad24d383a6b1cd4182000000000000000000000000e4a13bc304682a903e
+  "result": {
+    "calls": [
+      {
+        "error": "internal failure",
+        "from": "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a",
+        "gas": "0x39ff0",
+        "gasUsed": "0x39ff0",
+        "input": "0x606060405234620000005760405160208062001fd283398101604052515b805b600a8054600160a060020a031916600160a060020a0383161790555b506001600d819055600e81905560408051808201909152600c8082527f566f
+        "type": "CREATE",
+        "value": "0x0"
+      }
+    ],
+    "error": "invalid jump destination",
+    "from": "0xe4a13bc304682a903e9472f469c33801dd18d9e8",
+    "gas": "0x435c8",
+    "gasUsed": "0x435c8",
+    "input": "0x3b91f506000000000000000000000000a14bdd7e5666d784dcce98ad24d383a6b1cd4182000000000000000000000000e4a13bc304682a903e9472f469c33801dd18d9e8",
+    "to": "0x1d3ddf7caf024f253487e18bc4a15b1a360c170a",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_instafail.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_instafail.json
new file mode 100644
index 00000000..86070d13
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_instafail.json
@@ -0,0 +1,72 @@
+{
+  "genesis": {
+    "difficulty": "117067574",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712380",
+    "hash": "0xe05db05eeb3f288041ecb10a787df121c0ed69499355716e17c307de313a4486",
+    "miner": "0x0c062b329265c965deef1eede55183b3acb8f611",
+    "mixHash": "0xb669ae39118a53d2c65fd3b1e1d3850dd3f8c6842030698ed846a2762d68b61d",
+    "nonce": "0x2b469722b8e28c45",
+    "number": "24973",
+    "stateRoot": "0x532a5c3f75453a696428db078e32ae283c85cb97e4d8560dbdf022adac6df369",
+    "timestamp": "1479891145",
+    "totalDifficulty": "1892250259406",
+    "alloc": {
+      "0x6c06b16512b332e6cd8293a2974872674716ce18": {
+        "balance": "0x0",
+        "nonce": "1",
+        "code": "0x60606040526000357c0100000000000000000000000000000000000000000000000000000000900480632e1a7d4d146036575b6000565b34600057604e60048080359060200190919050506050565b005b3373ffffffffffffffff
+        "storage": {}
+      },
+      "0x66fdfd05e46126a07465ad24e40cc0597bc1ef31": {
+        "balance": "0x229ebbb36c3e0f20",
+        "nonce": "3",
+        "code": "0x",
+        "storage": {}
+      }
+    },
+    "config": {
+      "chainId": 3,
+      "homesteadBlock": 0,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "byzantiumBlock": 1700000,
+      "constantinopleBlock": 4230000,
+      "petersburgBlock": 4939394,
+      "istanbulBlock": 6485846,
+      "muirGlacierBlock": 7117117,
+      "ethash": {}
+    }
+  },
+  "context": {
+    "number": "24974",
+    "difficulty": "117067574",
+    "timestamp": "1479891162",
+    "gasLimit": "4712388",
+    "miner": "0xc822ef32e6d26e170b70cf761e204c1806265914"
+  },
+  "input": "0xf889038504a81557008301f97e946c06b16512b332e6cd8293a2974872674716ce1880a42e1a7d4d00000000000000000000000000000000000000000000000014d1120d7b1600002aa0e2a6558040c5d72bc59f2fb62a38993a314c849c
+  "result": {
+    "type": "CALL",
+    "from": "0x66fdfd05e46126a07465ad24e40cc0597bc1ef31",
+    "to": "0x6c06b16512b332e6cd8293a2974872674716ce18",
+    "value": "0x0",
+    "gas": "0x1a466",
+    "gasUsed": "0x1dc6",
+    "input": "0x2e1a7d4d00000000000000000000000000000000000000000000000014d1120d7b160000",
+    "output": "0x",
+    "calls": [
+      {
+        "type": "CALL",
+        "from": "0x6c06b16512b332e6cd8293a2974872674716ce18",
+        "to": "0x66fdfd05e46126a07465ad24e40cc0597bc1ef31",
+        "value": "0x14d1120d7b160000",
+        "error":"internal failure",
+        "input": "0x"
+      }
+    ]
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_throw_outer_revert.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_throw_outer_revert.json
new file mode 100644
index 00000000..ec2ceb42
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/inner_throw_outer_revert.json
```

```
@@ -0,0 +1,81 @@
+{
+  "context": {
+    "difficulty": "3956606365",
+    "gasLimit": "5413248",
+    "miner": "0x00d8ae40d9a06d0e7a2877b62e32eb959afbe16d",
+    "number": "2295104",
+    "timestamp": "1513681256"
+  },
+  "genesis": {
+    "alloc": {
+      "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76": {
+        "balance": "0x0",
+        "code": "0x6060604052600436106101015e576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff1680625b4487146101a257806311df9995146101cb578063278ecde11461022057806330adce0
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000000": "0x0000000000000000000000008d69d00910d0b2afb2a99ed6c16c8129fa8e1751",
+          "0x0000000000000000000000000000000000000000000000000000000000000003": "0x000000000000000000000000e819f024b41358d2c08e3a868a5c5dd0566078d4",
+          "0x0000000000000000000000000000000000000000000000000000000000000007": "0x000000000000000000000000000000000000000000000000000000005a388981",
+          "0x0000000000000000000000000000000000000000000000000000000000000008": "0x000000000000000000000000000000000000000000000000000000005a3b38e6"
+        }
+      },
+      "0xd4fcab9f0a6dc0493af47c864f6f17a8a5e2e826": {
+        "balance": "0x2a2dd979a35cf000",
+        "code": "0x",
+        "nonce": "0",
+        "storage": {}
+      },
+      "0xe819f024b41358d2c08e3a868a5c5dd0566078d4": {
+        "balance": "0x0",
+        "code": "0x6060604052600436106100ba576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806306fdde03146100bf578063095ea7b31461014d57806318160ddd146101a757806323b87
+        "nonce": "1",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3956606365",
+    "extraData": "0x566961425443",
+    "gasLimit": "5418523",
+    "hash": "0x6f37eb930a25da673ea1bb80fd9e32ddac19cdf7cd4bb2eac62cc13598624077",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "mixHash": "0x10971cde68c587c750c23b8589ae868ce82c2c646636b97e7d9856470c5297c7",
+    "nonce": "0x810f923ff4b450a1",
+    "number": "2295103",
+    "stateRoot": "0xff403612573d76dfdaf4fea2429b77dbe9764021ae0e38dc8ac79a3cf551179e",
+    "timestamp": "1513681246",
+    "totalDifficulty": "7162347056825919"
+  },
+  "input": "0xf86d808504e3b292008307dfa69433056b5dcac09a9b4becad0e1dcf92c19bd0af76880e92596fd62900008029a0e5f27bb66431f7081bb7f1f242003056d7f3f35414c352cd3d1848b52716dac2a07d0be78980edb0bd2a0678fc53aa90
+  "result": {
+    "calls": [
+      {
+        "error": "invalid opcode: INVALID",
+        "from": "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76",
+        "gas": "0x75fe3",
+        "gasUsed": "0x75fe3",
+        "input": "0xa9059cbb000000000000000000000000d4fcab9f0a6dc0493af47c864f6f17a8a5e2e82600000000000000000000000000000000000000000000000000000000000002f4",
+        "to": "0xe819f024b41358d2c08e3a868a5c5dd0566078d4",
+        "type": "CALL",
+        "value": "0x0"
+      }
+    ],
+    "error": "execution reverted",
+    "from": "0xd4fcab9f0a6dc0493af47c864f6f17a8a5e2e826",
+    "gas": "0x78d9e",
+    "gasUsed": "0x76fc0",
+    "input": "0x",
+    "to": "0x33056b5dcac09a9b4becad0e1dcf92c19bd0af76",
+    "type": "CALL",
+    "value": "0xe92596fd6290000"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/oog.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/oog.json
new file mode 100644
index 00000000..de4fed6a
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/oog.json
@@ -0,0 +1,60 @@
+{
+  "context": {
+    "difficulty": "3699098917",
+    "gasLimit": "5258985",
+    "miner": "0xd049bfd667cb46aa3ef5df0da3e57db3be39e511",
+    "number": "2294631",
+    "timestamp": "1513675366"
+  },
+  "genesis": {
+    "alloc": {
+      "0x43064693d3d38ad6a7cb579e0d6d9718c8aa6b62": {
+        "balance": "0x0",
+        "code": "0x6060604052600436106100ba576000357c0100000000000000000000000000000000000000000000000000000000900463ffffffff16806306fdde03146100bf578063095ea7b31461014d57806318160ddd146101a757806323b87
+        "nonce": "1",
+        "storage": {
+          "0x296b66049cc4f9c8bf3d4f14752add261d1a980b39bdd194a7897baf39ac7579": "0x0000000000000000000000000000000000000000033b2e3c9fc9653f9e72b1e0"
+        }
+      },
+      "0x94194bc2aaf494501d7880b61274a169f6502a54": {
+        "balance": "0xea8c39a876d19888d",
+        "code": "0x",
+        "nonce": "265",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3699098917",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "5263953",
+    "hash": "0x03a0f62a8106793dafcfae7b75fd2654322062d585a19cea568314d7205790dc",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x15482cc64b7c00a947f5bf015dfc010db1a6a668c74df61974d6a7848c174408",
+    "nonce": "0xd1bdb150f6fd170e",
+    "number": "2294630",
+    "stateRoot": "0x1ab1a534e84cc787cda1db21e0d5920ab06017948075b759166cfea7274657a1",
+    "timestamp": "1513675347",
```

```diff
+    "totalDifficulty": "7160543502214733"
+  },
+  "input": "0xf8ab820109855d21dba00082ca1d9443064693d3d38ad6a7cb579e0d6d9718c8aa6b6280b844a9059cbb000000000000000000000000e77b1ac803616503510bed0086e3a7be2627a699000000000000000000000000000000000
+  "result": {
+    "error": "out of gas",
+    "from": "0x94194bc2aaf494501d7880b61274a169f6502a54",
+    "gas": "0x7045",
+    "gasUsed": "0x7045",
+    "input": "0xa9059cbb000000000000000000000000e77b1ac803616503510bed0086e3a7be2627a6990000000000000000000000000000000000000000000000000000009502f9000",
+    "to": "0x43064693d3d38ad6a7cb579e0d6d9718c8aa6b62",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert.json
new file mode 100644
index 00000000..059040a1
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert.json
@@ -0,0 +1,58 @@
+{
+  "context": {
+    "difficulty": "3665057456",
+    "gasLimit": "5232723",
+    "miner": "0xf4d8e706cfb25c0decbbdd4d2e2cc10c66376a3f",
+    "number": "2294501",
+    "timestamp": "1513673601"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0f6cef2b7fbb504782e35aa82a2207e816a2b7a9": {
+        "balance": "0x2a3fc32bcc019283",
+        "code": "0x",
+        "nonce": "10",
+        "storage": {}
+      },
+      "0xabbcd5b340c80b5f1c0545c04c987b87310296ae": {
+        "balance": "0x0",
+        "code": "0x606060405236156100755763ffffffff7c0100000000000000000000000000000000000000000000000000000000600035041663 2d0335ab811461007a578063548db174146100ab5780637f649783146100fc578063b092145e146
+        "nonce": "1",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3672229776",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "5227619",
+    "hash": "0xa07b3d6c6bf63f5f981016db9f2d1d93033833f2c17e8bf7209e85f1faf08076",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0x806e151ce2817be922e93e8d5921fa0f0d0fd213d6b2b9a3fa17458e74a163d0",
+    "nonce": "0xbc5d43adc2c30c7d",
+    "number": "2294500",
+    "stateRoot": "0xca645b335888352ef9d8b1ef083e9019648180b259026572e3139717270de97d",
+    "timestamp": "1513673552",
+    "totalDifficulty": "7160066586979149"
+  },
+  "input": "0xf9018b0a8505d21dba00832dc6c094abbcd5b340c80b5f1c0545c04c987b87310296ae80b9012473b40a5c000000000000000000000000400de2e016bda6577407dfc379faba9899bc73ef0000000000000000000000002cc31912b2b0f3
+  "result": {
+    "error": "execution reverted",
+    "from": "0x0f6cef2b7fbb504782e35aa82a2207e816a2b7a9",
+    "gas": "0x2d55e8",
+    "gasUsed": "0xc3",
+    "input": "0x73b40a5c000000000000000000000000400de2e016bda6577407dfc379faba9899bc73ef0000000000000000000000002cc31912b2b0f3075a87b3640923d45a26cef3ee0000000000000000000000000000000000000000
+    "to": "0xabbcd5b340c80b5f1c0545c04c987b87310296ae",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert_reason.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert_reason.json
new file mode 100644
index 00000000..094b0446
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/revert_reason.json
@@ -0,0 +1,64 @@
+{
+  "context": {
+    "difficulty": "2",
+    "gasLimit": "8000000",
+    "miner": "0x0000000000000000000000000000000000000000",
+    "number": "3212651",
+    "timestamp": "1597246515"
+  },
+  "genesis": {
+    "alloc": {
+      "0xf58833cf0c791881b494eb79d461e08a1f043f52": {
+        "balance": "0x0",
+        "code": "0x608060405234801561001057600080fd5b50600436106100a5576000357c0100000000000000000000000000000000000000000000000000000000900480630609ff1bd11610078578063609ff1bd146101af5780639e7b8d6114610
+        "nonce": "1",
+        "storage": {
+          "0x6200beec95762de01ce05f2a0e58ce3299dbb53c68c9f3254a242121223cdf58": "0x0000000000000000000000000000000000000000000000000000000000000000"
+        }
+      },
+      "0xf7579c3d8a669c89d5ed246a22eb6db8f6fedbf1": {
+        "balance": "0x57af9d6b3df812900",
+        "code": "0x",
+        "nonce": "6",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 0,
+      "constantinopleBlock": 0,
+      "petersburgBlock": 0,
+      "IstanbulBlock":1561651,
+      "chainId": 5,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x0000000000000000000000000000000000000000000000000000000000000000",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf888068449504f80832dc6c094f58833cf0c791881b494eb79d461e08a1f043f5280a45c19a95c000000000000000000000000f7579c3d8a669c89d5ed246a22eb6db8f6fedbf12da0264664db3e71fae1dbdaf2f53954be149ad3b7ba8a
+  "result": {
+    "error": "execution reverted",
+    "from": "0xf7579c3d8a669c89d5ed246a22eb6db8f6fedbf1",
+    "gas": "0x2d7308",
+    "gasUsed": "0x588",
+    "input": "0x5c19a95c000000000000000000000000f7579c3d8a669c89d5ed246a22eb6db8f6fedbf1",
+    "to": "0xf58833cf0c791881b494eb79d461e08a1f043f52",
+    "type": "CALL",
+    "value": "0x0",
+    "output": "0x08c379a00000000000000000000000000000000000000000000000000000000000000020000000000000000000000000000000000000000000000000000000000000001e53656c662d64656c65676174696f6e20697320646973616c6
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/selfdestruct.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/selfdestruct.json
new file mode 100644
index 00000000..132cefa1
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/selfdestruct.json
@@ -0,0 +1,73 @@
+{
+  "context": {
+    "difficulty": "3502894804",
+    "gasLimit": "4722976",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2289806",
+    "timestamp": "1513601314"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "22",
+        "storage": {}
+      },
+      "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe": {
+        "balance": "0x4d87094125a369d9bd5",
+        "code": "0x61deadff",
+        "nonce": "1",
+        "storage": {}
+      },
+      "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb": {
+        "balance": "0x1780d77678137ac1b775",
+        "code": "0x",
+        "nonce": "29072",
+        "storage": {}
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf88b8271908506fc23ac0083015f90943b873a919aa0512d5a0f09e6dcceaa4a6727fafe80a463e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c52aa0bdce0b59e8761854e857fe64015f06dd08a4
+  "result": {
+    "calls": [
+      {
+        "from": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+        "input": "0x",
+        "to": "0x000000000000000000000000000000000000dEaD",
+        "type": "SELFDESTRUCT",
+        "value": "0x4d87094125a369d9bd5"
+      }
+    ],
+    "from": "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb",
+    "gas": "0x10738",
+    "gasUsed": "0x7533",
+    "input": "0x63e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+    "output": "0x",
+    "to": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/simple.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/simple.json
new file mode 100644
index 00000000..b4643212
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/simple.json
@@ -0,0 +1,78 @@
+{
+  "context": {
+    "difficulty": "3502894804",
+    "gasLimit": "4722976",
+    "miner": "0x1585936b53834b021f68cc13eeefdec2efc8e724",
+    "number": "2289806",
+    "timestamp": "1513601314"
+  },
+  "genesis": {
+    "alloc": {
+      "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5": {
+        "balance": "0x0",
+        "code": "0x",
+        "nonce": "22",
+        "storage": {}
+      },
+      "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe": {
+        "balance": "0x4d87094125a369d9bd5",
+        "code": "0x606060405236156100935763ffffffff60e060020a60003504166311ee8382811461009c57806313af4035146100be5780631f5e8f4c146100ee57806324daddc5146101125780634921a91a1461013b57806363e4bff4146101575
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000000": "0x0000000000000000000000001b436ba50d378d4bbc8660d312a13df6af6e89dfb",
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x00000000000000000000000000000000000000000000000006f05b59d3b20000",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x000000000000000000000000000000000000000000000000000000000000003c",
+          "0x0000000000000000000000000000000000000000000000000000000000000003": "0x000000000000000000000000000000000000000000000000000000005a37b834"
+        }
+      },
+      "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb": {
+        "balance": "0x1780d77678137ac1b775",
+        "code": "0x",
+        "nonce": "29072",
+        "storage": {}
+      }
+    },
+    "config": {
```

```diff
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "3509749784",
+    "extraData": "0x4554482e45544846414e532e4f52472d4641313738394444",
+    "gasLimit": "4727564",
+    "hash": "0x609948ac3bd3c00b7736b933248891d6c901ee28f066241bddb28f4e00a9f440",
+    "miner": "0xbbf5029fd710d227630c8b7d338051b8e76d50b3",
+    "mixHash": "0xb131e4507c93c7377de00e7c271bf409ec7492767142ff0f45c882f8068c2ada",
+    "nonce": "0x4eb12e19c16d43da",
+    "number": "2289805",
+    "stateRoot": "0xc7f10f352bff82fac3c2999d3085093d12652e19c7fd32591de49dc5d91b4f1f",
+    "timestamp": "1513601261",
+    "totalDifficulty": "7143276353481064"
+  },
+  "input": "0xf88b8271908506fc23ac0083015f90943b873a919aa0512d5a0f09e6dcceaa4a6727fafe80a463e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c52aa0bdce0b59e8761854e857fe64015f06dd08a4
+  "result": {
+    "calls": [
+      {
+        "from": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+        "input": "0x",
+        "to": "0x0024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+        "type": "CALL",
+        "value": "0x6f05b59d3b20000"
+      }
+    ],
+    "from": "0xb436ba50d378d4bbc8660d312a13df6af6e89dfb",
+    "gas": "0x10738",
+    "gasUsed": "0x3ef9",
+    "input": "0x63e4bff40000000000000000000000000024f658a46fbb89d8ac105e98d7ac7cbbaf27c5",
+    "output": "0x0000000000000000000000000000000000000000000000000000000000000001",
+    "to": "0x3b873a919aa0512d5a0f09e6dcceaa4a6727fafe",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/throw.json b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/throw.json
new file mode 100644
index 00000000..09cf4497
--- /dev/null
+++ b/eth/tracers/internal/tracetest/testdata/call_tracer_legacy/throw.json
@@ -0,0 +1,62 @@
+{
+  "context": {
+    "difficulty": "117009631",
+    "gasLimit": "4712388",
+    "miner": "0x294e5d6c39a36ce38af1dca70c1060f78dee8070",
+    "number": "25009",
+    "timestamp": "1479891666"
+  },
+  "genesis": {
+    "alloc": {
+      "0x70c9217d814985faef62b124420f8dfbddd96433": {
+        "balance": "0x4ecd70668f5d854a",
+        "code": "0x",
+        "nonce": "1638",
+        "storage": {}
+      },
+      "0xc212e03b9e060e36facad5fd8f4435412ca22e6b": {
+        "balance": "0x0",
+        "code": "0x606060405236156101745760e060020a600035046302d05d3f811461017c57806304a7fdbc1461018e5780630e90f957146101fb5780630fb5a6b4146102125780631
4baa1b61461021b57806317fc45e21461023a5780632b09692
+        "nonce": "1",
+        "storage": {
+          "0x0000000000000000000000000000000000000000000000000000000000000001": "0x0000000000000000000000002cccf5e0538493c235d1c5ef6580f77d99e91396",
+          "0x0000000000000000000000000000000000000000000000000000000000000002": "0x00000000000000000000000000000000000000000000000000000000000061a9",
+          "0x0000000000000000000000000000000000000000000000000000000000000005": "0x00000000000000000000000070c9217d814985faef62b124420f8dfbddd96433"
+        }
+      }
+    },
+    "config": {
+      "byzantiumBlock": 1700000,
+      "chainId": 3,
+      "daoForkSupport": true,
+      "eip150Block": 0,
+      "eip150Hash": "0x41941023680923e0fe4d74a34bdac8141f2540e3ae90623718e47d66d1ca4a2d",
+      "eip155Block": 10,
+      "eip158Block": 10,
+      "ethash": {},
+      "homesteadBlock": 0
+    },
+    "difficulty": "117066792",
+    "extraData": "0xd783010502846765746887676f312e372e33856c696e7578",
+    "gasLimit": "4712388",
+    "hash": "0xe23e8d4562a1045b70cbc99fefb20c101a8f0fc8559a80d65fea8896e2f1d46e",
+    "miner": "0x71842f946b98800fe6feb49f0ae4e253259031c9",
+    "mixHash": "0x0aada9d6e93dd4db0d09c0488dc0a048fca2ccdc1f3fc7b83ba2a8d393a3a4ff",
+    "nonce": "0x70849d5838dee2e9",
+    "number": "25008",
+    "stateRoot": "0x1e01d2161794768c5b917069e73d86e8dca80cd7f3168c0597de420ab93a3b7b",
+    "timestamp": "1479891641",
+    "totalDifficulty": "1896347038589"
+  },
+  "input": "0xf88b8206668504a817c8008303d09094c212e03b9e060e36facad5fd8f4435412ca22e6b80a451a34eb8000000000000000000000000000000000000000000000027fad02094277c000029a0692a3b4e7b2842f8dd7832e712c21e09f451
+  "result": {
+    "error": "invalid jump destination",
+    "from": "0x70c9217d814985faef62b124420f8dfbddd96433",
+    "gas": "0x37b38",
+    "gasUsed": "0x37b38",
+    "input": "0x51a34eb8000000000000000000000000000000000000000000000027fad02094277c0000",
+    "to": "0xc212e03b9e060e36facad5fd8f4435412ca22e6b",
+    "type": "CALL",
+    "value": "0x0"
+  }
+}
diff --git a/eth/tracers/js/bigint.go b/eth/tracers/js/bigint.go
new file mode 100644
index 00000000..858fd057
--- /dev/null
+++ b/eth/tracers/js/bigint.go
@@ -0,0 +1,30 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
```

```
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package js
+
+// bigIntegerJS is the minified version of https://github.com/peterolson/BigInteger.js.
+const bigIntegerJS = `var bigInt=function(undefined){"use strict";var BASE=1e7,LOG_BASE=7,MAX_INT=9007199254740992,MAX_INT_ARR=smallToArray(MAX_INT),LOG_MAX_INT=Math.log(MAX_INT);function Integer(v,radi
diff --git a/eth/tracers/js/internal/tracers/4byte_tracer_legacy.js b/eth/tracers/js/internal/tracers/4byte_tracer_legacy.js
new file mode 100644
index 00000000..c27f9ae0
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/4byte_tracer_legacy.js
@@ -0,0 +1,96 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// 4byteTracer searches for 4byte-identifiers, and collects them for post-processing.
+// It collects the methods identifiers along with the size of the supplied data, so
+// a reversed signature can be matched against the size of the data.
+//
+// Example:
+//   > debug.traceTransaction( "0x214e597e35da083692f5386141e69f47e973b2c56e7a8073b1ea08fd7571e9de", {tracer: "4byteTracer"})
+//   {
+//     0x27dc297e-128: 1,
+//     0x38cc4831-0: 2,
+//     0x524f3889-96: 1,
+//     0xadf59f99-288: 1,
+//     0xc281d19e-0: 1
+//   }
+{
+       // ids aggregates the 4byte ids found.
+       ids : {},
+
+       // callType returns 'false' for non-calls, or the peek-index for the first param
+       // after 'value', i.e. meminstart.
+       callType: function(opstr){
+               switch(opstr){
+               case "CALL": case "CALLCODE":
+                       // gas, addr, val, memin, meminsz, memout, memoutsz
+                       return 3; // stack ptr to memin
+
+               case "DELEGATECALL": case "STATICCALL":
+                       // gas, addr, memin, meminsz, memout, memoutsz
+                       return 2; // stack ptr to memin
+               }
+               return false;
+       },
+
+       // store save the given indentifier and datasize.
+       store: function(id, size){
+               var key = "" + toHex(id) + "-" + size;
+               this.ids[key] = this.ids[key] + 1 || 1;
+       },
+
+       // step is invoked for every opcode that the VM executes.
+       step: function(log, db) {
+               // Skip any opcodes that are not internal calls
+               var ct = this.callType(log.op.toString());
+               if (!ct) {
+                       return;
+               }
+               // Skip any pre-compile invocations, those are just fancy opcodes
+               if (isPrecompiled(toAddress(log.stack.peek(1).toString(16)))) {
+                       return;
+               }
+               // Gather internal call details
+               var inSz = log.stack.peek(ct + 1).valueOf();
+               if (inSz >= 4) {
+                       var inOff = log.stack.peek(ct).valueOf();
+                       this.store(log.memory.slice(inOff, inOff + 4), inSz-4);
+               }
+       },
+
+       // fault is invoked when the actual execution of an opcode fails.
+       fault: function(log, db) { },
+
+       // result is invoked when all the opcodes have been iterated over and returns
+       // the final result of the tracing.
+       result: function(ctx) {
+               // Save the outer calldata also
+               if (ctx.input.length >= 4) {
+                       this.store(slice(ctx.input, 0, 4), ctx.input.length-4)
+               }
+               return this.ids;
+       },
+}
diff --git a/eth/tracers/js/internal/tracers/assets.go b/eth/tracers/js/internal/tracers/assets.go
new file mode 100644
index 00000000..a2bb69de
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/assets.go
@@ -0,0 +1,481 @@
+// Code generated by go-bindata. DO NOT EDIT.
+// sources:
+// 4byte_tracer_legacy.js (2.933kB)
+// bigram_tracer.js (1.712kB)
+// call_tracer_js.js (3.497kB)
+// call_tracer_legacy.js (8.956kB)
+// evmdis_tracer.js (4.215kB)
+// noop_tracer.js (1.271kB)
+// opcount_tracer.js (1.372kB)
+// prestate_tracer.js (4.287kB)
+// trigram_tracer.js (1.788kB)
+// unigram_tracer.js (1.469kB)
+
+package tracers
+
+import (
+       "bytes"
+       "compress/gzip"
+       "crypto/sha256"
+       "fmt"
+       "io"
+       "io/ioutil"
+       "os"
+       "path/filepath"
+       "strings"
+       "time"
+)
+
+func bindataRead(data []byte, name string) ([]byte, error) {
+       gz, err := gzip.NewReader(bytes.NewBuffer(data))
+       if err != nil {
+               return nil, fmt.Errorf("read %q: %w", name, err)
+       }
+
+       var buf bytes.Buffer
+       _, err = io.Copy(&buf, gz)
+       clErr := gz.Close()
+
+       if err != nil {
+               return nil, fmt.Errorf("read %q: %w", name, err)
+       }
+       if clErr != nil {
+               return nil, err
+       }
+
+       return buf.Bytes(), nil
+}
+
+type asset struct {
+       bytes  []byte
+       info   os.FileInfo
+       digest [sha256.Size]byte
+}
+
+type bindataFileInfo struct {
+       name    string
+       size    int64
+       mode    os.FileMode
+       modTime time.Time
+}
+
+func (fi bindataFileInfo) Name() string {
+       return fi.name
+}
+func (fi bindataFileInfo) Size() int64 {
+       return fi.size
+}
+func (fi bindataFileInfo) Mode() os.FileMode {
+       return fi.mode
+}
+func (fi bindataFileInfo) ModTime() time.Time {
+       return fi.modTime
+}
+func (fi bindataFileInfo) IsDir() bool {
+       return false
+}
+func (fi bindataFileInfo) Sys() interface{} {
+       return nil
+}
+
+var __4byte_tracer_legacyJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x94\x56\x5b\x6f\xdb\x4a\x0e\x7e\xb6\x7f\x05\xd7\x2f\xb5\x51\x59\x8e\x2f\x89\x2f\xd9\x16\xf0\xe6\xa4\x6d\x80\x9c\x24\x88\xdd
+
+func _4byte_tracer_legacyJsBytes() ([]byte, error) {
+       return bindataRead(
+               __4byte_tracer_legacyJs,
+               "4byte_tracer_legacy.js",
+       )
+}
+
+func _4byte_tracer_legacyJs() (*asset, error) {
+       bytes, err := _4byte_tracer_legacyJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "4byte_tracer_legacy.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0xb4, 0xc5, 0x48, 0x2d, 0xd9, 0x43, 0x95, 0x93, 0x3b, 0x93, 0x2c, 0x47, 0x8c, 0x84, 0x32, 0x3c, 0x8b, 0x2e, 0xf3, 0x72, 0xc4, 0x57, 0xe6, 0
+       return a, nil
+}
+
+var _bigram_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x54\x5b\x6f\xdb\x36\x14\x7e\xf7\xaf\xf8\xde\x92\x20\xae\xd4\x6e\x2f\x83\x33\x0f\xd0\xb2\xa4\x35\x90\xda\x81\xad\xac\x30\x86\x3
+
+func bigram_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _bigram_tracerJs,
+               "bigram_tracer.js",
+       )
+}
+
+func bigram_tracerJs() (*asset, error) {
+       bytes, err := bigram_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "bigram_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x77, 0x6c, 0xd, 0x24, 0xf2, 0x49, 0xbd, 0x58, 0x8b, 0xb5, 0xd1, 0xc9, 0xcd, 0xcf, 0x5b, 0x3e, 0x5c, 0xfb, 0x14, 0x50, 0xe7, 0xe3, 0xb9, 0x
+       return a, nil
+}
+
+var _call_tracer_jsJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x56\x5f\x6f\xdb\x38\x0c\x7f\x8e\x3f\x05\xaf\x0f\x4b\x82\x65\x71\xbb\x03\xf6\xd0\x2d\x03\x72\x45\xbb\x05\xe8\xb5\x45\x9a\xde\x
+
+func call_tracer_jsJsBytes() ([]byte, error) {
+       return bindataRead(
+               _call_tracer_jsJs,
+               "call_tracer_js.js",
+       )
+}
+
+func call_tracer_jsJs() (*asset, error) {
+       bytes, err := call_tracer_jsJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "call_tracer_js.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x42, 0x13, 0x7a, 0x14, 0xbf, 0xa7, 0x49, 0x4f, 0xb4, 0x4f, 0x45, 0x1, 0xbc, 0x9e, 0xd1, 0x8e, 0xc7, 0xee, 0x61, 0xfa, 0x82, 0x52, 0xa4, 0x
+       return a, nil
+}
+
+var _call_tracer_legacyJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\xd4\x5a\xdf\x6f\x1b\x37\xf2\x7f\x96\xfe\x8a\x89\x1f\x6a\x09\x51\x24\x39\xe9\xb7\x5f\xc0\xae\x7a\x50\x1d\x25\x35\xe0\xc6\x81\x
+
+func call_tracer_legacyJsBytes() ([]byte, error) {
+       return bindataRead(
+               _call_tracer_legacyJs,
+               "call_tracer_legacy.js",
+       )
```

```
+}
+
+func call_tracer_legacyJs() (*asset, error) {
+       bytes, err := call_tracer_legacyJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "call_tracer_legacy.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x46, 0x79, 0xb6, 0xbc, 0xd2, 0xc, 0x25, 0xb1, 0x22, 0x56, 0xef, 0x77, 0xb9, 0x5e, 0x2e, 0xf4, 0xda, 0xb2, 0x2f, 0x53, 0xa4, 0xff, 0xc8, 0x
+       return a, nil
+}
+
+var _evmdis_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\xac\x97\x6f\x6f\xda\x48\x13\xc0\x5f\xc3\xa7\x18\xe5\x15\xa8\x14\xb0\x31\x04\x9c\xcb\x49\x3c\x29\xe9\xe5\x51\x9a\x44\x40\xee\x54\xa
+
+func evmdis_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _evmdis_tracerJs,
+               "evmdis_tracer.js",
+       )
+}
+
+func evmdis_tracerJs() (*asset, error) {
+       bytes, err := evmdis_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "evmdis_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x13, 0xeb, 0xca, 0x1f, 0x5f, 0xd3, 0x29, 0x81, 0xbb, 0xd8, 0xc8, 0x4a, 0x3a, 0x38, 0x10, 0xe2, 0xe7, 0xa4, 0xcd, 0xde, 0x78, 0x85, 0xc2, 0
+       return a, nil
+}
+
+var _noop_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x93\x4f\x6f\xdb\x46\x10\xc5\xcf\xe6\xa7\x78\xc7\x04\x50\xc5\xfe\x39\x14\x70\x8a\x02\xac\x61\x27\x2a\x1c\xdb\x90\xe8\x06\x3e\x0e\
+
+func noop_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _noop_tracerJs,
+               "noop_tracer.js",
+       )
+}
+
+func noop_tracerJs() (*asset, error) {
+       bytes, err := noop_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "noop_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0xe3, 0xf, 0x1c, 0x6f, 0x65, 0xaf, 0x90, 0x31, 0xab, 0xf, 0xe0, 0xca, 0x54, 0x7, 0xfd, 0xd3, 0xa1, 0x4a, 0x14, 0x1, 0x2a, 0x9d, 0xdc, 0xb9,
+       return a, nil
+}
+
+var _opcount_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x94\xcf\x6e\xdb\x46\x10\x87\xcf\xe2\x53\xfc\x8e\x09\xa2\x92\x69\x7b\x28\xe0\x16\x05\x58\xc3\x4e\x04\xd8\xb2\x21\xd1\x09\x7c\x
+
+func opcount_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _opcount_tracerJs,
+               "opcount_tracer.js",
+       )
+}
+
+func opcount_tracerJs() (*asset, error) {
+       bytes, err := opcount_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "opcount_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x27, 0xe, 0x97, 0x88, 0x9b, 0x53, 0xbb, 0x20, 0x44, 0xd8, 0xf5, 0xeb, 0x41, 0xd2, 0x7e, 0xd6, 0xda, 0x6b, 0xf5, 0xaf, 0x0, 0x75, 0x9f, 0xd
+       return a, nil
+}
+
+var _prestate_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x9c\x57\xdd\x6f\xdb\x38\x12\x7f\xb6\xfe\x8a\x41\x5f\x6c\x5d\x5d\xb9\xcd\x02\x7b\x80\x73\x39\x40\x75\xdd\x36\x40\x36\x09\x6c\xe7\
+
+func prestate_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _prestate_tracerJs,
+               "prestate_tracer.js",
+       )
+}
+
+func prestate_tracerJs() (*asset, error) {
+       bytes, err := prestate_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "prestate_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0xd4, 0x9, 0xf9, 0x44, 0x13, 0x31, 0x89, 0xf7, 0x35, 0x9a, 0xc6, 0xf0, 0x86, 0x9d, 0xb2, 0xe3, 0x57, 0xe2, 0xc0, 0xde, 0xc9, 0x3a, 0x4c, 0x
+       return a, nil
+}
+
+var _trigram_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x94\x4f\x6f\xe3\x36\x10\xc5\xef\xfe\x14\xaf\x27\x27\x88\xd7\x4a\xda\x4b\xe1\xd4\x05\xdc\x6c\xb2\x6b\x20\x6b\x07\xb6\xd2\x45\x
+
+func trigram_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _trigram_tracerJs,
+               "trigram_tracer.js",
+       )
+}
+
+func trigram_tracerJs() (*asset, error) {
+       bytes, err := trigram_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "trigram_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0x40, 0x63, 0xe1, 0x42, 0x60, 0x7, 0x1b, 0x79, 0x47, 0x1, 0xa1, 0xbf, 0xc4, 0x66, 0x19, 0x9b, 0x2b, 0x5a, 0x1f, 0x82, 0x3d, 0xcf, 0xee, 0xe
+       return a, nil
+}
+
+var _unigram_tracerJs = []byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x8c\x94\x41\x6f\xdb\xc6\x13\xc5\xef\xfa\x14\xef\x68\x23\xfa\x8b\xc9\xbf\x97\x42\x69\x0a\xb0\x86\x9d\x08\x70\x64\x43\xa2\x1b\x18\x
+
+func unigram_tracerJsBytes() ([]byte, error) {
+       return bindataRead(
+               _unigram_tracerJs,
+               "unigram_tracer.js",
+       )
+}
+
+func unigram_tracerJs() (*asset, error) {
+       bytes, err := unigram_tracerJsBytes()
+       if err != nil {
+               return nil, err
+       }
+
+       info := bindataFileInfo{name: "unigram_tracer.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
+       a := &asset{bytes: bytes, info: info, digest: [32]uint8{0xc, 0xe6, 0x5c, 0x88, 0x18, 0xa7, 0x85, 0x61, 0x18, 0xc6, 0xec, 0x17, 0xfc, 0xdf, 0x9d, 0xc0, 0x1b, 0x49, 0xf8, 0x8d, 0xf1, 0xeb, 0x35, 0x
+       return a, nil
+}
+
+// Asset loads and returns the asset for the given name.
+// It returns an error if the asset could not be found or
+// could not be loaded.
+func Asset(name string) ([]byte, error) {
+       canonicalName := strings.Replace(name, "\\", "/", -1)
+       if f, ok := _bindata[canonicalName]; ok {
+               a, err := f()
+               if err != nil {
+                       return nil, fmt.Errorf("Asset %s can't read by error: %v", name, err)
+               }
+               return a.bytes, nil
+       }
+       return nil, fmt.Errorf("Asset %s not found", name)
+}
+
+// AssetString returns the asset contents as a string (instead of a []byte).
+func AssetString(name string) (string, error) {
+       data, err := Asset(name)
+       return string(data), err
+}
+
+// MustAsset is like Asset but panics when Asset would return an error.
+// It simplifies safe initialization of global variables.
+func MustAsset(name string) []byte {
+       a, err := Asset(name)
+       if err != nil {
+               panic("asset: Asset(" + name + "): " + err.Error())
+       }
+
+       return a
+}
+
+// MustAssetString is like AssetString but panics when Asset would return an
+// error. It simplifies safe initialization of global variables.
+func MustAssetString(name string) string {
+       return string(MustAsset(name))
+}
+
+// AssetInfo loads and returns the asset info for the given name.
+// It returns an error if the asset could not be found or
+// could not be loaded.
+func AssetInfo(name string) (os.FileInfo, error) {
+       canonicalName := strings.Replace(name, "\\", "/", -1)
+       if f, ok := _bindata[canonicalName]; ok {
+               a, err := f()
+               if err != nil {
+                       return nil, fmt.Errorf("AssetInfo %s can't read by error: %v", name, err)
+               }
+               return a.info, nil
+       }
+       return nil, fmt.Errorf("AssetInfo %s not found", name)
+}
+
+// AssetDigest returns the digest of the file with the given name. It returns an
+// error if the asset could not be found or the digest could not be loaded.
+func AssetDigest(name string) ([sha256.Size]byte, error) {
+       canonicalName := strings.Replace(name, "\\", "/", -1)
+       if f, ok := _bindata[canonicalName]; ok {
+               a, err := f()
+               if err != nil {
+                       return [sha256.Size]byte{}, fmt.Errorf("AssetDigest %s can't read by error: %v", name, err)
+               }
+               return a.digest, nil
+       }
+       return [sha256.Size]byte{}, fmt.Errorf("AssetDigest %s not found", name)
+}
+
+// Digests returns a map of all known files and their checksums.
+func Digests() (map[string][sha256.Size]byte, error) {
+       mp := make(map[string][sha256.Size]byte, len(_bindata))
+       for name := range _bindata {
+               a, err := _bindata[name]()
+               if err != nil {
+                       return nil, err
+               }
+               mp[name] = a.digest
+       }
+       return mp, nil
+}
+
+// AssetNames returns the names of the assets.
+func AssetNames() []string {
+       names := make([]string, 0, len(_bindata))
+       for name := range _bindata {
+               names = append(names, name)
+       }
+       return names
+}
+
+// _bindata is a table, holding each asset generator, mapped to its name.
+var _bindata = map[string]func() (*asset, error){
+       "4byte_tracer_legacy.js": _4byte_tracer_legacyJs,
+       "bigram_tracer.js":       bigram_tracerJs,
+       "call_tracer_js.js":      call_tracer_jsJs,
+       "call_tracer_legacy.js":  call_tracer_legacyJs,
+       "evmdis_tracer.js":       evmdis_tracerJs,
+       "noop_tracer.js":         noop_tracerJs,
+       "opcount_tracer.js":      opcount_tracerJs,
+       "prestate_tracer.js":     prestate_tracerJs,
+       "trigram_tracer.js":      trigram_tracerJs,
+       "unigram_tracer.js":      unigram_tracerJs,
+}
+
+// AssetDebug is true if the assets were built with the debug flag enabled.
+const AssetDebug = false
+
+// AssetDir returns the file names below a certain
+// directory embedded in the file by go-bindata.
+// For example if you run go-bindata on data/... and data contains the
+// following hierarchy:
+//     data/
+//       foo.txt
+//       img/
+//         a.png
+//         b.png
+// then AssetDir("data") would return []string{"foo.txt", "img"},
+// AssetDir("data/img") would return []string{"a.png", "b.png"},
+// AssetDir("foo.txt") and AssetDir("notexist") would return an error, and
+// AssetDir("") will return []string{"data"}.
+func AssetDir(name string) ([]string, error) {
+       node := _bintree
+       if len(name) != 0 {
+               canonicalName := strings.Replace(name, "\\", "/", -1)
+               pathList := strings.Split(canonicalName, "/")
+               for _, p := range pathList {
+                       node = node.Children[p]
+                       if node == nil {
+                               return nil, fmt.Errorf("Asset %s not found", name)
+                       }
+               }
+       }
+       if node.Func != nil {
+               return nil, fmt.Errorf("Asset %s not found", name)
+       }
+       rv := make([]string, 0, len(node.Children))
+       for childName := range node.Children {
+               rv = append(rv, childName)
+       }
+       return rv, nil
+}
+
+type bintree struct {
+       Func     func() (*asset, error)
+       Children map[string]*bintree
+}
+
+var _bintree = &bintree{nil, map[string]*bintree{
+       "4byte_tracer_legacy.js": {_4byte_tracer_legacyJs, map[string]*bintree{}},
+       "bigram_tracer.js":       {bigram_tracerJs, map[string]*bintree{}},
+       "call_tracer_js.js":      {call_tracer_jsJs, map[string]*bintree{}},
+       "call_tracer_legacy.js":  {call_tracer_legacyJs, map[string]*bintree{}},
+       "evmdis_tracer.js":       {evmdis_tracerJs, map[string]*bintree{}},
+       "noop_tracer.js":         {noop_tracerJs, map[string]*bintree{}},
+       "opcount_tracer.js":      {opcount_tracerJs, map[string]*bintree{}},
+       "prestate_tracer.js":     {prestate_tracerJs, map[string]*bintree{}},
+       "trigram_tracer.js":      {trigram_tracerJs, map[string]*bintree{}},
+       "unigram_tracer.js":      {unigram_tracerJs, map[string]*bintree{}},
+}}
+
+// RestoreAsset restores an asset under the given directory.
+func RestoreAsset(dir, name string) error {
+       data, err := Asset(name)
+       if err != nil {
+               return err
+       }
+       info, err := AssetInfo(name)
+       if err != nil {
+               return err
+       }
+       err = os.MkdirAll(_filePath(dir, filepath.Dir(name)), os.FileMode(0755))
+       if err != nil {
+               return err
+       }
+       err = ioutil.WriteFile(_filePath(dir, name), data, info.Mode())
+       if err != nil {
+               return err
+       }
+       return os.Chtimes(_filePath(dir, name), info.ModTime(), info.ModTime())
+}
+
+// RestoreAssets restores an asset under the given directory recursively.
+func RestoreAssets(dir, name string) error {
+       children, err := AssetDir(name)
+       // File
+       if err != nil {
+               return RestoreAsset(dir, name)
+       }
+       // Dir
+       for _, child := range children {
+               err = RestoreAssets(dir, filepath.Join(name, child))
+               if err != nil {
+                       return err
+               }
+       }
+       return nil
+}
+
+func _filePath(dir, name string) string {
+       canonicalName := strings.Replace(name, "\\", "/", -1)
+       return filepath.Join(append([]string{dir}, strings.Split(canonicalName, "/")...)...)
+}
diff --git a/eth/tracers/js/internal/tracers/bigram_tracer.js b/eth/tracers/js/internal/tracers/bigram_tracer.js
new file mode 100644
index 00000000..24f9af95
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/bigram_tracer.js
@@ -0,0 +1,57 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2018 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+{
+    // hist is the counters of opcode bigrams
+    hist: {},
+    // lastOp is last operation
+    lastOp: '',
+    // execution depth of last op
+    lastDepth: 0,
+    // step is invoked for every opcode that the VM executes.
+    step: function(log, db) {
+        var op = log.op.toString();
+        var depth = log.getDepth();
+        if (depth == this.lastDepth){
+            var key = this.lastOp+'-'+op;
+            if (this.hist[key]){
+                this.hist[key]++;
+            }
+            else {
+                this.hist[key] = 1;
+            }
+        }
+        this.lastOp = op;
+        this.lastDepth = depth;
+    },
+    // fault is invoked when the actual execution of an opcode fails.
+    fault: function(log, db) {},
+    // result is invoked when all the opcodes have been iterated over and returns
+    // the final result of the tracing.
+    result: function(ctx) {
+        return this.hist;
+    },
+}
diff --git a/eth/tracers/js/internal/tracers/call_tracer_js.js b/eth/tracers/js/internal/tracers/call_tracer_js.js
new file mode 100644
index 00000000..31c20a28
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/call_tracer_js.js
```

```
@@ -0,0 +1,122 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+
+// callFrameTracer uses the new call frame tracing methods to report useful information
+// about internal messages of a transaction.
+{
+	callstack: [{}],
+	fault: function(log, db) {},
+	result: function(ctx, db) {
+		// Prepare outer message info
+		var result = {
+			type:    ctx.type,
+			from:    toHex(ctx.from),
+			to:      toHex(ctx.to),
+			value:   '0x' + ctx.value.toString(16),
+			gas:     '0x' + bigInt(ctx.gas).toString(16),
+			gasUsed: '0x' + bigInt(ctx.gasUsed).toString(16),
+			input:   toHex(ctx.input),
+			output:  toHex(ctx.output),
+		}
+		if (this.callstack[0].calls !== undefined) {
+			result.calls = this.callstack[0].calls
+		}
+		if (this.callstack[0].error !== undefined) {
+			result.error = this.callstack[0].error
+		} else if (ctx.error !== undefined) {
+			result.error = ctx.error
+		}
+		if (result.error !== undefined && (result.error !== "execution reverted" || result.output ==="0x")) {
+			delete result.output
+		}
+
+		return this.finalize(result)
+	},
+	enter: function(frame) {
+		var call = {
+			type: frame.getType(),
+			from: toHex(frame.getFrom()),
+			to: toHex(frame.getTo()),
+			input: toHex(frame.getInput()),
+			gas: '0x' + bigInt(frame.getGas()).toString('16'),
+		}
+		if (frame.getValue() !== undefined){
+			call.value='0x' + bigInt(frame.getValue()).toString(16)
+		}
+		this.callstack.push(call)
+	},
+	exit: function(frameResult) {
+		var len = this.callstack.length
+		if (len > 1) {
+			var call = this.callstack.pop()
+			call.gasUsed = '0x' + bigInt(frameResult.getGasUsed()).toString('16')
+			var error = frameResult.getError()
+			if (error === undefined) {
+				call.output = toHex(frameResult.getOutput())
+			} else {
+				call.error = error
+				if (call.type === 'CREATE' || call.type === 'CREATE2') {
+					delete call.to
+				}
+			}
+			len -= 1
+			if (this.callstack[len-1].calls === undefined) {
+				this.callstack[len-1].calls = []
+			}
+			this.callstack[len-1].calls.push(call)
+		}
+	},
+	// finalize recreates a call object using the final desired field oder for json
+	// serialization. This is a nicety feature to pass meaningfully ordered results
+	// to users who don't interpret it, just display it.
+	finalize: function(call) {
+		var sorted = {
+			type:    call.type,
+			from:    call.from,
+			to:      call.to,
+			value:   call.value,
+			gas:     call.gas,
+			gasUsed: call.gasUsed,
+			input:   call.input,
+			output:  call.output,
+			error:   call.error,
+			time:    call.time,
+			calls:   call.calls,
+		}
+		for (var key in sorted) {
+			if (sorted[key] === undefined) {
+				delete sorted[key]
+			}
+		}
+		if (sorted.calls !== undefined) {
+			for (var i=0; i<sorted.calls.length; i++) {
+				sorted.calls[i] = this.finalize(sorted.calls[i])
+			}
+		}
+		return sorted
+	}
+}
diff --git a/eth/tracers/js/internal/tracers/call_tracer_legacy.js b/eth/tracers/js/internal/tracers/call_tracer_legacy.js
new file mode 100644
index 00000000..7081dfaa
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/call_tracer_legacy.js
@@ -0,0 +1,262 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
```

```
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// callTracer is a full blown transaction tracer that extracts and reports all
+// the internal calls made by a transaction, along with any useful information.
+{
+	// callstack is the current recursive call stack of the EVM execution.
+	callstack: [{}],
+
+	// descended tracks whether we've just descended from an outer transaction into
+	// an inner call.
+	descended: false,
+
+	// step is invoked for every opcode that the VM executes.
+	step: function(log, db) {
+		// Capture any errors immediately
+		var error = log.getError();
+		if (error !== undefined) {
+			this.fault(log, db);
+			return;
+		}
+		// We only care about system opcodes, faster if we pre-check once
+		var syscall = (log.op.toNumber() & 0xf0) == 0xf0;
+		if (syscall) {
+			var op = log.op.toString();
+		}
+		// If a new contract is being created, add to the call stack
+		if (syscall && (op == 'CREATE' || op == "CREATE2")) {
+			var inOff = log.stack.peek(1).valueOf();
+			var inEnd = inOff + log.stack.peek(2).valueOf();
+
+			// Assemble the internal call report and store for completion
+			var call = {
+				type:    op,
+				from:    toHex(log.contract.getAddress()),
+				input:   toHex(log.memory.slice(inOff, inEnd)),
+				gasIn:   log.getGas(),
+				gasCost: log.getCost(),
+				value:   '0x' + log.stack.peek(0).toString(16)
+			};
+			this.callstack.push(call);
+			this.descended = true
+			return;
+		}
+		// If a contract is being self destructed, gather that as a subcall too
+		if (syscall && op == 'SELFDESTRUCT') {
+			var left = this.callstack.length;
+			if (this.callstack[left-1].calls === undefined) {
+				this.callstack[left-1].calls = [];
+			}
+			this.callstack[left-1].calls.push({
+				type:    op,
+				from:    toHex(log.contract.getAddress()),
+				to:      toHex(toAddress(log.stack.peek(0).toString(16))),
+				gasIn:   log.getGas(),
+				gasCost: log.getCost(),
+				value:   '0x' + db.getBalance(log.contract.getAddress()).toString(16)
+			});
+			return
+		}
+		// If a new method invocation is being done, add to the call stack
+		if (syscall && (op == 'CALL' || op == 'CALLCODE' || op == 'DELEGATECALL' || op == 'STATICCALL')) {
+			// Skip any pre-compile invocations, those are just fancy opcodes
+			var to = toAddress(log.stack.peek(1).toString(16));
+			if (isPrecompiled(to)) {
+				return
+			}
+			var off = (op == 'DELEGATECALL' || op == 'STATICCALL' ? 0 : 1);
+
+			var inOff = log.stack.peek(2 + off).valueOf();
+			var inEnd = inOff + log.stack.peek(3 + off).valueOf();
+
+			// Assemble the internal call report and store for completion
+			var call = {
+				type:    op,
+				from:    toHex(log.contract.getAddress()),
+				to:      toHex(to),
+				input:   toHex(log.memory.slice(inOff, inEnd)),
+				gasIn:   log.getGas(),
+				gasCost: log.getCost(),
+				outOff:  log.stack.peek(4 + off).valueOf(),
+				outLen:  log.stack.peek(5 + off).valueOf()
+			};
+			if (op != 'DELEGATECALL' && op != 'STATICCALL') {
+				call.value = '0x' + log.stack.peek(2).toString(16);
+			}
+			this.callstack.push(call);
+			this.descended = true
+			return;
+		}
+		// If we've just descended into an inner call, retrieve it's true allowance. We
+		// need to extract if from within the call as there may be funky gas dynamics
+		// with regard to requested and actually given gas (2300 stipend, 63/64 rule).
+		if (this.descended) {
+			if (log.getDepth() >= this.callstack.length) {
+				this.callstack[this.callstack.length - 1].gas = log.getGas();
+			} else {
+				// TODO(karalabe): The call was made to a plain account. We currently don't
+				// have access to the true gas amount inside the call and so any amount will
+				// mostly be wrong since it depends on a lot of input args. Skip gas for now.
+			}
+			this.descended = false;
+		}
+		// If an existing call is returning, pop off the call stack
+		if (syscall && op == 'REVERT') {
+			this.callstack[this.callstack.length - 1].error = "execution reverted";
+			return;
+		}
+		if (log.getDepth() == this.callstack.length - 1) {
+			// Pop off the last call and get the execution results
+			var call = this.callstack.pop();
+
+			if (call.type == 'CREATE' || call.type == "CREATE2") {
+				// If the call was a CREATE, retrieve the contract address and output code
+				call.gasUsed = '0x' + bigInt(call.gasIn - call.gasCost - log.getGas()).toString(16);
+				delete call.gasIn; delete call.gasCost;
+
+				var ret = log.stack.peek(0);
+				if (!ret.equals(0)) {
+					call.to     = toHex(toAddress(ret.toString(16)));
+					call.output = toHex(db.getCode(toAddress(ret.toString(16))));
+				} else if (call.error === undefined) {
+					call.error = "internal failure"; // TODO(karalabe): surface these faults somehow
+				}
+			} else {
+				// If the call was a contract call, retrieve the gas usage and output
+				if (call.gas !== undefined) {
+					call.gasUsed = '0x' + bigInt(call.gasIn - call.gasCost + call.gas - log.getGas()).toString(16);
+				}
+				var ret = log.stack.peek(0);
+				if (!ret.equals(0)) {
+					call.output = toHex(log.memory.slice(call.outOff, call.outOff + call.outLen));
+				} else if (call.error === undefined) {
+					call.error = "internal failure"; // TODO(karalabe): surface these faults somehow
+				}
+				delete call.gasIn; delete call.gasCost;
+				delete call.outOff; delete call.outLen;
+			}
+			if (call.gas !== undefined) {
+				call.gas = '0x' + bigInt(call.gas).toString(16);
+			}
+			// Inject the call into the previous one
+			var left = this.callstack.length;
+			if (this.callstack[left-1].calls === undefined) {
+				this.callstack[left-1].calls = [];
+			}
+			this.callstack[left-1].calls.push(call);
+		}
+	},
+
+	// fault is invoked when the actual execution of an opcode fails.
+	fault: function(log, db) {
+		// If the topmost call already reverted, don't handle the additional fault again
+		if (this.callstack[this.callstack.length - 1].error !== undefined) {
+			return;
+		}
+		// Pop off the just failed call
+		var call = this.callstack.pop();
+		call.error = log.getError();
+
+		// Consume all available gas and clean any leftovers
+		if (call.gas !== undefined) {
+			call.gas = '0x' + bigInt(call.gas).toString(16);
+			call.gasUsed = call.gas
+		}
+		delete call.gasIn; delete call.gasCost;
+		delete call.outOff; delete call.outLen;
+
+		// Flatten the failed call into its parent
+		var left = this.callstack.length;
+		if (left > 0) {
+			if (this.callstack[left-1].calls === undefined) {
+				this.callstack[left-1].calls = [];
+			}
+			this.callstack[left-1].calls.push(call);
+			return;
+		}
+		// Last call failed too, leave it in the stack
+		this.callstack.push(call);
+	},
+
+	// result is invoked when all the opcodes have been iterated over and returns
+	// the final result of the tracing.
+	result: function(ctx, db) {
+		var result = {
+			type:    ctx.type,
+			from:    toHex(ctx.from),
+			to:      toHex(ctx.to),
+			value:   '0x' + ctx.value.toString(16),
+			gas:     '0x' + bigInt(ctx.gas).toString(16),
+			gasUsed: '0x' + bigInt(ctx.gasUsed).toString(16),
+			input:   toHex(ctx.input),
+			output:  toHex(ctx.output),
+			time:    ctx.time,
+		};
+		if (this.callstack[0].calls !== undefined) {
+			result.calls = this.callstack[0].calls;
+		}
+		if (this.callstack[0].error !== undefined) {
+			result.error = this.callstack[0].error;
+		} else if (ctx.error !== undefined) {
+			result.error = ctx.error;
+		}
+		if (result.error !== undefined && (result.error !== "execution reverted" || result.output ==="0x")) {
+			delete result.output;
+		}
+		return this.finalize(result);
+	},
+
+	// finalize recreates a call object using the final desired field oder for json
+	// serialization. This is a nicety feature to pass meaningfully ordered results
+	// to users who don't interpret it, just display it.
+	finalize: function(call) {
+		var sorted = {
+			type:    call.type,
+			from:    call.from,
+			to:      call.to,
+			value:   call.value,
+			gas:     call.gas,
+			gasUsed: call.gasUsed,
+			input:   call.input,
+			output:  call.output,
+			error:   call.error,
+			time:    call.time,
+			calls:   call.calls,
+		}
+		for (var key in sorted) {
+			if (sorted[key] === undefined) {
+				delete sorted[key];
+			}
+		}
+		if (sorted.calls !== undefined) {
+			for (var i=0; i<sorted.calls.length; i++) {
+				sorted.calls[i] = this.finalize(sorted.calls[i]);
+			}
+		}
+		return sorted;
+	}
+}
diff --git a/eth/tracers/js/internal/tracers/evmdis_tracer.js b/eth/tracers/js/internal/tracers/evmdis_tracer.js
new file mode 100644
index 00000000..db3422ed
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/evmdis_tracer.js
@@ -0,0 +1,103 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
```

```
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// evmdisTracer returns sufficient information from a trace to perform evmdis-style
+// disassembly.
+{
+       stack: [{ops: []}],
+
+       npushes: {0: 0, 1: 1, 2: 1, 3: 1, 4: 1, 5: 1, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1, 11: 1, 16: 1, 17: 1, 18: 1, 19: 1, 20: 1, 21: 1, 22: 1, 23: 1, 24: 1, 25: 1, 26: 1, 32: 1, 48: 1, 49: 1, 50: 1, 51: 1,
+
+       // result is invoked when all the opcodes have been iterated over and returns
+       // the final result of the tracing.
+       result: function() { return this.stack[0].ops; },
+
+       // fault is invoked when the actual execution of an opcode fails.
+       fault: function(log, db) { },
+
+       // step is invoked for every opcode that the VM executes.
+       step: function(log, db) {
+               var frame = this.stack[this.stack.length - 1];
+
+               var error = log.getError();
+               if (error) {
+                       frame["error"] = error;
+               } else if (log.getDepth() == this.stack.length) {
+                       opinfo = {
+                               op:     log.op.toNumber(),
+                               depth : log.getDepth(),
+                               result: [],
+                       };
+                       if (frame.ops.length > 0) {
+                               var prevop = frame.ops[frame.ops.length - 1];
+                               for(var i = 0; i < this.npushes[prevop.op]; i++)
+                                       prevop.result.push(log.stack.peek(i).toString(16));
+                       }
+                       switch(log.op.toString()) {
+                       case "CALL": case "CALLCODE":
+                               var instart = log.stack.peek(3).valueOf();
+                               var insize = log.stack.peek(4).valueOf();
+                               opinfo["gas"] = log.stack.peek(0).valueOf();
+                               opinfo["to"] = log.stack.peek(1).toString(16);
+                               opinfo["value"] = log.stack.peek(2).toString();
+                               opinfo["input"] = log.memory.slice(instart, instart + insize);
+                               opinfo["error"] = null;
+                               opinfo["return"] = null;
+                               opinfo["ops"] = [];
+                               this.stack.push(opinfo);
+                               break;
+                       case "DELEGATECALL": case "STATICCALL":
+                               var instart = log.stack.peek(2).valueOf();
+                               var insize = log.stack.peek(3).valueOf();
+                               opinfo["op"] =  log.op.toString();
+                               opinfo["gas"] =  log.stack.peek(0).valueOf();
+                               opinfo["to"] =  log.stack.peek(1).toString(16);
+                               opinfo["input"] =  log.memory.slice(instart, instart + insize);
+                               opinfo["error"] =  null;
+                               opinfo["return"] =  null;
+                               opinfo["ops"] = [];
+                               this.stack.push(opinfo);
+                               break;
+                       case "RETURN": case "REVERT":
+                               var out = log.stack.peek(0).valueOf();
+                               var outsize = log.stack.peek(1).valueOf();
+                               frame.return = log.memory.slice(out, out + outsize);
+                               break;
+                       case "STOP": case "SELFDESTRUCT":
+                               frame.return = log.memory.slice(0, 0);
+                               break;
+                       case "JUMPDEST":
+                               opinfo["pc"] = log.getPC();
+                       }
+                       if(log.op.isPush()) {
+                               opinfo["len"] = log.op.toNumber() - 0x5e;
+                       }
+                       frame.ops.push(opinfo);
+               } else {
+                       this.stack = this.stack.slice(0, log.getDepth());
+               }
+       }
+}
diff --git a/eth/tracers/js/internal/tracers/noop_tracer.js b/eth/tracers/js/internal/tracers/noop_tracer.js
new file mode 100644
index 00000000..c6881a43
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/noop_tracer.js
@@ -0,0 +1,39 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// noopTracer is just the barebone boilerplate code required from a JavaScript
+// object to be usable as a transaction tracer.
+{
+       // step is invoked for every opcode that the VM executes.
+       step: function(log, db) { },
+
+       // fault is invoked when the actual execution of an opcode fails.
+       fault: function(log, db) { },
+
+       // result is invoked when all the opcodes have been iterated over and returns
+       // the final result of the tracing.
+       result: function(ctx, db) { return {}; }
+}
diff --git a/eth/tracers/js/internal/tracers/opcount_tracer.js b/eth/tracers/js/internal/tracers/opcount_tracer.js
new file mode 100644
index 00000000..b0b307fa
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/opcount_tracer.js
@@ -0,0 +1,42 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// opcountTracer is a sample tracer that just counts the number of instructions
+// executed by the EVM before the transaction terminated.
+{
+       // count tracks the number of EVM instructions executed.
+       count: 0,
+
+       // step is invoked for every opcode that the VM executes.
+       step: function(log, db) { this.count++ },
+
+       // fault is invoked when the actual execution of an opcode fails.
+       fault: function(log, db) { },
+
+       // result is invoked when all the opcodes have been iterated over and returns
+       // the final result of the tracing.
+       result: function(ctx, db) { return this.count }
+}
diff --git a/eth/tracers/js/internal/tracers/prestate_tracer.js b/eth/tracers/js/internal/tracers/prestate_tracer.js
new file mode 100644
index 00000000..a264b7f1
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/prestate_tracer.js
@@ -0,0 +1,118 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// prestateTracer outputs sufficient information to create a local execution of
+// the transaction from a custom assembled genesis block.
+{
+       // prestate is the genesis that we're building.
+       prestate: null,
+
+       // lookupAccount injects the specified account into the prestate object.
+       lookupAccount: function(addr, db){
+               var acc = toHex(addr);
+               if (this.prestate[acc] === undefined) {
+                       this.prestate[acc] = {
+                               balance: '0x' + db.getBalance(addr).toString(16),
+                               nonce:   db.getNonce(addr),
+                               code:    toHex(db.getCode(addr)),
+                               storage: {}
+                       };
+               }
+       },
+
+       // lookupStorage injects the specified storage entry of the given account into
+       // the prestate object.
+       lookupStorage: function(addr, key, db){
+               var acc = toHex(addr);
+               var idx = toHex(key);
+
+               if (this.prestate[acc].storage[idx] === undefined) {
+                       this.prestate[acc].storage[idx] = toHex(db.getState(addr, key));
+               }
+       },
+
+       // result is invoked when all the opcodes have been iterated over and returns
+       // the final result of the tracing.
+       result: function(ctx, db) {
+               // At this point, we need to deduct the 'value' from the
+               // outer transaction, and move it back to the origin
+               this.lookupAccount(ctx.from, db);
+
+               var fromBal = bigInt(this.prestate[toHex(ctx.from)].balance.slice(2), 16);
+               var toBal   = bigInt(this.prestate[toHex(ctx.to)].balance.slice(2), 16);
+
+               this.prestate[toHex(ctx.to)].balance   = '0x'+toBal.subtract(ctx.value).toString(16);
+               this.prestate[toHex(ctx.from)].balance = '0x'+fromBal.add(ctx.value).add((ctx.gasUsed + ctx.intrinsicGas) * ctx.gasPrice).toString(16);
+
+               // Decrement the caller's nonce, and remove empty create targets
+               this.prestate[toHex(ctx.from)].nonce--;
```

```
+		if (ctx.type == 'CREATE') {
+			// We can blibdly delete the contract prestate, as any existing state would
+			// have caused the transaction to be rejected as invalid in the first place.
+			delete this.prestate[toHex(ctx.to)];
+		}
+		// Return the assembled allocations (prestate)
+		return this.prestate;
+	},
+
+	// step is invoked for every opcode that the VM executes.
+	step: function(log, db) {
+		// Add the current account if we just started tracing
+		if (this.prestate === null){
+			this.prestate = {};
+			// Balance will potentially be wrong here, since this will include the value
+			// sent along with the message. We fix that in 'result()'.
+			this.lookupAccount(log.contract.getAddress(), db);
+		}
+		// Whenever new state is accessed, add it to the prestate
+		switch (log.op.toString()) {
+			case "EXTCODECOPY": case "EXTCODESIZE": case "BALANCE":
+				this.lookupAccount(toAddress(log.stack.peek(0).toString(16)), db);
+				break;
+			case "CREATE":
+				var from = log.contract.getAddress();
+				this.lookupAccount(toContract(from, db.getNonce(from)), db);
+				break;
+			case "CREATE2":
+				var from = log.contract.getAddress();
+				// stack: salt, size, offset, endowment
+				var offset = log.stack.peek(1).valueOf()
+				var size = log.stack.peek(2).valueOf()
+				var end = offset + size
+				this.lookupAccount(toContract2(from, log.stack.peek(3).toString(16), log.memory.slice(offset, end)), db);
+				break;
+			case "CALL": case "CALLCODE": case "DELEGATECALL": case "STATICCALL":
+				this.lookupAccount(toAddress(log.stack.peek(1).toString(16)), db);
+				break;
+			case 'SSTORE':case 'SLOAD':
+				this.lookupStorage(log.contract.getAddress(), toWord(log.stack.peek(0).toString(16)), db);
+				break;
+		}
+	},
+
+	// fault is invoked when the actual execution of an opcode fails.
+	fault: function(log, db) {}
+}
diff --git a/eth/tracers/js/internal/tracers/tracers.go b/eth/tracers/js/internal/tracers/tracers.go
new file mode 100644
index 00000000..c921db72
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/tracers.go
@@ -0,0 +1,31 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+//go:generate go-bindata -nometadata -o assets.go -pkg tracers -ignore tracers.go -ignore assets.go ./...
+//go:generate gofmt -s -w assets.go
+
+// Package tracers contains the actual JavaScript tracer assets.
+package tracers
diff --git a/eth/tracers/js/internal/tracers/trigram_tracer.js b/eth/tracers/js/internal/tracers/trigram_tracer.js
new file mode 100644
index 00000000..f43c690b
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/trigram_tracer.js
@@ -0,0 +1,59 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2018 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+{
+    // hist is the map of trigram counters
+    hist: {},
+    // lastOp is last operation
+    lastOps: ['',''],
+    lastDepth: 0,
+        // step is invoked for every opcode that the VM executes.
+    step: function(log, db) {
+        var depth = log.getDepth();
+        if (depth != this.lastDepth){
+            this.lastOps = ['',''];
+            this.lastDepth = depth;
+            return;
+        }
+        var op = log.op.toString();
+        var key = this.lastOps[0]+'-'+this.lastOps[1]+'-'+op;
+        if (this.hist[key]){
+            this.hist[key]++;
+        }
+        else {
+            this.hist[key] = 1;
+        }
+        this.lastOps[0] = this.lastOps[1];
+        this.lastOps[1] = op;
+    },
+    // fault is invoked when the actual execution of an opcode fails.
+    fault: function(log, db) {},
+    // result is invoked when all the opcodes have been iterated over and returns
+    // the final result of the tracing.
+    result: function(ctx) {
+        return this.hist;
+    },
+}
diff --git a/eth/tracers/js/internal/tracers/unigram_tracer.js b/eth/tracers/js/internal/tracers/unigram_tracer.js
new file mode 100644
index 00000000..8ca82381
--- /dev/null
+++ b/eth/tracers/js/internal/tracers/unigram_tracer.js
@@ -0,0 +1,51 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2018 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+{
+    // hist is the map of opcodes to counters
+    hist: {},
+    // nops counts number of ops
+    nops: 0,
+    // step is invoked for every opcode that the VM executes.
+    step: function(log, db) {
+        var op = log.op.toString();
+        if (this.hist[op]){
+            this.hist[op]++;
+        }
+        else {
+            this.hist[op] = 1;
+        }
+        this.nops++;
+    },
+    // fault is invoked when the actual execution of an opcode fails.
+    fault: function(log, db) {},
+
+    // result is invoked when all the opcodes have been iterated over and returns
+    // the final result of the tracing.
+    result: function(ctx) {
+        return this.hist;
+    },
+}
diff --git a/eth/tracers/js/tracer.go b/eth/tracers/js/tracer.go
new file mode 100644
index 00000000..4fdf7904
--- /dev/null
+++ b/eth/tracers/js/tracer.go
@@ -0,0 +1,890 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+// package js is a collection of tracers written in javascript.
+package js
+
+import (
+       "encoding/json"
+       "errors"
+       "fmt"
+       "math/big"
+       "strings"
+       "sync/atomic"
+       "time"
+       "unicode"
+       "unsafe"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/ethereum/go-ethereum/crypto"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/vm"
+       tracers2 "github.com/flare-foundation/coreth/eth/tracers"
+       "github.com/flare-foundation/coreth/eth/tracers/js/internal/tracers"
+       "gopkg.in/olebedev/go-duktape.v3"
+)
+
+// camel converts a snake cased input string into a camel cased output.
+func camel(str string) string {
+       pieces := strings.Split(str, "_")
+       for i := 1; i < len(pieces); i++ {
+               pieces[i] = string(unicode.ToUpper(rune(pieces[i][0]))) + pieces[i][1:]
```

```
+	}
+	return strings.Join(pieces, "")
+}
+
+var assetTracers = make(map[string]string)
+
+// init retrieves the JavaScript transaction tracers included in go-ethereum.
+func init() {
+	for _, file := range tracers.AssetNames() {
+		name := camel(strings.TrimSuffix(file, ".js"))
+		assetTracers[name] = string(tracers.MustAsset(file))
+	}
+	tracers2.RegisterLookup(true, newJsTracer)
+}
+
+// makeSlice convert an unsafe memory pointer with the given type into a Go byte
+// slice.
+//
+// Note, the returned slice uses the same memory area as the input arguments.
+// If those are duktape stack items, popping them off **will** make the slice
+// contents change.
+func makeSlice(ptr unsafe.Pointer, size uint) []byte {
+	var sl = struct {
+		addr uintptr
+		len  int
+		cap  int
+	}{uintptr(ptr), int(size), int(size)}
+
+	return *(*[]byte)(unsafe.Pointer(&sl))
+}
+
+// popSlice pops a buffer off the JavaScript stack and returns it as a slice.
+func popSlice(ctx *duktape.Context) []byte {
+	blob := common.CopyBytes(makeSlice(ctx.GetBuffer(-1)))
+	ctx.Pop()
+	return blob
+}
+
+// pushBigInt create a JavaScript BigInteger in the VM.
+func pushBigInt(n *big.Int, ctx *duktape.Context) {
+	ctx.GetGlobalString("bigInt")
+	ctx.PushString(n.String())
+	ctx.Call(1)
+}
+
+// opWrapper provides a JavaScript wrapper around OpCode.
+type opWrapper struct {
+	op vm.OpCode
+}
+
+// pushObject assembles a JSVM object wrapping a swappable opcode and pushes it
+// onto the VM stack.
+func (ow *opWrapper) pushObject(vm *duktape.Context) {
+	obj := vm.PushObject()
+
+	vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushInt(int(ow.op)); return 1 })
+	vm.PutPropString(obj, "toNumber")
+
+	vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushString(ow.op.String()); return 1 })
+	vm.PutPropString(obj, "toString")
+
+	vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushBoolean(ow.op.IsPush()); return 1 })
+	vm.PutPropString(obj, "isPush")
+}
+
+// memoryWrapper provides a JavaScript wrapper around vm.Memory.
+type memoryWrapper struct {
+	memory *vm.Memory
+}
+
+// slice returns the requested range of memory as a byte slice.
+func (mw *memoryWrapper) slice(begin, end int64) []byte {
+	if end == begin {
+		return []byte{}
+	}
+	if end < begin || begin < 0 {
+		// TODO(karalabe): We can't js-throw from Go inside duktape inside Go. The Go
+		// runtime goes belly up https://github.com/golang/go/issues/15639.
+		log.Warn("Tracer accessed out of bound memory", "offset", begin, "end", end)
+		return nil
+	}
+	if mw.memory.Len() < int(end) {
+		// TODO(karalabe): We can't js-throw from Go inside duktape inside Go. The Go
+		// runtime goes belly up https://github.com/golang/go/issues/15639.
+		log.Warn("Tracer accessed out of bound memory", "available", mw.memory.Len(), "offset", begin, "size", end-begin)
+		return nil
+	}
+	return mw.memory.GetCopy(begin, end-begin)
+}
+
+// getUint returns the 32 bytes at the specified address interpreted as a uint.
+func (mw *memoryWrapper) getUint(addr int64) *big.Int {
+	if mw.memory.Len() < int(addr)+32 || addr < 0 {
+		// TODO(karalabe): We can't js-throw from Go inside duktape inside Go. The Go
+		// runtime goes belly up https://github.com/golang/go/issues/15639.
+		log.Warn("Tracer accessed out of bound memory", "available", mw.memory.Len(), "offset", addr, "size", 32)
+		return new(big.Int)
+	}
+	return new(big.Int).SetBytes(mw.memory.GetPtr(addr, 32))
+}
+
+// pushObject assembles a JSVM object wrapping a swappable memory and pushes it
+// onto the VM stack.
+func (mw *memoryWrapper) pushObject(vm *duktape.Context) {
+	obj := vm.PushObject()
+
+	// Generate the `slice` method which takes two ints and returns a buffer
+	vm.PushGoFunction(func(ctx *duktape.Context) int {
+		blob := mw.slice(int64(ctx.GetInt(-2)), int64(ctx.GetInt(-1)))
+		ctx.Pop2()
+
+		ptr := ctx.PushFixedBuffer(len(blob))
+		copy(makeSlice(ptr, uint(len(blob))), blob)
+		return 1
+	})
+	vm.PutPropString(obj, "slice")
+
+	// Generate the `getUint` method which takes an int and returns a bigint
+	vm.PushGoFunction(func(ctx *duktape.Context) int {
+		offset := int64(ctx.GetInt(-1))
+		ctx.Pop()
+
+		pushBigInt(mw.getUint(offset), ctx)
+		return 1
+	})
+	vm.PutPropString(obj, "getUint")
+}
+
+// stackWrapper provides a JavaScript wrapper around vm.Stack.
+type stackWrapper struct {
+	stack *vm.Stack
+}
+
+// peek returns the nth-from-the-top element of the stack.
```

```
+func (sw *stackWrapper) peek(idx int) *big.Int {
+       if len(sw.stack.Data()) <= idx || idx < 0 {
+               // TODO(karalabe): We can't js-throw from Go inside duktape inside Go. The Go
+               // runtime goes belly up https://github.com/golang/go/issues/15639.
+               log.Warn("Tracer accessed out of bound stack", "size", len(sw.stack.Data()), "index", idx)
+               return new(big.Int)
+       }
+       return sw.stack.Back(idx).ToBig()
+}
+
+// pushObject assembles a JSVM object wrapping a swappable stack and pushes it
+// onto the VM stack.
+func (sw *stackWrapper) pushObject(vm *duktape.Context) {
+       obj := vm.PushObject()
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushInt(len(sw.stack.Data())); return 1 })
+       vm.PutPropString(obj, "length")
+
+       // Generate the `peek` method which takes an int and returns a bigint
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               offset := ctx.GetInt(-1)
+               ctx.Pop()
+
+               pushBigInt(sw.peek(offset), ctx)
+               return 1
+       })
+       vm.PutPropString(obj, "peek")
+}
+
+// dbWrapper provides a JavaScript wrapper around vm.Database.
+type dbWrapper struct {
+       db vm.StateDB
+}
+
+// pushObject assembles a JSVM object wrapping a swappable database and pushes it
+// onto the VM stack.
+func (dw *dbWrapper) pushObject(vm *duktape.Context) {
+       obj := vm.PushObject()
+
+       // Push the wrapper for statedb.GetBalance
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               pushBigInt(dw.db.GetBalance(common.BytesToAddress(popSlice(ctx))), ctx)
+               return 1
+       })
+       vm.PutPropString(obj, "getBalance")
+
+       // Push the wrapper for statedb.GetNonce
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               ctx.PushInt(int(dw.db.GetNonce(common.BytesToAddress(popSlice(ctx)))))
+               return 1
+       })
+       vm.PutPropString(obj, "getNonce")
+
+       // Push the wrapper for statedb.GetCode
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               code := dw.db.GetCode(common.BytesToAddress(popSlice(ctx)))
+
+               ptr := ctx.PushFixedBuffer(len(code))
+               copy(makeSlice(ptr, uint(len(code))), code)
+               return 1
+       })
+       vm.PutPropString(obj, "getCode")
+
+       // Push the wrapper for statedb.GetState
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               hash := popSlice(ctx)
+               addr := popSlice(ctx)
+
+               state := dw.db.GetState(common.BytesToAddress(addr), common.BytesToHash(hash))
+
+               ptr := ctx.PushFixedBuffer(len(state))
+               copy(makeSlice(ptr, uint(len(state))), state[:])
+               return 1
+       })
+       vm.PutPropString(obj, "getState")
+
+       // Push the wrapper for statedb.Exists
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               ctx.PushBoolean(dw.db.Exist(common.BytesToAddress(popSlice(ctx))))
+               return 1
+       })
+       vm.PutPropString(obj, "exists")
+}
+
+// contractWrapper provides a JavaScript wrapper around vm.Contract
+type contractWrapper struct {
+       contract *vm.Contract
+}
+
+// pushObject assembles a JSVM object wrapping a swappable contract and pushes it
+// onto the VM stack.
+func (cw *contractWrapper) pushObject(vm *duktape.Context) {
+       obj := vm.PushObject()
+
+       // Push the wrapper for contract.Caller
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               ptr := ctx.PushFixedBuffer(20)
+               copy(makeSlice(ptr, 20), cw.contract.Caller().Bytes())
+               return 1
+       })
+       vm.PutPropString(obj, "getCaller")
+
+       // Push the wrapper for contract.Address
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               ptr := ctx.PushFixedBuffer(20)
+               copy(makeSlice(ptr, 20), cw.contract.Address().Bytes())
+               return 1
+       })
+       vm.PutPropString(obj, "getAddress")
+
+       // Push the wrapper for contract.Value
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               pushBigInt(cw.contract.Value(), ctx)
+               return 1
+       })
+       vm.PutPropString(obj, "getValue")
+
+       // Push the wrapper for contract.Input
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               blob := cw.contract.Input
+
+               ptr := ctx.PushFixedBuffer(len(blob))
+               copy(makeSlice(ptr, uint(len(blob))), blob)
+               return 1
+       })
+       vm.PutPropString(obj, "getInput")
+}
+
+type frame struct {
+       typ   *string
+       from  *common.Address
+       to    *common.Address
+       input []byte
+       gas   *uint
+       value *big.Int
+}
+
+func newFrame() *frame {
+       return &frame{
+               typ:  new(string),
+               from: new(common.Address),
+               to:   new(common.Address),
+               gas:  new(uint),
+       }
+}
+
+func (f *frame) pushObject(vm *duktape.Context) {
+       obj := vm.PushObject()
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, *f.typ); return 1 })
+       vm.PutPropString(obj, "getType")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, *f.from); return 1 })
+       vm.PutPropString(obj, "getFrom")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, *f.to); return 1 })
+       vm.PutPropString(obj, "getTo")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, f.input); return 1 })
+       vm.PutPropString(obj, "getInput")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, *f.gas); return 1 })
+       vm.PutPropString(obj, "getGas")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               if f.value != nil {
+                       pushValue(ctx, f.value)
+               } else {
+                       ctx.PushUndefined()
+               }
+               return 1
+       })
+       vm.PutPropString(obj, "getValue")
+}
+
+type frameResult struct {
+       gasUsed    *uint
+       output     []byte
+       errorValue *string
+}
+
+func newFrameResult() *frameResult {
+       return &frameResult{
+               gasUsed: new(uint),
+       }
+}
+
+func (r *frameResult) pushObject(vm *duktape.Context) {
+       obj := vm.PushObject()
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, *r.gasUsed); return 1 })
+       vm.PutPropString(obj, "getGasUsed")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int { pushValue(ctx, r.output); return 1 })
+       vm.PutPropString(obj, "getOutput")
+
+       vm.PushGoFunction(func(ctx *duktape.Context) int {
+               if r.errorValue != nil {
+                       pushValue(ctx, *r.errorValue)
+               } else {
+                       ctx.PushUndefined()
+               }
+               return 1
+       })
+       vm.PutPropString(obj, "getError")
+}
+
+// jsTracer provides an implementation of Tracer that evaluates a Javascript
+// function for each VM execution step.
+type jsTracer struct {
+       vm  *duktape.Context // Javascript VM instance
+       env *vm.EVM          // EVM instance executing the code being traced
+
+       tracerObject int // Stack index of the tracer JavaScript object
+       stateObject  int // Stack index of the global state to pull arguments from
+
+       opWrapper       *opWrapper       // Wrapper around the VM opcode
+       stackWrapper    *stackWrapper    // Wrapper around the VM stack
+       memoryWrapper   *memoryWrapper   // Wrapper around the VM memory
+       contractWrapper *contractWrapper // Wrapper around the contract object
+       dbWrapper       *dbWrapper       // Wrapper around the VM environment
+
+       pcValue     *uint   // Swappable pc value wrapped by a log accessor
+       gasValue    *uint   // Swappable gas value wrapped by a log accessor
+       costValue   *uint   // Swappable cost value wrapped by a log accessor
+       depthValue  *uint   // Swappable depth value wrapped by a log accessor
+       errorValue  *string // Swappable error value wrapped by a log accessor
+       refundValue *uint   // Swappable refund value wrapped by a log accessor
+
+       frame       *frame       // Represents entry into call frame. Fields are swappable
+       frameResult *frameResult // Represents exit from a call frame. Fields are swappable
+
+       ctx map[string]interface{} // Transaction context gathered throughout execution
+       err error                  // Error, if one has occurred
+
+       interrupt uint32 // Atomic flag to signal execution interruption
+       reason    error  // Textual reason for the interruption
+
+       activePrecompiles []common.Address // Updated on CaptureStart based on given rules
+       traceSteps        bool             // When true, will invoke step() on each opcode
+       traceCallFrames   bool             // When true, will invoke enter() and exit() js funcs
+}
+
+// New instantiates a new tracer instance. code specifies a Javascript snippet,
+// which must evaluate to an expression returning an object with 'step', 'fault'
+// and 'result' functions.
+func newJsTracer(code string, ctx *tracers2.Context) (tracers2.Tracer, error) {
+       if c, ok := assetTracers[code]; ok {
+               code = c
+       }
+       if ctx == nil {
+               ctx = new(tracers2.Context)
+       }
+       tracer := &jsTracer{
+               vm:              duktape.New(),
+               ctx:             make(map[string]interface{}),
+               opWrapper:       new(opWrapper),
+               stackWrapper:    new(stackWrapper),
+               memoryWrapper:   new(memoryWrapper),
+               contractWrapper: new(contractWrapper),
+               dbWrapper:       new(dbWrapper),
+               pcValue:         new(uint),
+               gasValue:        new(uint),
+               costValue:       new(uint),
+               depthValue:      new(uint),
+               refundValue:     new(uint),
+               frame:           newFrame(),
+               frameResult:     newFrameResult(),
+       }
+       if ctx.BlockHash != (common.Hash{}) {
+               tracer.ctx["blockHash"] = ctx.BlockHash
+
+               if ctx.TxHash != (common.Hash{}) {
+                       tracer.ctx["txIndex"] = ctx.TxIndex
+                       tracer.ctx["txHash"] = ctx.TxHash
+               }
+       }
+       // Set up builtins for this environment
+       tracer.vm.PushGlobalGoFunction("toHex", func(ctx *duktape.Context) int {
+               ctx.PushString(hexutil.Encode(popSlice(ctx)))
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("toWord", func(ctx *duktape.Context) int {
+               var word common.Hash
+               if ptr, size := ctx.GetBuffer(-1); ptr != nil {
+                       word = common.BytesToHash(makeSlice(ptr, size))
+               } else {
+                       word = common.HexToHash(ctx.GetString(-1))
+               }
+               ctx.Pop()
+               copy(makeSlice(ctx.PushFixedBuffer(32), 32), word[:])
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("toAddress", func(ctx *duktape.Context) int {
+               var addr common.Address
+               if ptr, size := ctx.GetBuffer(-1); ptr != nil {
+                       addr = common.BytesToAddress(makeSlice(ptr, size))
+               } else {
+                       addr = common.HexToAddress(ctx.GetString(-1))
+               }
+               ctx.Pop()
+               copy(makeSlice(ctx.PushFixedBuffer(20), 20), addr[:])
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("toContract", func(ctx *duktape.Context) int {
+               var from common.Address
+               if ptr, size := ctx.GetBuffer(-2); ptr != nil {
+                       from = common.BytesToAddress(makeSlice(ptr, size))
+               } else {
+                       from = common.HexToAddress(ctx.GetString(-2))
+               }
+               nonce := uint64(ctx.GetInt(-1))
+               ctx.Pop2()
+
+               contract := crypto.CreateAddress(from, nonce)
+               copy(makeSlice(ctx.PushFixedBuffer(20), 20), contract[:])
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("toContract2", func(ctx *duktape.Context) int {
+               var from common.Address
+               if ptr, size := ctx.GetBuffer(-3); ptr != nil {
+                       from = common.BytesToAddress(makeSlice(ptr, size))
+               } else {
+                       from = common.HexToAddress(ctx.GetString(-3))
+               }
+               // Retrieve salt hex string from js stack
+               salt := common.HexToHash(ctx.GetString(-2))
+               // Retrieve code slice from js stack
+               var code []byte
+               if ptr, size := ctx.GetBuffer(-1); ptr != nil {
+                       code = common.CopyBytes(makeSlice(ptr, size))
+               } else {
+                       code = common.FromHex(ctx.GetString(-1))
+               }
+               codeHash := crypto.Keccak256(code)
+               ctx.Pop3()
+               contract := crypto.CreateAddress2(from, salt, codeHash)
+               copy(makeSlice(ctx.PushFixedBuffer(20), 20), contract[:])
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("isPrecompiled", func(ctx *duktape.Context) int {
+               addr := common.BytesToAddress(popSlice(ctx))
+               for _, p := range tracer.activePrecompiles {
+                       if p == addr {
+                               ctx.PushBoolean(true)
+                               return 1
+                       }
+               }
+               ctx.PushBoolean(false)
+               return 1
+       })
+       tracer.vm.PushGlobalGoFunction("slice", func(ctx *duktape.Context) int {
+               start, end := ctx.GetInt(-2), ctx.GetInt(-1)
+               ctx.Pop2()
+
+               blob := popSlice(ctx)
+               size := end - start
+
+               if start < 0 || start > end || end > len(blob) {
+                       // TODO(karalabe): We can't js-throw from Go inside duktape inside Go. The Go
+                       // runtime goes belly up https://github.com/golang/go/issues/15639.
+                       log.Warn("Tracer accessed out of bound memory", "available", len(blob), "offset", start, "size", size)
+                       ctx.PushFixedBuffer(0)
+                       return 1
+               }
+               copy(makeSlice(ctx.PushFixedBuffer(size), uint(size)), blob[start:end])
+               return 1
+       })
+       // Push the JavaScript tracer as object #0 onto the JSVM stack and validate it
+       if err := tracer.vm.PevalString("(" + code + ")"); err != nil {
+               log.Warn("Failed to compile tracer", "err", err)
+               return nil, err
+       }
+       tracer.tracerObject = 0 // yeah, nice, eval can't return the index itself
+
+       hasStep := tracer.vm.GetPropString(tracer.tracerObject, "step")
+       tracer.vm.Pop()
+
+       if !tracer.vm.GetPropString(tracer.tracerObject, "fault") {
+               return nil, fmt.Errorf("trace object must expose a function fault()")
+       }
+       tracer.vm.Pop()
+
+       if !tracer.vm.GetPropString(tracer.tracerObject, "result") {
+               return nil, fmt.Errorf("trace object must expose a function result()")
+       }
+       tracer.vm.Pop()
+
+       hasEnter := tracer.vm.GetPropString(tracer.tracerObject, "enter")
+       tracer.vm.Pop()
+       hasExit := tracer.vm.GetPropString(tracer.tracerObject, "exit")
+       tracer.vm.Pop()
+       if hasEnter != hasExit {
+               return nil, fmt.Errorf("trace object must expose either both or none of enter() and exit()")
+       }
+       tracer.traceCallFrames = hasEnter && hasExit
+       tracer.traceSteps = hasStep
+
+       // Tracer is valid, inject the big int library to access large numbers
+       tracer.vm.EvalString(bigIntegerJS)
+       tracer.vm.PutGlobalString("bigInt")
+
+       // Push the global environment state as object #1 into the JSVM stack
+       tracer.stateObject = tracer.vm.PushObject()
+
+       logObject := tracer.vm.PushObject()
+
+       tracer.opWrapper.pushObject(tracer.vm)
+       tracer.vm.PutPropString(logObject, "op")
+
+       tracer.stackWrapper.pushObject(tracer.vm)
+       tracer.vm.PutPropString(logObject, "stack")
+
+       tracer.memoryWrapper.pushObject(tracer.vm)
+       tracer.vm.PutPropString(logObject, "memory")
+
+       tracer.contractWrapper.pushObject(tracer.vm)
+       tracer.vm.PutPropString(logObject, "contract")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushUint(*tracer.pcValue); return 1 })
+       tracer.vm.PutPropString(logObject, "getPC")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushUint(*tracer.gasValue); return 1 })
+       tracer.vm.PutPropString(logObject, "getGas")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushUint(*tracer.costValue); return 1 })
+       tracer.vm.PutPropString(logObject, "getCost")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushUint(*tracer.depthValue); return 1 })
+       tracer.vm.PutPropString(logObject, "getDepth")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int { ctx.PushUint(*tracer.refundValue); return 1 })
+       tracer.vm.PutPropString(logObject, "getRefund")
+
+       tracer.vm.PushGoFunction(func(ctx *duktape.Context) int {
+               if tracer.errorValue != nil {
+                       ctx.PushString(*tracer.errorValue)
+               } else {
+                       ctx.PushUndefined()
+               }
+               return 1
+       })
+       tracer.vm.PutPropString(logObject, "getError")
+
+       tracer.vm.PutPropString(tracer.stateObject, "log")
+
+       tracer.frame.pushObject(tracer.vm)
+       tracer.vm.PutPropString(tracer.stateObject, "frame")
+
+       tracer.frameResult.pushObject(tracer.vm)
+       tracer.vm.PutPropString(tracer.stateObject, "frameResult")
+
+       tracer.dbWrapper.pushObject(tracer.vm)
+       tracer.vm.PutPropString(tracer.stateObject, "db")
+
+       return tracer, nil
+}
+
+// Stop terminates execution of the tracer at the first opportune moment.
+func (jst *jsTracer) Stop(err error) {
+       jst.reason = err
+       atomic.StoreUint32(&jst.interrupt, 1)
+}
+
+// call executes a method on a JS object, catching any errors, formatting and
+// returning them as error objects.
+func (jst *jsTracer) call(noret bool, method string, args ...string) (json.RawMessage, error) {
+       // Execute the JavaScript call and return any error
+       jst.vm.PushString(method)
+       for _, arg := range args {
+               jst.vm.GetPropString(jst.stateObject, arg)
+       }
+       code := jst.vm.PcallProp(jst.tracerObject, len(args))
+       defer jst.vm.Pop()
+
+       if code != 0 {
+               err := jst.vm.SafeToString(-1)
+               return nil, errors.New(err)
+       }
+       // No error occurred, extract return value and return
+       if noret {
+               return nil, nil
+       }
+       // Push a JSON marshaller onto the stack. We can't marshal from the out-
+       // side because duktape can crash on large nestings and we can't catch
+       // C++ exceptions ourselves from Go. TODO(karalabe): Yuck, why wrap?!
+       jst.vm.PushString("(JSON.stringify)")
+       jst.vm.Eval()
+
+       jst.vm.Swap(-1, -2)
+       if code = jst.vm.Pcall(1); code != 0 {
+               err := jst.vm.SafeToString(-1)
+               return nil, errors.New(err)
+       }
+       return json.RawMessage(jst.vm.SafeToString(-1)), nil
+}
+
+func wrapError(context string, err error) error {
+       return fmt.Errorf("%v    in server-side tracer function '%v'", err, context)
+}
+
+// CaptureStart implements the Tracer interface to initialize the tracing operation.
+func (jst *jsTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+       jst.env = env
+       jst.ctx["type"] = "CALL"
+       if create {
+               jst.ctx["type"] = "CREATE"
+       }
+       jst.ctx["from"] = from
+       jst.ctx["to"] = to
+       jst.ctx["input"] = input
+       jst.ctx["gas"] = gas
+       jst.ctx["gasPrice"] = env.TxContext.GasPrice
+       jst.ctx["value"] = value
+
+       // Initialize the context
+       jst.ctx["block"] = env.Context.BlockNumber.Uint64()
+       jst.dbWrapper.db = env.StateDB
+       // Update list of precompiles based on current block
+       rules := env.ChainConfig().AvalancheRules(env.Context.BlockNumber, env.Context.Time)
+       jst.activePrecompiles = vm.ActivePrecompiles(rules)
+
+       // Compute intrinsic gas
+       isHomestead := env.ChainConfig().IsHomestead(env.Context.BlockNumber)
+       isIstanbul := env.ChainConfig().IsIstanbul(env.Context.BlockNumber)
+       intrinsicGas, err := core.IntrinsicGas(input, nil, jst.ctx["type"] == "CREATE", isHomestead, isIstanbul)
+       if err != nil {
+               return
+       }
+       jst.ctx["intrinsicGas"] = intrinsicGas
+}
+
+// CaptureState implements the Tracer interface to trace a single step of VM execution.
+func (jst *jsTracer) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+       if !jst.traceSteps {
+               return
+       }
+       if jst.err != nil {
+               return
+       }
+       // If tracing was interrupted, set the error and stop
+       if atomic.LoadUint32(&jst.interrupt) > 0 {
+               jst.err = jst.reason
+               jst.env.Cancel()
+               return
+       }
+       jst.opWrapper.op = op
+       jst.stackWrapper.stack = scope.Stack
+       jst.memoryWrapper.memory = scope.Memory
+       jst.contractWrapper.contract = scope.Contract
+
+       *jst.pcValue = uint(pc)
+       *jst.gasValue = uint(gas)
+       *jst.costValue = uint(cost)
+       *jst.depthValue = uint(depth)
+       *jst.refundValue = uint(jst.env.StateDB.GetRefund())
+
+       jst.errorValue = nil
+       if err != nil {
+               jst.errorValue = new(string)
+               *jst.errorValue = err.Error()
+       }
+
+       if _, err := jst.call(true, "step", "log", "db"); err != nil {
+               jst.err = wrapError("step", err)
+       }
+}
+
+// CaptureFault implements the Tracer interface to trace an execution fault
+func (jst *jsTracer) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+       if jst.err != nil {
+               return
+       }
+       // Apart from the error, everything matches the previous invocation
+       jst.errorValue = new(string)
+       *jst.errorValue = err.Error()
+
+       if _, err := jst.call(true, "fault", "log", "db"); err != nil {
+               jst.err = wrapError("fault", err)
+       }
+}
+
+// CaptureEnd is called after the call finishes to finalize the tracing.
+func (jst *jsTracer) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {
+       jst.ctx["output"] = output
+       jst.ctx["time"] = t.String()
+       jst.ctx["gasUsed"] = gasUsed
+
+       if err != nil {
+               jst.ctx["error"] = err.Error()
+       }
+}
+
+// CaptureEnter is called when EVM enters a new scope (via call, create or selfdestruct).
+func (jst *jsTracer) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+       if !jst.traceCallFrames {
+               return
+       }
+       if jst.err != nil {
+               return
+       }
+       // If tracing was interrupted, set the error and stop
+       if atomic.LoadUint32(&jst.interrupt) > 0 {
+               jst.err = jst.reason
+               return
+       }
+
+       *jst.frame.typ = typ.String()
+       *jst.frame.from = from
+       *jst.frame.to = to
+       jst.frame.input = common.CopyBytes(input)
+       *jst.frame.gas = uint(gas)
+       jst.frame.value = nil
+       if value != nil {
+               jst.frame.value = new(big.Int).SetBytes(value.Bytes())
+       }
+
+       if _, err := jst.call(true, "enter", "frame"); err != nil {
+               jst.err = wrapError("enter", err)
+       }
+}
+
+// CaptureExit is called when EVM exits a scope, even if the scope didn't
+// execute any code.
+func (jst *jsTracer) CaptureExit(output []byte, gasUsed uint64, err error) {
+       if !jst.traceCallFrames {
+               return
+       }
+       // If tracing was interrupted, set the error and stop
+       if atomic.LoadUint32(&jst.interrupt) > 0 {
+               jst.err = jst.reason
+               return
+       }
+
+       jst.frameResult.output = common.CopyBytes(output)
+       *jst.frameResult.gasUsed = uint(gasUsed)
+       jst.frameResult.errorValue = nil
+       if err != nil {
+               jst.frameResult.errorValue = new(string)
+               *jst.frameResult.errorValue = err.Error()
+       }
+
+       if _, err := jst.call(true, "exit", "frameResult"); err != nil {
+               jst.err = wrapError("exit", err)
+       }
+}
+
+// GetResult calls the Javascript 'result' function and returns its value, or any accumulated error
+func (jst *jsTracer) GetResult() (json.RawMessage, error) {
+       // Transform the context into a JavaScript object and inject into the state
+       obj := jst.vm.PushObject()
+
+       for key, val := range jst.ctx {
+               jst.addToObj(obj, key, val)
+       }
+       jst.vm.PutPropString(jst.stateObject, "ctx")
+
+       // Finalize the trace and return the results
+       result, err := jst.call(false, "result", "ctx", "db")
+       if err != nil {
+               jst.err = wrapError("result", err)
+       }
+       // Clean up the JavaScript environment
+       jst.vm.DestroyHeap()
+       jst.vm.Destroy()
+
+       return result, jst.err
+}
+
+// addToObj pushes a field to a JS object.
```

```
+func (jst *jsTracer) addToObj(obj int, key string, val interface{}) {
+       pushValue(jst.vm, val)
+       jst.vm.PutPropString(obj, key)
+}
+
+func pushValue(ctx *duktape.Context, val interface{}) {
+       switch val := val.(type) {
+       case uint64:
+               ctx.PushUint(uint(val))
+       case string:
+               ctx.PushString(val)
+       case []byte:
+               ptr := ctx.PushFixedBuffer(len(val))
+               copy(makeSlice(ptr, uint(len(val))), val)
+       case common.Address:
+               ptr := ctx.PushFixedBuffer(20)
+               copy(makeSlice(ptr, 20), val[:])
+       case *big.Int:
+               pushBigInt(val, ctx)
+       case int:
+               ctx.PushInt(val)
+       case uint:
+               ctx.PushUint(val)
+       case common.Hash:
+               ptr := ctx.PushFixedBuffer(32)
+               copy(makeSlice(ptr, 32), val[:])
+       default:
+               panic(fmt.Sprintf("unsupported type: %T", val))
+       }
+}
diff --git a/eth/tracers/js/tracer_test.go b/eth/tracers/js/tracer_test.go
new file mode 100644
index 00000000..a43eb9b3
--- /dev/null
+++ b/eth/tracers/js/tracer_test.go
@@ -0,0 +1,267 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package js
+
+import (
+       "encoding/json"
+       "errors"
+       "math/big"
+       "testing"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/eth/tracers"
+       "github.com/flare-foundation/coreth/params"
+)
+
+type account struct{}
+
+func (account) SubBalance(amount *big.Int)                          {}
+func (account) AddBalance(amount *big.Int)                          {}
+func (account) SetAddress(common.Address)                           {}
+func (account) Value() *big.Int                                     { return nil }
+func (account) SetBalance(*big.Int)                                 {}
+func (account) SetNonce(uint64)                                     {}
+func (account) Balance() *big.Int                                   { return nil }
+func (account) Address() common.Address                             { return common.Address{} }
+func (account) SetCode(common.Hash, []byte)                         {}
+func (account) ForEachStorage(cb func(key, value common.Hash) bool) {}
+
+type dummyStatedb struct {
+       state.StateDB
+}
+
+func (*dummyStatedb) GetRefund() uint64                       { return 1337 }
+func (*dummyStatedb) GetBalance(addr common.Address) *big.Int { return new(big.Int) }
+
+type vmContext struct {
+       blockCtx vm.BlockContext
+       txCtx    vm.TxContext
+}
+
+func testCtx() *vmContext {
+       return &vmContext{blockCtx: vm.BlockContext{BlockNumber: big.NewInt(1)}, txCtx: vm.TxContext{GasPrice: big.NewInt(100000)}}
+}
+
+func runTrace(tracer tracers.Tracer, vmctx *vmContext, chaincfg *params.ChainConfig) (json.RawMessage, error) {
+       var (
+               env             = vm.NewEVM(vmctx.blockCtx, vmctx.txCtx, &dummyStatedb{}, chaincfg, vm.Config{Debug: true, Tracer: tracer})
+               startGas uint64 = 10000
+               value           = big.NewInt(0)
+               contract        = vm.NewContract(account{}, account{}, value, startGas)
+       )
+       contract.Code = []byte{byte(vm.PUSH1), 0x1, byte(vm.PUSH1), 0x1, 0x0}
+
+       tracer.CaptureStart(env, contract.Caller(), contract.Address(), false, []byte{}, startGas, value)
+       ret, err := env.Interpreter().Run(contract, []byte{}, false)
+       tracer.CaptureEnd(ret, startGas-contract.Gas, 1, err)
+       if err != nil {
+               return nil, err
+       }
+       return tracer.GetResult()
+}
+
+func TestTracer(t *testing.T) {
+       execTracer := func(code string) ([]byte, string) {
+               t.Helper()
+               tracer, err := newJsTracer(code, nil)
+               if err != nil {
+                       t.Fatal(err)
+               }
+               ret, err := runTrace(tracer, testCtx(), params.TestChainConfig)
+               if err != nil {
```

```
+                       return nil, err.Error() // Stringify to allow comparison without nil checks
+               }
+               return ret, ""
+       }
+       for i, tt := range []struct {
+               code string
+               want string
+               fail string
+       }{
+               { // tests that we don't panic on bad arguments to memory access
+                       code: "{depths: [], step: function(log) { this.depths.push(log.memory.slice(-1,-2)); }, fault: function() {}, result: function() { return this.depths; }}",
+                       want: `[{},{},{}]`,
+               }, { // tests that we don't panic on bad arguments to stack peeks
+                       code: "{depths: [], step: function(log) { this.depths.push(log.stack.peek(-1)); }, fault: function() {}, result: function() { return this.depths; }}",
+                       want: `["0","0","0"]`,
+               }, { //  tests that we don't panic on bad arguments to memory getUint
+                       code: "{ depths: [], step: function(log, db) { this.depths.push(log.memory.getUint(-64));}, fault: function() {}, result: function() { return this.depths; }}",
+                       want: `["0","0","0"]`,
+               }, { // tests some general counting
+                       code: "{count: 0, step: function() { this.count += 1; }, fault: function() {}, result: function() { return this.count; }}",
+                       want: `3`,
+               }, { // tests that depth is reported correctly
+                       code: "{depths: [], step: function(log) { this.depths.push(log.stack.length()); }, fault: function() {}, result: function() { return this.depths; }}",
+                       want: `[0,1,2]`,
+               }, { // tests to-string of opcodes
+                       code: "{opcodes: [], step: function(log) { this.opcodes.push(log.op.toString()); }, fault: function() {}, result: function() { return this.opcodes; }}",
+                       want: `["PUSH1","PUSH1","STOP"]`,
+               }, { // tests intrinsic gas
+                       code: "{depths: [], step: function() {}, fault: function() {}, result: function(ctx) { return ctx.gasPrice+'.'+ctx.gasUsed+'.'+ctx.intrinsicGas; }}",
+                       want: `"100000.6.21000"`,
+               }, { // tests too deep object / serialization crash
+                       code: "{step: function() {}, fault: function() {}, result: function() { var o={}; var x=o; for (var i=0; i<1000; i++){  o.foo={}; o=o.foo; } return x; }}",
+                       fail: "RangeError: json encode recursion limit    in server-side tracer function 'result'",
+               },
+       } {
+               if have, err := execTracer(tt.code); tt.want != string(have) || tt.fail != err {
+                       t.Errorf("testcase %d: expected return value to be '%s' got '%s', error to be '%s' got '%s'\n\tcode: %v", i, tt.want, string(have), tt.fail, err, tt.code)
+               }
+       }
+}
+
+func TestHalt(t *testing.T) {
+       t.Skip("duktape doesn't support abortion")
+       timeout := errors.New("stahp")
+       tracer, err := newJsTracer("{step: function() { while(1); }, result: function() { return null; }, fault: function(){}}", nil)
+       if err != nil {
+               t.Fatal(err)
+       }
+       go func() {
+               time.Sleep(1 * time.Second)
+               tracer.Stop(timeout)
+       }()
+       if _, err = runTrace(tracer, testCtx(), params.TestChainConfig); err.Error() != "stahp    in server-side tracer function 'step'" {
+               t.Errorf("Expected timeout error, got %v", err)
+       }
+}
+
+func TestHaltBetweenSteps(t *testing.T) {
+       tracer, err := newJsTracer("{step: function() {}, fault: function() {}, result: function() { return null; }}", nil)
+       if err != nil {
+               t.Fatal(err)
+       }
+       env := vm.NewEVM(vm.BlockContext{BlockNumber: big.NewInt(1)}, vm.TxContext{GasPrice: big.NewInt(1)}, &dummyStatedb{}, params.TestChainConfig, vm.Config{Debug: true, Tracer: tracer})
+       scope := &vm.ScopeContext{
+               Contract: vm.NewContract(&account{}, &account{}, big.NewInt(0), 0),
+       }
+       tracer.CaptureStart(env, common.Address{}, common.Address{}, false, []byte{}, 0, big.NewInt(0))
+       tracer.CaptureState(0, 0, 0, 0, scope, nil, 0, nil)
+       timeout := errors.New("stahp")
+       tracer.Stop(timeout)
+       tracer.CaptureState(0, 0, 0, 0, scope, nil, 0, nil)
+
+       if _, err := tracer.GetResult(); err.Error() != timeout.Error() {
+               t.Errorf("Expected timeout error, got %v", err)
+       }
+}
+
+// TestNoStepExec tests a regular value transfer (no exec), and accessing the statedb
+// in 'result'
+func TestNoStepExec(t *testing.T) {
+       execTracer := func(code string) []byte {
+               t.Helper()
+               tracer, err := newJsTracer(code, nil)
+               if err != nil {
+                       t.Fatal(err)
+               }
+               env := vm.NewEVM(vm.BlockContext{BlockNumber: big.NewInt(1)}, vm.TxContext{GasPrice: big.NewInt(100)}, &dummyStatedb{}, params.TestChainConfig, vm.Config{Debug: true, Tracer: tracer})
+               tracer.CaptureStart(env, common.Address{}, common.Address{}, false, []byte{}, 1000, big.NewInt(0))
+               tracer.CaptureEnd(nil, 0, 1, nil)
+               ret, err := tracer.GetResult()
+               if err != nil {
+                       t.Fatal(err)
+               }
+               return ret
+       }
+       for i, tt := range []struct {
+               code string
+               want string
+       }{
+               { // tests that we don't panic on accessing the db methods
+                       code: "{depths: [], step: function() {}, fault: function() {},  result: function(ctx, db){ return db.getBalance(ctx.to)} }",
+                       want: `"0"`,
+               },
+       } {
+               if have := execTracer(tt.code); tt.want != string(have) {
+                       t.Errorf("testcase %d: expected return value to be %s got %s\n\tcode: %v", i, tt.want, string(have), tt.code)
+               }
+       }
+}
+
+func TestIsPrecompile(t *testing.T) {
+       chaincfg := &params.ChainConfig{ChainID: big.NewInt(1), HomesteadBlock: big.NewInt(0), DAOForkBlock: nil, DAOForkSupport: false, EIP150Block: big.NewInt(0), EIP150Hash: common.Hash{}, EIP155Block
+       chaincfg.ByzantiumBlock = big.NewInt(100)
+       chaincfg.IstanbulBlock = big.NewInt(200)
+       chaincfg.ApricotPhase2BlockTimestamp = big.NewInt(300)
+       txCtx := vm.TxContext{GasPrice: big.NewInt(100000)}
+       tracer, err := newJsTracer("{addr: toAddress('0000000000000000000000000000000000000009'), res: null, step: function() { this.res = isPrecompiled(this.addr); }, fault: function() {}, result: funct
+       if err != nil {
+               t.Fatal(err)
+       }
+
+       blockCtx := vm.BlockContext{BlockNumber: big.NewInt(150)}
+       res, err := runTrace(tracer, &vmContext{blockCtx, txCtx}, chaincfg)
+       if err != nil {
+               t.Error(err)
+       }
+       if string(res) != "false" {
+               t.Errorf("Tracer should not consider blake2f as precompile in byzantium")
+       }
+
+       tracer, _ = newJsTracer("{addr: toAddress('0000000000000000000000000000000000000009'), res: null, step: function() { this.res = isPrecompiled(this.addr); }, fault: function() {}, result: function
+       blockCtx = vm.BlockContext{BlockNumber: big.NewInt(250)}
+       res, err = runTrace(tracer, &vmContext{blockCtx, txCtx}, chaincfg)
+       if err != nil {
```

```
+               t.Error(err)
+       }
+       if string(res) != "true" {
+               t.Errorf("Tracer should consider blake2f as precompile in istanbul")
+       }
+}
+
+func TestEnterExit(t *testing.T) {
+       // test that either both or none of enter() and exit() are defined
+       if _, err := newJsTracer("{step: function() {}, fault: function() {}, result: function() { return null; }, enter: function() {}}", new(tracers.Context)); err == nil {
+               t.Fatal("tracer creation should've failed without exit() definition")
+       }
+       if _, err := newJsTracer("{step: function() {}, fault: function() {}, result: function() { return null; }, enter: function() {}, exit: function() {}}", new(tracers.Context)); err != nil {
+               t.Fatal(err)
+       }
+       // test that the enter and exit method are correctly invoked and the values passed
+       tracer, err := newJsTracer("{enters: 0, exits: 0, enterGas: 0, gasUsed: 0, step: function() {}, fault: function() {}, result: function() { return {enters: this.enters, exits: this.exits, enterGas
+       if err != nil {
+               t.Fatal(err)
+       }
+       scope := &vm.ScopeContext{
+               Contract: vm.NewContract(&account{}, &account{}, big.NewInt(0), 0),
+       }
+       tracer.CaptureEnter(vm.CALL, scope.Contract.Caller(), scope.Contract.Address(), []byte{}, 1000, new(big.Int))
+       tracer.CaptureExit([]byte{}, 400, nil)
+
+       have, err := tracer.GetResult()
+       if err != nil {
+               t.Fatal(err)
+       }
+       want := `{"enters":1,"exits":1,"enterGas":1000,"gasUsed":400}`
+       if string(have) != want {
+               t.Errorf("Number of invocations of enter() and exit() is wrong. Have %s, want %s\n", have, want)
+       }
+}
diff --git a/eth/tracers/logger/access_list_tracer.go b/eth/tracers/logger/access_list_tracer.go
new file mode 100644
index 00000000..72ea7f98
--- /dev/null
+++ b/eth/tracers/logger/access_list_tracer.go
@@ -0,0 +1,185 @@
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package logger
+
+import (
+       "math/big"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+)
+
+// accessList is an accumulator for the set of accounts and storage slots an EVM
+// contract execution touches.
+type accessList map[common.Address]accessListSlots
+
+// accessListSlots is an accumulator for the set of storage slots within a single
+// contract that an EVM contract execution touches.
+type accessListSlots map[common.Hash]struct{}
+
+// newAccessList creates a new accessList.
+func newAccessList() accessList {
+       return make(map[common.Address]accessListSlots)
+}
+
+// addAddress adds an address to the accesslist.
+func (al accessList) addAddress(address common.Address) {
+       // Set address if not previously present
+       if _, present := al[address]; !present {
+               al[address] = make(map[common.Hash]struct{})
+       }
+}
+
+// addSlot adds a storage slot to the accesslist.
+func (al accessList) addSlot(address common.Address, slot common.Hash) {
+       // Set address if not previously present
+       al.addAddress(address)
+
+       // Set the slot on the surely existent storage set
+       al[address][slot] = struct{}{}
+}
+
+// equal checks if the content of the current access list is the same as the
+// content of the other one.
+func (al accessList) equal(other accessList) bool {
+       // Cross reference the accounts first
+       if len(al) != len(other) {
+               return false
+       }
+       for addr := range al {
+               if _, ok := other[addr]; !ok {
+                       return false
+               }
+       }
+       for addr := range other {
+               if _, ok := al[addr]; !ok {
+                       return false
+               }
+       }
+       // Accounts match, cross reference the storage slots too
+       for addr, slots := range al {
+               otherslots := other[addr]
+
+               if len(slots) != len(otherslots) {
+                       return false
+               }
+               for hash := range slots {
+                       if _, ok := otherslots[hash]; !ok {
+                               return false
+                       }
+               }
+               for hash := range otherslots {
+                       if _, ok := slots[hash]; !ok {
+                               return false
+                       }
+               }
+       }
+       return true
```

```
+}
+
+// accesslist converts the accesslist to a types.AccessList.
+func (al accessList) accessList() types.AccessList {
+	acl := make(types.AccessList, 0, len(al))
+	for addr, slots := range al {
+		tuple := types.AccessTuple{Address: addr, StorageKeys: []common.Hash{}}
+		for slot := range slots {
+			tuple.StorageKeys = append(tuple.StorageKeys, slot)
+		}
+		acl = append(acl, tuple)
+	}
+	return acl
+}
+
+// AccessListTracer is a tracer that accumulates touched accounts and storage
+// slots into an internal set.
+type AccessListTracer struct {
+	excl map[common.Address]struct{} // Set of account to exclude from the list
+	list accessList                  // Set of accounts and storage slots touched
+}
+
+// NewAccessListTracer creates a new tracer that can generate AccessLists.
+// An optional AccessList can be specified to occupy slots and addresses in
+// the resulting accesslist.
+func NewAccessListTracer(acl types.AccessList, from, to common.Address, precompiles []common.Address) *AccessListTracer {
+	excl := map[common.Address]struct{}{
+		from: {}, to: {},
+	}
+	for _, addr := range precompiles {
+		excl[addr] = struct{}{}
+	}
+	list := newAccessList()
+	for _, al := range acl {
+		if _, ok := excl[al.Address]; !ok {
+			list.addAddress(al.Address)
+		}
+		for _, slot := range al.StorageKeys {
+			list.addSlot(al.Address, slot)
+		}
+	}
+	return &AccessListTracer{
+		excl: excl,
+		list: list,
+	}
+}
+
+func (a *AccessListTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+}
+
+// CaptureState captures all opcodes that touch storage or addresses and adds them to the accesslist.
+func (a *AccessListTracer) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+	stack := scope.Stack
+	stackData := stack.Data()
+	stackLen := len(stackData)
+	if (op == vm.SLOAD || op == vm.SSTORE) && stackLen >= 1 {
+		slot := common.Hash(stackData[stackLen-1].Bytes32())
+		a.list.addSlot(scope.Contract.Address(), slot)
+	}
+	if (op == vm.EXTCODECOPY || op == vm.EXTCODEHASH || op == vm.EXTCODESIZE || op == vm.BALANCE || op == vm.SELFDESTRUCT) && stackLen >= 1 {
+		addr := common.Address(stackData[stackLen-1].Bytes20())
+		if _, ok := a.excl[addr]; !ok {
+			a.list.addAddress(addr)
+		}
+	}
+	if (op == vm.DELEGATECALL || op == vm.CALL || op == vm.STATICCALL || op == vm.CALLCODE) && stackLen >= 5 {
+		addr := common.Address(stackData[stackLen-2].Bytes20())
+		if _, ok := a.excl[addr]; !ok {
+			a.list.addAddress(addr)
+		}
+	}
+}
+
+func (*AccessListTracer) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+}
+
+func (*AccessListTracer) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {}
+
+func (*AccessListTracer) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+}
+
+func (*AccessListTracer) CaptureExit(output []byte, gasUsed uint64, err error) {}
+
+// AccessList returns the current accesslist maintained by the tracer.
+func (a *AccessListTracer) AccessList() types.AccessList {
+	return a.list.accessList()
+}
+
+// Equal returns if the content of two access list traces are equal.
+func (a *AccessListTracer) Equal(other *AccessListTracer) bool {
+	return a.list.equal(other.list)
+}
diff --git a/eth/tracers/logger/gen_structlog.go b/eth/tracers/logger/gen_structlog.go
new file mode 100644
index 00000000..20569f62
--- /dev/null
+++ b/eth/tracers/logger/gen_structlog.go
@@ -0,0 +1,110 @@
+// Code generated by github.com/fjl/gencodec. DO NOT EDIT.
+
+package logger
+
+import (
+	"encoding/json"
+
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/ethereum/go-ethereum/common/hexutil"
+	"github.com/ethereum/go-ethereum/common/math"
+	"github.com/holiman/uint256"
+)
+
+var _ = (*structLogMarshaling)(nil)
+
+// MarshalJSON marshals as JSON.
+func (s StructLog) MarshalJSON() ([]byte, error) {
+	type StructLog struct {
+		Pc            uint64                      `json:"pc"`
+		Op            vm.OpCode                   `json:"op"`
+		Gas           math.HexOrDecimal64         `json:"gas"`
+		GasCost       math.HexOrDecimal64         `json:"gasCost"`
+		Memory        hexutil.Bytes               `json:"memory"`
+		MemorySize    int                         `json:"memSize"`
+		Stack         []uint256.Int               `json:"stack"`
+		ReturnData    hexutil.Bytes               `json:"returnData"`
+		Storage       map[common.Hash]common.Hash `json:"-"`
+		Depth         int                         `json:"depth"`
+		RefundCounter uint64                      `json:"refund"`
+		Err           error                       `json:"-"`
+		OpName        string                      `json:"opName"`
+		ErrorString   string                      `json:"error"`
+	}
+	var enc StructLog
+	enc.Pc = s.Pc
```

```
+       enc.Op = s.Op
+       enc.Gas = math.HexOrDecimal64(s.Gas)
+       enc.GasCost = math.HexOrDecimal64(s.GasCost)
+       enc.Memory = s.Memory
+       enc.MemorySize = s.MemorySize
+       enc.Stack = s.Stack
+       enc.ReturnData = s.ReturnData
+       enc.Storage = s.Storage
+       enc.Depth = s.Depth
+       enc.RefundCounter = s.RefundCounter
+       enc.Err = s.Err
+       enc.OpName = s.OpName()
+       enc.ErrorString = s.ErrorString()
+       return json.Marshal(&enc)
+}
+
+// UnmarshalJSON unmarshals from JSON.
+func (s *StructLog) UnmarshalJSON(input []byte) error {
+       type StructLog struct {
+               Pc            *uint64                     `json:"pc"`
+               Op            *vm.OpCode                  `json:"op"`
+               Gas           *math.HexOrDecimal64        `json:"gas"`
+               GasCost       *math.HexOrDecimal64        `json:"gasCost"`
+               Memory        *hexutil.Bytes              `json:"memory"`
+               MemorySize    *int                        `json:"memSize"`
+               Stack         []uint256.Int               `json:"stack"`
+               ReturnData    *hexutil.Bytes              `json:"returnData"`
+               Storage       map[common.Hash]common.Hash `json:"-"`
+               Depth         *int                        `json:"depth"`
+               RefundCounter *uint64                     `json:"refund"`
+               Err           error                       `json:"-"`
+       }
+       var dec StructLog
+       if err := json.Unmarshal(input, &dec); err != nil {
+               return err
+       }
+       if dec.Pc != nil {
+               s.Pc = *dec.Pc
+       }
+       if dec.Op != nil {
+               s.Op = *dec.Op
+       }
+       if dec.Gas != nil {
+               s.Gas = uint64(*dec.Gas)
+       }
+       if dec.GasCost != nil {
+               s.GasCost = uint64(*dec.GasCost)
+       }
+       if dec.Memory != nil {
+               s.Memory = *dec.Memory
+       }
+       if dec.MemorySize != nil {
+               s.MemorySize = *dec.MemorySize
+       }
+       if dec.Stack != nil {
+               s.Stack = dec.Stack
+       }
+       if dec.ReturnData != nil {
+               s.ReturnData = *dec.ReturnData
+       }
+       if dec.Storage != nil {
+               s.Storage = dec.Storage
+       }
+       if dec.Depth != nil {
+               s.Depth = *dec.Depth
+       }
+       if dec.RefundCounter != nil {
+               s.RefundCounter = *dec.RefundCounter
+       }
+       if dec.Err != nil {
+               s.Err = dec.Err
+       }
+       return nil
+}
diff --git a/eth/tracers/logger/logger.go b/eth/tracers/logger/logger.go
new file mode 100644
index 00000000..5b5f73ec
--- /dev/null
+++ b/eth/tracers/logger/logger.go
@@ -0,0 +1,349 @@
+// Copyright 2015 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package logger
+
+import (
+       "encoding/hex"
+       "fmt"
+       "io"
+       "math/big"
+       "strings"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/ethereum/go-ethereum/common/math"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/holiman/uint256"
+)
+
+// Storage represents a contract's storage.
+type Storage map[common.Hash]common.Hash
+
+// Copy duplicates the current storage.
+func (s Storage) Copy() Storage {
+       cpy := make(Storage)
+       for key, value := range s {
+               cpy[key] = value
+       }
+       return cpy
+}
+
+// Config are the configuration options for structured logger the EVM
+type Config struct {
+       EnableMemory     bool // enable memory capture
+       DisableStack     bool // disable stack capture
+       DisableStorage   bool // disable storage capture
+       EnableReturnData bool // enable return data capture
+       Debug            bool // print output during capture end
+       Limit            int  // maximum length of output, but zero means unlimited
+       // Chain overrides, can be used to execute a trace using future fork rules
+       Overrides *params.ChainConfig `json:"overrides,omitempty"`
+}
+
+//go:generate gencodec -type StructLog -field-override structLogMarshaling -out gen_structlog.go
+
+// StructLog is emitted to the EVM each cycle and lists information about the current internal state
+// prior to the execution of the statement.
+type StructLog struct {
+       Pc            uint64                      `json:"pc"`
+       Op            vm.OpCode                   `json:"op"`
+       Gas           uint64                      `json:"gas"`
+       GasCost       uint64                      `json:"gasCost"`
+       Memory        []byte                      `json:"memory"`
+       MemorySize    int                         `json:"memSize"`
+       Stack         []uint256.Int               `json:"stack"`
+       ReturnData    []byte                      `json:"returnData"`
+       Storage       map[common.Hash]common.Hash `json:"-"`
+       Depth         int                         `json:"depth"`
+       RefundCounter uint64                      `json:"refund"`
+       Err           error                       `json:"-"`
+}
+
+// overrides for gencodec
+type structLogMarshaling struct {
+       Gas         math.HexOrDecimal64
+       GasCost     math.HexOrDecimal64
+       Memory      hexutil.Bytes
+       ReturnData  hexutil.Bytes
+       OpName      string `json:"opName"` // adds call to OpName() in MarshalJSON
+       ErrorString string `json:"error"`  // adds call to ErrorString() in MarshalJSON
+}
+
+// OpName formats the operand name in a human-readable format.
+func (s *StructLog) OpName() string {
+       return s.Op.String()
+}
+
+// ErrorString formats the log's error as a string.
+func (s *StructLog) ErrorString() string {
+       if s.Err != nil {
+               return s.Err.Error()
+       }
+       return ""
+}
+
+// StructLogger is an EVM state logger and implements EVMLogger.
+//
+// StructLogger can capture state based on the given Log configuration and also keeps
+// a track record of modified storage which is used in reporting snapshots of the
+// contract their storage.
+type StructLogger struct {
+       cfg Config
+       env *vm.EVM
+
+       storage map[common.Address]Storage
+       logs    []StructLog
+       output  []byte
+       err     error
+}
+
+// NewStructLogger returns a new logger
+func NewStructLogger(cfg *Config) *StructLogger {
+       logger := &StructLogger{
+               storage: make(map[common.Address]Storage),
+       }
+       if cfg != nil {
+               logger.cfg = *cfg
+       }
+       return logger
+}
+
+// Reset clears the data held by the logger.
+func (l *StructLogger) Reset() {
+       l.storage = make(map[common.Address]Storage)
+       l.output = make([]byte, 0)
+       l.logs = l.logs[:0]
+       l.err = nil
+}
+
+// CaptureStart implements the EVMLogger interface to initialize the tracing operation.
+func (l *StructLogger) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+       l.env = env
+}
+
+// CaptureState logs a new structured log message and pushes it out to the environment
+//
+// CaptureState also tracks SLOAD/SSTORE ops to track storage change.
+func (l *StructLogger) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+       memory := scope.Memory
+       stack := scope.Stack
+       contract := scope.Contract
+       // check if already accumulated the specified number of logs
+       if l.cfg.Limit != 0 && l.cfg.Limit <= len(l.logs) {
+               return
+       }
+       // Copy a snapshot of the current memory state to a new buffer
+       var mem []byte
+       if l.cfg.EnableMemory {
+               mem = make([]byte, len(memory.Data()))
+               copy(mem, memory.Data())
+       }
+       // Copy a snapshot of the current stack state to a new buffer
+       var stck []uint256.Int
+       if !l.cfg.DisableStack {
+               stck = make([]uint256.Int, len(stack.Data()))
+               for i, item := range stack.Data() {
+                       stck[i] = item
+               }
+       }
+       stackData := stack.Data()
+       stackLen := len(stackData)
+       // Copy a snapshot of the current storage to a new container
+       var storage Storage
+       if !l.cfg.DisableStorage && (op == vm.SLOAD || op == vm.SSTORE) {
+               // initialise new changed values storage container for this contract
+               // if not present.
+               if l.storage[contract.Address()] == nil {
+                       l.storage[contract.Address()] = make(Storage)
+               }
+               // capture SLOAD opcodes and record the read entry in the local storage
+               if op == vm.SLOAD && stackLen >= 1 {
+                       var (
+                               address = common.Hash(stackData[stackLen-1].Bytes32())
+                               value   = l.env.StateDB.GetState(contract.Address(), address)
+                       )
+                       l.storage[contract.Address()][address] = value
+                       storage = l.storage[contract.Address()].Copy()
+               } else if op == vm.SSTORE && stackLen >= 2 {
+                       // capture SSTORE opcodes and record the written entry in the local storage.
+                       var (
+                               value   = common.Hash(stackData[stackLen-2].Bytes32())
+                               address = common.Hash(stackData[stackLen-1].Bytes32())
+                       )
+                       l.storage[contract.Address()][address] = value
+                       storage = l.storage[contract.Address()].Copy()
+               }
+       }
+       var rdata []byte
+       if l.cfg.EnableReturnData {
+               rdata = make([]byte, len(rData))
+               copy(rdata, rData)
+       }
+       // create a new snapshot of the EVM.
+       log := StructLog{pc, op, gas, cost, mem, memory.Len(), stck, rdata, storage, depth, l.env.StateDB.GetRefund(), err}
+       l.logs = append(l.logs, log)
+}
+
+// CaptureFault implements the EVMLogger interface to trace an execution fault
+// while running an opcode.
+func (l *StructLogger) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+}
+
+// CaptureEnd is called after the call finishes to finalize the tracing.
+func (l *StructLogger) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {
+       l.output = output
+       l.err = err
+       if l.cfg.Debug {
+               fmt.Printf("0x%x\n", output)
+               if err != nil {
+                       fmt.Printf(" error: %v\n", err)
+               }
+       }
+}
+
+func (l *StructLogger) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+}
+
+func (l *StructLogger) CaptureExit(output []byte, gasUsed uint64, err error) {}
+
+// StructLogs returns the captured log entries.
+func (l *StructLogger) StructLogs() []StructLog { return l.logs }
+
+// Error returns the VM error captured by the trace.
+func (l *StructLogger) Error() error { return l.err }
+
+// Output returns the VM return value captured by the trace.
+func (l *StructLogger) Output() []byte { return l.output }
+
+// WriteTrace writes a formatted trace to the given writer
+func WriteTrace(writer io.Writer, logs []StructLog) {
+       for _, log := range logs {
+               fmt.Fprintf(writer, "%-16spc=%08d gas=%v cost=%v", log.Op, log.Pc, log.Gas, log.GasCost)
+               if log.Err != nil {
+                       fmt.Fprintf(writer, " ERROR: %v", log.Err)
+               }
+               fmt.Fprintln(writer)
+
+               if len(log.Stack) > 0 {
+                       fmt.Fprintln(writer, "Stack:")
+                       for i := len(log.Stack) - 1; i >= 0; i-- {
+                               fmt.Fprintf(writer, "%08d  %s\n", len(log.Stack)-i-1, log.Stack[i].Hex())
+                       }
+               }
+               if len(log.Memory) > 0 {
+                       fmt.Fprintln(writer, "Memory:")
+                       fmt.Fprint(writer, hex.Dump(log.Memory))
+               }
+               if len(log.Storage) > 0 {
+                       fmt.Fprintln(writer, "Storage:")
+                       for h, item := range log.Storage {
+                               fmt.Fprintf(writer, "%x: %x\n", h, item)
+                       }
+               }
+               if len(log.ReturnData) > 0 {
+                       fmt.Fprintln(writer, "ReturnData:")
+                       fmt.Fprint(writer, hex.Dump(log.ReturnData))
+               }
+               fmt.Fprintln(writer)
+       }
+}
+
+// WriteLogs writes vm logs in a readable format to the given writer
+func WriteLogs(writer io.Writer, logs []*types.Log) {
+       for _, log := range logs {
+               fmt.Fprintf(writer, "LOG%d: %x bn=%d txi=%x\n", len(log.Topics), log.Address, log.BlockNumber, log.TxIndex)
+
+               for i, topic := range log.Topics {
+                       fmt.Fprintf(writer, "%08d  %x\n", i, topic)
+               }
+
+               fmt.Fprint(writer, hex.Dump(log.Data))
+               fmt.Fprintln(writer)
+       }
+}
+
+type mdLogger struct {
+       out io.Writer
+       cfg *Config
+       env *vm.EVM
+}
+
+// NewMarkdownLogger creates a logger which outputs information in a format adapted
+// for human readability, and is also a valid markdown table
+func NewMarkdownLogger(cfg *Config, writer io.Writer) *mdLogger {
+       l := &mdLogger{out: writer, cfg: cfg}
+       if l.cfg == nil {
+               l.cfg = &Config{}
+       }
+       return l
+}
+
+func (t *mdLogger) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+       t.env = env
+       if !create {
+               fmt.Fprintf(t.out, "From: `%v`\nTo: `%v`\nData: `0x%x`\nGas: `%d`\nValue `%v` wei\n",
+                       from.String(), to.String(),
+                       input, gas, value)
+       } else {
+               fmt.Fprintf(t.out, "From: `%v`\nCreate at: `%v`\nData: `0x%x`\nGas: `%d`\nValue `%v` wei\n",
+                       from.String(), to.String(),
+                       input, gas, value)
+       }
+
+       fmt.Fprintf(t.out, `
+|  Pc   |      Op     | Cost |   Stack   |   RStack  |  Refund |
+|-------|-------------|------|-----------|-----------|---------|
+`)
+}
+
+// CaptureState also tracks SLOAD/SSTORE ops to track storage change.
+func (t *mdLogger) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+       stack := scope.Stack
+       fmt.Fprintf(t.out, "| %4d  | %10v  |  %3d |", pc, op, cost)
+
+       if !t.cfg.DisableStack {
+               // format stack
```

```
+		var a []string
+		for _, elem := range stack.Data() {
+			a = append(a, elem.Hex())
+		}
+		b := fmt.Sprintf("[%v]", strings.Join(a, ","))
+		fmt.Fprintf(t.out, "%10v |", b)
+	}
+	fmt.Fprintf(t.out, "%10v |", t.env.StateDB.GetRefund())
+	fmt.Fprintln(t.out, "")
+	if err != nil {
+		fmt.Fprintf(t.out, "Error: %v\n", err)
+	}
+}
+
+func (t *mdLogger) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+	fmt.Fprintf(t.out, "\nError: at pc=%d, op=%v: %v\n", pc, op, err)
+}
+
+func (t *mdLogger) CaptureEnd(output []byte, gasUsed uint64, tm time.Duration, err error) {
+	fmt.Fprintf(t.out, "\nOutput: `0x%x`\nConsumed gas: `%d`\nError: `%v`\n",
+		output, gasUsed, err)
+}
+
+func (t *mdLogger) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+}
+
+func (t *mdLogger) CaptureExit(output []byte, gasUsed uint64, err error) {}
diff --git a/eth/tracers/logger/logger_json.go b/eth/tracers/logger/logger_json.go
new file mode 100644
index 00000000..5d222466
--- /dev/null
+++ b/eth/tracers/logger/logger_json.go
@@ -0,0 +1,100 @@
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package logger
+
+import (
+	"encoding/json"
+	"io"
+	"math/big"
+	"time"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/ethereum/go-ethereum/common/math"
+	"github.com/flare-foundation/coreth/core/vm"
+)
+
+type JSONLogger struct {
+	encoder *json.Encoder
+	cfg     *Config
+	env     *vm.EVM
+}
+
+// NewJSONLogger creates a new EVM tracer that prints execution steps as JSON objects
+// into the provided stream.
+func NewJSONLogger(cfg *Config, writer io.Writer) *JSONLogger {
+	l := &JSONLogger{encoder: json.NewEncoder(writer), cfg: cfg}
+	if l.cfg == nil {
+		l.cfg = &Config{}
+	}
+	return l
+}
+
+func (l *JSONLogger) CaptureStart(env *vm.EVM, from, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+	l.env = env
+}
+
+func (l *JSONLogger) CaptureFault(pc uint64, op vm.OpCode, gas uint64, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+	// TODO: Add rData to this interface as well
+	l.CaptureState(pc, op, gas, cost, scope, nil, depth, err)
+}
+
+// CaptureState outputs state information on the logger.
+func (l *JSONLogger) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+	memory := scope.Memory
+	stack := scope.Stack
+
+	log := StructLog{
+		Pc:            pc,
+		Op:            op,
+		Gas:           gas,
+		GasCost:       cost,
+		MemorySize:    memory.Len(),
+		Depth:         depth,
+		RefundCounter: l.env.StateDB.GetRefund(),
+		Err:           err,
+	}
+	if l.cfg.EnableMemory {
+		log.Memory = memory.Data()
+	}
+	if !l.cfg.DisableStack {
+		log.Stack = stack.Data()
+	}
+	if l.cfg.EnableReturnData {
+		log.ReturnData = rData
+	}
+	l.encoder.Encode(log)
+}
+
+// CaptureEnd is triggered at end of execution.
+func (l *JSONLogger) CaptureEnd(output []byte, gasUsed uint64, t time.Duration, err error) {
+	type endLog struct {
+		Output  string              `json:"output"`
+		GasUsed math.HexOrDecimal64 `json:"gasUsed"`
+		Time    time.Duration       `json:"time"`
+		Err     string              `json:"error,omitempty"`
+	}
+	var errMsg string
+	if err != nil {
+		errMsg = err.Error()
+	}
+	l.encoder.Encode(endLog{common.Bytes2Hex(output), math.HexOrDecimal64(gasUsed), t, errMsg})
+}
+
+func (l *JSONLogger) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+}
+
+func (l *JSONLogger) CaptureExit(output []byte, gasUsed uint64, err error) {}
diff --git a/eth/tracers/logger/logger_test.go b/eth/tracers/logger/logger_test.go
```

```
new file mode 100644
index 00000000..954cc119
--- /dev/null
+++ b/eth/tracers/logger/logger_test.go
@@ -0,0 +1,74 @@
+// Copyright 2016 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package logger
+
+import (
+	"math/big"
+	"testing"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/params"
+)
+
+type dummyContractRef struct {
+	calledForEach bool
+}
+
+func (dummyContractRef) Address() common.Address     { return common.Address{} }
+func (dummyContractRef) Value() *big.Int             { return new(big.Int) }
+func (dummyContractRef) SetCode(common.Hash, []byte) {}
+func (d *dummyContractRef) ForEachStorage(callback func(key, value common.Hash) bool) {
+	d.calledForEach = true
+}
+func (d *dummyContractRef) SubBalance(amount *big.Int) {}
+func (d *dummyContractRef) AddBalance(amount *big.Int) {}
+func (d *dummyContractRef) SetBalance(*big.Int)        {}
+func (d *dummyContractRef) SetNonce(uint64)            {}
+func (d *dummyContractRef) Balance() *big.Int          { return new(big.Int) }
+
+type dummyStatedb struct {
+	state.StateDB
+}
+
+func (*dummyStatedb) GetRefund() uint64                                       { return 1337 }
+func (*dummyStatedb) GetState(_ common.Address, _ common.Hash) common.Hash    { return common.Hash{} }
+func (*dummyStatedb) SetState(_ common.Address, _ common.Hash, _ common.Hash) {}
+
+func TestStoreCapture(t *testing.T) {
+	var (
+		logger   = NewStructLogger(nil)
+		env      = vm.NewEVM(vm.BlockContext{}, vm.TxContext{}, &dummyStatedb{}, params.TestChainConfig, vm.Config{Debug: true, Tracer: logger})
+		contract = vm.NewContract(&dummyContractRef{}, &dummyContractRef{}, new(big.Int), 100000)
+	)
+	contract.Code = []byte{byte(vm.PUSH1), 0x1, byte(vm.PUSH1), 0x0, byte(vm.SSTORE)}
+	var index common.Hash
+	logger.CaptureStart(env, common.Address{}, contract.Address(), false, nil, 0, nil)
+	_, err := env.Interpreter().Run(contract, []byte{}, false)
+	if err != nil {
+		t.Fatal(err)
+	}
+	if len(logger.storage[contract.Address()]) == 0 {
+		t.Fatalf("expected exactly 1 changed value on address %x, got %d", contract.Address(),
+			len(logger.storage[contract.Address()]))
+	}
+	exp := common.BigToHash(big.NewInt(1))
+	if logger.storage[contract.Address()][index] != exp {
+		t.Errorf("expected %x, got %x", exp, logger.storage[contract.Address()][index])
+	}
+}
diff --git a/eth/tracers/native/4byte.go b/eth/tracers/native/4byte.go
new file mode 100644
index 00000000..473abc02
--- /dev/null
+++ b/eth/tracers/native/4byte.go
@@ -0,0 +1,158 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package native
+
+import (
+	"encoding/json"
+	"math/big"
+	"strconv"
+	"sync/atomic"
+	"time"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/tracers"
+)
+
+func init() {
+	register("4byteTracer", newFourByteTracer)
+}
+
+// fourByteTracer searches for 4byte-identifiers, and collects them for post-processing.
+// It collects the methods identifiers along with the size of the supplied data, so
+// a reversed signature can be matched against the size of the data.
+//
+// Example:
```

```
+//   > debug.traceTransaction( "0x214e597e35da083692f5386141e69f47e973b2c56e7a8073b1ea08fd7571e9de", {tracer: "4byteTracer"})
+//   {
+//     0x27dc297e-128: 1,
+//     0x38cc4831-0: 2,
+//     0x524f3889-96: 1,
+//     0xadf59f99-288: 1,
+//     0xc281d19e-0: 1
+//   }
+type fourByteTracer struct {
+       env               *vm.EVM
+       ids               map[string]int   // ids aggregates the 4byte ids found
+       interrupt         uint32           // Atomic flag to signal execution interruption
+       reason            error            // Textual reason for the interruption
+       activePrecompiles []common.Address // Updated on CaptureStart based on given rules
+}
+
+// newFourByteTracer returns a native go tracer which collects
+// 4 byte-identifiers of a tx, and implements vm.EVMLogger.
+func newFourByteTracer() tracers.Tracer {
+       t := &fourByteTracer{
+               ids: make(map[string]int),
+       }
+       return t
+}
+
+// isPrecompiled returns whether the addr is a precompile. Logic borrowed from newJsTracer in eth/tracers/js/tracer.go
+func (t *fourByteTracer) isPrecompiled(addr common.Address) bool {
+       for _, p := range t.activePrecompiles {
+               if p == addr {
+                       return true
+               }
+       }
+       return false
+}
+
+// store saves the given identifier and datasize.
+func (t *fourByteTracer) store(id []byte, size int) {
+       key := bytesToHex(id) + "-" + strconv.Itoa(size)
+       t.ids[key] += 1
+}
+
+// CaptureStart implements the EVMLogger interface to initialize the tracing operation.
+func (t *fourByteTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+       t.env = env
+
+       // Update list of precompiles based on current block
+       rules := env.ChainConfig().AvalancheRules(env.Context.BlockNumber, env.Context.Time)
+       t.activePrecompiles = vm.ActivePrecompiles(rules)
+
+       // Save the outer calldata also
+       if len(input) >= 4 {
+               t.store(input[0:4], len(input)-4)
+       }
+}
+
+// CaptureState implements the EVMLogger interface to trace a single step of VM execution.
+func (t *fourByteTracer) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+}
+
+// CaptureEnter is called when EVM enters a new scope (via call, create or selfdestruct).
+func (t *fourByteTracer) CaptureEnter(op vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
+       // Skip if tracing was interrupted
+       if atomic.LoadUint32(&t.interrupt) > 0 {
+               t.env.Cancel()
+               return
+       }
+       if len(input) < 4 {
+               return
+       }
+       // primarily we want to avoid CREATE/CREATE2/SELFDESTRUCT
+       if op != vm.DELEGATECALL && op != vm.STATICCALL &&
+               op != vm.CALL && op != vm.CALLCODE {
+               return
+       }
+       // Skip any pre-compile invocations, those are just fancy opcodes
+       if t.isPrecompiled(to) {
+               return
+       }
+       t.store(input[0:4], len(input)-4)
+}
+
+// CaptureExit is called when EVM exits a scope, even if the scope didn't
+// execute any code.
+func (t *fourByteTracer) CaptureExit(output []byte, gasUsed uint64, err error) {
+}
+
+// CaptureFault implements the EVMLogger interface to trace an execution fault.
+func (t *fourByteTracer) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, depth int, err error) {
+}
+
+// CaptureEnd is called after the call finishes to finalize the tracing.
+func (t *fourByteTracer) CaptureEnd(output []byte, gasUsed uint64, _ time.Duration, err error) {
+}
+
+// GetResult returns the json-encoded nested list of call traces, and any
+// error arising from the encoding or forceful termination (via `Stop`).
+func (t *fourByteTracer) GetResult() (json.RawMessage, error) {
+       res, err := json.Marshal(t.ids)
+       if err != nil {
+               return nil, err
+       }
+       return res, t.reason
+}
+
+// Stop terminates execution of the tracer at the first opportune moment.
+func (t *fourByteTracer) Stop(err error) {
+       t.reason = err
+       atomic.StoreUint32(&t.interrupt, 1)
+}
diff --git a/eth/tracers/native/call.go b/eth/tracers/native/call.go
index 757a7653..f248b3d5 100644
--- a/eth/tracers/native/call.go
+++ b/eth/tracers/native/call.go
@@ -1,4 +1,4 @@
-// (c) 2019-2020, Ava Labs, Inc.
+// (c) 2020-2021, Ava Labs, Inc.
 //
 // This file is a derived work, based on the go-ethereum library whose original
 // notices appear below.
@@ -35,13 +35,13 @@ import (
        "sync/atomic"
        "time"
 
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/eth/tracers"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/eth/tracers"
 )
 
 func init() {
-       tracers.RegisterNativeTracer("callTracer", NewCallTracer)
+       register("callTracer", newCallTracer)
 }
```

```diff
 type callFrame struct {
@@ -58,21 +58,24 @@ type callFrame struct {
 }
 
 type callTracer struct {
+	env       *vm.EVM
 	callstack []callFrame
 	interrupt uint32 // Atomic flag to signal execution interruption
 	reason    error  // Textual reason for the interruption
 }
 
-// NewCallTracer returns a native go tracer which tracks
+// newCallTracer returns a native go tracer which tracks
 // call frames of a tx, and implements vm.EVMLogger.
-func NewCallTracer() tracers.Tracer {
+func newCallTracer() tracers.Tracer {
 	// First callframe contains tx context info
 	// and is populated on start and end.
 	t := &callTracer{callstack: make([]callFrame, 1)}
 	return t
 }
 
+// CaptureStart implements the EVMLogger interface to initialize the tracing operation.
 func (t *callTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
+	t.env = env
 	t.callstack[0] = callFrame{
 		Type:  "CALL",
 		From:  addrToHex(from),
@@ -86,6 +89,7 @@ func (t *callTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Ad
 	}
 }
 
+// CaptureEnd is called after the call finishes to finalize the tracing.
 func (t *callTracer) CaptureEnd(output []byte, gasUsed uint64, _ time.Duration, err error) {
 	t.callstack[0].GasUsed = uintToHex(gasUsed)
 	if err != nil {
@@ -98,16 +102,19 @@ func (t *callTracer) CaptureEnd(output []byte, gasUsed uint64, _ time.Duration,
 	}
 }
 
-func (t *callTracer) CaptureState(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+// CaptureState implements the EVMLogger interface to trace a single step of VM execution.
+func (t *callTracer) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
 }
 
-func (t *callTracer) CaptureFault(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, _ *vm.ScopeContext, depth int, err error) {
+// CaptureFault implements the EVMLogger interface to trace an execution fault.
+func (t *callTracer) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, _ *vm.ScopeContext, depth int, err error) {
 }
 
+// CaptureEnter is called when EVM enters a new scope (via call, create or selfdestruct).
 func (t *callTracer) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
 	// Skip if tracing was interrupted
 	if atomic.LoadUint32(&t.interrupt) > 0 {
-		// TODO: env.Cancel()
+		t.env.Cancel()
 		return
 	}
 
@@ -122,6 +129,8 @@ func (t *callTracer) CaptureEnter(typ vm.OpCode, from common.Address, to common.
 	t.callstack = append(t.callstack, call)
 }
 
+// CaptureExit is called when EVM exits a scope, even if the scope didn't
+// execute any code.
 func (t *callTracer) CaptureExit(output []byte, gasUsed uint64, err error) {
 	size := len(t.callstack)
 	if size <= 1 {
@@ -144,6 +153,8 @@ func (t *callTracer) CaptureExit(output []byte, gasUsed uint64, err error) {
 	t.callstack[size-1].Calls = append(t.callstack[size-1].Calls, call)
 }
 
+// GetResult returns the json-encoded nested list of call traces, and any
+// error arising from the encoding or forceful termination (via `Stop`).
 func (t *callTracer) GetResult() (json.RawMessage, error) {
 	if len(t.callstack) != 1 {
 		return nil, errors.New("incorrect number of top-level calls")
@@ -155,6 +166,7 @@ func (t *callTracer) GetResult() (json.RawMessage, error) {
 	return json.RawMessage(res), t.reason
 }
 
+// Stop terminates execution of the tracer at the first opportune moment.
 func (t *callTracer) Stop(err error) {
 	t.reason = err
 	atomic.StoreUint32(&t.interrupt, 1)
diff --git a/eth/tracers/native/noop.go b/eth/tracers/native/noop.go
index 2b0eb09c..cddbbc9f 100644
--- a/eth/tracers/native/noop.go
+++ b/eth/tracers/native/noop.go
@@ -1,3 +1,29 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
 package native
 
 import (
@@ -5,42 +31,54 @@ import (
 	"math/big"
 	"time"
 
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/eth/tracers"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/tracers"
 )
 
 func init() {
-	tracers.RegisterNativeTracer("noopTracerNative", NewNoopTracer)
+	register("noopTracerNative", newNoopTracer)
```

```
 }

+// noopTracer is a go implementation of the Tracer interface which
+// performs no action. It's mostly useful for testing purposes.
 type noopTracer struct{}

-func NewNoopTracer() tracers.Tracer {
+// newNoopTracer returns a new noop tracer.
+func newNoopTracer() tracers.Tracer {
        return &noopTracer{}
 }

+// CaptureStart implements the EVMLogger interface to initialize the tracing operation.
 func (t *noopTracer) CaptureStart(env *vm.EVM, from common.Address, to common.Address, create bool, input []byte, gas uint64, value *big.Int) {
 }

+// CaptureEnd is called after the call finishes to finalize the tracing.
 func (t *noopTracer) CaptureEnd(output []byte, gasUsed uint64, _ time.Duration, err error) {
 }

-func (t *noopTracer) CaptureState(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
+// CaptureState implements the EVMLogger interface to trace a single step of VM execution.
+func (t *noopTracer) CaptureState(pc uint64, op vm.OpCode, gas, cost uint64, scope *vm.ScopeContext, rData []byte, depth int, err error) {
 }

-func (t *noopTracer) CaptureFault(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost uint64, _ *vm.ScopeContext, depth int, err error) {
+// CaptureFault implements the EVMLogger interface to trace an execution fault.
+func (t *noopTracer) CaptureFault(pc uint64, op vm.OpCode, gas, cost uint64, _ *vm.ScopeContext, depth int, err error) {
 }

+// CaptureEnter is called when EVM enters a new scope (via call, create or selfdestruct).
 func (t *noopTracer) CaptureEnter(typ vm.OpCode, from common.Address, to common.Address, input []byte, gas uint64, value *big.Int) {
 }

+// CaptureExit is called when EVM exits a scope, even if the scope didn't
+// execute any code.
 func (t *noopTracer) CaptureExit(output []byte, gasUsed uint64, err error) {
 }

+// GetResult returns an empty json object.
 func (t *noopTracer) GetResult() (json.RawMessage, error) {
        return json.RawMessage(`{}`), nil
 }

+// Stop terminates execution of the tracer at the first opportune moment.
 func (t *noopTracer) Stop(err error) {
 }
diff --git a/eth/tracers/native/tracer.go b/eth/tracers/native/tracer.go
new file mode 100644
index 00000000..b60883ab
--- /dev/null
+++ b/eth/tracers/native/tracer.go
@@ -0,0 +1,89 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+/*
+Package native is a collection of tracers written in go.
+
+In order to add a native tracer and have it compiled into the binary, a new
+file needs to be added to this folder, containing an implementation of the
+`eth.tracers.Tracer` interface.
+
+Aside from implementing the tracer, it also needs to register itself, using the
+`register` method -- and this needs to be done in the package initialization.
+
+Example:
+
+```golang
+func init() {
+       register("noopTracerNative", newNoopTracer)
+}
+```
+*/
+package native
+
+import (
+       "errors"
+
+       "github.com/flare-foundation/coreth/eth/tracers"
+)
+
+// init registers itself this packages as a lookup for tracers.
+func init() {
+       tracers.RegisterLookup(false, lookup)
+}
+
+/*
+ctors is a map of package-local tracer constructors.
+
+We cannot be certain about the order of init-functions within a package,
+The go spec (https://golang.org/ref/spec#Package_initialization) says
+
+> To ensure reproducible initialization behavior, build systems
+> are encouraged to present multiple files belonging to the same
+> package in lexical file name order to a compiler.
+
+Hence, we cannot make the map in init, but must make it upon first use.
+*/
+var ctors map[string]func() tracers.Tracer
+
+// register is used by native tracers to register their presence.
+func register(name string, ctor func() tracers.Tracer) {
+       if ctors == nil {
+               ctors = make(map[string]func() tracers.Tracer)
+       }
+       ctors[name] = ctor
+}
+
+// lookup returns a tracer, if one can be matched to the given name.
+func lookup(name string, ctx *tracers.Context) (tracers.Tracer, error) {
```

```
+       if ctors == nil {
+               ctors = make(map[string]func() tracers.Tracer)
+       }
+       if ctor, ok := ctors[name]; ok {
+               return ctor(), nil
+       }
+       return nil, errors.New("no tracer found")
+}
diff --git a/eth/tracers/testing/calltrace_test.go b/eth/tracers/testing/calltrace_test.go
index 8cac229d..072f624c 100644
--- a/eth/tracers/testing/calltrace_test.go
+++ b/eth/tracers/testing/calltrace_test.go
@@ -10,19 +10,20 @@ import (
        "testing"
        "unicode"
 
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/eth/tracers"
-       "github.com/ava-labs/coreth/tests"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/rlp"
 
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/eth/tracers"
+       "github.com/flare-foundation/coreth/tests"
+
        // Force-load the native, to trigger registration
-       _ "github.com/ava-labs/coreth/eth/tracers/native"
+       _ "github.com/flare-foundation/coreth/eth/tracers/native"
 )
 
 type callContext struct {
@@ -106,7 +107,7 @@ func testCallTracer(tracerName string, dirPath string, t *testing.T) {
                                context = vm.BlockContext{
                                        CanTransfer: core.CanTransfer,
                                        Transfer:    core.Transfer,
-                                       Coinbase:    test.Context.Miner,
+                                       Coinbase:    common.HexToAddress("0x0100000000000000000000000000000000000000"),
                                        BlockNumber: new(big.Int).SetUint64(uint64(test.Context.Number)),
                                        Time:        new(big.Int).SetUint64(uint64(test.Context.Time)),
                                        Difficulty:  (*big.Int)(test.Context.Difficulty),
@@ -217,7 +218,7 @@ func benchTracer(tracerName string, test *callTracerTest, b *testing.B) {
        context := vm.BlockContext{
                CanTransfer: core.CanTransfer,
                Transfer:    core.Transfer,
-               Coinbase:    test.Context.Miner,
+               Coinbase:    common.HexToAddress("0x0100000000000000000000000000000000000000"),
                BlockNumber: new(big.Int).SetUint64(uint64(test.Context.Number)),
                Time:        new(big.Int).SetUint64(uint64(test.Context.Time)),
                Difficulty:  (*big.Int)(test.Context.Difficulty),
diff --git a/eth/tracers/tracer.go b/eth/tracers/tracer.go
index 416c085b..8b531633 100644
--- a/eth/tracers/tracer.go
+++ b/eth/tracers/tracer.go
@@ -35,13 +35,15 @@ import (
        "time"
        "unsafe"
 
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/vm"
+       "gopkg.in/olebedev/go-duktape.v3"
+
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
-       "gopkg.in/olebedev/go-duktape.v3"
+
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/vm"
 )
 
 // bigIntegerJS is the minified version of https://github.com/peterolson/BigInteger.js.
diff --git a/eth/tracers/tracer_test.go b/eth/tracers/tracer_test.go
index 72a62026..10c64b64 100644
--- a/eth/tracers/tracer_test.go
+++ b/eth/tracers/tracer_test.go
@@ -33,10 +33,11 @@ import (
        "testing"
        "time"
 
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/params"
 )
 
 type account struct{}
diff --git a/eth/tracers/tracers.go b/eth/tracers/tracers.go
index d90cec62..908084c0 100644
--- a/eth/tracers/tracers.go
+++ b/eth/tracers/tracers.go
@@ -1,13 +1,3 @@
-// (c) 2019-2020, Ava Labs, Inc.
-//
-// This file is a derived work, based on the go-ethereum library whose original
-// notices appear below.
-//
-// It is distributed under a license compatible with the licensing terms of the
-// original code from which it is derived.
-//
-// Much love to the original authors for their work.
-// **********
 // Copyright 2017 The go-ethereum Authors
 // This file is part of the go-ethereum library.
 //
@@ -24,19 +14,25 @@
 // You should have received a copy of the GNU Lesser General Public License
 // along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
 
-// Package tracers is a collection of JavaScript transaction tracers.package tracers
-
+// Package tracers is a manager for transaction tracing engines.
 package tracers
 
 import (
        "encoding/json"
-       "strings"
-       "unicode"
+       "errors"
 
-       "github.com/ava-labs/coreth/core/vm"
```

```
-	"github.com/ava-labs/coreth/eth/tracers/internal/tracers"
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/vm"
 )
 
+// Context contains some contextual infos for a transaction execution that is not
+// available from within the EVM object.
+type Context struct {
+	BlockHash common.Hash // Hash of the block the tx is contained within (zero if dangling tx or call)
+	TxIndex   int         // Index of the transaction within a block (zero if dangling tx or call)
+	TxHash    common.Hash // Hash of the transaction being traced (zero if dangling call)
+}
+
 // Tracer interface extends vm.EVMLogger and additionally
 // allows collecting the tracing result.
 type Tracer interface {
@@ -46,50 +42,31 @@ type Tracer interface {
 	Stop(err error)
 }
 
+type lookupFunc func(string, *Context) (Tracer, error)
+
 var (
-	nativeTracers map[string]func() Tracer = make(map[string]func() Tracer)
-	jsTracers                              = make(map[string]string)
+	lookups []lookupFunc
 )
 
-// RegisterNativeTracer makes native tracers which adhere
-// to the `Tracer` interface available to the rest of the codebase.
-// It is typically invoked in the `init()` function, e.g. see the `native/call.go`.
-func RegisterNativeTracer(name string, ctor func() Tracer) {
-	nativeTracers[name] = ctor
-}
-
-// New returns a new instance of a tracer,
-// 1. If 'code' is the name of a registered native tracer, then that tracer
-//   is instantiated and returned
-// 2. If 'code' is the name of a registered js-tracer, then that tracer is
-//   instantiated and returned
-// 3. Otherwise, the code is interpreted as the js code of a js-tracer, and
-//   is evaluated and returned.
-func New(code string, ctx *Context) (Tracer, error) {
-	// Resolve native tracer
-	if fn, ok := nativeTracers[code]; ok {
-		return fn(), nil
+// RegisterLookup registers a method as a lookup for tracers, meaning that
+// users can invoke a named tracer through that lookup. If 'wildcard' is true,
+// then the lookup will be placed last. This is typically meant for interpreted
+// engines (js) which can evaluate dynamic user-supplied code.
+func RegisterLookup(wildcard bool, lookup lookupFunc) {
+	if wildcard {
+		lookups = append(lookups, lookup)
+	} else {
+		lookups = append([]lookupFunc{lookup}, lookups...)
 	}
-	// Resolve js-tracers by name and assemble the tracer object
-	if tracer, ok := jsTracers[code]; ok {
-		code = tracer
-	}
-	return newJsTracer(code, ctx)
 }
 
-// camel converts a snake cased input string into a camel cased output.
-func camel(str string) string {
-	pieces := strings.Split(str, "_")
-	for i := 1; i < len(pieces); i++ {
-		pieces[i] = string(unicode.ToUpper(rune(pieces[i][0]))) + pieces[i][1:]
-	}
-	return strings.Join(pieces, "")
-}
-
-// init retrieves the JavaScript transaction tracers included in go-ethereum.
-func init() {
-	for _, file := range tracers.AssetNames() {
-		name := camel(strings.TrimSuffix(file, ".js"))
-		jsTracers[name] = string(tracers.MustAsset(file))
+// New returns a new instance of a tracer, by iterating through the
+// registered lookups.
+func New(code string, ctx *Context) (Tracer, error) {
+	for _, lookup := range lookups {
+		if tracer, err := lookup(code, ctx); err == nil {
+			return tracer, nil
+		}
 	}
+	return nil, errors.New("tracer not found")
 }
diff --git a/eth/tracers/tracers_test.go b/eth/tracers/tracers_test.go
index 3421d7c0..d6a2217d 100644
--- a/eth/tracers/tracers_test.go
+++ b/eth/tracers/tracers_test.go
@@ -27,74 +27,21 @@
 package tracers
 
 import (
-	"crypto/ecdsa"
-	"crypto/rand"
-	"encoding/json"
 	"math/big"
-	"reflect"
 	"testing"
 
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/tests"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/tracers/logger"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/tests"
 )
 
-// To generate a new callTracer test, copy paste the makeTest method below into
-// a Geth console and call it with a transaction hash you which to export.
-
-/*
-// makeTest generates a callTracer test by running a prestate reassembled and a
-// call trace run, assembling all the gathered information into a test case.
-var makeTest = function(tx, rewind) {
-  // Generate the genesis block from the block, transaction and prestate data
-  var block   = eth.getBlock(eth.getTransaction(tx).blockHash);
-  var genesis = eth.getBlock(block.parentHash);
-
-  delete genesis.gasUsed;
-  delete genesis.logsBloom;
-  delete genesis.parentHash;
-  delete genesis.receiptsRoot;
-  delete genesis.sha3Uncles;
-  delete genesis.size;
-  delete genesis.transactions;
-  delete genesis.transactionsRoot;
-  delete genesis.uncles;
-
-  genesis.gasLimit  = genesis.gasLimit.toString();
-  genesis.number    = genesis.number.toString();
-  genesis.timestamp = genesis.timestamp.toString();
-
-  genesis.alloc = debug.traceTransaction(tx, {tracer: "prestateTracer", rewind: rewind});
-  for (var key in genesis.alloc) {
-    genesis.alloc[key].nonce = genesis.alloc[key].nonce.toString();
-  }
-  genesis.config = admin.nodeInfo.protocols.eth.config;
-
-  // Generate the call trace and produce the test input
-  var result = debug.traceTransaction(tx, {tracer: "callTracer", rewind: rewind});
-  delete result.time;
-
-  console.log(JSON.stringify({
-    genesis: genesis,
-    context: {
-      number:     block.number.toString(),
-      difficulty: block.difficulty,
-      timestamp:  block.timestamp.toString(),
-      gasLimit:   block.gasLimit.toString(),
-      miner:      block.miner,
-    },
-    input:  eth.getRawTransaction(tx),
-    result: result,
-  }, null, 2));
-}
-*/
-
 // callTrace is the result of a callTracer run.
 type callTrace struct {
        Type    string          `json:"type"`
@@ -109,184 +56,6 @@ type callTrace struct {
        Calls   []callTrace     `json:"calls,omitempty"`
 }

-// TestZeroValueToNotExitCall tests the calltracer(s) on the following:
-// Tx to A, A calls B with zero value. B does not already exist.
-// Expected: that enter/exit is invoked and the inner call is shown in the result
-func TestZeroValueToNotExitCall(t *testing.T) {
-       var to = common.HexToAddress("0x00000000000000000000000000000000deadbeef")
-       privkey, err := crypto.HexToECDSA("0000000000000000deadbeef00000000000000000000000000000000deadbeef")
-       if err != nil {
-               t.Fatalf("err %v", err)
-       }
-       signer := types.NewEIP155Signer(big.NewInt(1))
-       tx, err := types.SignNewTx(privkey, signer, &types.LegacyTx{
-               GasPrice: big.NewInt(0),
-               Gas:      50000,
-               To:       &to,
-       })
-       if err != nil {
-               t.Fatalf("err %v", err)
-       }
-       origin, _ := signer.Sender(tx)
-       txContext := vm.TxContext{
-               Origin:   origin,
-               GasPrice: big.NewInt(1),
-       }
-       context := vm.BlockContext{
-               CanTransfer: core.CanTransfer,
-               Transfer:    core.Transfer,
-               Coinbase:    common.Address{},
-               BlockNumber: new(big.Int).SetUint64(8000000),
-               Time:        new(big.Int).SetUint64(5),
-               Difficulty:  big.NewInt(0x30000),
-               GasLimit:    uint64(6000000),
-       }
-       var code = []byte{
-               byte(vm.PUSH1), 0x0, byte(vm.DUP1), byte(vm.DUP1), byte(vm.DUP1), // in and outs zero
-               byte(vm.DUP1), byte(vm.PUSH1), 0xff, byte(vm.GAS), // value=0,address=0xff, gas=GAS
-               byte(vm.CALL),
-       }
-       var alloc = core.GenesisAlloc{
-               to: core.GenesisAccount{
-                       Nonce: 1,
-                       Code:  code,
-               },
-               origin: core.GenesisAccount{
-                       Nonce:   0,
-                       Balance: big.NewInt(500000000000000),
-               },
-       }
-       _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), alloc, false)
-       // Create the tracer, the EVM environment and run it
-       tracer, err := New("callTracerJs", new(Context))
-       if err != nil {
-               t.Fatalf("failed to create call tracer: %v", err)
-       }
-       evm := vm.NewEVM(context, txContext, statedb, params.AvalancheMainnetChainConfig, vm.Config{Debug: true, Tracer: tracer})
-       msg, err := tx.AsMessage(signer, nil)
-       if err != nil {
-               t.Fatalf("failed to prepare transaction for tracing: %v", err)
-       }
-       st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
-       if _, err = st.TransitionDb(); err != nil {
-               t.Fatalf("failed to execute transaction: %v", err)
-       }
-       // Retrieve the trace result and compare against the etalon
-       res, err := tracer.GetResult()
-       if err != nil {
-               t.Fatalf("failed to retrieve trace result: %v", err)
-       }
-       have := new(callTrace)
-       if err := json.Unmarshal(res, have); err != nil {
-               t.Fatalf("failed to unmarshal trace result: %v", err)
-       }
-       wantStr := `{"type":"CALL","from":"0x682a80a6f560eec50d54e63cbeda1c324c5f8d1b","to":"0x00000000000000000000000000000000deadbeef","value":"0x0","gas":"0x7148","gasUsed":"0x2d0","input":"0x","outpu
-       want := new(callTrace)
-       json.Unmarshal([]byte(wantStr), want)
-       if !jsonEqual(have, want) {
-               t.Error("have != want")
-       }
-}
-
-func TestPrestateTracerCreate2(t *testing.T) {
-       unsignedTx := types.NewTransaction(1, common.HexToAddress("0x00000000000000000000000000000000deadbeef"),
-               new(big.Int), 5000000, big.NewInt(1), []byte{})
-
-       privateKeyECDSA, err := ecdsa.GenerateKey(crypto.S256(), rand.Reader)
-       if err != nil {
-               t.Fatalf("err %v", err)
-       }
-       signer := types.NewEIP155Signer(big.NewInt(1))
-       tx, err := types.SignTx(unsignedTx, signer, privateKeyECDSA)
-       if err != nil {
-               t.Fatalf("err %v", err)
```

```diff
-       }
-       /**
-               This comes from one of the test-vectors on the Skinny Create2 - EIP
-
-           address 0x00000000000000000000000000000000deadbeef
-           salt 0x00000000000000000000000000000000000000000000000000000000cafebabe
-           init_code 0xdeadbeef
-           gas (assuming no mem expansion): 32006
-           result: 0x60f3f640a8508fC6a86d45DF051962668E1e8AC7
-       */
-       origin, _ := signer.Sender(tx)
-       txContext := vm.TxContext{
-               Origin:   origin,
-               GasPrice: big.NewInt(1),
-       }
-       context := vm.BlockContext{
-               CanTransfer: core.CanTransfer,
-               Transfer:    core.Transfer,
-               Coinbase:    common.Address{},
-               BlockNumber: new(big.Int).SetUint64(8000000),
-               Time:        new(big.Int).SetUint64(5),
-               Difficulty:  big.NewInt(0x30000),
-               GasLimit:    uint64(6000000),
-       }
-       alloc := core.GenesisAlloc{}
-
-       // The code pushes 'deadbeef' into memory, then the other params, and calls CREATE2, then returns
-       // the address
-       alloc[common.HexToAddress("0x00000000000000000000000000000000deadbeef")] = core.GenesisAccount{
-               Nonce:   1,
-               Code:    hexutil.MustDecode("0x63deadbeef60005263cafebabe6004601c6000F560005260206000F3"),
-               Balance: big.NewInt(1),
-       }
-       alloc[origin] = core.GenesisAccount{
-               Nonce:   1,
-               Code:    []byte{},
-               Balance: big.NewInt(500000000000000),
-       }
-       _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), alloc, false)
-
-       // Create the tracer, the EVM environment and run it
-       tracer, err := New("prestateTracer", new(Context))
-       if err != nil {
-               t.Fatalf("failed to create call tracer: %v", err)
-       }
-       evm := vm.NewEVM(context, txContext, statedb, params.AvalancheMainnetChainConfig, vm.Config{Debug: true, Tracer: tracer})
-
-       msg, err := tx.AsMessage(signer, nil)
-       if err != nil {
-               t.Fatalf("failed to prepare transaction for tracing: %v", err)
-       }
-       st := core.NewStateTransition(evm, msg, new(core.GasPool).AddGas(tx.Gas()))
-       if _, err = st.TransitionDb(); err != nil {
-               t.Fatalf("failed to execute transaction: %v", err)
-       }
-       // Retrieve the trace result and compare against the etalon
-       res, err := tracer.GetResult()
-       if err != nil {
-               t.Fatalf("failed to retrieve trace result: %v", err)
-       }
-       ret := make(map[string]interface{})
-       if err := json.Unmarshal(res, &ret); err != nil {
-               t.Fatalf("failed to unmarshal trace result: %v", err)
-       }
-       if _, has := ret["0x60f3f640a8508fc6a86d45df051962668e1e8ac7"]; !has {
-               t.Fatalf("Expected 0x60f3f640a8508fc6a86d45df051962668e1e8ac7 in result")
-       }
-}
-
-// jsonEqual is similar to reflect.DeepEqual, but does a 'bounce' via json prior to
-// comparison
-func jsonEqual(x, y interface{}) bool {
-       xTrace := new(callTrace)
-       yTrace := new(callTrace)
-       if xj, err := json.Marshal(x); err == nil {
-               json.Unmarshal(xj, xTrace)
-       } else {
-               return false
-       }
-       if yj, err := json.Marshal(y); err == nil {
-               json.Unmarshal(yj, yTrace)
-       } else {
-               return false
-       }
-       return reflect.DeepEqual(xTrace, yTrace)
-}
-
 func BenchmarkTransactionTrace(b *testing.B) {
        key, _ := crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f291")
        from := crypto.PubkeyToAddress(key.PublicKey)
@@ -337,7 +106,7 @@ func BenchmarkTransactionTrace(b *testing.B) {
        }
        _, statedb := tests.MakePreState(rawdb.NewMemoryDatabase(), alloc, false)
        // Create the tracer, the EVM environment and run it
-       tracer := vm.NewStructLogger(&vm.LogConfig{
+       tracer := logger.NewStructLogger(&logger.Config{
                Debug: false,
                //DisableStorage: true,
                //EnableMemory: false,
diff --git a/ethclient/client_interface_test.go b/ethclient/client_interface_test.go
new file mode 100644
index 00000000..80783bcc
--- /dev/null
+++ b/ethclient/client_interface_test.go
@@ -0,0 +1,17 @@
+package ethclient
+
+import (
+       "reflect"
+       "testing"
+)
+
+func TestInterfaceStructOneToOne(t *testing.T) {
+       // checks struct provides at least the methods signatures in the interface
+       var _ Client = (*client)(nil)
+       // checks interface and struct have the same number of methods
+       clientType := reflect.TypeOf(&client{})
+       ClientType := reflect.TypeOf((*Client)(nil)).Elem()
+       if clientType.NumMethod() != ClientType.NumMethod() {
+               t.Fatalf("no 1 to 1 compliance between struct methods (%v) and interface methods (%v)", clientType.NumMethod(), ClientType.NumMethod())
+       }
+}
diff --git a/ethclient/corethclient/corethclient.go b/ethclient/corethclient/corethclient.go
index 8d7654fc..a0881dd5 100644
--- a/ethclient/corethclient/corethclient.go
+++ b/ethclient/corethclient/corethclient.go
@@ -33,12 +33,12 @@ import (
        "runtime"
        "runtime/debug"
 
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/ethclient"
-       "github.com/ava-labs/coreth/interfaces"
-       "github.com/ava-labs/coreth/rpc"
```

```
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/common/hexutil"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/ethclient"
+	"github.com/flare-foundation/coreth/interfaces"
+	"github.com/flare-foundation/coreth/rpc"
 )

 // Client is a wrapper around rpc.Client that implements geth-specific functionality.
diff --git a/ethclient/ethclient.go b/ethclient/ethclient.go
index 81d988e2..b0d6e182 100644
--- a/ethclient/ethclient.go
+++ b/ethclient/ethclient.go
@@ -34,46 +34,84 @@ import (
	"fmt"
	"math/big"

-	"github.com/ava-labs/avalanchego/ids"
-	"github.com/ava-labs/coreth/accounts/abi/bind"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/interfaces"
-	"github.com/ava-labs/coreth/rpc"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/common/hexutil"
+	"github.com/flare-foundation/coreth/accounts/abi/bind"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/interfaces"
+	"github.com/flare-foundation/coreth/rpc"
+	"github.com/flare-foundation/flare/ids"
 )

 // Verify that Client implements required interfaces
 var (
-	_ bind.AcceptedContractCaller = (*Client)(nil)
-	_ bind.ContractBackend        = (*Client)(nil)
-	_ bind.ContractFilterer       = (*Client)(nil)
-	_ bind.ContractTransactor     = (*Client)(nil)
-	_ bind.DeployBackend          = (*Client)(nil)
-
-	_ interfaces.ChainReader            = (*Client)(nil)
-	_ interfaces.ChainStateReader       = (*Client)(nil)
-	_ interfaces.TransactionReader      = (*Client)(nil)
-	_ interfaces.TransactionSender      = (*Client)(nil)
-	_ interfaces.ContractCaller         = (*Client)(nil)
-	_ interfaces.GasEstimator           = (*Client)(nil)
-	_ interfaces.GasPricer              = (*Client)(nil)
-	_ interfaces.LogFilterer            = (*Client)(nil)
-	_ interfaces.AcceptedStateReader    = (*Client)(nil)
-	_ interfaces.AcceptedContractCaller = (*Client)(nil)
+	_ bind.AcceptedContractCaller = (*client)(nil)
+	_ bind.ContractBackend        = (*client)(nil)
+	_ bind.ContractFilterer       = (*client)(nil)
+	_ bind.ContractTransactor     = (*client)(nil)
+	_ bind.DeployBackend          = (*client)(nil)
+
+	_ interfaces.ChainReader            = (*client)(nil)
+	_ interfaces.ChainStateReader       = (*client)(nil)
+	_ interfaces.TransactionReader      = (*client)(nil)
+	_ interfaces.TransactionSender      = (*client)(nil)
+	_ interfaces.ContractCaller         = (*client)(nil)
+	_ interfaces.GasEstimator           = (*client)(nil)
+	_ interfaces.GasPricer              = (*client)(nil)
+	_ interfaces.LogFilterer            = (*client)(nil)
+	_ interfaces.AcceptedStateReader    = (*client)(nil)
+	_ interfaces.AcceptedContractCaller = (*client)(nil)
+
+	_ Client = (*client)(nil)
 )

-// Client defines typed wrappers for the Ethereum RPC API.
-type Client struct {
+// Client defines interface for typed wrappers for the Ethereum RPC API.
+type Client interface {
+	Close()
+	ChainID(context.Context) (*big.Int, error)
+	BlockByHash(context.Context, common.Hash) (*types.Block, error)
+	BlockByNumber(context.Context, *big.Int) (*types.Block, error)
+	BlockNumber(context.Context) (uint64, error)
+	HeaderByHash(context.Context, common.Hash) (*types.Header, error)
+	HeaderByNumber(context.Context, *big.Int) (*types.Header, error)
+	TransactionByHash(context.Context, common.Hash) (tx *types.Transaction, isPending bool, err error)
+	TransactionSender(context.Context, *types.Transaction, common.Hash, uint) (common.Address, error)
+	TransactionCount(context.Context, common.Hash) (uint, error)
+	TransactionInBlock(context.Context, common.Hash, uint) (*types.Transaction, error)
+	TransactionReceipt(context.Context, common.Hash) (*types.Receipt, error)
+	SubscribeNewAcceptedTransactions(context.Context, chan<- *common.Hash) (interfaces.Subscription, error)
+	SubscribeNewPendingTransactions(context.Context, chan<- *common.Hash) (interfaces.Subscription, error)
+	SubscribeNewHead(context.Context, chan<- *types.Header) (interfaces.Subscription, error)
+	NetworkID(context.Context) (*big.Int, error)
+	BalanceAt(context.Context, common.Address, *big.Int) (*big.Int, error)
+	AssetBalanceAt(context.Context, common.Address, ids.ID, *big.Int) (*big.Int, error)
+	StorageAt(context.Context, common.Address, common.Hash, *big.Int) ([]byte, error)
+	CodeAt(context.Context, common.Address, *big.Int) ([]byte, error)
+	NonceAt(context.Context, common.Address, *big.Int) (uint64, error)
+	FilterLogs(context.Context, interfaces.FilterQuery) ([]types.Log, error)
+	SubscribeFilterLogs(context.Context, interfaces.FilterQuery, chan<- types.Log) (interfaces.Subscription, error)
+	AcceptedCodeAt(context.Context, common.Address) ([]byte, error)
+	AcceptedNonceAt(context.Context, common.Address) (uint64, error)
+	AcceptedCallContract(context.Context, interfaces.CallMsg) ([]byte, error)
+	CallContract(context.Context, interfaces.CallMsg, *big.Int) ([]byte, error)
+	SuggestGasPrice(context.Context) (*big.Int, error)
+	SuggestGasTipCap(context.Context) (*big.Int, error)
+	EstimateGas(context.Context, interfaces.CallMsg) (uint64, error)
+	EstimateBaseFee(context.Context) (*big.Int, error)
+	SendTransaction(context.Context, *types.Transaction) error
+}
+
+// client defines implementation for typed wrappers for the Ethereum RPC API.
+type client struct {
	c *rpc.Client
 }

 // Dial connects a client to the given URL.
-func Dial(rawurl string) (*Client, error) {
+func Dial(rawurl string) (Client, error) {
	return DialContext(context.Background(), rawurl)
 }

-func DialContext(ctx context.Context, rawurl string) (*Client, error) {
+func DialContext(ctx context.Context, rawurl string) (Client, error) {
	c, err := rpc.DialContext(ctx, rawurl)
	if err != nil {
		return nil, err
@@ -82,18 +120,18 @@ func DialContext(ctx context.Context, rawurl string) (*Client, error) {
 }

 // NewClient creates a client that uses the given RPC client.
-func NewClient(c *rpc.Client) *Client {
-	return &Client{c}
+func NewClient(c *rpc.Client) Client {
+	return &client{c}
 }

-func (ec *Client) Close() {
```

```diff
+func (ec *client) Close() {
        ec.c.Close()
 }
 
 // Blockchain Access
 
 // ChainID retrieves the current chain ID for transaction replay protection.
-func (ec *Client) ChainID(ctx context.Context) (*big.Int, error) {
+func (ec *client) ChainID(ctx context.Context) (*big.Int, error) {
        var result hexutil.Big
        err := ec.c.CallContext(ctx, &result, "eth_chainId")
        if err != nil {
@@ -106,7 +144,7 @@ func (ec *Client) ChainID(ctx context.Context) (*big.Int, error) {
 //
 // Note that loading full blocks requires two requests. Use HeaderByHash
 // if you don't need all transactions or uncle headers.
-func (ec *Client) BlockByHash(ctx context.Context, hash common.Hash) (*types.Block, error) {
+func (ec *client) BlockByHash(ctx context.Context, hash common.Hash) (*types.Block, error) {
        return ec.getBlock(ctx, "eth_getBlockByHash", hash, true)
 }
 
@@ -115,12 +153,12 @@ func (ec *Client) BlockByHash(ctx context.Context, hash common.Hash) (*types.Blo
 //
 // Note that loading full blocks requires two requests. Use HeaderByNumber
 // if you don't need all transactions or uncle headers.
-func (ec *Client) BlockByNumber(ctx context.Context, number *big.Int) (*types.Block, error) {
+func (ec *client) BlockByNumber(ctx context.Context, number *big.Int) (*types.Block, error) {
        return ec.getBlock(ctx, "eth_getBlockByNumber", ToBlockNumArg(number), true)
 }
 
 // BlockNumber returns the most recent block number
-func (ec *Client) BlockNumber(ctx context.Context) (uint64, error) {
+func (ec *client) BlockNumber(ctx context.Context) (uint64, error) {
        var result hexutil.Uint64
        err := ec.c.CallContext(ctx, &result, "eth_blockNumber")
        return uint64(result), err
@@ -134,7 +172,7 @@ type rpcBlock struct {
        BlockExtraData *hexutil.Bytes   `json:"blockExtraData"`
 }
 
-func (ec *Client) getBlock(ctx context.Context, method string, args ...interface{}) (*types.Block, error) {
+func (ec *client) getBlock(ctx context.Context, method string, args ...interface{}) (*types.Block, error) {
        var raw json.RawMessage
        err := ec.c.CallContext(ctx, &raw, method, args...)
        if err != nil {
@@ -200,7 +238,7 @@ func (ec *Client) getBlock(ctx context.Context, method string, args ...interface
 }
 
 // HeaderByHash returns the block header with the given hash.
-func (ec *Client) HeaderByHash(ctx context.Context, hash common.Hash) (*types.Header, error) {
+func (ec *client) HeaderByHash(ctx context.Context, hash common.Hash) (*types.Header, error) {
        var head *types.Header
        err := ec.c.CallContext(ctx, &head, "eth_getBlockByHash", hash, false)
        if err == nil && head == nil {
@@ -211,7 +249,7 @@ func (ec *Client) HeaderByHash(ctx context.Context, hash common.Hash) (*types.He
 
 // HeaderByNumber returns a block header from the current canonical chain. If number is
 // nil, the latest known header is returned.
-func (ec *Client) HeaderByNumber(ctx context.Context, number *big.Int) (*types.Header, error) {
+func (ec *client) HeaderByNumber(ctx context.Context, number *big.Int) (*types.Header, error) {
        var head *types.Header
        err := ec.c.CallContext(ctx, &head, "eth_getBlockByNumber", ToBlockNumArg(number), false)
        if err == nil && head == nil {
@@ -239,7 +277,7 @@ func (tx *rpcTransaction) UnmarshalJSON(msg []byte) error {
 }
 
 // TransactionByHash returns the transaction with the given hash.
-func (ec *Client) TransactionByHash(ctx context.Context, hash common.Hash) (tx *types.Transaction, isPending bool, err error) {
+func (ec *client) TransactionByHash(ctx context.Context, hash common.Hash) (tx *types.Transaction, isPending bool, err error) {
        var json *rpcTransaction
        err = ec.c.CallContext(ctx, &json, "eth_getTransactionByHash", hash)
        if err != nil {
@@ -261,12 +299,14 @@ func (ec *Client) TransactionByHash(ctx context.Context, hash common.Hash) (tx *
 //
 // There is a fast-path for transactions retrieved by TransactionByHash and
 // TransactionInBlock. Getting their sender address can be done without an RPC interaction.
-func (ec *Client) TransactionSender(ctx context.Context, tx *types.Transaction, block common.Hash, index uint) (common.Address, error) {
+func (ec *client) TransactionSender(ctx context.Context, tx *types.Transaction, block common.Hash, index uint) (common.Address, error) {
        // Try to load the address from the cache.
        sender, err := types.Sender(&senderFromServer{blockhash: block}, tx)
        if err == nil {
                return sender, nil
        }
+
+       // It was not found in cache, ask the server.
        var meta struct {
                Hash common.Hash
                From common.Address
@@ -281,14 +321,14 @@ func (ec *Client) TransactionSender(ctx context.Context, tx *types.Transaction,
 }
 
 // TransactionCount returns the total number of transactions in the given block.
-func (ec *Client) TransactionCount(ctx context.Context, blockHash common.Hash) (uint, error) {
+func (ec *client) TransactionCount(ctx context.Context, blockHash common.Hash) (uint, error) {
        var num hexutil.Uint
        err := ec.c.CallContext(ctx, &num, "eth_getBlockTransactionCountByHash", blockHash)
        return uint(num), err
 }
 
 // TransactionInBlock returns a single transaction at index in the given block.
-func (ec *Client) TransactionInBlock(ctx context.Context, blockHash common.Hash, index uint) (*types.Transaction, error) {
+func (ec *client) TransactionInBlock(ctx context.Context, blockHash common.Hash, index uint) (*types.Transaction, error) {
        var json *rpcTransaction
        err := ec.c.CallContext(ctx, &json, "eth_getTransactionByBlockHashAndIndex", blockHash, hexutil.Uint64(index))
        if err != nil {
@@ -307,7 +347,7 @@ func (ec *Client) TransactionInBlock(ctx context.Context, blockHash common.Hash,
 
 // TransactionReceipt returns the receipt of a transaction by transaction hash.
 // Note that the receipt is not available for pending transactions.
-func (ec *Client) TransactionReceipt(ctx context.Context, txHash common.Hash) (*types.Receipt, error) {
+func (ec *client) TransactionReceipt(ctx context.Context, txHash common.Hash) (*types.Receipt, error) {
        var r *types.Receipt
        err := ec.c.CallContext(ctx, &r, "eth_getTransactionReceipt", txHash)
        if err == nil {
@@ -319,25 +359,25 @@ func (ec *Client) TransactionReceipt(ctx context.Context, txHash common.Hash) (*
 }
 
 // SubscribeNewAcceptedTransactions subscribes to notifications about the accepted transaction hashes on the given channel.
-func (ec *Client) SubscribeNewAcceptedTransactions(ctx context.Context, ch chan<- *common.Hash) (interfaces.Subscription, error) {
+func (ec *client) SubscribeNewAcceptedTransactions(ctx context.Context, ch chan<- *common.Hash) (interfaces.Subscription, error) {
        return ec.c.EthSubscribe(ctx, ch, "newAcceptedTransactions")
 }
 
 // SubscribeNewAcceptedTransactions subscribes to notifications about the accepted transaction hashes on the given channel.
-func (ec *Client) SubscribeNewPendingTransactions(ctx context.Context, ch chan<- *common.Hash) (interfaces.Subscription, error) {
+func (ec *client) SubscribeNewPendingTransactions(ctx context.Context, ch chan<- *common.Hash) (interfaces.Subscription, error) {
        return ec.c.EthSubscribe(ctx, ch, "newPendingTransactions")
 }
 
 // SubscribeNewHead subscribes to notifications about the current blockchain head
 // on the given channel.
-func (ec *Client) SubscribeNewHead(ctx context.Context, ch chan<- *types.Header) (interfaces.Subscription, error) {
+func (ec *client) SubscribeNewHead(ctx context.Context, ch chan<- *types.Header) (interfaces.Subscription, error) {
        return ec.c.EthSubscribe(ctx, ch, "newHeads")
 }
 
 // State Access
 
 // NetworkID returns the network ID (also known as the chain ID) for this chain.
-func (ec *Client) NetworkID(ctx context.Context) (*big.Int, error) {
+func (ec *client) NetworkID(ctx context.Context) (*big.Int, error) {
        version := new(big.Int)
        var ver string
        if err := ec.c.CallContext(ctx, &ver, "net_version"); err != nil {
@@ -351,7 +391,7 @@ func (ec *Client) NetworkID(ctx context.Context) (*big.Int, error) {
 
 // BalanceAt returns the wei balance of the given account.
 // The block number can be nil, in which case the balance is taken from the latest known block.
-func (ec *Client) BalanceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (*big.Int, error) {
+func (ec *client) BalanceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (*big.Int, error) {
        var result hexutil.Big
        err := ec.c.CallContext(ctx, &result, "eth_getBalance", account, ToBlockNumArg(blockNumber))
        return (*big.Int)(&result), err
@@ -359,7 +399,7 @@ func (ec *Client) BalanceAt(ctx context.Context, account common.Address, blockNu
 
 // AssetBalanceAt returns the [assetID] balance of the given account
 // The block number can be nil, in which case the balance is taken from the latest known block.
-func (ec *Client) AssetBalanceAt(ctx context.Context, account common.Address, assetID ids.ID, blockNumber *big.Int) (*big.Int, error) {
+func (ec *client) AssetBalanceAt(ctx context.Context, account common.Address, assetID ids.ID, blockNumber *big.Int) (*big.Int, error) {
        var result hexutil.Big
        err := ec.c.CallContext(ctx, &result, "eth_getAssetBalance", account, ToBlockNumArg(blockNumber), assetID)
        return (*big.Int)(&result), err
@@ -367,7 +407,7 @@ func (ec *Client) AssetBalanceAt(ctx context.Context, account common.Address, as
 
 // StorageAt returns the value of key in the contract storage of the given account.
 // The block number can be nil, in which case the value is taken from the latest known block.
-func (ec *Client) StorageAt(ctx context.Context, account common.Address, key common.Hash, blockNumber *big.Int) ([]byte, error) {
+func (ec *client) StorageAt(ctx context.Context, account common.Address, key common.Hash, blockNumber *big.Int) ([]byte, error) {
        var result hexutil.Bytes
        err := ec.c.CallContext(ctx, &result, "eth_getStorageAt", account, key, ToBlockNumArg(blockNumber))
        return result, err
@@ -375,7 +415,7 @@ func (ec *Client) StorageAt(ctx context.Context, account common.Address, key com
 
 // CodeAt returns the contract code of the given account.
 // The block number can be nil, in which case the code is taken from the latest known block.
-func (ec *Client) CodeAt(ctx context.Context, account common.Address, blockNumber *big.Int) ([]byte, error) {
+func (ec *client) CodeAt(ctx context.Context, account common.Address, blockNumber *big.Int) ([]byte, error) {
        var result hexutil.Bytes
        err := ec.c.CallContext(ctx, &result, "eth_getCode", account, ToBlockNumArg(blockNumber))
        return result, err
@@ -383,7 +423,7 @@ func (ec *Client) CodeAt(ctx context.Context, account common.Address, blockNumbe
 
 // NonceAt returns the account nonce of the given account.
 // The block number can be nil, in which case the nonce is taken from the latest known block.
-func (ec *Client) NonceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (uint64, error) {
+func (ec *client) NonceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (uint64, error) {
        var result hexutil.Uint64
        err := ec.c.CallContext(ctx, &result, "eth_getTransactionCount", account, ToBlockNumArg(blockNumber))
        return uint64(result), err
@@ -392,7 +432,7 @@ func (ec *Client) NonceAt(ctx context.Context, account common.Address, blockNumb
 // Filters
 
 // FilterLogs executes a filter query.
-func (ec *Client) FilterLogs(ctx context.Context, q interfaces.FilterQuery) ([]types.Log, error) {
+func (ec *client) FilterLogs(ctx context.Context, q interfaces.FilterQuery) ([]types.Log, error) {
        var result []types.Log
        arg, err := toFilterArg(q)
        if err != nil {
@@ -403,7 +443,7 @@ func (ec *Client) FilterLogs(ctx context.Context, q interfaces.FilterQuery) ([]t
 }
 
 // SubscribeFilterLogs subscribes to the results of a streaming filter query.
-func (ec *Client) SubscribeFilterLogs(ctx context.Context, q interfaces.FilterQuery, ch chan<- types.Log) (interfaces.Subscription, error) {
+func (ec *client) SubscribeFilterLogs(ctx context.Context, q interfaces.FilterQuery, ch chan<- types.Log) (interfaces.Subscription, error) {
        arg, err := toFilterArg(q)
        if err != nil {
                return nil, err
@@ -433,19 +473,19 @@ func toFilterArg(q interfaces.FilterQuery) (interface{}, error) {
 }
 
 // AcceptedCodeAt returns the contract code of the given account in the accepted state.
-func (ec *Client) AcceptedCodeAt(ctx context.Context, account common.Address) ([]byte, error) {
+func (ec *client) AcceptedCodeAt(ctx context.Context, account common.Address) ([]byte, error) {
        return ec.CodeAt(ctx, account, nil)
 }
 
 // AcceptedNonceAt returns the account nonce of the given account in the accepted state.
 // This is the nonce that should be used for the next transaction.
-func (ec *Client) AcceptedNonceAt(ctx context.Context, account common.Address) (uint64, error) {
+func (ec *client) AcceptedNonceAt(ctx context.Context, account common.Address) (uint64, error) {
        return ec.NonceAt(ctx, account, nil)
 }
 
 // AcceptedCallContract executes a message call transaction in the accepted
 // state.
-func (ec *Client) AcceptedCallContract(ctx context.Context, msg interfaces.CallMsg) ([]byte, error) {
+func (ec *client) AcceptedCallContract(ctx context.Context, msg interfaces.CallMsg) ([]byte, error) {
        return ec.CallContract(ctx, msg, nil)
 }
 
@@ -457,7 +497,7 @@ func (ec *Client) AcceptedCallContract(ctx context.Context, msg interfaces.CallM
 // blockNumber selects the block height at which the call runs. It can be nil, in which
 // case the code is taken from the latest known block. Note that state from very old
 // blocks might not be available.
-func (ec *Client) CallContract(ctx context.Context, msg interfaces.CallMsg, blockNumber *big.Int) ([]byte, error) {
+func (ec *client) CallContract(ctx context.Context, msg interfaces.CallMsg, blockNumber *big.Int) ([]byte, error) {
        var hex hexutil.Bytes
        err := ec.c.CallContext(ctx, &hex, "eth_call", toCallArg(msg), ToBlockNumArg(blockNumber))
        if err != nil {
@@ -468,7 +508,7 @@ func (ec *Client) CallContract(ctx context.Context, msg interfaces.CallMsg, bloc
 
 // SuggestGasPrice retrieves the currently suggested gas price to allow a timely
 // execution of a transaction.
-func (ec *Client) SuggestGasPrice(ctx context.Context) (*big.Int, error) {
+func (ec *client) SuggestGasPrice(ctx context.Context) (*big.Int, error) {
        var hex hexutil.Big
        if err := ec.c.CallContext(ctx, &hex, "eth_gasPrice"); err != nil {
                return nil, err
@@ -478,7 +518,7 @@ func (ec *Client) SuggestGasPrice(ctx context.Context) (*big.Int, error) {
 
 // SuggestGasTipCap retrieves the currently suggested gas tip cap after 1559 to
 // allow a timely execution of a transaction.
-func (ec *Client) SuggestGasTipCap(ctx context.Context) (*big.Int, error) {
+func (ec *client) SuggestGasTipCap(ctx context.Context) (*big.Int, error) {
        var hex hexutil.Big
        if err := ec.c.CallContext(ctx, &hex, "eth_maxPriorityFeePerGas"); err != nil {
                return nil, err
@@ -490,7 +530,7 @@ func (ec *Client) SuggestGasTipCap(ctx context.Context) (*big.Int, error) {
 // the current pending state of the backend blockchain. There is no guarantee that this is
 // the true gas limit requirement as other transactions may be added or removed by miners,
 // but it should provide a basis for setting a reasonable default.
-func (ec *Client) EstimateGas(ctx context.Context, msg interfaces.CallMsg) (uint64, error) {
+func (ec *client) EstimateGas(ctx context.Context, msg interfaces.CallMsg) (uint64, error) {
        var hex hexutil.Uint64
        err := ec.c.CallContext(ctx, &hex, "eth_estimateGas", toCallArg(msg))
        if err != nil {
@@ -502,7 +542,7 @@ func (ec *Client) EstimateGas(ctx context.Context, msg interfaces.CallMsg) (uint
 // EstimateBaseFee tries to estimate the base fee for the next block if it were created
 // immediately. There is no guarantee that this will be the base fee used in the next block
 // or that the next base fee will be higher or lower than the returned value.
-func (ec *Client) EstimateBaseFee(ctx context.Context) (*big.Int, error) {
+func (ec *client) EstimateBaseFee(ctx context.Context) (*big.Int, error) {
        var hex hexutil.Big
        err := ec.c.CallContext(ctx, &hex, "eth_baseFee")
        if err != nil {
@@ -515,7 +555,7 @@ func (ec *Client) EstimateBaseFee(ctx context.Context) (*big.Int, error) {
 //
 // If the transaction was a contract creation use the TransactionReceipt method to get the
 // contract address after the transaction has been mined.
-func (ec *Client) SendTransaction(ctx context.Context, tx *types.Transaction) error {
+func (ec *client) SendTransaction(ctx context.Context, tx *types.Transaction) error {
        data, err := tx.MarshalBinary()
        if err != nil {
                return err
diff --git a/ethclient/signer.go b/ethclient/signer.go
index dafa943b..43baf1bc 100644
--- a/ethclient/signer.go
+++ b/ethclient/signer.go
@@ -30,8 +30,8 @@ import (
        "errors"
        "math/big"
 
-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/core/types"
 )
 
 // senderFromServer is a types.Signer that remembers the sender address returned by the RPC
@@ -55,7 +55,7 @@ func (s *senderFromServer) Equal(other types.Signer) bool {
 }
 
 func (s *senderFromServer) Sender(tx *types.Transaction) (common.Address, error) {
-       if s.blockhash == (common.Hash{}) {
+       if s.addr == (common.Address{}) {
                return common.Address{}, errNotCached
        }
        return s.addr, nil
diff --git a/ethdb/dbtest/testsuite.go b/ethdb/dbtest/testsuite.go
index 90c92ee3..2209bf79 100644
--- a/ethdb/dbtest/testsuite.go
+++ b/ethdb/dbtest/testsuite.go
@@ -32,7 +32,7 @@ import (
        "sort"
        "testing"
 
-       "github.com/ava-labs/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb"
 )
 
 // TestDatabaseSuite runs a suite of tests against a KeyValueStore database
diff --git a/ethdb/leveldb/leveldb.go b/ethdb/leveldb/leveldb.go
new file mode 100644
index 00000000..1bb02f19
--- /dev/null
+++ b/ethdb/leveldb/leveldb.go
@@ -0,0 +1,531 @@
+// (c) 2021-2022, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2018 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+//go:build !js
+// +build !js
+
+// Package leveldb implements the key-value database layer based on LevelDB.
+package leveldb
+
+import (
+       "fmt"
+       "strconv"
+       "strings"
+       "sync"
+       "time"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/ethereum/go-ethereum/metrics"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/syndtr/goleveldb/leveldb"
+       "github.com/syndtr/goleveldb/leveldb/errors"
+       "github.com/syndtr/goleveldb/leveldb/filter"
+       "github.com/syndtr/goleveldb/leveldb/opt"
+       "github.com/syndtr/goleveldb/leveldb/util"
+)
+
+const (
+       // degradationWarnInterval specifies how often warning should be printed if the
+       // leveldb database cannot keep up with requested writes.
+       degradationWarnInterval = time.Minute
+
+       // minCache is the minimum amount of memory in megabytes to allocate to leveldb
+       // read and write caching, split half and half.
+       minCache = 16
+
+       // minHandles is the minimum number of files handles to allocate to the open
+       // database files.
+       minHandles = 16
+
+       // metricsGatheringInterval specifies the interval to retrieve leveldb database
+       // compaction, io and pause stats to report to the user.
+       metricsGatheringInterval = 3 * time.Second
+)
+
+// Database is a persistent key-value store. Apart from basic data storage
+// functionality it also supports batch writes and iterating over the keyspace in
+// binary-alphabetical order.
+type Database struct {
+       fn string      // filename for reporting
+       db *leveldb.DB // LevelDB instance
+
+       compTimeMeter      metrics.Meter // Meter for measuring the total time spent in database compaction
+       compReadMeter      metrics.Meter // Meter for measuring the data read during compaction
+       compWriteMeter     metrics.Meter // Meter for measuring the data written during compaction
+       writeDelayNMeter   metrics.Meter // Meter for measuring the write delay number due to database compaction
+       writeDelayMeter    metrics.Meter // Meter for measuring the write delay duration due to database compaction
+       diskSizeGauge      metrics.Gauge // Gauge for tracking the size of all the levels in the database
+       diskReadMeter      metrics.Meter // Meter for measuring the effective amount of data read
+       diskWriteMeter     metrics.Meter // Meter for measuring the effective amount of data written
+       memCompGauge       metrics.Gauge // Gauge for tracking the number of memory compaction
+       level0CompGauge    metrics.Gauge // Gauge for tracking the number of table compaction in level0
+       nonlevel0CompGauge metrics.Gauge // Gauge for tracking the number of table compaction in non0 level
+       seekCompGauge      metrics.Gauge // Gauge for tracking the number of table compaction caused by read opt
+
+       quitLock sync.Mutex      // Mutex protecting the quit channel access
+       quitChan chan chan error // Quit channel to stop the metrics collection before closing the database
+
+       log log.Logger // Contextual logger tracking the database path
+}
+
+// New returns a wrapped LevelDB object. The namespace is the prefix that the
+// metrics reporting should use for surfacing internal stats.
+func New(file string, cache int, handles int, namespace string, readonly bool) (*Database, error) {
+       return NewCustom(file, namespace, func(options *opt.Options) {
+               // Ensure we have some minimal caching and file guarantees
+               if cache < minCache {
+                       cache = minCache
+               }
+               if handles < minHandles {
+                       handles = minHandles
+               }
+               // Set default options
+               options.OpenFilesCacheCapacity = handles
+               options.BlockCacheCapacity = cache / 2 * opt.MiB
+               options.WriteBuffer = cache / 4 * opt.MiB // Two of these are used internally
+               if readonly {
+                       options.ReadOnly = true
+               }
+       })
+}
+
+// NewCustom returns a wrapped LevelDB object. The namespace is the prefix that the
+// metrics reporting should use for surfacing internal stats.
+// The customize function allows the caller to modify the leveldb options.
+func NewCustom(file string, namespace string, customize func(options *opt.Options)) (*Database, error) {
+       options := configureOptions(customize)
+       logger := log.New("database", file)
+       usedCache := options.GetBlockCacheCapacity() + options.GetWriteBuffer()*2
+       logCtx := []interface{}{"cache", common.StorageSize(usedCache), "handles", options.GetOpenFilesCacheCapacity()}
+       if options.ReadOnly {
+               logCtx = append(logCtx, "readonly", "true")
+       }
+       logger.Info("Allocated cache and file handles", logCtx...)
+
+       // Open the db and recover any potential corruptions
+       db, err := leveldb.OpenFile(file, options)
+       if _, corrupted := err.(*errors.ErrCorrupted); corrupted {
+               db, err = leveldb.RecoverFile(file, nil)
+       }
+       if err != nil {
+               return nil, err
+       }
+       // Assemble the wrapper with all the registered metrics
+       ldb := &Database{
+               fn:       file,
+               db:       db,
+               log:      logger,
+               quitChan: make(chan chan error),
+       }
+       ldb.compTimeMeter = metrics.NewRegisteredMeter(namespace+"compact/time", nil)
+       ldb.compReadMeter = metrics.NewRegisteredMeter(namespace+"compact/input", nil)
+       ldb.compWriteMeter = metrics.NewRegisteredMeter(namespace+"compact/output", nil)
+       ldb.diskSizeGauge = metrics.NewRegisteredGauge(namespace+"disk/size", nil)
+       ldb.diskReadMeter = metrics.NewRegisteredMeter(namespace+"disk/read", nil)
+       ldb.diskWriteMeter = metrics.NewRegisteredMeter(namespace+"disk/write", nil)
+       ldb.writeDelayMeter = metrics.NewRegisteredMeter(namespace+"compact/writedelay/duration", nil)
+       ldb.writeDelayNMeter = metrics.NewRegisteredMeter(namespace+"compact/writedelay/counter", nil)
+       ldb.memCompGauge = metrics.NewRegisteredGauge(namespace+"compact/memory", nil)
+       ldb.level0CompGauge = metrics.NewRegisteredGauge(namespace+"compact/level0", nil)
+       ldb.nonlevel0CompGauge = metrics.NewRegisteredGauge(namespace+"compact/nonlevel0", nil)
+       ldb.seekCompGauge = metrics.NewRegisteredGauge(namespace+"compact/seek", nil)
+
+       // Start up the metrics gathering and return
+       go ldb.meter(metricsGatheringInterval)
+       return ldb, nil
+}
+
+// configureOptions sets some default options, then runs the provided setter.
+func configureOptions(customizeFn func(*opt.Options)) *opt.Options {
+       // Set default options
+       options := &opt.Options{
+               Filter:                 filter.NewBloomFilter(10),
+               DisableSeeksCompaction: true,
+       }
+       // Allow caller to make custom modifications to the options
+       if customizeFn != nil {
+               customizeFn(options)
+       }
+       return options
+}
+
+// Close stops the metrics collection, flushes any pending data to disk and closes
+// all io accesses to the underlying key-value store.
+func (db *Database) Close() error {
+       db.quitLock.Lock()
+       defer db.quitLock.Unlock()
+
+       if db.quitChan != nil {
+               errc := make(chan error)
+               db.quitChan <- errc
+               if err := <-errc; err != nil {
+                       db.log.Error("Metrics collection failed", "err", err)
+               }
+               db.quitChan = nil
+       }
+       return db.db.Close()
+}
+
+// Has retrieves if a key is present in the key-value store.
+func (db *Database) Has(key []byte) (bool, error) {
+       return db.db.Has(key, nil)
+}
+
+// Get retrieves the given key if it's present in the key-value store.
+func (db *Database) Get(key []byte) ([]byte, error) {
+       dat, err := db.db.Get(key, nil)
+       if err != nil {
+               return nil, err
+       }
+       return dat, nil
+}
+
+// Put inserts the given value into the key-value store.
+func (db *Database) Put(key []byte, value []byte) error {
+       return db.db.Put(key, value, nil)
+}
```

```
+
+// Delete removes the key from the key-value store.
+func (db *Database) Delete(key []byte) error {
+       return db.db.Delete(key, nil)
+}
+
+// NewBatch creates a write-only key-value store that buffers changes to its host
+// database until a final write is called.
+func (db *Database) NewBatch() ethdb.Batch {
+       return &batch{
+               db: db.db,
+               b:  new(leveldb.Batch),
+       }
+}
+
+// NewIterator creates a binary-alphabetical iterator over a subset
+// of database content with a particular key prefix, starting at a particular
+// initial key (or after, if it does not exist).
+func (db *Database) NewIterator(prefix []byte, start []byte) ethdb.Iterator {
+       return db.db.NewIterator(bytesPrefixRange(prefix, start), nil)
+}
+
+// Stat returns a particular internal stat of the database.
+func (db *Database) Stat(property string) (string, error) {
+       return db.db.GetProperty(property)
+}
+
+// Compact flattens the underlying data store for the given key range. In essence,
+// deleted and overwritten versions are discarded, and the data is rearranged to
+// reduce the cost of operations needed to access them.
+//
+// A nil start is treated as a key before all keys in the data store; a nil limit
+// is treated as a key after all keys in the data store. If both is nil then it
+// will compact entire data store.
+func (db *Database) Compact(start []byte, limit []byte) error {
+       return db.db.CompactRange(util.Range{Start: start, Limit: limit})
+}
+
+// Path returns the path to the database directory.
+func (db *Database) Path() string {
+       return db.fn
+}
+
+// meter periodically retrieves internal leveldb counters and reports them to
+// the metrics subsystem.
+//
+// This is how a LevelDB stats table looks like (currently):
+//   Compactions
+//    Level |   Tables   |    Size(MB)   |    Time(sec)  |    Read(MB)   |   Write(MB)
+//   -------+------------+---------------+---------------+---------------+---------------
+//      0   |          0 |       0.00000 |       1.27969 |       0.00000 |      12.31098
+//      1   |         85 |     109.27913 |      28.09293 |     213.92493 |     214.26294
+//      2   |        523 |    1000.37159 |       7.26059 |      66.86342 |      66.77884
+//      3   |        570 |    1113.18458 |       0.00000 |       0.00000 |       0.00000
+//
+// This is how the write delay look like (currently):
+// DelayN:5 Delay:406.604657ms Paused: false
+//
+// This is how the iostats look like (currently):
+// Read(MB):3895.04860 Write(MB):3654.64712
+func (db *Database) meter(refresh time.Duration) {
+       // Create the counters to store current and previous compaction values
+       compactions := make([][]float64, 2)
+       for i := 0; i < 2; i++ {
+               compactions[i] = make([]float64, 4)
+       }
+       // Create storage for iostats.
+       var iostats [2]float64
+
+       // Create storage and warning log tracer for write delay.
+       var (
+               delaystats      [2]int64
+               lastWritePaused time.Time
+       )
+
+       var (
+               errc chan error
+               merr error
+       )
+
+       timer := time.NewTimer(refresh)
+       defer timer.Stop()
+
+       // Iterate ad infinitum and collect the stats
+       for i := 1; errc == nil && merr == nil; i++ {
+               // Retrieve the database stats
+               stats, err := db.db.GetProperty("leveldb.stats")
+               if err != nil {
+                       db.log.Error("Failed to read database stats", "err", err)
+                       merr = err
+                       continue
+               }
+               // Find the compaction table, skip the header
+               lines := strings.Split(stats, "\n")
+               for len(lines) > 0 && strings.TrimSpace(lines[0]) != "Compactions" {
+                       lines = lines[1:]
+               }
+               if len(lines) <= 3 {
+                       db.log.Error("Compaction leveldbTable not found")
+                       merr = errors.New("compaction leveldbTable not found")
+                       continue
+               }
+               lines = lines[3:]
+
+               // Iterate over all the leveldbTable rows, and accumulate the entries
+               for j := 0; j < len(compactions[i%2]); j++ {
+                       compactions[i%2][j] = 0
+               }
+               for _, line := range lines {
+                       parts := strings.Split(line, "|")
+                       if len(parts) != 6 {
+                               break
+                       }
+                       for idx, counter := range parts[2:] {
+                               value, err := strconv.ParseFloat(strings.TrimSpace(counter), 64)
+                               if err != nil {
+                                       db.log.Error("Compaction entry parsing failed", "err", err)
+                                       merr = err
+                                       continue
+                               }
+                               compactions[i%2][idx] += value
+                       }
+               }
+               // Update all the requested meters
+               if db.diskSizeGauge != nil {
+                       db.diskSizeGauge.Update(int64(compactions[i%2][0] * 1024 * 1024))
+               }
+               if db.compTimeMeter != nil {
+                       db.compTimeMeter.Mark(int64((compactions[i%2][1] - compactions[(i-1)%2][1]) * 1000 * 1000 * 1000))
+               }
+               if db.compReadMeter != nil {
+                       db.compReadMeter.Mark(int64((compactions[i%2][2] - compactions[(i-1)%2][2]) * 1024 * 1024))
+               }
+               if db.compWriteMeter != nil {
```

```
+			db.compWriteMeter.Mark(int64((compactions[i%2][3] - compactions[(i-1)%2][3]) * 1024 * 1024))
+		}
+		// Retrieve the write delay statistic
+		writedelay, err := db.db.GetProperty("leveldb.writedelay")
+		if err != nil {
+			db.log.Error("Failed to read database write delay statistic", "err", err)
+			merr = err
+			continue
+		}
+		var (
+			delayN        int64
+			delayDuration string
+			duration      time.Duration
+			paused        bool
+		)
+		if n, err := fmt.Sscanf(writedelay, "DelayN:%d Delay:%s Paused:%t", &delayN, &delayDuration, &paused); n != 3 || err != nil {
+			db.log.Error("Write delay statistic not found")
+			merr = err
+			continue
+		}
+		duration, err = time.ParseDuration(delayDuration)
+		if err != nil {
+			db.log.Error("Failed to parse delay duration", "err", err)
+			merr = err
+			continue
+		}
+		if db.writeDelayNMeter != nil {
+			db.writeDelayNMeter.Mark(delayN - delaystats[0])
+		}
+		if db.writeDelayMeter != nil {
+			db.writeDelayMeter.Mark(duration.Nanoseconds() - delaystats[1])
+		}
+		// If a warning that db is performing compaction has been displayed, any subsequent
+		// warnings will be withheld for one minute not to overwhelm the user.
+		if paused && delayN-delaystats[0] == 0 && duration.Nanoseconds()-delaystats[1] == 0 &&
+			time.Now().After(lastWritePaused.Add(degradationWarnInterval)) {
+			db.log.Warn("Database compacting, degraded performance")
+			lastWritePaused = time.Now()
+		}
+		delaystats[0], delaystats[1] = delayN, duration.Nanoseconds()
+
+		// Retrieve the database iostats.
+		ioStats, err := db.db.GetProperty("leveldb.iostats")
+		if err != nil {
+			db.log.Error("Failed to read database iostats", "err", err)
+			merr = err
+			continue
+		}
+		var nRead, nWrite float64
+		parts := strings.Split(ioStats, " ")
+		if len(parts) < 2 {
+			db.log.Error("Bad syntax of ioStats", "ioStats", ioStats)
+			merr = fmt.Errorf("bad syntax of ioStats %s", ioStats)
+			continue
+		}
+		if n, err := fmt.Sscanf(parts[0], "Read(MB):%f", &nRead); n != 1 || err != nil {
+			db.log.Error("Bad syntax of read entry", "entry", parts[0])
+			merr = err
+			continue
+		}
+		if n, err := fmt.Sscanf(parts[1], "Write(MB):%f", &nWrite); n != 1 || err != nil {
+			db.log.Error("Bad syntax of write entry", "entry", parts[1])
+			merr = err
+			continue
+		}
+		if db.diskReadMeter != nil {
+			db.diskReadMeter.Mark(int64((nRead - iostats[0]) * 1024 * 1024))
+		}
+		if db.diskWriteMeter != nil {
+			db.diskWriteMeter.Mark(int64((nWrite - iostats[1]) * 1024 * 1024))
+		}
+		iostats[0], iostats[1] = nRead, nWrite
+
+		compCount, err := db.db.GetProperty("leveldb.compcount")
+		if err != nil {
+			db.log.Error("Failed to read database iostats", "err", err)
+			merr = err
+			continue
+		}
+
+		var (
+			memComp       uint32
+			level0Comp    uint32
+			nonLevel0Comp uint32
+			seekComp      uint32
+		)
+		if n, err := fmt.Sscanf(compCount, "MemComp:%d Level0Comp:%d NonLevel0Comp:%d SeekComp:%d", &memComp, &level0Comp, &nonLevel0Comp, &seekComp); n != 4 || err != nil {
+			db.log.Error("Compaction count statistic not found")
+			merr = err
+			continue
+		}
+		db.memCompGauge.Update(int64(memComp))
+		db.level0CompGauge.Update(int64(level0Comp))
+		db.nonlevel0CompGauge.Update(int64(nonLevel0Comp))
+		db.seekCompGauge.Update(int64(seekComp))
+
+		// Sleep a bit, then repeat the stats collection
+		select {
+		case errc = <-db.quitChan:
+			// Quit requesting, stop hammering the database
+		case <-timer.C:
+			timer.Reset(refresh)
+			// Timeout, gather a new set of stats
+		}
+	}
+
+	if errc == nil {
+		errc = <-db.quitChan
+	}
+	errc <- merr
+}
+
+// batch is a write-only leveldb batch that commits changes to its host database
+// when Write is called. A batch cannot be used concurrently.
+type batch struct {
+	db   *leveldb.DB
+	b    *leveldb.Batch
+	size int
+}
+
+// Put inserts the given value into the batch for later committing.
+func (b *batch) Put(key, value []byte) error {
+	b.b.Put(key, value)
+	b.size += len(value)
+	return nil
+}
+
+// Delete inserts the a key removal into the batch for later committing.
+func (b *batch) Delete(key []byte) error {
+	b.b.Delete(key)
+	b.size += len(key)
+	return nil
+}
+
+// ValueSize retrieves the amount of data queued up for writing.
+func (b *batch) ValueSize() int {
+       return b.size
+}
+
+// Write flushes any accumulated data to disk.
+func (b *batch) Write() error {
+       return b.db.Write(b.b, nil)
+}
+
+// Reset resets the batch for reuse.
+func (b *batch) Reset() {
+       b.b.Reset()
+       b.size = 0
+}
+
+// Replay replays the batch contents.
+func (b *batch) Replay(w ethdb.KeyValueWriter) error {
+       return b.b.Replay(&replayer{writer: w})
+}
+
+// replayer is a small wrapper to implement the correct replay methods.
+type replayer struct {
+       writer  ethdb.KeyValueWriter
+       failure error
+}
+
+// Put inserts the given value into the key-value data store.
+func (r *replayer) Put(key, value []byte) {
+       // If the replay already failed, stop executing ops
+       if r.failure != nil {
+               return
+       }
+       r.failure = r.writer.Put(key, value)
+}
+
+// Delete removes the key from the key-value data store.
+func (r *replayer) Delete(key []byte) {
+       // If the replay already failed, stop executing ops
+       if r.failure != nil {
+               return
+       }
+       r.failure = r.writer.Delete(key)
+}
+
+// bytesPrefixRange returns key range that satisfy
+// - the given prefix, and
+// - the given seek position
+func bytesPrefixRange(prefix, start []byte) *util.Range {
+       r := util.BytesPrefix(prefix)
+       r.Start = append(r.Start, start...)
+       return r
+}
diff --git a/ethdb/leveldb/leveldb_test.go b/ethdb/leveldb/leveldb_test.go
new file mode 100644
index 00000000..fff2a97d
--- /dev/null
+++ b/ethdb/leveldb/leveldb_test.go
@@ -0,0 +1,50 @@
+// (c) 2021-2022, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2019 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package leveldb
+
+import (
+       "testing"
+
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb/dbtest"
+       "github.com/syndtr/goleveldb/leveldb"
+       "github.com/syndtr/goleveldb/leveldb/storage"
+)
+
+func TestLevelDB(t *testing.T) {
+       t.Run("DatabaseSuite", func(t *testing.T) {
+               dbtest.TestDatabaseSuite(t, func() ethdb.KeyValueStore {
+                       db, err := leveldb.Open(storage.NewMemStorage(), nil)
+                       if err != nil {
+                               t.Fatal(err)
+                       }
+                       return &Database{
+                               db: db,
+                       }
+               })
+       })
+}
diff --git a/ethdb/memorydb/memorydb.go b/ethdb/memorydb/memorydb.go
index 769581ff..6b56519d 100644
--- a/ethdb/memorydb/memorydb.go
+++ b/ethdb/memorydb/memorydb.go
@@ -33,8 +33,8 @@ import (
        "strings"
        "sync"

-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 var (
diff --git a/ethdb/memorydb/memorydb_test.go b/ethdb/memorydb/memorydb_test.go
index 34361e9f..92df1f67 100644
--- a/ethdb/memorydb/memorydb_test.go
+++ b/ethdb/memorydb/memorydb_test.go
@@ -29,8 +29,8 @@ package memorydb
 import (
        "testing"

-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/ethdb/dbtest"
+       "github.com/flare-foundation/coreth/ethdb"
```

```diff
+	"github.com/flare-foundation/coreth/ethdb/dbtest"
 )
 
 func TestMemoryDB(t *testing.T) {
diff --git a/go.mod b/go.mod
index f607b56d..1262c316 100644
--- a/go.mod
+++ b/go.mod
@@ -1,28 +1,30 @@
-module github.com/ava-labs/coreth
+module github.com/flare-foundation/coreth
 
 go 1.16
 
 require (
 	github.com/VictoriaMetrics/fastcache v1.6.0
-	github.com/ava-labs/avalanchego v1.6.4
 	github.com/btcsuite/btcd v0.21.0-beta // indirect
 	github.com/cespare/cp v0.1.0
 	github.com/davecgh/go-spew v1.1.1
 	github.com/deckarep/golang-set v1.7.1
 	github.com/ethereum/go-ethereum v1.10.12
 	github.com/fjl/memsize v0.0.0-20190710130421-bcb5799ab5e5
+	github.com/flare-foundation/flare v0.5.0
 	github.com/gballet/go-libpcsclite v0.0.0-20191108122812-4678299bea08
 	github.com/google/uuid v1.1.5
 	github.com/gorilla/rpc v1.2.0
 	github.com/gorilla/websocket v1.4.2
 	github.com/hashicorp/go-bexpr v0.1.10
-	github.com/hashicorp/go-plugin v1.3.0
+	github.com/hashicorp/go-plugin v1.4.3
 	github.com/hashicorp/golang-lru v0.5.5-0.20210104140557-80c98217689d
 	github.com/holiman/bloomfilter/v2 v2.0.3
 	github.com/holiman/uint256 v1.2.0
 	github.com/mattn/go-colorable v0.1.8
 	github.com/mattn/go-isatty v0.0.12
 	github.com/olekukonko/tablewriter v0.0.5
+	github.com/prometheus/client_golang v1.7.1
+	github.com/prometheus/client_model v0.2.0
 	github.com/prometheus/tsdb v0.10.0 // indirect
 	github.com/rjeczalik/notify v0.9.2
 	github.com/spf13/cast v1.3.1
@@ -30,8 +32,10 @@ require (
 	github.com/spf13/viper v1.7.1
 	github.com/status-im/keycard-go v0.0.0-20200402102358-957c09536969
 	github.com/stretchr/testify v1.7.0
+	github.com/syndtr/goleveldb v1.0.1-0.20210819022825-2ae1ddf74ef7
 	github.com/tyler-smith/go-bip39 v1.0.2
 	golang.org/x/crypto v0.0.0-20210322153248-0c34fe9e7dc2
+	golang.org/x/sync v0.0.0-20210220032951-036812b2e83c
 	golang.org/x/text v0.3.6
 	golang.org/x/time v0.0.0-20210723032227-1f47c861a9ac
 	gopkg.in/olebedev/go-duktape.v3 v3.0.0-20200619000410-60c24ae608a6
diff --git a/go.sum b/go.sum
index 5b333235..93546743 100644
--- a/go.sum
+++ b/go.sum
@@ -40,7 +40,6 @@ github.com/BurntSushi/toml v0.3.1/go.mod h1:xHWCNGjB5oqiDr8zfno3MHue2Ht5sIBksp03
 github.com/BurntSushi/xgb v0.0.0-20160522181843-27f122750802/go.mod h1:IVnqGOEym/WlBOVXweHU+Q+/VP0lqqI8lqeDx9IjBqo=
 github.com/DATA-DOG/go-sqlmock v1.3.3/go.mod h1:f/Ixk793poVmq4qj/V1dPUg2JEAKC73Q5eFN3EC/SaM=
 github.com/Microsoft/go-winio v0.4.14/go.mod h1:qXqCSQ3Xa7+6tgxaGTIe4Kpcdsi+P8jBhyzoq1bpyYA=
-github.com/NYTimes/gziphandler v1.1.1 h1:ZUDjpQae29j0ryrS0u/B8HZfJBtBQHjqw2rQ2cqUQ3I=
 github.com/NYTimes/gziphandler v1.1.1/go.mod h1:n/CVRwUEOgIxrgPvAQhUUr9oeUtvrhMomdKFjzJNB0c=
 github.com/OneOfOne/xxhash v1.2.2/go.mod h1:HSdplMjZKSmBqAxg5vPj2TmRDmfkzw+cTzAElWljhcU=
 github.com/StackExchange/wmi v0.0.0-20180116203802-5d049714c4a6 h1:fLjPD/aNc3UIOA6tDi6QXUemppXK3P9BI7mr2hd6gx8=
@@ -60,8 +59,6 @@ github.com/apache/arrow/go/arrow v0.0.0-20191024131854-af6fa24be0db/go.mod h1:VT
 github.com/armon/circbuf v0.0.0-20150827004946-bbbad097214e/go.mod h1:3U/XgcO3hCbHZ8TKRvWD2dDTCfh9M9ya+I9JpbB7O8o=
 github.com/armon/go-metrics v0.0.0-20180917152333-f0300d1749da/go.mod h1:Q73ZrmVTwzkszR9V5SSuryQ31EELlFMUz1kKyl939pY=
 github.com/armon/go-radix v0.0.0-20180808171621-7fddfc383310/go.mod h1:ufUuZ+zHj4x4TnLV4JWEpy2hxWSpsRywHrMgIH9cCH8=
-github.com/ava-labs/avalanchego v1.6.4 h1:EbjGqyU9MqpsRVC9wmNmwUNlFM8aiTTKmNAwuiHDmPs=
-github.com/ava-labs/avalanchego v1.6.4/go.mod h1:DzxlGkF8hj3GiwRoQfHq9gJfyt4EyeIpAsdfBOBg6mI=
 github.com/aws/aws-sdk-go-v2 v1.2.0/go.mod h1:zEQs02YRBw1DjK0PoJv3ygDYOFTre1ejlJWl8FwAuQo=
 github.com/aws/aws-sdk-go-v2/config v1.1.1/go.mod h1:0XsVy9lBI/BCXm+2Tuvt39YmdHwS5unDQmxZOYe8F5Y=
 github.com/aws/aws-sdk-go-v2/credentials v1.1.1/go.mod h1:mM2iIjwl7LULWtS6JCACyInboHirisUUdkBPoTHMOUo=
@@ -157,6 +154,8 @@ github.com/fatih/color v1.7.0 h1:DkWD4oS2D8LGGgTQ6IvwJJXSL5Vp2ffcQg58nFV38Ys=
 github.com/fatih/color v1.7.0/go.mod h1:Zm6kSWBoL9eyXnKyktHP6abPY2pDugNf5KwzbycvMj4=
 github.com/fjl/memsize v0.0.0-20190710130421-bcb5799ab5e5 h1:FtmdgXiUlNeRsoNMFlKLDt+S+6hbjVMEW6RGQ7aUf7c=
 github.com/fjl/memsize v0.0.0-20190710130421-bcb5799ab5e5/go.mod h1:VvhXpOYNQvB+uIk2RvXzuaQtkQJzzIx6lSBe1xv7hi0=
+github.com/flare-foundation/flare v0.5.0 h1:pQJhItsDwEC8U7QOlFCSbBRkMcD03KR2/Ov7SsfbHLc=
+github.com/flare-foundation/flare v0.5.0/go.mod h1:ai++PWjLrLOZYheY7qutUJbReWsJSe5XO47QcIQQW0U=
 github.com/fogleman/gg v1.2.1-0.20190220221249-0403632d5b90/go.mod h1:R/bRT+9gY/C5z7JzPU0zXsXHKM4/ayA+zqcVNZzPa1k=
 github.com/fogleman/gg v1.3.0/go.mod h1:R/bRT+9gY/C5z7JzPU0zXsXHKM4/ayA+zqcVNZzPa1k=
 github.com/fortytw2/leaktest v1.3.0 h1:u8491cBMTQ8ft8aeV+adlcytMZylmA5nnwwkRZjI8vw=
@@ -256,9 +255,7 @@ github.com/googleapis/gax-go/v2 v2.0.4/go.mod h1:0Wqv26UfaUD9n4G6kQubkQ+KchISgw+
 github.com/googleapis/gax-go/v2 v2.0.5/go.mod h1:DWXyrwAJ9X0FpwwEdw+IPEYBICEFu5mhpdKc/us6bOk=
 github.com/gopherjs/gopherjs v0.0.0-20181017120253-0766667cb4d1 h1:EGx4pi6eqNxGaHF6qqu48+N2wcFQ5qg5FXgOdqsJ5d8=
 github.com/gopherjs/gopherjs v0.0.0-20181017120253-0766667cb4d1/go.mod h1:wJfORRmW1u3UXTncJ5qlYoELFm8eSnnEO6hX4iZ3EWY=
-github.com/gorilla/handlers v1.4.2 h1:0QniY0USkHQ1RGCLfKxeNHK9bkDHGRYGNDFBCS+YARg=
 github.com/gorilla/handlers v1.4.2/go.mod h1:Qkdc/uu4tH4g6mTK6auzZ766c4CA0Ng8+o/OAirnOIQ=
-github.com/gorilla/mux v1.8.0 h1:i40aqfkR1h2SlN9hojwV5ZA91wcXFOvkdNIeFDP5koI=
 github.com/gorilla/mux v1.8.0/go.mod h1:DVbg23sWSpFRCP0SfiEN6jmj59UnW/n46BH5rLB71So=
 github.com/gorilla/rpc v1.2.0 h1:WvvdC2lNeT1SP32zrIce5l0ECBfbAlmrmSBsuc57wfk=
 github.com/gorilla/rpc v1.2.0/go.mod h1:V4h9r+4sF5HnzqbwIez0fKSpANP0zlYd3qR7p36jkTQ=
@@ -281,8 +278,9 @@ github.com/hashicorp/go-hclog v0.14.1/go.mod h1:whpDNt7SSdeAju8AWKIWsul05p54N/39
 github.com/hashicorp/go-immutable-radix v1.0.0/go.mod h1:0y9vanUI8NX6FsYoO3zeMjhV/C5i9g4Q3DwcSNZ4P60=
 github.com/hashicorp/go-msgpack v0.5.3/go.mod h1:ahLV/dePpqEmjfWmKiqvPkv/twdG7iPBM1vqhUKIvfM=
 github.com/hashicorp/go-multierror v1.0.0/go.mod h1:dHtQlpGsu+cZNNAkkCN/P3hoUDHhCYQXV3UM06sGGrk=
-github.com/hashicorp/go-plugin v1.3.0 h1:4d/wJojzvHV1I4i/rrjVaeuyxWrLzDE1mDCyDy8fXS8=
 github.com/hashicorp/go-plugin v1.3.0/go.mod h1:F9eH4LrE/ZsRdbwhfjs9k9HoDUwAHnYtXdgmf1AVNs0=
+github.com/hashicorp/go-plugin v1.4.3 h1:DXmvivbWD5qdiBts9TpBC7BYL1Aia5sxbRgQB+v6UZM=
+github.com/hashicorp/go-plugin v1.4.3/go.mod h1:5fGEH17QVwTTcR0zV7yhDPLLmFX9YSZ38b18Udy6vYQ=
 github.com/hashicorp/go-rootcerts v1.0.0/go.mod h1:K6zTfqpRlCUIjkwsN4Z+hiSfzSTQa6eBIzfwKfwNnHU=
 github.com/hashicorp/go-sockaddr v1.0.0/go.mod h1:7Xibr9yA9JjQq1JpNB2Vw7kxv8xerXegt+ozgdvDeDU=
 github.com/hashicorp/go-syslog v1.0.0/go.mod h1:qPfqrKkXGihmCqbJM2mZgkZGvKG1dFdvsLplgctolz4=
diff --git a/interfaces/interfaces.go b/interfaces/interfaces.go
index 100e658c..4e75ef41 100644
--- a/interfaces/interfaces.go
+++ b/interfaces/interfaces.go
@@ -32,8 +32,8 @@ import (
 	"errors"
 	"math/big"
 
-	"github.com/ava-labs/coreth/core/types"
 	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/coreth/core/types"
 )
 
 // NotFound is returned by API methods if the requested item does not exist.
diff --git a/internal/ethapi/api.go b/internal/ethapi/api.go
index 60e0242c..7ed7af3b 100644
--- a/internal/ethapi/api.go
+++ b/internal/ethapi/api.go
@@ -34,16 +34,6 @@ import (
 	"strings"
 	"time"
 
-	"github.com/ava-labs/avalanchego/ids"
-	"github.com/ava-labs/coreth/accounts"
-	"github.com/ava-labs/coreth/accounts/keystore"
-	"github.com/ava-labs/coreth/accounts/scwallet"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
```

```diff
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/davecgh/go-spew/spew"
 	"github.com/ethereum/go-ethereum/accounts/abi"
 	"github.com/ethereum/go-ethereum/common"
@@ -52,6 +42,17 @@ import (
 	"github.com/ethereum/go-ethereum/crypto"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/accounts"
+	"github.com/flare-foundation/coreth/accounts/keystore"
+	"github.com/flare-foundation/coreth/accounts/scwallet"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/eth/tracers/logger"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/rpc"
+	"github.com/flare-foundation/flare/ids"
 	"github.com/tyler-smith/go-bip39"
 )
 
@@ -599,9 +600,9 @@ func NewPublicBlockChainAPI(b Backend) *PublicBlockChainAPI {
 }
 
 // ChainId is the EIP-155 replay-protection chain id for the current ethereum chain config.
-func (api *PublicBlockChainAPI) ChainId() (*hexutil.Big, error) {
+func (s *PublicBlockChainAPI) ChainId() (*hexutil.Big, error) {
 	// if current block is at or past the EIP-155 replay-protection fork block, return chainID from config
-	if config := api.b.ChainConfig(); config.IsEIP155(api.b.CurrentBlock().Number()) {
+	if config := s.b.ChainConfig(); config.IsEIP155(s.b.CurrentBlock().Number()) {
 		return (*hexutil.Big)(config.ChainID), nil
 	}
 	return nil, fmt.Errorf("chain not synced beyond EIP-155 replay-protection fork block")
@@ -900,7 +901,11 @@ func DoCall(ctx context.Context, b Backend, args TransactionArgs, blockNrOrHash
 	if blkNumber, isNum := blockNrOrHash.Number(); isNum && blkNumber == rpc.PendingBlockNumber {
 		// Override header with a copy to ensure the original header is not modified
 		header = types.CopyHeader(header)
+		// Grab the hash of the unmodified header, so that the modified header can point to the
+		// prior block as its parent.
+		parentHash := header.Hash()
 		header.Time = uint64(time.Now().Unix())
+		header.ParentHash = parentHash
 		header.Number = new(big.Int).Add(header.Number, big.NewInt(1))
 		estimatedBaseFee, err := b.EstimateBaseFee(ctx)
 		if err != nil {
@@ -1157,7 +1162,7 @@ type StructLogRes struct {
 }
 
 // FormatLogs formats EVM returned structured logs for json output
-func FormatLogs(logs []vm.StructLog) []StructLogRes {
+func FormatLogs(logs []logger.StructLog) []StructLogRes {
 	formatted := make([]StructLogRes, len(logs))
 	for index, trace := range logs {
 		formatted[index] = StructLogRes{
@@ -1270,7 +1275,9 @@ func RPCMarshalBlock(block *types.Block, inclTx bool, fullTx bool, config *param
 // a `PublicBlockchainAPI`.
 func (s *PublicBlockChainAPI) rpcMarshalHeader(ctx context.Context, header *types.Header) map[string]interface{} {
 	fields := RPCMarshalHeader(header)
-	fields["totalDifficulty"] = (*hexutil.Big)(s.b.GetTd(ctx, header.Hash()))
+	// Note: Coreth enforces that the difficulty of a block is always 1, such that the total difficulty of a block
+	// will be equivalent to its height.
+	fields["totalDifficulty"] = (*hexutil.Big)(header.Number)
 	return fields
 }
 
@@ -1282,7 +1289,9 @@ func (s *PublicBlockChainAPI) rpcMarshalBlock(ctx context.Context, b *types.Bloc
 		return nil, err
 	}
 	if inclTx {
-		fields["totalDifficulty"] = (*hexutil.Big)(s.b.GetTd(ctx, b.Hash()))
+		// Note: Coreth enforces that the difficulty of a block is always 1, such that the total difficulty of a block
+		// will be equivalent to its height.
+		fields["totalDifficulty"] = (*hexutil.Big)(b.Number())
 	}
 	return fields, err
 }
@@ -1459,9 +1468,9 @@ func AccessList(ctx context.Context, b Backend, blockNrOrHash rpc.BlockNumberOrH
 	precompiles := vm.ActivePrecompiles(b.ChainConfig().AvalancheRules(header.Number, new(big.Int).SetUint64(header.Time)))
 
 	// Create an initial tracer
-	prevTracer := vm.NewAccessListTracer(nil, args.from(), to, precompiles)
+	prevTracer := logger.NewAccessListTracer(nil, args.from(), to, precompiles)
 	if args.AccessList != nil {
-		prevTracer = vm.NewAccessListTracer(*args.AccessList, args.from(), to, precompiles)
+		prevTracer = logger.NewAccessListTracer(*args.AccessList, args.from(), to, precompiles)
 	}
 	for {
 		// Retrieve the current access list to expand
@@ -1488,7 +1497,7 @@ func AccessList(ctx context.Context, b Backend, blockNrOrHash rpc.BlockNumberOrH
 		}
 
 		// Apply the transaction with the access list tracer
-		tracer := vm.NewAccessListTracer(accessList, args.from(), to, precompiles)
+		tracer := logger.NewAccessListTracer(accessList, args.from(), to, precompiles)
 		config := vm.Config{Tracer: tracer, Debug: true, NoBaseFee: true}
 		vmenv, _, err := b.GetEVM(ctx, msg, statedb, header, &config)
 		if err != nil {
diff --git a/internal/ethapi/backend.go b/internal/ethapi/backend.go
index 0f0795fa..f96e4de6 100644
--- a/internal/ethapi/backend.go
+++ b/internal/ethapi/backend.go
@@ -32,18 +32,18 @@ import (
 	"math/big"
 	"time"
 
-	"github.com/ava-labs/coreth/accounts"
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/bloombits"
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/core/vm"
-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/accounts"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/bloombits"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/core/vm"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/rpc"
 )
 
 // Backend interface provides the common API services (that are provided by
@@ -74,7 +74,6 @@ type Backend interface {
 	StateAndHeaderByNumber(ctx context.Context, number rpc.BlockNumber) (*state.StateDB, *types.Header, error)
 	StateAndHeaderByNumberOrHash(ctx context.Context, blockNrOrHash rpc.BlockNumberOrHash) (*state.StateDB, *types.Header, error)
 	GetReceipts(ctx context.Context, hash common.Hash) (types.Receipts, error)
-	GetTd(ctx context.Context, hash common.Hash) *big.Int
 	GetEVM(ctx context.Context, msg core.Message, state *state.StateDB, header *types.Header, vmConfig *vm.Config) (*vm.EVM, func() error, error)
 	SubscribeChainEvent(ch chan<- core.ChainEvent) event.Subscription
 	SubscribeChainHeadEvent(ch chan<- core.ChainHeadEvent) event.Subscription
@@ -113,40 +112,48 @@ func GetAPIs(apiBackend Backend) []rpc.API {
 			Version:   "1.0",
 			Service:   NewPublicEthereumAPI(apiBackend),
 			Public:    true,
+			Name:      "internal-public-eth",
 		}, {
 			Namespace: "eth",
 			Version:   "1.0",
 			Service:   NewPublicBlockChainAPI(apiBackend),
 			Public:    true,
+			Name:      "internal-public-blockchain",
 		}, {
 			Namespace: "eth",
 			Version:   "1.0",
 			Service:   NewPublicTransactionPoolAPI(apiBackend, nonceLock),
 			Public:    true,
+			Name:      "internal-public-transaction-pool",
 		}, {
 			Namespace: "txpool",
 			Version:   "1.0",
 			Service:   NewPublicTxPoolAPI(apiBackend),
 			Public:    true,
+			Name:      "internal-public-tx-pool",
 		}, {
 			Namespace: "debug",
 			Version:   "1.0",
 			Service:   NewPublicDebugAPI(apiBackend),
 			Public:    true,
+			Name:      "internal-public-debug",
 		}, {
 			Namespace: "debug",
 			Version:   "1.0",
 			Service:   NewPrivateDebugAPI(apiBackend),
+			Name:      "internal-private-debug",
 		}, {
 			Namespace: "eth",
 			Version:   "1.0",
 			Service:   NewPublicAccountAPI(apiBackend.AccountManager()),
 			Public:    true,
+			Name:      "internal-public-account",
 		}, {
 			Namespace: "personal",
 			Version:   "1.0",
 			Service:   NewPrivateAccountAPI(apiBackend, nonceLock),
 			Public:    false,
+			Name:      "internal-private-personal",
 		},
 	}
 }
diff --git a/internal/ethapi/transaction_args.go b/internal/ethapi/transaction_args.go
index 0048b910..8feab39c 100644
--- a/internal/ethapi/transaction_args.go
+++ b/internal/ethapi/transaction_args.go
@@ -33,12 +33,12 @@ import (
 	"fmt"
 	"math/big"
 
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/rpc"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/hexutil"
 	"github.com/ethereum/go-ethereum/common/math"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/rpc"
 )
 
 // TransactionArgs represents the arguments to construct a new transaction
diff --git a/internal/shutdowncheck/shutdown_tracker.go b/internal/shutdowncheck/shutdown_tracker.go
new file mode 100644
index 00000000..bfb2fbde
--- /dev/null
+++ b/internal/shutdowncheck/shutdown_tracker.go
@@ -0,0 +1,95 @@
+// (c) 2020-2021, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2021 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package shutdowncheck
+
+import (
+	"time"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
+)
+
+// ShutdownTracker is a service that reports previous unclean shutdowns
+// upon start. It needs to be started after a successful start-up and stopped
+// after a successful shutdown, just before the db is closed.
+type ShutdownTracker struct {
+	db     ethdb.Database
+	stopCh chan struct{}
+}
+
+// NewShutdownTracker creates a new ShutdownTracker instance and has
+// no other side-effect.
+func NewShutdownTracker(db ethdb.Database) *ShutdownTracker {
+	return &ShutdownTracker{
+		db:     db,
+		stopCh: make(chan struct{}),
+	}
+}
+
+// MarkStartup is to be called in the beginning when the node starts. It will:
+// - Push a new startup marker to the db
+// - Report previous unclean shutdowns
+func (t *ShutdownTracker) MarkStartup() {
+       if uncleanShutdowns, discards, err := rawdb.PushUncleanShutdownMarker(t.db); err != nil {
+               log.Error("Could not update unclean-shutdown-marker list", "error", err)
+       } else {
+               if discards > 0 {
+                       log.Warn("Old unclean shutdowns found", "count", discards)
+               }
+               for _, tstamp := range uncleanShutdowns {
+                       t := time.Unix(int64(tstamp), 0)
+                       log.Warn("Unclean shutdown detected", "booted", t,
+                               "age", common.PrettyAge(t))
+               }
+       }
+}
+
+// Start runs an event loop that updates the current marker's timestamp every 5 minutes.
+func (t *ShutdownTracker) Start() {
+       go func() {
+               ticker := time.NewTicker(5 * time.Minute)
+               defer ticker.Stop()
+               for {
+                       select {
+                       case <-ticker.C:
+                               rawdb.UpdateUncleanShutdownMarker(t.db)
+                       case <-t.stopCh:
+                               return
+                       }
+               }
+       }()
+}
+
+// Stop will stop the update loop and clear the current marker.
+func (t *ShutdownTracker) Stop() {
+       // Stop update loop.
+       t.stopCh <- struct{}{}
+       // Clear last marker.
+       rawdb.PopUncleanShutdownMarker(t.db)
+}
diff --git a/metrics/prometheus/prometheus.go b/metrics/prometheus/prometheus.go
new file mode 100644
index 00000000..83b92a7e
--- /dev/null
+++ b/metrics/prometheus/prometheus.go
@@ -0,0 +1,185 @@
+// (c) 2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package prometheus
+
+import (
+       "sort"
+       "strings"
+
+       "github.com/ethereum/go-ethereum/metrics"
+
+       "github.com/prometheus/client_golang/prometheus"
+
+       dto "github.com/prometheus/client_model/go"
+)
+
+var (
+       pv             = []float64{.5, .75, .95, .99, .999, .9999}
+       pvShortPercent = []float64{50, 95, 99}
+       pvShort        = []float64{.50, .95, .99}
+)
+
+type gatherer struct {
+       reg metrics.Registry
+}
+
+func (g gatherer) Gather() ([]*dto.MetricFamily, error) {
+       // Gather and pre-sort the metrics to avoid random listings
+       var names []string
+       g.reg.Each(func(name string, i interface{}) {
+               names = append(names, name)
+       })
+       sort.Strings(names)
+
+       mfs := make([]*dto.MetricFamily, 0, len(names))
+       for _, name := range names {
+               mIntf := g.reg.Get(name)
+               name := strings.Replace(name, "/", "_", -1)
+
+               switch m := mIntf.(type) {
+               case metrics.Counter:
+                       val := m.Snapshot().Count()
+                       valFloat := float64(val)
+                       mfs = append(mfs, &dto.MetricFamily{
+                               Name: &name,
+                               Type: dto.MetricType_COUNTER.Enum(),
+                               Metric: []*dto.Metric{{
+                                       Counter: &dto.Counter{
+                                               Value: &valFloat,
+                                       },
+                               }},
+                       })
+               case metrics.Gauge:
+                       val := m.Snapshot().Value()
+                       valFloat := float64(val)
+                       mfs = append(mfs, &dto.MetricFamily{
+                               Name: &name,
+                               Type: dto.MetricType_GAUGE.Enum(),
+                               Metric: []*dto.Metric{{
+                                       Gauge: &dto.Gauge{
+                                               Value: &valFloat,
+                                       },
+                               }},
+                       })
+               case metrics.GaugeFloat64:
+                       val := m.Snapshot().Value()
+                       mfs = append(mfs, &dto.MetricFamily{
+                               Name: &name,
+                               Type: dto.MetricType_GAUGE.Enum(),
+                               Metric: []*dto.Metric{{
+                                       Gauge: &dto.Gauge{
+                                               Value: &val,
+                                       },
+                               }},
+                       })
+               case metrics.Histogram:
+                       snapshot := m.Snapshot()
+                       count := snapshot.Count()
+                       countUint := uint64(count)
+                       sum := snapshot.Sum()
+                       sumFloat := float64(sum)
+
+                       ps := m.Percentiles(pv)
+                       qs := make([]*dto.Quantile, len(pv))
+                       for i := range ps {
+                               v := pv[i]
+                               s := ps[i]
+				qs[i] = &dto.Quantile{
+					Quantile: &v,
+					Value:    &s,
+				}
+			}
+
+			mfs = append(mfs, &dto.MetricFamily{
+				Name: &name,
+				Type: dto.MetricType_SUMMARY.Enum(),
+				Metric: []*dto.Metric{{
+					Summary: &dto.Summary{
+						SampleCount: &countUint,
+						SampleSum:   &sumFloat,
+						Quantile:    qs,
+					},
+				}},
+			})
+		case metrics.Meter:
+			val := m.Snapshot().Count()
+			valFloat := float64(val)
+			mfs = append(mfs, &dto.MetricFamily{
+				Name: &name,
+				Type: dto.MetricType_GAUGE.Enum(),
+				Metric: []*dto.Metric{{
+					Gauge: &dto.Gauge{
+						Value: &valFloat,
+					},
+				}},
+			})
+		case metrics.Timer:
+			snapshot := m.Snapshot()
+			count := snapshot.Count()
+			countUint := uint64(count)
+			sum := snapshot.Sum()
+			sumFloat := float64(sum)
+
+			ps := m.Percentiles(pv)
+			qs := make([]*dto.Quantile, len(pv))
+			for i := range ps {
+				v := pv[i]
+				s := ps[i]
+				qs[i] = &dto.Quantile{
+					Quantile: &v,
+					Value:    &s,
+				}
+			}
+
+			mfs = append(mfs, &dto.MetricFamily{
+				Name: &name,
+				Type: dto.MetricType_SUMMARY.Enum(),
+				Metric: []*dto.Metric{{
+					Summary: &dto.Summary{
+						SampleCount: &countUint,
+						SampleSum:   &sumFloat,
+						Quantile:    qs,
+					},
+				}},
+			})
+		case metrics.ResettingTimer:
+			snapshot := m.Snapshot()
+
+			vals := snapshot.Values()
+			count := uint64(len(vals))
+			if count == 0 {
+				continue
+			}
+
+			ps := m.Percentiles(pvShortPercent)
+			qs := make([]*dto.Quantile, len(pv))
+			for i := range pvShort {
+				v := pv[i]
+				s := float64(ps[i])
+				qs[i] = &dto.Quantile{
+					Quantile: &v,
+					Value:    &s,
+				}
+			}
+
+			mfs = append(mfs, &dto.MetricFamily{
+				Name: &name,
+				Type: dto.MetricType_SUMMARY.Enum(),
+				Metric: []*dto.Metric{{
+					Summary: &dto.Summary{
+						SampleCount: &count,
+						// TODO: do we need to specify SampleSum here? and if so
+						// what should that be?
+						Quantile: qs,
+					},
+				}},
+			})
+		}
+	}
+	return mfs, nil
+}
+
+func Gatherer(reg metrics.Registry) prometheus.Gatherer {
+	return gatherer{reg: reg}
+}
diff --git a/miner/miner.go b/miner/miner.go
index 263439c8..f05ae5c2 100644
--- a/miner/miner.go
+++ b/miner/miner.go
@@ -28,12 +28,13 @@
 package miner
 
 import (
-	"github.com/ava-labs/coreth/consensus"
-	"github.com/ava-labs/coreth/core"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/event"
+	"github.com/flare-foundation/coreth/consensus"
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/flare/utils/timer/mockable"
 )
 
 // Backend wraps all methods required for mining.
@@ -51,9 +52,9 @@ type Miner struct {
 	worker *worker
 }
 
-func New(eth Backend, config *Config, chainConfig *params.ChainConfig, mux *event.TypeMux, engine consensus.Engine) *Miner {
+func New(eth Backend, config *Config, chainConfig *params.ChainConfig, mux *event.TypeMux, engine consensus.Engine, clock *mockable.Clock) *Miner {
 	return &Miner{
-		worker: newWorker(config, chainConfig, engine, eth, mux),
+		worker: newWorker(config, chainConfig, engine, eth, mux, clock),
 	}
 }
 
diff --git a/miner/worker.go b/miner/worker.go
index 1e6c3df8..40702504 100644
```

```
--- a/miner/worker.go
+++ b/miner/worker.go
@@ -36,16 +36,17 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/coreth/consensus"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/consensus/misc"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/event"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/consensus"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/consensus/misc"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/flare/utils/timer/mockable"
 )

 // environment is the worker's current environment and holds all of the current state information.
@@ -81,9 +82,10 @@ type worker struct {
        mux      *event.TypeMux // TODO replace
        mu       sync.RWMutex   // The lock used to protect the coinbase and extra fields
        coinbase common.Address
+       clock    *mockable.Clock // Allows us mock the clock for testing
 }

-func newWorker(config *Config, chainConfig *params.ChainConfig, engine consensus.Engine, eth Backend, mux *event.TypeMux) *worker {
+func newWorker(config *Config, chainConfig *params.ChainConfig, engine consensus.Engine, eth Backend, mux *event.TypeMux, clock *mockable.Clock) *worker {
        worker := &worker{
                config:      config,
                chainConfig: chainConfig,
@@ -91,6 +93,7 @@ func newWorker(config *Config, chainConfig *params.ChainConfig, engine consensus
                eth:         eth,
                mux:         mux,
                chain:       eth.BlockChain(),
+               clock:       clock,
        }

        return worker
@@ -108,7 +111,7 @@ func (w *worker) commitNewWork() (*types.Block, error) {
        w.mu.RLock()
        defer w.mu.RUnlock()

-       tstart := time.Now()
+       tstart := w.clock.Time()
        timestamp := tstart.Unix()
        parent := w.chain.CurrentBlock()
        // Note: in order to support asynchronous block production, blocks are allowed to have
@@ -119,11 +122,11 @@ func (w *worker) commitNewWork() (*types.Block, error) {
        }

        var gasLimit uint64
-       if w.chainConfig.IsApricotPhase1(big.NewInt(timestamp)) {
+       if w.chainConfig.IsApricotPhase5(big.NewInt(timestamp)) {
+               gasLimit = params.ApricotPhase5GasLimit
+       } else if w.chainConfig.IsApricotPhase1(big.NewInt(timestamp)) {
                gasLimit = params.ApricotPhase1GasLimit
        } else {
-               // The gas limit is set in phase1 to ApricotPhase1GasLimit because the ceiling and floor were set to the same value
-               // such that the gas limit converged to it. Since this is hardbaked now, we remove the ability to configure it.
                gasLimit = core.CalcGasLimit(parent.GasUsed(), parent.GasLimit(), params.ApricotPhase1GasLimit, params.ApricotPhase1GasLimit)
        }
        num := parent.Number()
diff --git a/node/api.go b/node/api.go
index c38b2df2..9f5ce07a 100644
--- a/node/api.go
+++ b/node/api.go
@@ -27,24 +27,28 @@
 package node

 import (
-       "github.com/ava-labs/coreth/internal/debug"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/internal/debug"
+       "github.com/flare-foundation/coreth/rpc"
 )

 // apis returns the collection of built-in RPC APIs.
 func (n *Node) apis() []rpc.API {
-       return []rpc.API{{
-               Namespace: "debug",
-               Version:   "1.0",
-               Service:   debug.Handler,
-       }, {
-               Namespace: "web3",
-               Version:   "1.0",
-               Service:   &publicWeb3API{n},
-               Public:    true,
-       },
+       return []rpc.API{
+               {
+                       Namespace: "debug",
+                       Version:   "1.0",
+                       Service:   debug.Handler,
+                       Name:      "debug-handler",
+               },
+               {
+                       Namespace: "web3",
+                       Version:   "1.0",
+                       Service:   &publicWeb3API{n},
+                       Public:    true,
+                       Name:      "web3",
+               },
        }
 }

diff --git a/node/config.go b/node/config.go
index 5b7be4e3..ab20ff59 100644
--- a/node/config.go
+++ b/node/config.go
@@ -32,11 +32,11 @@ import (
        "os"
        "path/filepath"

-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/accounts/external"
-       "github.com/ava-labs/coreth/accounts/keystore"
-       "github.com/ava-labs/coreth/rpc"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/accounts/external"
+       "github.com/flare-foundation/coreth/accounts/keystore"
+       "github.com/flare-foundation/coreth/rpc"
 )
 // Config represents a small collection of configuration values to fine tune the
diff --git a/node/defaults.go b/node/defaults.go
index e4c826b9..1d8728fb 100644
--- a/node/defaults.go
+++ b/node/defaults.go
@@ -27,7 +27,7 @@
 package node
 
 import (
-       "github.com/ava-labs/coreth/rpc"
+       "github.com/flare-foundation/coreth/rpc"
 )
 
 const (
diff --git a/node/node.go b/node/node.go
index 8d5cc504..819f32a5 100644
--- a/node/node.go
+++ b/node/node.go
@@ -27,59 +27,18 @@
 package node
 
 import (
-       "sync"
-
-       "github.com/ava-labs/coreth/accounts"
-       "github.com/ava-labs/coreth/rpc"
-       "github.com/ethereum/go-ethereum/event"
+       "github.com/flare-foundation/coreth/accounts"
+       "github.com/flare-foundation/coreth/rpc"
 )
 
 // Node is a container on which services can be registered.
 type Node struct {
-       eventmux *event.TypeMux
-       config   *Config
-       accman   *accounts.Manager
-       // log      log.Logger
-       // ephemKeystore string // if non-empty, the key directory that will be removed by Stop
-       // dirLock       fileutil.Releaser // prevents concurrent use of instance directory
-       // stop          chan struct{}     // Channel to wait for termination notifications
-       // server        *p2p.Server // Currently running P2P networking layer
-       // startStopLock sync.Mutex // Start/Stop are protected by an additional lock
-       // state int // Tracks state of node lifecycle
-
-       lock    sync.Mutex
-       rpcAPIs []rpc.API // List of APIs currently provided by the node
-       // inprocHandler *rpc.Server // In-process RPC request handler to process the API requests
-
-       // databases map[*closeTrackingDB]struct{} // All open databases
+       config *Config
+       accman *accounts.Manager
 
        corethVersion string
 }
 
-// const (
-//     initializingState = iota
-//     closedState
-// )
-
-// func (n *Node) openDataDir() error {
-//     if n.config.DataDir == "" {
-//             return nil // ephemeral
-//     }
-//
-//     instdir := filepath.Join(n.config.DataDir, n.config.name())
-//     if err := os.MkdirAll(instdir, 0700); err != nil {
-//             return err
-//     }
-//     // Lock the instance directory to prevent concurrent use by another instance as well as
-//     // accidental use of the instance directory as a database.
-//     release, _, err := fileutil.Flock(filepath.Join(instdir, "LOCK"))
-//     if err != nil {
-//             return convertFileLockError(err)
-//     }
-//     n.dirLock = release
-//     return nil
-// }
-
 // New creates a new P2P node, ready for protocol registration.
 func New(conf *Config) (*Node, error) {
        // Copy config and resolve the datadir so future changes to the current
@@ -87,13 +46,7 @@ func New(conf *Config) (*Node, error) {
        confCopy := *conf
        conf = &confCopy
 
-       node := &Node{
-               config:   conf,
-               eventmux: new(event.TypeMux),
-       }
-
-       // Register built-in APIs.
-       node.rpcAPIs = append(node.rpcAPIs, node.apis()...)
+       node := &Node{config: conf}
 
        // Ensure that the AccountManager method works before the node has started. We rely on
        // this in cmd/geth.
@@ -103,8 +56,6 @@ func New(conf *Config) (*Node, error) {
        }
        node.accman = am
 
-       // Configure RPC servers.
-
        return node, nil
 }
 
@@ -118,16 +69,7 @@ func (n *Node) AccountManager() *accounts.Manager {
        return n.accman
 }
 
-// EventMux retrieves the event multiplexer used by all the network services in
-// the current protocol stack.
-func (n *Node) EventMux() *event.TypeMux {
-       return n.eventmux
-}
-
 // RegisterAPIs registers the APIs a service provides on the node.
-func (n *Node) RegisterAPIs(apis []rpc.API) {
-       n.lock.Lock()
-       defer n.lock.Unlock()
-
-       n.rpcAPIs = append(n.rpcAPIs, apis...)
+func (n *Node) APIs() []rpc.API {
+       return n.apis()
 }
diff --git a/params/avalanche_params.go b/params/avalanche_params.go
index a6124df6..df871bf0 100644
--- a/params/avalanche_params.go
+++ b/params/avalanche_params.go
@@ -4,11 +4,13 @@
 package params
 
 import (
-       "github.com/ava-labs/avalanchego/utils/units"
+       "math/big"
+
+       "github.com/flare-foundation/flare/utils/units"
 )

 // Minimum Gas Price
-var (
+const (
        // MinGasPrice is the number of nAVAX required per gas unit for a
        // transaction to be valid, measured in wei
        LaunchMinGasPrice        int64 = 470_000_000_000
@@ -17,11 +19,30 @@ var (
        AvalancheAtomicTxFee = units.MilliAvax

        ApricotPhase1GasLimit uint64 = 8_000_000
+       ApricotPhase5GasLimit uint64 = 30_000_000
+
+       ApricotPhase3ExtraDataSize                   = 80
+       ApricotPhase3MinBaseFee               int64  = 75_000_000_000
+       ApricotPhase3MaxBaseFee               int64  = 225_000_000_000
+       ApricotPhase3InitialBaseFee           int64  = 225_000_000_000
+       ApricotPhase3TargetGas                uint64 = 10_000_000
+       ApricotPhase4MinBaseFee               int64  = 25_000_000_000
+       ApricotPhase4MaxBaseFee               int64  = 1_000_000_000_000
+       ApricotPhase4BaseFeeChangeDenominator uint64 = 12
+       ApricotPhase5TargetGas                uint64 = 150_000_000
+       ApricotPhase5BaseFeeChangeDenominator uint64 = 36

-       ApricotPhase3ExtraDataSize        = 80
-       ApricotPhase3MinBaseFee     int64 = 75_000_000_000
-       ApricotPhase3MaxBaseFee     int64 = 225_000_000_000
-       ApricotPhase3InitialBaseFee int64 = 225_000_000_000
-       ApricotPhase4MinBaseFee     int64 = 25_000_000_000
-       ApricotPhase4MaxBaseFee     int64 = 1_000_000_000_000
+       // The base cost to charge per atomic transaction. Added in Apricot Phase 5.
+       AtomicTxBaseCost uint64 = 10_000
+)
+
+var (
+       // The atomic gas limit specifies the maximum amount of gas that can be consumed by the atomic
+       // transactions included in a block and is enforced as of ApricotPhase5. Prior to ApricotPhase5,
+       // a block included a single atomic transaction. As of ApricotPhase5, each block can include a set
+       // of atomic transactions where the cumulative atomic gas consumed is capped by the atomic gas limit,
+       // similar to the block gas limit.
+       //
+       // This value must always remain <= MaxUint64.
+       AtomicGasLimit *big.Int = big.NewInt(100_000)
 )
diff --git a/params/config.go b/params/config.go
index 5bb4a55c..596f0cd4 100644
--- a/params/config.go
+++ b/params/config.go
@@ -37,20 +37,22 @@ import (

 // Avalanche ChainIDs
 var (
-       // AvalancheMainnetChainID ...
-       AvalancheMainnetChainID = big.NewInt(43114)
-       // AvalancheFujiChainID ...
-       AvalancheFujiChainID = big.NewInt(43113)
-       // AvalancheLocalChainID ...
-       AvalancheLocalChainID = big.NewInt(43112)
+       // FlareChainID ...
+       FlareChainID = big.NewInt(14)
+       // SongbirdChainID ...
+       SongbirdChainID = big.NewInt(19)
+       // CostonChainID ...
+       CostonChainID = big.NewInt(16)
+       // LocalChainID ...
+       LocalChainID = big.NewInt(4294967295)

        errNonGenesisForkByHeight = errors.New("coreth only supports forking by height at the genesis block")
 )

 var (
-       // AvalancheMainnetChainConfig is the configuration for Avalanche Main Network
-       AvalancheMainnetChainConfig = &ChainConfig{
-               ChainID:                     AvalancheMainnetChainID,
+       // FlareChainConfig is the configuration for Flare main network.
+       FlareChainConfig = &ChainConfig{
+               ChainID:                     FlareChainID,
                HomesteadBlock:              big.NewInt(0),
                DAOForkBlock:                big.NewInt(0),
                DAOForkSupport:              true,
@@ -63,15 +65,16 @@ var (
                PetersburgBlock:             big.NewInt(0),
                IstanbulBlock:               big.NewInt(0),
                MuirGlacierBlock:            big.NewInt(0),
-               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2021, time.March, 31, 14, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2021, time.May, 10, 11, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2021, time.August, 24, 14, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2021, time.September, 22, 21, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase5BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
        }

-       // AvalancheFujiChainConfig is the configuration for the Fuji Test Network
-       AvalancheFujiChainConfig = &ChainConfig{
-               ChainID:                     AvalancheFujiChainID,
+       // SongbirdChainConfig is the configuration for the Songbird canary network.
+       SongbirdChainConfig = &ChainConfig{
+               ChainID:                     SongbirdChainID,
                HomesteadBlock:              big.NewInt(0),
                DAOForkBlock:                big.NewInt(0),
                DAOForkSupport:              true,
@@ -84,15 +87,16 @@ var (
                PetersburgBlock:             big.NewInt(0),
                IstanbulBlock:               big.NewInt(0),
                MuirGlacierBlock:            big.NewInt(0),
-               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2021, time.March, 26, 14, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2021, time.May, 5, 14, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2021, time.August, 16, 19, 0, 0, 0, time.UTC).Unix()),
-               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2021, time.September, 16, 21, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase5BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
        }

-       // AvalancheLocalChainConfig is the configuration for the Avalanche Local Network
-       AvalancheLocalChainConfig = &ChainConfig{
-               ChainID:                     AvalancheLocalChainID,
+       // CostonChainConfig is the configuration for the Coston test network.
+       CostonChainConfig = &ChainConfig{
+               ChainID:                     CostonChainID,
                HomesteadBlock:              big.NewInt(0),
                DAOForkBlock:                big.NewInt(0),
                DAOForkSupport:              true,
@@ -105,18 +109,42 @@ var (
```

```diff
                PetersburgBlock:             big.NewInt(0),
                IstanbulBlock:               big.NewInt(0),
                MuirGlacierBlock:            big.NewInt(0),
-               ApricotPhase1BlockTimestamp: big.NewInt(0),
-               ApricotPhase2BlockTimestamp: big.NewInt(0),
-               ApricotPhase3BlockTimestamp: big.NewInt(0),
-               ApricotPhase4BlockTimestamp: big.NewInt(0),
-       }
-
-       TestChainConfig         = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
-       TestLaunchConfig        = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
-       TestApricotPhase1Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
-       TestApricotPhase2Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
-       TestApricotPhase3Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
-       TestApricotPhase4Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase5BlockTimestamp: big.NewInt(time.Date(2100, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+       }
+
+       // LocalChainConfig is the configuration for the local network.
+       LocalChainConfig = &ChainConfig{
+               ChainID:                     LocalChainID,
+               HomesteadBlock:              big.NewInt(0),
+               DAOForkBlock:                big.NewInt(0),
+               DAOForkSupport:              true,
+               EIP150Block:                 big.NewInt(0),
+               EIP150Hash:                  common.HexToHash("0x2086799aeebeae135c246c65021c82b4e15a2c451340993aacfd2751886514f0"),
+               EIP155Block:                 big.NewInt(0),
+               EIP158Block:                 big.NewInt(0),
+               ByzantiumBlock:              big.NewInt(0),
+               ConstantinopleBlock:         big.NewInt(0),
+               PetersburgBlock:             big.NewInt(0),
+               IstanbulBlock:               big.NewInt(0),
+               MuirGlacierBlock:            big.NewInt(0),
+               ApricotPhase1BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase2BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase3BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase4BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+               ApricotPhase5BlockTimestamp: big.NewInt(time.Date(2000, time.January, 1, 0, 0, 0, 0, time.UTC).Unix()),
+       }
+
+       TestChainConfig         = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestLaunchConfig        = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestApricotPhase1Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestApricotPhase2Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestApricotPhase3Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestApricotPhase4Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
+       TestApricotPhase5Config = &ChainConfig{big.NewInt(1), big.NewInt(0), nil, false, big.NewInt(0), common.Hash{}, big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewInt(0), big.NewIn
        TestRules               = TestChainConfig.AvalancheRules(new(big.Int), new(big.Int))
 )

@@ -155,11 +183,13 @@ type ChainConfig struct {
        ApricotPhase3BlockTimestamp *big.Int `json:"apricotPhase3BlockTimestamp,omitempty"`
        // Apricot Phase 4 introduces the notion of a block fee to the dynamic fee algorithm (nil = no fork, 0 = already activated)
        ApricotPhase4BlockTimestamp *big.Int `json:"apricotPhase4BlockTimestamp,omitempty"`
+       // Apricot Phase 5 introduces a batch of atomic transactions with a maximum atomic gas limit per block. (nil = no fork, 0 = already activated)
+       ApricotPhase5BlockTimestamp *big.Int `json:"apricotPhase5BlockTimestamp,omitempty"`
 }

 // String implements the fmt.Stringer interface.
 func (c *ChainConfig) String() string {
-       return fmt.Sprintf("{ChainID: %v Homestead: %v DAO: %v DAOSupport: %v EIP150: %v EIP155: %v EIP158: %v Byzantium: %v Constantinople: %v Petersburg: %v Istanbul: %v, Muir Glacier: %v, Apricot Phas
+       return fmt.Sprintf("{ChainID: %v Homestead: %v DAO: %v DAOSupport: %v EIP150: %v EIP155: %v EIP158: %v Byzantium: %v Constantinople: %v Petersburg: %v Istanbul: %v, Muir Glacier: %v, Apricot Phas
                c.ChainID,
                c.HomesteadBlock,
                c.DAOForkBlock,
@@ -176,6 +206,7 @@ func (c *ChainConfig) String() string {
                c.ApricotPhase2BlockTimestamp,
                c.ApricotPhase3BlockTimestamp,
                c.ApricotPhase4BlockTimestamp,
+               c.ApricotPhase5BlockTimestamp,
        )
 }

@@ -257,15 +288,22 @@ func (c *ChainConfig) IsApricotPhase4(blockTimestamp *big.Int) bool {
        return isForked(c.ApricotPhase4BlockTimestamp, blockTimestamp)
 }

+// IsApricotPhase5 returns whether [blockTimestamp] represents a block
+// with a timestamp after the Apricot Phase 5 upgrade time.
+func (c *ChainConfig) IsApricotPhase5(blockTimestamp *big.Int) bool {
+       return isForked(c.ApricotPhase5BlockTimestamp, blockTimestamp)
+}
+
 // CheckCompatible checks whether scheduled fork transitions have been imported
 // with a mismatching chain configuration.
-func (c *ChainConfig) CheckCompatible(newcfg *ChainConfig, height uint64) *ConfigCompatError {
+func (c *ChainConfig) CheckCompatible(newcfg *ChainConfig, height uint64, timestamp uint64) *ConfigCompatError {
        bhead := new(big.Int).SetUint64(height)
+       bheadTimestamp := new(big.Int).SetUint64(timestamp)

        // Iterate checkCompatible to find the lowest conflict.
        var lasterr *ConfigCompatError
        for {
-               err := c.checkCompatible(newcfg, bhead)
+               err := c.checkCompatible(newcfg, bhead, bheadTimestamp)
                if err == nil || (lasterr != nil && err.RewindTo == lasterr.RewindTo) {
                        break
                }
@@ -356,48 +394,63 @@ func (c *ChainConfig) CheckConfigForkOrder() error {
        return nil
 }

-func (c *ChainConfig) checkCompatible(newcfg *ChainConfig, head *big.Int) *ConfigCompatError {
-       if isForkIncompatible(c.HomesteadBlock, newcfg.HomesteadBlock, head) {
+func (c *ChainConfig) checkCompatible(newcfg *ChainConfig, headHeight *big.Int, headTimestamp *big.Int) *ConfigCompatError {
+       if isForkIncompatible(c.HomesteadBlock, newcfg.HomesteadBlock, headHeight) {
                return newCompatError("Homestead fork block", c.HomesteadBlock, newcfg.HomesteadBlock)
        }
-       if isForkIncompatible(c.DAOForkBlock, newcfg.DAOForkBlock, head) {
+       if isForkIncompatible(c.DAOForkBlock, newcfg.DAOForkBlock, headHeight) {
                return newCompatError("DAO fork block", c.DAOForkBlock, newcfg.DAOForkBlock)
        }
-       if c.IsDAOFork(head) && c.DAOForkSupport != newcfg.DAOForkSupport {
+       if c.IsDAOFork(headHeight) && c.DAOForkSupport != newcfg.DAOForkSupport {
                return newCompatError("DAO fork support flag", c.DAOForkBlock, newcfg.DAOForkBlock)
        }
-       if isForkIncompatible(c.EIP150Block, newcfg.EIP150Block, head) {
+       if isForkIncompatible(c.EIP150Block, newcfg.EIP150Block, headHeight) {
                return newCompatError("EIP150 fork block", c.EIP150Block, newcfg.EIP150Block)
        }
-       if isForkIncompatible(c.EIP155Block, newcfg.EIP155Block, head) {
+       if isForkIncompatible(c.EIP155Block, newcfg.EIP155Block, headHeight) {
                return newCompatError("EIP155 fork block", c.EIP155Block, newcfg.EIP155Block)
        }
-       if isForkIncompatible(c.EIP158Block, newcfg.EIP158Block, head) {
+       if isForkIncompatible(c.EIP158Block, newcfg.EIP158Block, headHeight) {
                return newCompatError("EIP158 fork block", c.EIP158Block, newcfg.EIP158Block)
        }
-       if c.IsEIP158(head) && !configNumEqual(c.ChainID, newcfg.ChainID) {
+       if c.IsEIP158(headHeight) && !configNumEqual(c.ChainID, newcfg.ChainID) {
```

```
                return newCompatError("EIP158 chain ID", c.EIP158Block, newcfg.EIP158Block)
        }
-       if isForkIncompatible(c.ByzantiumBlock, newcfg.ByzantiumBlock, head) {
+       if isForkIncompatible(c.ByzantiumBlock, newcfg.ByzantiumBlock, headHeight) {
                return newCompatError("Byzantium fork block", c.ByzantiumBlock, newcfg.ByzantiumBlock)
        }
-       if isForkIncompatible(c.ConstantinopleBlock, newcfg.ConstantinopleBlock, head) {
+       if isForkIncompatible(c.ConstantinopleBlock, newcfg.ConstantinopleBlock, headHeight) {
                return newCompatError("Constantinople fork block", c.ConstantinopleBlock, newcfg.ConstantinopleBlock)
        }
-       if isForkIncompatible(c.PetersburgBlock, newcfg.PetersburgBlock, head) {
+       if isForkIncompatible(c.PetersburgBlock, newcfg.PetersburgBlock, headHeight) {
                // the only case where we allow Petersburg to be set in the past is if it is equal to Constantinople
                // mainly to satisfy fork ordering requirements which state that Petersburg fork be set if Constantinople fork is set
-               if isForkIncompatible(c.ConstantinopleBlock, newcfg.PetersburgBlock, head) {
+               if isForkIncompatible(c.ConstantinopleBlock, newcfg.PetersburgBlock, headHeight) {
                        return newCompatError("Petersburg fork block", c.PetersburgBlock, newcfg.PetersburgBlock)
                }
        }
-       if isForkIncompatible(c.IstanbulBlock, newcfg.IstanbulBlock, head) {
+       if isForkIncompatible(c.IstanbulBlock, newcfg.IstanbulBlock, headHeight) {
                return newCompatError("Istanbul fork block", c.IstanbulBlock, newcfg.IstanbulBlock)
        }
-       if isForkIncompatible(c.MuirGlacierBlock, newcfg.MuirGlacierBlock, head) {
+       if isForkIncompatible(c.MuirGlacierBlock, newcfg.MuirGlacierBlock, headHeight) {
                return newCompatError("Muir Glacier fork block", c.MuirGlacierBlock, newcfg.MuirGlacierBlock)
        }
-       // TODO(aaronbuchwald) ensure that Avalanche Blocktimestamps are not modified
+       if isForkIncompatible(c.ApricotPhase1BlockTimestamp, newcfg.ApricotPhase1BlockTimestamp, headTimestamp) {
+               return newCompatError("ApricotPhase1 fork block timestamp", c.ApricotPhase1BlockTimestamp, newcfg.ApricotPhase1BlockTimestamp)
+       }
+       if isForkIncompatible(c.ApricotPhase2BlockTimestamp, newcfg.ApricotPhase2BlockTimestamp, headTimestamp) {
+               return newCompatError("ApricotPhase2 fork block timestamp", c.ApricotPhase2BlockTimestamp, newcfg.ApricotPhase2BlockTimestamp)
+       }
+       if isForkIncompatible(c.ApricotPhase3BlockTimestamp, newcfg.ApricotPhase3BlockTimestamp, headTimestamp) {
+               return newCompatError("ApricotPhase3 fork block timestamp", c.ApricotPhase3BlockTimestamp, newcfg.ApricotPhase3BlockTimestamp)
+       }
+       if isForkIncompatible(c.ApricotPhase4BlockTimestamp, newcfg.ApricotPhase4BlockTimestamp, headTimestamp) {
+               return newCompatError("ApricotPhase4 fork block timestamp", c.ApricotPhase4BlockTimestamp, newcfg.ApricotPhase4BlockTimestamp)
+       }
+       if isForkIncompatible(c.ApricotPhase5BlockTimestamp, newcfg.ApricotPhase5BlockTimestamp, headTimestamp) {
+               return newCompatError("ApricotPhase5 fork block timestamp", c.ApricotPhase5BlockTimestamp, newcfg.ApricotPhase5BlockTimestamp)
+       }
+
        return nil
 }

@@ -467,10 +520,7 @@ type Rules struct {
        IsByzantium, IsConstantinople, IsPetersburg, IsIstanbul bool

        // Rules for Avalanche releases
-       IsApricotPhase1 bool
-       IsApricotPhase2 bool
-       IsApricotPhase3 bool
-       IsApricotPhase4 bool
+       IsApricotPhase1, IsApricotPhase2, IsApricotPhase3, IsApricotPhase4, IsApricotPhase5 bool
 }

 // Rules ensures c's ChainID is not nil.
@@ -501,5 +551,6 @@ func (c *ChainConfig) AvalancheRules(blockNum, blockTimestamp *big.Int) Rules {
        rules.IsApricotPhase2 = c.IsApricotPhase2(blockTimestamp)
        rules.IsApricotPhase3 = c.IsApricotPhase3(blockTimestamp)
        rules.IsApricotPhase4 = c.IsApricotPhase4(blockTimestamp)
+       rules.IsApricotPhase5 = c.IsApricotPhase5(blockTimestamp)
        return rules
 }
diff --git a/params/config_test.go b/params/config_test.go
new file mode 100644
index 00000000..62fea4be
--- /dev/null
+++ b/params/config_test.go
@@ -0,0 +1,138 @@
+// (c) 2019-2020, Ava Labs, Inc.
+//
+// This file is a derived work, based on the go-ethereum library whose original
+// notices appear below.
+//
+// It is distributed under a license compatible with the licensing terms of the
+// original code from which it is derived.
+//
+// Much love to the original authors for their work.
+// **********
+// Copyright 2017 The go-ethereum Authors
+// This file is part of the go-ethereum library.
+//
+// The go-ethereum library is free software: you can redistribute it and/or modify
+// it under the terms of the GNU Lesser General Public License as published by
+// the Free Software Foundation, either version 3 of the License, or
+// (at your option) any later version.
+//
+// The go-ethereum library is distributed in the hope that it will be useful,
+// but WITHOUT ANY WARRANTY; without even the implied warranty of
+// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
+// GNU Lesser General Public License for more details.
+//
+// You should have received a copy of the GNU Lesser General Public License
+// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
+
+package params
+
+import (
+       "math/big"
+       "reflect"
+       "testing"
+)
+
+func TestCheckCompatible(t *testing.T) {
+       type test struct {
+               stored, new              *ChainConfig
+               headHeight, headTimestamp uint64
+               wantErr                  *ConfigCompatError
+       }
+       tests := []test{
+               {stored: TestChainConfig, new: TestChainConfig, headHeight: 0, headTimestamp: 0, wantErr: nil},
+               {stored: TestChainConfig, new: TestChainConfig, headHeight: 100, headTimestamp: 1000, wantErr: nil},
+               {
+                       stored:        &ChainConfig{EIP150Block: big.NewInt(10)},
+                       new:           &ChainConfig{EIP150Block: big.NewInt(20)},
+                       headHeight:    9,
+                       headTimestamp: 90,
+                       wantErr:       nil,
+               },
+               {
+                       stored:        TestChainConfig,
+                       new:           &ChainConfig{HomesteadBlock: nil},
+                       headHeight:    3,
+                       headTimestamp: 30,
+                       wantErr: &ConfigCompatError{
+                               What:         "Homestead fork block",
+                               StoredConfig: big.NewInt(0),
+                               NewConfig:    nil,
+                               RewindTo:     0,
+                       },
+               },
+		{
+			stored:        TestChainConfig,
+			new:           &ChainConfig{HomesteadBlock: big.NewInt(1)},
+			headHeight:    3,
+			headTimestamp: 30,
+			wantErr: &ConfigCompatError{
+				What:         "Homestead fork block",
+				StoredConfig: big.NewInt(0),
+				NewConfig:    big.NewInt(1),
+				RewindTo:     0,
+			},
+		},
+		{
+			stored:        &ChainConfig{HomesteadBlock: big.NewInt(30), EIP150Block: big.NewInt(10)},
+			new:           &ChainConfig{HomesteadBlock: big.NewInt(25), EIP150Block: big.NewInt(20)},
+			headHeight:    25,
+			headTimestamp: 250,
+			wantErr: &ConfigCompatError{
+				What:         "EIP150 fork block",
+				StoredConfig: big.NewInt(10),
+				NewConfig:    big.NewInt(20),
+				RewindTo:     9,
+			},
+		},
+		{
+			stored:        &ChainConfig{ConstantinopleBlock: big.NewInt(30)},
+			new:           &ChainConfig{ConstantinopleBlock: big.NewInt(30), PetersburgBlock: big.NewInt(30)},
+			headHeight:    40,
+			headTimestamp: 400,
+			wantErr:       nil,
+		},
+		{
+			stored:        &ChainConfig{ConstantinopleBlock: big.NewInt(30)},
+			new:           &ChainConfig{ConstantinopleBlock: big.NewInt(30), PetersburgBlock: big.NewInt(31)},
+			headHeight:    40,
+			headTimestamp: 400,
+			wantErr: &ConfigCompatError{
+				What:         "Petersburg fork block",
+				StoredConfig: nil,
+				NewConfig:    big.NewInt(31),
+				RewindTo:     30,
+			},
+		},
+		{
+			stored:        TestChainConfig,
+			new:           TestApricotPhase4Config,
+			headHeight:    0,
+			headTimestamp: 0,
+			wantErr: &ConfigCompatError{
+				What:         "ApricotPhase5 fork block timestamp",
+				StoredConfig: big.NewInt(0),
+				NewConfig:    nil,
+				RewindTo:     0,
+			},
+		},
+		{
+			stored:        TestChainConfig,
+			new:           TestApricotPhase4Config,
+			headHeight:    10,
+			headTimestamp: 100,
+			wantErr: &ConfigCompatError{
+				What:         "ApricotPhase5 fork block timestamp",
+				StoredConfig: big.NewInt(0),
+				NewConfig:    nil,
+				RewindTo:     0,
+			},
+		},
+	}
+
+	for _, test := range tests {
+		err := test.stored.CheckCompatible(test.new, test.headHeight, test.headTimestamp)
+		if !reflect.DeepEqual(err, test.wantErr) {
+			t.Errorf("error mismatch:\nstored: %v\nnew: %v\nheadHeight: %v\nerr: %v\nwant: %v", test.stored, test.new, test.headHeight, err, test.wantErr)
+		}
+	}
+}
diff --git a/params/protocol_params.go b/params/protocol_params.go
index 00db2bfd..2137387a 100644
--- a/params/protocol_params.go
+++ b/params/protocol_params.go
@@ -29,9 +29,10 @@ package params
 import "math/big"
 
 const (
-	GasLimitBoundDivisor uint64 = 1024    // The bound divisor of the gas limit, used in update calculations.
-	MinGasLimit          uint64 = 5000    // Minimum the gas limit may ever be.
-	GenesisGasLimit      uint64 = 4712388 // Gas limit of the Genesis block.
+	GasLimitBoundDivisor uint64 = 1024               // The bound divisor of the gas limit, used in update calculations.
+	MinGasLimit          uint64 = 5000               // Minimum the gas limit may ever be.
+	MaxGasLimit          uint64 = 0x7fffffffffffffff // Maximum the gas limit (2^63-1).
+	GenesisGasLimit      uint64 = 4712388            // Gas limit of the Genesis block.
 
 	// Note: MaximumExtraDataSize has been reduced to 32 in Geth, but is kept the same in Coreth for
 	// backwards compatibility.
@@ -47,8 +48,8 @@ const (
 	LogDataGas            uint64 = 8     // Per byte in a LOG* operation's data.
 	CallStipend           uint64 = 2300  // Free gas given at beginning of call.
 
-	Sha3Gas     uint64 = 30 // Once per SHA3 operation.
-	Sha3WordGas uint64 = 6  // Once per word of the SHA3 operation's data.
+	Keccak256Gas     uint64 = 30 // Once per KECCAK256 operation.
+	Keccak256WordGas uint64 = 6  // Once per word of the KECCAK256 operation's data.
 
 	SstoreSetGas    uint64 = 20000 // Once per SSTORE operation.
 	SstoreResetGas  uint64 = 5000  // Once per SSTORE operation if the zeroness changes from zero.
@@ -130,8 +131,6 @@ const (
 	// Introduced in Tangerine Whistle (Eip 150)
 	CreateBySelfdestructGas uint64 = 25000
 
-	BaseFeeChangeDenominator = 12 // Bounds the amount the base fee can change between blocks.
-
 	MaxCodeSize = 24576 // Maximum bytecode to permit for a contract
 
 	// Precompiled contract gas prices
diff --git a/params/version.go b/params/version.go
index 22334a6a..a6df0da8 100644
--- a/params/version.go
+++ b/params/version.go
@@ -33,7 +33,7 @@ import (
 const (
 	VersionMajor = 1        // Major version component of the current release
 	VersionMinor = 10       // Minor version component of the current release
-	VersionPatch = 12       // Patch version component of the current release
+	VersionPatch = 15       // Patch version component of the current release
 	VersionMeta  = "stable" // Version metadata to append to the version string
 )
 
diff --git a/peer/client.go b/peer/client.go
new file mode 100644
index 00000000..6b46f7ca
--- /dev/null
+++ b/peer/client.go
@@ -0,0 +1,52 @@
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package peer
+
+import (
+       "github.com/flare-foundation/flare/version"
+)
+
+var _ Client = &client{}
+
+// Client defines ability to send request / response through the Network
+type Client interface {
+       // RequestAny synchronously sends request to the first connected peer that matches the specified minVersion in
+       // random order.
+       // A peer is considered a match if its version is greater than or equal to the specified minVersion
+       // Returns errNoPeersMatchingVersion if no peer could be found matching specified version
+       RequestAny(minVersion version.Application, request []byte) ([]byte, bool, error)
+
+       // Gossip sends given gossip message to peers
+       Gossip(gossip []byte) error
+}
+
+// client implements Client interface
+// provides ability to send request / responses through the Network
+type client struct {
+       network Network
+}
+
+// RequestAny synchronously sends request to the first connected peer that matches the specified minVersion in
+// random order.
+// Returns response bytes, whether the request failed and optional error
+// Returns errNoPeersMatchingVersion if no peer could be found matching specified version
+// This function is blocks until a response is received from the peer
+func (c *client) RequestAny(minVersion version.Application, request []byte) ([]byte, bool, error) {
+       waitingHandler := newWaitingResponseHandler()
+       if err := c.network.RequestAny(minVersion, request, waitingHandler); err != nil {
+               return nil, true, err
+       }
+       return <-waitingHandler.responseChan, waitingHandler.failed, nil
+}
+
+func (c *client) Gossip(gossip []byte) error {
+       return c.network.Gossip(gossip)
+}
+
+// NewClient returns Client for a given network
+func NewClient(network Network) Client {
+       return &client{
+               network: network,
+       }
+}
diff --git a/peer/network.go b/peer/network.go
new file mode 100644
index 00000000..700f4bb2
--- /dev/null
+++ b/peer/network.go
@@ -0,0 +1,343 @@
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package peer
+
+import (
+       "context"
+       "errors"
+       "fmt"
+       "sync"
+       "time"
+
+       "github.com/flare-foundation/coreth/plugin/evm/message"
+
+       "github.com/flare-foundation/flare/snow/validators"
+
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow/engine/common"
+       "github.com/flare-foundation/flare/version"
+       "golang.org/x/sync/semaphore"
+)
+
+// Minimum amount of time to handle a request
+const minRequestHandlingDuration = 100 * time.Millisecond
+
+var (
+       errAcquiringSemaphore                      = errors.New("error acquiring semaphore")
+       _                     Network              = &network{}
+       _                     validators.Connector = &network{}
+       _                     common.AppHandler    = &network{}
+)
+
+type Network interface {
+       validators.Connector
+       common.AppHandler
+
+       // RequestAny synchronously sends request to the first connected peer that matches the specified minVersion in
+       // random order.
+       // A peer is considered a match if its version is greater than or equal to the specified minVersion
+       // Returns errNoPeersMatchingVersion if no peer could be found matching specified version
+       RequestAny(minVersion version.Application, message []byte, handler message.ResponseHandler) error
+
+       // Gossip sends given gossip message to peers
+       Gossip(gossip []byte) error
+
+       // Shutdown stops all peer channel listeners and marks the node to have stopped
+       // n.Start() can be called again but the peers will have to be reconnected
+       // by calling OnPeerConnected for each peer
+       Shutdown()
+
+       // SetGossipHandler sets the provided gossip handler as the gossip handler
+       SetGossipHandler(handler message.GossipHandler)
+
+       // SetRequestHandler sets the provided request handler as the request handler
+       SetRequestHandler(handler message.RequestHandler)
+
+       // Size returns the size of the network in number of connected peers
+       Size() uint32
+}
+
+// network is an implementation of Network that processes message requests for
+// each peer in linear fashion
+type network struct {
+       lock                          sync.RWMutex                        // lock for mutating state of this Network struct
+       self                          ids.ShortID                         // NodeID of this node
+       requestIDGen                  uint32                              // requestID counter used to track outbound requests
+       outstandingResponseHandlerMap map[uint32]message.ResponseHandler  // maps avalanchego requestID => response handler
+       activeRequests                *semaphore.Weighted                 // controls maximum number of active outbound requests
+       appSender                     common.AppSender                    // avalanchego AppSender for sending messages
+       codec                         codec.Manager                       // Codec used for parsing messages
+       requestHandler                message.RequestHandler              // maps request type => handler
+       gossipHandler                 message.GossipHandler               // maps gossip type => handler
+       peers                         map[ids.ShortID]version.Application // maps nodeID => version.Version
+}
+
+func NewNetwork(appSender common.AppSender, codec codec.Manager, self ids.ShortID, maxActiveRequests int64) Network {
+       return &network{
+               appSender:                     appSender,
+               codec:                         codec,
+               self:                          self,
+               outstandingResponseHandlerMap: make(map[uint32]message.ResponseHandler),
+               peers:                         make(map[ids.ShortID]version.Application),
+               activeRequests:                semaphore.NewWeighted(maxActiveRequests),
+       }
+}
+
+// RequestAny sends given request to the first connected peer that matches the specified minVersion
+// A peer is considered a match if its version is greater than or equal to the specified minVersion
+// If minVersion is nil, then the request will be sent to any peer regardless of their version
+// Returns a non-nil error if we were not able to send a request to a peer with >= [minVersion]
+// or we fail to send a request to the selected peer.
+func (n *network) RequestAny(minVersion version.Application, request []byte, handler message.ResponseHandler) error {
+       // Take a slot from total [activeRequests] and block until a slot becomes available.
+       if err := n.activeRequests.Acquire(context.Background(), 1); err != nil {
+               return errAcquiringSemaphore
+       }
+
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       for nodeID, nodeVersion := range n.peers {
+               // map iteration is sufficiently random to avoid hitting same peer so here
+               // we get a random peerID key that we compare minimum version that
+               // we have
+               if minVersion == nil || nodeVersion.Compare(minVersion) >= 0 {
+                       return n.request(nodeID, request, handler)
+               }
+       }
+
+       n.activeRequests.Release(1)
+       return fmt.Errorf("no peers found matching version %s out of %d peers", minVersion, len(n.peers))
+}
+
+// Request sends request message bytes to specified nodeID and adds [responseHandler] to [outstandingResponseHandlerMap]
+// so that it can be invoked when the network receives either a response or failure message.
+// Assumes [nodeID] is never [self] since we guarantee [self] will not be added to the [peers] map.
+// Returns an error if [appSender] is unable to make the request.
+// Assumes write lock is held
+func (n *network) request(nodeID ids.ShortID, request []byte, responseHandler message.ResponseHandler) error {
+       log.Debug("sending request to peer", "nodeID", nodeID, "requestLen", len(request))
+
+       // generate requestID
+       requestID := n.requestIDGen
+       n.requestIDGen++
+
+       n.outstandingResponseHandlerMap[requestID] = responseHandler
+
+       nodeIDs := ids.NewShortSet(1)
+       nodeIDs.Add(nodeID)
+
+       // send app request to the peer
+       // on failure: release the activeRequests slot, mark message as processed and return fatal error
+       // Send app request to [nodeID].
+       // On failure, release the slot from active requests and [outstandingResponseHandlerMap].
+       if err := n.appSender.SendAppRequest(nodeIDs, requestID, request); err != nil {
+               n.activeRequests.Release(1)
+               delete(n.outstandingResponseHandlerMap, requestID)
+               log.Error("could not send app message", "err", err, "nodeID", nodeID, "requestID", requestID)
+               return err
+       }
+
+       log.Debug("sent request message to peer", "nodeID", nodeID, "requestID", requestID)
+       return nil
+}
+
+// AppRequest is called by avalanchego -> VM when there is an incoming AppRequest from a peer
+// error returned by this function is expected to be treated as fatal by the engine
+// returns error if the requestHandler returns an error
+// sends a response back to the sender if length of response returned by the handler is >0
+// expects the deadline to not have been passed
+func (n *network) AppRequest(nodeID ids.ShortID, requestID uint32, deadline time.Time, request []byte) error {
+       n.lock.RLock()
+       defer n.lock.RUnlock()
+
+       log.Debug("received AppRequest from node", "nodeID", nodeID, "requestID", requestID, "requestLen", len(request))
+
+       var req message.Request
+       if _, err := n.codec.Unmarshal(request, &req); err != nil {
+               log.Debug("failed to unmarshal app request", "nodeID", nodeID, "requestID", requestID, "requestLen", len(request), "err", err)
+               return nil
+       }
+
+       // calculate how much time is left until the deadline
+       timeTillDeadline := time.Until(deadline)
+
+       // bufferedDeadline is half the time till actual deadline so that the message has a reasonable chance
+       // of completing its processing and sending the response to the peer.
+       timeTillDeadline = time.Duration(timeTillDeadline.Nanoseconds() / 2)
+       bufferedDeadline := time.Now().Add(timeTillDeadline)
+
+       // check if we have enough time to handle this request
+       if time.Until(bufferedDeadline) < minRequestHandlingDuration {
+               // Drop the request if we already missed the deadline to respond.
+               log.Debug("deadline to process AppRequest has expired, skipping", "nodeID", nodeID, "requestID", requestID, "type", req.Type())
+               return nil
+       }
+
+       log.Debug("processing incoming request", "nodeID", nodeID, "requestID", requestID, "type", req.Type())
+       ctx, cancel := context.WithDeadline(context.Background(), bufferedDeadline)
+       defer cancel()
+
+       responseBytes, err := req.Handle(ctx, nodeID, requestID, n.requestHandler)
+       switch {
+       case err != nil && err != context.DeadlineExceeded:
+               return err // Return a fatal error
+       case responseBytes != nil:
+               return n.appSender.SendAppResponse(nodeID, requestID, responseBytes) // Propagate fatal error
+       default:
+               return nil
+       }
+}
+
+// AppResponse is invoked when there is a response received from a peer regarding a request
+// Error returned by this function is expected to be treated as fatal by the engine
+// If [requestID] is not known, this function will emit a log and return a nil error.
+// If the response handler returns an error it is propagated as a fatal error.
+func (n *network) AppResponse(nodeID ids.ShortID, requestID uint32, response []byte) error {
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       log.Debug("received AppResponse from peer", "nodeID", nodeID, "requestID", requestID)
+
+       handler, exists := n.getRequestHandler(requestID)
+       if !exists {
+               // Should never happen since the engine should be managing outstanding requests
+               log.Error("received response to unknown request", "nodeID", nodeID, "requestID", requestID, "responseLen", len(response))
+               return nil
+       }
+
+       return handler.OnResponse(nodeID, requestID, response)
+}
+
+// AppRequestFailed can be called by the avalanchego -> VM in following cases:
+// - node is benched
+// - failed to send message to [nodeID] due to a network issue
+// - timeout
+// error returned by this function is expected to be treated as fatal by the engine
+// returns error only when the response handler returns an error
+func (n *network) AppRequestFailed(nodeID ids.ShortID, requestID uint32) error {
+       n.lock.Lock()
+       defer n.lock.Unlock()
+       log.Debug("received AppRequestFailed from peer", "nodeID", nodeID, "requestID", requestID)
+
+       handler, exists := n.getRequestHandler(requestID)
+       if !exists {
+               // Should never happen since the engine should be managing outstanding requests
+               log.Error("received request failed to unknown request", "nodeID", nodeID, "requestID", requestID)
+               return nil
+       }
+
+       return handler.OnFailure(nodeID, requestID)
+}
+
+// getRequestHandler fetches the handler for [requestID] and marks the request with [requestID] as having been fulfilled.
+// This is called by either [AppResponse] or [AppRequestFailed].
+// assumes that the write lock is held.
+func (n *network) getRequestHandler(requestID uint32) (message.ResponseHandler, bool) {
+       handler, exists := n.outstandingResponseHandlerMap[requestID]
+       if !exists {
+               return nil, false
+       }
+       // mark message as processed, release activeRequests slot
+       delete(n.outstandingResponseHandlerMap, requestID)
+       n.activeRequests.Release(1)
+       return handler, true
+}
+
+// Gossip sends given gossip message to peers
+func (n *network) Gossip(gossip []byte) error {
+       return n.appSender.SendAppGossip(gossip)
+}
+
+// AppGossip is called by avalanchego -> VM when there is an incoming AppGossip from a peer
+// error returned by this function is expected to be treated as fatal by the engine
+// returns error if request could not be parsed as message.Request or when the requestHandler returns an error
+func (n *network) AppGossip(nodeID ids.ShortID, gossipBytes []byte) error {
+       var gossipMsg message.Message
+       if _, err := n.codec.Unmarshal(gossipBytes, &gossipMsg); err != nil {
+               log.Debug("could not parse app gossip", "nodeID", nodeID, "gossipLen", len(gossipBytes), "err", err)
+               return nil
+       }
+
+       log.Debug("processing AppGossip from node", "nodeID", nodeID, "type", gossipMsg.Type(), "gossipLen", len(gossipBytes))
+       return gossipMsg.Handle(n.gossipHandler, nodeID)
+}
+
+// Connected adds the given nodeID to the peer list so that it can receive messages
+func (n *network) Connected(nodeID ids.ShortID, nodeVersion version.Application) error {
+       log.Debug("adding new peer", "nodeID", nodeID)
+
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       if nodeID == n.self {
+               log.Debug("skipping registering self as peer")
+               return nil
+       }
+
+       if storedVersion, exists := n.peers[nodeID]; exists {
+               // Peer is already connected, update the version if it has changed.
+               // Log a warning message since the consensus engine should never call Connected on a peer
+               // that we have already marked as Connected.
+               if nodeVersion.Compare(storedVersion) != 0 {
+                       n.peers[nodeID] = nodeVersion
+                       log.Warn("received Connected message for already connected peer, updating node version", "nodeID", nodeID, "storedVersion", storedVersion, "nodeVersion", nodeVersion)
+               } else {
+                       log.Warn("ignoring peer connected event for already connected peer with identical version", "nodeID", nodeID)
+               }
+               return nil
+       }
+
+       n.peers[nodeID] = nodeVersion
+       return nil
+}
+
+// Disconnected removes given [nodeID] from the peer list
+func (n *network) Disconnected(nodeID ids.ShortID) error {
+       log.Debug("disconnecting peer", "nodeID", nodeID)
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       // if this peer already exists, log a warning and ignore the request
+       if _, exists := n.peers[nodeID]; !exists {
+               // we're not connected to this peer, nothing to do here
+               log.Warn("received peer disconnect request to unconnected peer", "nodeID", nodeID)
+               return nil
+       }
+
+       delete(n.peers, nodeID)
+       return nil
+}
+
+// Shutdown disconnects all peers
+func (n *network) Shutdown() {
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       // reset peers map
+       n.peers = make(map[ids.ShortID]version.Application)
+}
+
+func (n *network) SetGossipHandler(handler message.GossipHandler) {
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       n.gossipHandler = handler
+}
+
+func (n *network) SetRequestHandler(handler message.RequestHandler) {
+       n.lock.Lock()
+       defer n.lock.Unlock()
+
+       n.requestHandler = handler
+}
+
+func (n *network) Size() uint32 {
+       n.lock.RLock()
+       defer n.lock.RUnlock()
+
+       return uint32(len(n.peers))
+}
diff --git a/peer/network_test.go b/peer/network_test.go
new file mode 100644
```

```
index 00000000..d6f8a072
--- /dev/null
+++ b/peer/network_test.go
@@ -0,0 +1,484 @@
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package peer
+
+import (
+	"context"
+	"errors"
+	"fmt"
+	"sync"
+	"sync/atomic"
+	"testing"
+	"time"
+
+	"github.com/flare-foundation/flare/snow/engine/common"
+
+	"github.com/flare-foundation/coreth/plugin/evm/message"
+
+	"github.com/flare-foundation/flare/codec"
+	"github.com/flare-foundation/flare/codec/linearcodec"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/version"
+	"github.com/stretchr/testify/assert"
+)
+
+var (
+	defaultPeerVersion = version.NewDefaultApplication("corethtest", 1, 0, 0)
+
+	_ message.Request = &HelloRequest{}
+	_                 = &HelloResponse{}
+	_                 = &GreetingRequest{}
+	_                 = &GreetingResponse{}
+	_                 = &TestMessage{}
+
+	_ message.RequestHandler = &HelloGreetingRequestHandler{}
+	_ message.RequestHandler = &testRequestHandler{}
+
+	_ common.AppSender      = testAppSender{}
+	_ message.Message       = HelloGossip{}
+	_ message.GossipHandler = &testGossipHandler{}
+)
+
+func TestNetworkDoesNotConnectToItself(t *testing.T) {
+	selfNodeID := ids.GenerateTestShortID()
+	n := NewNetwork(nil, nil, selfNodeID, 1)
+	assert.NoError(t, n.Connected(selfNodeID, version.NewDefaultApplication("avalanchego", 1, 0, 0)))
+	assert.EqualValues(t, 0, n.Size())
+}
+
+func TestRequestsRoutingAndResponse(t *testing.T) {
+	callNum := uint32(0)
+	senderWg := &sync.WaitGroup{}
+	var net Network
+	sender := testAppSender{
+		sendAppRequestFn: func(nodes ids.ShortSet, requestID uint32, requestBytes []byte) error {
+			nodeID, _ := nodes.Pop()
+			senderWg.Add(1)
+			go func() {
+				defer senderWg.Done()
+				if err := net.AppRequest(nodeID, requestID, time.Now().Add(5*time.Second), requestBytes); err != nil {
+					panic(err)
+				}
+			}()
+			return nil
+		},
+		sendAppResponseFn: func(nodeID ids.ShortID, requestID uint32, responseBytes []byte) error {
+			senderWg.Add(1)
+			go func() {
+				defer senderWg.Done()
+				if err := net.AppResponse(nodeID, requestID, responseBytes); err != nil {
+					panic(err)
+				}
+				atomic.AddUint32(&callNum, 1)
+			}()
+			return nil
+		},
+	}
+
+	codecManager := buildCodec(t, HelloRequest{}, HelloResponse{})
+	net = NewNetwork(sender, codecManager, ids.ShortEmpty, 16)
+	net.SetRequestHandler(&HelloGreetingRequestHandler{codec: codecManager})
+	client := NewClient(net)
+	nodeID := ids.GenerateTestShortID()
+	assert.NoError(t, net.Connected(nodeID, defaultPeerVersion))
+
+	requestMessage := HelloRequest{Message: "this is a request"}
+
+	defer net.Shutdown()
+	assert.NoError(t, net.Connected(nodeID, defaultPeerVersion))
+
+	totalRequests := 5000
+	numCallsPerRequest := 1 // on sending response
+	totalCalls := totalRequests * numCallsPerRequest
+
+	requestWg := &sync.WaitGroup{}
+	requestWg.Add(totalCalls)
+	for i := 0; i < totalCalls; i++ {
+		go func(wg *sync.WaitGroup) {
+			defer wg.Done()
+			requestBytes, err := message.RequestToBytes(codecManager, requestMessage)
+			assert.NoError(t, err)
+			responseBytes, failed, err := client.RequestAny(defaultPeerVersion, requestBytes)
+			assert.NoError(t, err)
+			assert.False(t, failed)
+			assert.NotNil(t, responseBytes)
+
+			var response TestMessage
+			if _, err = codecManager.Unmarshal(responseBytes, &response); err != nil {
+				panic(fmt.Errorf("unexpected error during unmarshal: %w", err))
+			}
+			assert.Equal(t, "Hi", response.Message)
+		}(requestWg)
+	}
+
+	requestWg.Wait()
+	senderWg.Wait()
+	assert.Equal(t, totalCalls, int(atomic.LoadUint32(&callNum)))
+}
+
+func TestRequestMinVersion(t *testing.T) {
+	callNum := uint32(0)
+	nodeID := ids.GenerateTestShortID()
+	codecManager := buildCodec(t, TestMessage{})
+
+	var net Network
+	sender := testAppSender{
+		sendAppRequestFn: func(nodes ids.ShortSet, reqID uint32, messageBytes []byte) error {
+			atomic.AddUint32(&callNum, 1)
+			assert.True(t, nodes.Contains(nodeID), "request nodes should contain expected nodeID")
+			assert.Len(t, nodes, 1, "request nodes should contain exactly one node")
+
+                       go func() {
+                               time.Sleep(200 * time.Millisecond)
+                               atomic.AddUint32(&callNum, 1)
+                               responseBytes, err := codecManager.Marshal(message.Version, TestMessage{Message: "this is a response"})
+                               if err != nil {
+                                       panic(err)
+                               }
+                               err = net.AppResponse(nodeID, reqID, responseBytes)
+                               assert.NoError(t, err)
+                       }()
+                       return nil
+               },
+       }
+
+       // passing nil as codec works because the net.AppRequest is never called
+       net = NewNetwork(sender, codecManager, ids.ShortEmpty, 1)
+       client := NewClient(net)
+       requestMessage := TestMessage{Message: "this is a request"}
+       requestBytes, err := message.RequestToBytes(codecManager, requestMessage)
+       assert.NoError(t, err)
+       assert.NoError(t, net.Connected(nodeID, version.NewDefaultApplication("corethtest", 1, 7, 1)))
+
+       // ensure version does not match
+       responseBytes, failed, err := client.RequestAny(version.NewDefaultApplication("corethtest", 2, 0, 0), requestBytes)
+       assert.Equal(t, err.Error(), "no peers found matching version corethtest/2.0.0 out of 1 peers")
+       assert.True(t, failed)
+       assert.Nil(t, responseBytes)
+
+       // ensure version matches and the request goes through
+       responseBytes, failed, err = client.RequestAny(version.NewDefaultApplication("corethtest", 1, 0, 0), requestBytes)
+       assert.NoError(t, err)
+       assert.False(t, failed)
+
+       var response TestMessage
+       if _, err = codecManager.Unmarshal(responseBytes, &response); err != nil {
+               t.Fatal("unexpected error during unmarshal", err)
+       }
+       assert.Equal(t, "this is a response", response.Message)
+}
+
+func TestOnRequestHonoursDeadline(t *testing.T) {
+       var net Network
+       responded := false
+       sender := testAppSender{
+               sendAppRequestFn: func(nodes ids.ShortSet, reqID uint32, message []byte) error {
+                       return nil
+               },
+               sendAppResponseFn: func(nodeID ids.ShortID, reqID uint32, message []byte) error {
+                       responded = true
+                       return nil
+               },
+       }
+
+       codecManager := buildCodec(t, TestMessage{})
+
+       requestBytes, err := marshalStruct(codecManager, TestMessage{Message: "hello there"})
+       assert.NoError(t, err)
+
+       requestHandler := &testRequestHandler{
+               processingDuration: 500 * time.Millisecond,
+       }
+       net = NewNetwork(sender, codecManager, ids.ShortEmpty, 1)
+       net.SetRequestHandler(requestHandler)
+       nodeID := ids.GenerateTestShortID()
+
+       requestHandler.response, err = marshalStruct(codecManager, TestMessage{Message: "hi there"})
+       assert.NoError(t, err)
+       err = net.AppRequest(nodeID, 1, time.Now().Add(1*time.Millisecond), requestBytes)
+       assert.NoError(t, err)
+       // ensure the handler didn't get called (as peer.Network would've dropped the request)
+       assert.EqualValues(t, requestHandler.calls, 0)
+
+       requestHandler.processingDuration = 0
+       err = net.AppRequest(nodeID, 2, time.Now().Add(250*time.Millisecond), requestBytes)
+       assert.NoError(t, err)
+       assert.True(t, responded)
+       assert.EqualValues(t, requestHandler.calls, 1)
+}
+
+func TestGossip(t *testing.T) {
+       codecManager := buildCodec(t, HelloGossip{})
+
+       nodeID := ids.GenerateTestShortID()
+       var clientNetwork Network
+       wg := &sync.WaitGroup{}
+       sentGossip := false
+       wg.Add(1)
+       sender := testAppSender{
+               sendAppGossipFn: func(msg []byte) error {
+                       go func() {
+                               defer wg.Done()
+                               err := clientNetwork.AppGossip(nodeID, msg)
+                               assert.NoError(t, err)
+                       }()
+                       sentGossip = true
+                       return nil
+               },
+       }
+
+       gossipHandler := &testGossipHandler{}
+       clientNetwork = NewNetwork(sender, codecManager, ids.ShortEmpty, 1)
+       clientNetwork.SetGossipHandler(gossipHandler)
+
+       assert.NoError(t, clientNetwork.Connected(nodeID, defaultPeerVersion))
+
+       client := NewClient(clientNetwork)
+       defer clientNetwork.Shutdown()
+
+       b, err := buildGossip(codecManager, HelloGossip{Msg: "hello there!"})
+       assert.NoError(t, err)
+
+       err = client.Gossip(b)
+       assert.NoError(t, err)
+
+       wg.Wait()
+       assert.True(t, sentGossip)
+       assert.True(t, gossipHandler.received)
+}
+
+func TestHandleInvalidMessages(t *testing.T) {
+       codecManager := buildCodec(t, HelloGossip{}, TestMessage{})
+
+       nodeID := ids.GenerateTestShortID()
+       requestID := uint32(1)
+       sender := testAppSender{}
+
+       clientNetwork := NewNetwork(sender, codecManager, ids.ShortEmpty, 1)
+       clientNetwork.SetGossipHandler(message.NoopMempoolGossipHandler{})
+       clientNetwork.SetRequestHandler(&testRequestHandler{})
+
+       assert.NoError(t, clientNetwork.Connected(nodeID, defaultPeerVersion))
+
+       defer clientNetwork.Shutdown()
```

```
+
+       // Ensure a valid gossip message sent as any App specific message type does not trigger a fatal error
+       gossipMsg, err := buildGossip(codecManager, HelloGossip{Msg: "hello there!"})
+       assert.NoError(t, err)
+
+       // Ensure a valid request message sent as any App specific message type does not trigger a fatal error
+       requestMessage, err := marshalStruct(codecManager, TestMessage{Message: "Hello"})
+       assert.NoError(t, err)
+
+       // Ensure a random message sent as any App specific message type does not trigger a fatal error
+       garbageResponse := make([]byte, 10)
+       // Ensure a zero-length message sent as any App specific message type does not trigger a fatal error
+       emptyResponse := make([]byte, 0)
+       // Ensure a nil byte slice sent as any App specific message type does not trigger a fatal error
+       var nilResponse []byte
+
+       // Check for edge cases
+       assert.NoError(t, clientNetwork.AppGossip(nodeID, gossipMsg))
+       assert.NoError(t, clientNetwork.AppGossip(nodeID, requestMessage))
+       assert.NoError(t, clientNetwork.AppGossip(nodeID, garbageResponse))
+       assert.NoError(t, clientNetwork.AppGossip(nodeID, emptyResponse))
+       assert.NoError(t, clientNetwork.AppGossip(nodeID, nilResponse))
+       assert.NoError(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), gossipMsg))
+       assert.NoError(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), requestMessage))
+       assert.NoError(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), garbageResponse))
+       assert.NoError(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), emptyResponse))
+       assert.NoError(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), nilResponse))
+       assert.NoError(t, clientNetwork.AppResponse(nodeID, requestID, gossipMsg))
+       assert.NoError(t, clientNetwork.AppResponse(nodeID, requestID, requestMessage))
+       assert.NoError(t, clientNetwork.AppResponse(nodeID, requestID, garbageResponse))
+       assert.NoError(t, clientNetwork.AppResponse(nodeID, requestID, emptyResponse))
+       assert.NoError(t, clientNetwork.AppResponse(nodeID, requestID, nilResponse))
+       assert.NoError(t, clientNetwork.AppRequestFailed(nodeID, requestID))
+}
+
+func TestNetworkPropagatesRequestHandlerError(t *testing.T) {
+       codecManager := buildCodec(t, TestMessage{})
+
+       nodeID := ids.GenerateTestShortID()
+       requestID := uint32(1)
+       sender := testAppSender{}
+
+       clientNetwork := NewNetwork(sender, codecManager, ids.ShortEmpty, 1)
+       clientNetwork.SetGossipHandler(message.NoopMempoolGossipHandler{})
+       clientNetwork.SetRequestHandler(&testRequestHandler{err: errors.New("fail")}) // Return an error from the request handler
+
+       assert.NoError(t, clientNetwork.Connected(nodeID, defaultPeerVersion))
+
+       defer clientNetwork.Shutdown()
+
+       // Ensure a valid request message sent as any App specific message type does not trigger a fatal error
+       requestMessage, err := marshalStruct(codecManager, TestMessage{Message: "Hello"})
+       assert.NoError(t, err)
+
+       // Check that if the request handler returns an error, it is propagated as a fatal error.
+       assert.Error(t, clientNetwork.AppRequest(nodeID, requestID, time.Now().Add(time.Second), requestMessage))
+}
+
+func buildCodec(t *testing.T, types ...interface{}) codec.Manager {
+       codecManager := codec.NewDefaultManager()
+       c := linearcodec.NewDefault()
+       for _, typ := range types {
+               assert.NoError(t, c.RegisterType(typ))
+       }
+       assert.NoError(t, codecManager.RegisterCodec(message.Version, c))
+       return codecManager
+}
+
+// marshalStruct is a helper method used to marshal an object as `interface{}`
+// so that the codec is able to include the TypeID in the resulting bytes
+func marshalStruct(codec codec.Manager, obj interface{}) ([]byte, error) {
+       return codec.Marshal(message.Version, &obj)
+}
+
+func buildGossip(codec codec.Manager, msg message.Message) ([]byte, error) {
+       return codec.Marshal(message.Version, &msg)
+}
+
+type testAppSender struct {
+       sendAppRequestFn  func(ids.ShortSet, uint32, []byte) error
+       sendAppResponseFn func(ids.ShortID, uint32, []byte) error
+       sendAppGossipFn   func([]byte) error
+}
+
+func (t testAppSender) SendAppGossipSpecific(ids.ShortSet, []byte) error {
+       panic("not implemented")
+}
+
+func (t testAppSender) SendAppRequest(nodeIDs ids.ShortSet, requestID uint32, message []byte) error {
+       return t.sendAppRequestFn(nodeIDs, requestID, message)
+}
+
+func (t testAppSender) SendAppResponse(nodeID ids.ShortID, requestID uint32, message []byte) error {
+       return t.sendAppResponseFn(nodeID, requestID, message)
+}
+
+func (t testAppSender) SendAppGossip(message []byte) error {
+       return t.sendAppGossipFn(message)
+}
+
+type HelloRequest struct {
+       Message string `serialize:"true"`
+}
+
+func (h HelloRequest) Handle(ctx context.Context, nodeID ids.ShortID, requestID uint32, handler message.RequestHandler) ([]byte, error) {
+       // casting is only necessary for test since RequestHandler does not implement anything at the moment
+       return handler.(TestRequestHandler).HandleHelloRequest(ctx, nodeID, requestID, &h)
+}
+
+func (h HelloRequest) Type() string {
+       return "hello-request"
+}
+
+type GreetingRequest struct {
+       Greeting string `serialize:"true"`
+}
+
+func (g GreetingRequest) Handle(ctx context.Context, nodeID ids.ShortID, requestID uint32, handler message.RequestHandler) ([]byte, error) {
+       // casting is only necessary for test since RequestHandler does not implement anything at the moment
+       return handler.(TestRequestHandler).HandleGreetingRequest(ctx, nodeID, requestID, &g)
+}
+
+func (g GreetingRequest) Type() string {
+       return "greeting-request"
+}
+
+type HelloResponse struct {
+       Response string `serialize:"true"`
+}
+
+type GreetingResponse struct {
+       Greet string `serialize:"true"`
+}
+
+type TestRequestHandler interface {
+       HandleHelloRequest(ctx context.Context, nodeID ids.ShortID, requestID uint32, request *HelloRequest) ([]byte, error)
+       HandleGreetingRequest(ctx context.Context, nodeID ids.ShortID, requestID uint32, request *GreetingRequest) ([]byte, error)
+}
+
+type HelloGreetingRequestHandler struct {
+       codec codec.Manager
+}
+
+func (h *HelloGreetingRequestHandler) HandleHelloRequest(ctx context.Context, nodeID ids.ShortID, requestID uint32, request *HelloRequest) ([]byte, error) {
+       return h.codec.Marshal(message.Version, HelloResponse{Response: "Hi"})
+}
+
+func (h *HelloGreetingRequestHandler) HandleGreetingRequest(ctx context.Context, nodeID ids.ShortID, requestID uint32, request *GreetingRequest) ([]byte, error) {
+       return h.codec.Marshal(message.Version, GreetingResponse{Greet: "Hey there"})
+}
+
+type TestMessage struct {
+       Message string `serialize:"true"`
+}
+
+func (t TestMessage) Handle(ctx context.Context, nodeID ids.ShortID, requestID uint32, handler message.RequestHandler) ([]byte, error) {
+       return handler.(*testRequestHandler).handleTestRequest(ctx, nodeID, requestID, &t)
+}
+
+func (t TestMessage) Type() string {
+       return "test-message"
+}
+
+type HelloGossip struct {
+       message.Message
+       Msg string `serialize:"true"`
+}
+
+func (h HelloGossip) Handle(handler message.GossipHandler, nodeID ids.ShortID) error {
+       return handler.HandleEthTxs(nodeID, nil)
+}
+
+func (h HelloGossip) Type() string {
+       return "hello-gossip"
+}
+
+func (h HelloGossip) initialize(_ []byte) {
+       // no op
+}
+
+func (h HelloGossip) Bytes() []byte {
+       // no op
+       return nil
+}
+
+type testGossipHandler struct {
+       received bool
+       nodeID   ids.ShortID
+       msg      []byte
+}
+
+func (t *testGossipHandler) HandleAtomicTx(nodeID ids.ShortID, _ *message.AtomicTx) error {
+       t.received = true
+       t.nodeID = nodeID
+       return nil
+}
+
+func (t *testGossipHandler) HandleEthTxs(nodeID ids.ShortID, _ *message.EthTxs) error {
+       t.received = true
+       t.nodeID = nodeID
+       return nil
+}
+
+type testRequestHandler struct {
+       calls              uint32
+       processingDuration time.Duration
+       response           []byte
+       err                error
+}
+
+func (r *testRequestHandler) handleTestRequest(ctx context.Context, _ ids.ShortID, _ uint32, _ *TestMessage) ([]byte, error) {
+       r.calls++
+       select {
+       case <-time.After(r.processingDuration):
+               break
+       case <-ctx.Done():
+               return nil, ctx.Err()
+       }
+       return r.response, r.err
+}
diff --git a/peer/waiting_handler.go b/peer/waiting_handler.go
new file mode 100644
index 00000000..b656df64
--- /dev/null
+++ b/peer/waiting_handler.go
@@ -0,0 +1,39 @@
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package peer
+
+import (
+       "github.com/flare-foundation/coreth/plugin/evm/message"
+       "github.com/flare-foundation/flare/ids"
+)
+
+var _ message.ResponseHandler = &waitingResponseHandler{}
+
+// waitingResponseHandler implements the ResponseHandler interface
+// Internally used to wait for response after making a request synchronously
+// responseChan may contain response bytes if the original request has not failed
+// responseChan is closed in either fail or success scenario
+type waitingResponseHandler struct {
+       responseChan chan []byte // blocking channel with response bytes
+       failed       bool        // whether the original request is failed
+}
+
+// OnResponse passes the response bytes to the responseChan and closes the channel
+func (w *waitingResponseHandler) OnResponse(_ ids.ShortID, _ uint32, response []byte) error {
+       w.responseChan <- response
+       close(w.responseChan)
+       return nil
+}
+
+// OnFailure sets the failed flag to true and closes the channel
+func (w *waitingResponseHandler) OnFailure(ids.ShortID, uint32) error {
+       w.failed = true
+       close(w.responseChan)
+       return nil
+}
+
+// newWaitingResponseHandler returns new instance of the waitingResponseHandler
+func newWaitingResponseHandler() *waitingResponseHandler {
+       return &waitingResponseHandler{responseChan: make(chan []byte)}
+}
diff --git a/plugin/evm/admin.go b/plugin/evm/admin.go
index 0cdeb2ad..b9418795 100644
--- a/plugin/evm/admin.go
```

```
+++ b/plugin/evm/admin.go
@@ -7,9 +7,9 @@ import (
        "fmt"
        "net/http"

-       "github.com/ava-labs/avalanchego/api"
-       "github.com/ava-labs/avalanchego/utils/profiler"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/flare/api"
+       "github.com/flare-foundation/flare/utils/profiler"
 )

 // Admin is the API service for admin API calls
diff --git a/plugin/evm/atomic_trie.go b/plugin/evm/atomic_trie.go
new file mode 100644
index 00000000..44c2531c
--- /dev/null
+++ b/plugin/evm/atomic_trie.go
@@ -0,0 +1,437 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+       "encoding/binary"
+       "fmt"
+       "time"
+
+       "github.com/flare-foundation/flare/database"
+       "github.com/flare-foundation/flare/database/prefixdb"
+       "github.com/flare-foundation/flare/database/versiondb"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/trie"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/wrappers"
+)
+
+const (
+       commitHeightInterval = uint64(4096)
+       progressLogUpdate    = 30 * time.Second
+)
+
+var (
+       lastCommittedKey = []byte("atomicTrieLastCommittedBlock")
+)
+
+// AtomicTrie maintains an index of atomic operations by blockchainIDs for every block
+// height containing atomic transactions. The backing data structure for this index is
+// a Trie. The keys of the trie are block heights and the values (leaf nodes)
+// are the atomic operations applied to shared memory while processing the block accepted
+// at the corresponding height.
+type AtomicTrie interface {
+       // Index indexes the given atomicOps at the specified block height
+       // Returns an optional root hash
+       // A non-empty root hash is returned when the atomic trie has been committed
+       // Atomic trie is committed if the block height is multiple of commit interval
+       Index(height uint64, atomicOps map[ids.ID]*atomic.Requests) error
+
+       // Iterator returns an AtomicTrieIterator to iterate the trie at the given
+       // root hash
+       Iterator(hash common.Hash, startHeight uint64) (AtomicTrieIterator, error)
+
+       // LastCommitted returns the last committed hash and corresponding block height
+       LastCommitted() (common.Hash, uint64)
+
+       // TrieDB returns the underlying trie database
+       TrieDB() *trie.Database
+
+       // Root returns hash if it exists at specified height
+       // if trie was not committed at provided height, it returns
+       // common.Hash{} instead
+       Root(height uint64) (common.Hash, error)
+}
+
+// AtomicTrieIterator is a stateful iterator that iterates the leafs of an AtomicTrie
+type AtomicTrieIterator interface {
+       // Next advances the iterator to the next node in the atomic trie and
+       // returns true if there are more nodes to iterate
+       Next() bool
+
+       // BlockNumber returns the current block number
+       BlockNumber() uint64
+
+       // BlockchainID returns the current blockchain ID at the current block number
+       BlockchainID() ids.ID
+
+       // AtomicOps returns a map of blockchainIDs to the set of atomic requests
+       // for that blockchainID at the current block number
+       AtomicOps() *atomic.Requests
+
+       // Error returns error, if any encountered during this iteration
+       Error() error
+}
+
+// atomicTrie implements the AtomicTrie interface
+// using the eth trie.Trie implementation
+type atomicTrie struct {
+       commitHeightInterval uint64              // commit interval, same as commitHeightInterval by default
+       db                   *versiondb.Database // Underlying database
+       bonusBlocks          map[uint64]ids.ID   // Map of height to blockID for blocks to skip indexing
+       metadataDB           database.Database   // Underlying database containing the atomic trie metadata
+       atomicTrieDB         database.Database   // Underlying database containing the atomic trie
+       trieDB               *trie.Database      // Trie database
+       trie                 *trie.Trie          // Atomic trie.Trie mapping key (height+blockchainID) and value (codec serialized atomic.Requests)
+       repo                 AtomicTxRepository
+       lastCommittedHash    common.Hash // trie root hash of the most recent commit
+       lastCommittedHeight  uint64      // index height of the most recent commit
+       codec                codec.Manager
+       log                  log.Logger // struct logger
+}
+
+var _ AtomicTrie = &atomicTrie{}
+
+// NewAtomicTrie returns a new instance of a atomicTrie with the default commitHeightInterval.
+// Initializes the trie before returning it.
+func NewAtomicTrie(db *versiondb.Database, bonusBlocks map[uint64]ids.ID, repo AtomicTxRepository, codec codec.Manager, lastAcceptedHeight uint64) (AtomicTrie, error) {
+       return newAtomicTrie(db, bonusBlocks, repo, codec, lastAcceptedHeight, commitHeightInterval)
+}
+
+// newAtomicTrie returns a new instance of a atomicTrie with a configurable commitHeightInterval, used in testing.
+// Initializes the trie before returning it.
+func newAtomicTrie(
+       db *versiondb.Database, bonusBlocks map[uint64]ids.ID, repo AtomicTxRepository, codec codec.Manager, lastAcceptedHeight uint64, commitHeightInterval uint64,
+) (*atomicTrie, error) {
+       atomicTrieDB := prefixdb.New(atomicTrieDBPrefix, db)
+       metadataDB := prefixdb.New(atomicTrieMetaDBPrefix, db)
+       root, height, err := lastCommittedRootIfExists(metadataDB)
+       if err != nil {
+               return nil, err
+       }
+
+       triedb := trie.NewDatabaseWithConfig(
+               Database{atomicTrieDB},
+               &trie.Config{
+                       Cache:     10,    // Allocate 10MB of memory for clean cache
+                       Preimages: false, // Keys are not hashed, so there is no need for preimages
+               },
+       )
+       t, err := trie.New(root, triedb)
+       if err != nil {
+               return nil, err
+       }
+
+       atomicTrie := &atomicTrie{
+               commitHeightInterval: commitHeightInterval,
+               db:                   db,
+               bonusBlocks:          bonusBlocks,
+               atomicTrieDB:         atomicTrieDB,
+               metadataDB:           metadataDB,
+               trieDB:               triedb,
+               trie:                 t,
+               repo:                 repo,
+               codec:                codec,
+               lastCommittedHash:    root,
+               lastCommittedHeight:  height,
+               log:                  log.New("c", "atomicTrie"),
+       }
+       return atomicTrie, atomicTrie.initialize(lastAcceptedHeight)
+}
+
+// lastCommittedRootIfExists returns the last committed trie root and height if it exists
+// else returns empty common.Hash{} and 0
+// returns error only if there are issues with the underlying data store
+// or if values present in the database are not as expected
+func lastCommittedRootIfExists(db database.Database) (common.Hash, uint64, error) {
+       // read the last committed entry if it exists and set the root hash
+       lastCommittedHeightBytes, err := db.Get(lastCommittedKey)
+       switch {
+       case err == database.ErrNotFound:
+               return common.Hash{}, 0, nil
+       case err != nil:
+               return common.Hash{}, 0, err
+       case len(lastCommittedHeightBytes) != wrappers.LongLen:
+               return common.Hash{}, 0, fmt.Errorf("expected value of lastCommittedKey to be %d but was %d", wrappers.LongLen, len(lastCommittedHeightBytes))
+       }
+       height := binary.BigEndian.Uint64(lastCommittedHeightBytes)
+       hash, err := db.Get(lastCommittedHeightBytes)
+       if err != nil {
+               return common.Hash{}, 0, fmt.Errorf("committed hash does not exist for committed height: %d: %w", height, err)
+       }
+       return common.BytesToHash(hash), height, nil
+}
+
+// nearestCommitheight returns the nearest multiple of commitInterval less than or equal to blockNumber
+func nearestCommitHeight(blockNumber uint64, commitInterval uint64) uint64 {
+       return blockNumber - (blockNumber % commitInterval)
+}
+
+// initializes the atomic trie using the atomic repository height index.
+// Iterating from the last indexed height to lastAcceptedBlockNumber, making a single commit at the
+// most recent height divisible by the commitInterval.
+// Subsequent updates to this trie are made using the Index call as blocks are accepted.
+func (a *atomicTrie) initialize(lastAcceptedBlockNumber uint64) error {
+       start := time.Now()
+       a.log.Info("initializing atomic trie", "lastAcceptedBlockNumber", lastAcceptedBlockNumber)
+       // finalCommitHeight is the highest block that can be committed i.e. is divisible by b.commitHeightInterval
+       // Txs from heights greater than commitHeight are to be included in the trie corresponding to the block at
+       // finalCommitHeight+b.commitHeightInterval, which has not been accepted yet.
+       finalCommitHeight := nearestCommitHeight(lastAcceptedBlockNumber, a.commitHeightInterval)
+       uncommittedOpsMap := make(map[uint64]map[ids.ID]*atomic.Requests, lastAcceptedBlockNumber-finalCommitHeight)
+
+       heightBytes := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(heightBytes, a.lastCommittedHeight)
+       // iterate by height, from lastCommittedHeight to the lastAcceptedBlockNumber
+       iter := a.repo.IterateByHeight(heightBytes)
+       defer iter.Release()
+
+       preCommitBlockIndexed := 0
+       postCommitTxIndexed := 0
+       lastUpdate := time.Now()
+
+       // keep track of the latest generated trie's root and height.
+       lastHash := common.Hash{}
+       lastHeight := a.lastCommittedHeight
+       for iter.Next() {
+               // Get the height and transactions for this iteration (from the key and value, respectively)
+               // iterate over the transactions, indexing them if the height is < commit height
+               // otherwise, add the atomic operations from the transaction to the uncommittedOpsMap
+               height := binary.BigEndian.Uint64(iter.Key())
+               txs, err := ExtractAtomicTxs(iter.Value(), true, a.codec)
+               if err != nil {
+                       return err
+               }
+
+               // combine atomic operations from all transactions at this block height
+               combinedOps, err := mergeAtomicOps(txs)
+               if err != nil {
+                       return err
+               }
+
+               if _, skipBonusBlock := a.bonusBlocks[height]; skipBonusBlock {
+                       // If [height] is a bonus block, do not index the atomic operations into the trie
+               } else if height > finalCommitHeight {
+                       // if height is greater than commit height, add it to the map so that we can write it later
+                       // this is to ensure we have all the data before the commit height so that we can commit the
+                       // trie
+                       uncommittedOpsMap[height] = combinedOps
+               } else {
+                       if err := a.updateTrie(height, combinedOps); err != nil {
+                               return err
+                       }
+                       preCommitBlockIndexed++
+               }
+
+               if time.Since(lastUpdate) > progressLogUpdate {
+                       a.log.Info("imported entries into atomic trie pre-commit", "heightsIndexed", preCommitBlockIndexed)
+                       lastUpdate = time.Now()
+               }
+
+               // if height has reached or skipped over the next commit interval,
+               // keep track of progress and keep commit size under commitSizeCap
+               commitHeight := nearestCommitHeight(height, a.commitHeightInterval)
+               if lastHeight < commitHeight {
+                       hash, _, err := a.trie.Commit(nil)
+                       if err != nil {
+                               return err
+                       }
+                       // Dereference lashHash to avoid writing more intermediary
+                       // trie nodes than needed to disk, while keeping the commit
+                       // size under commitSizeCap (approximately).
+                       // Check [lastHash != hash] here to avoid dereferencing the
+                       // trie root in case there were no atomic txs since the
+                       // last commit.
+                       if (lastHash != common.Hash{} && lastHash != hash) {
+                               a.trieDB.Dereference(lastHash)
+                       }
+                       storage, _ := a.trieDB.Size()
+                       if storage > commitSizeCap {
+                               a.log.Info("committing atomic trie progress", "storage", storage)
+                               a.commit(commitHeight)
+                               // Flush any remaining changes that have not been committed yet in the versiondb.
+                               if err := a.db.Commit(); err != nil {
+                                       return err
+                               }
+                       }
+                       lastHash = hash
+                       lastHeight = commitHeight
+               }
+       }
+       if err := iter.Error(); err != nil {
+               return err
+       }
+
+       // Note: we should never create a commit at the genesis block (should not contain any atomic txs)
+       if lastAcceptedBlockNumber == 0 {
+               return nil
+       }
+       // now that all heights < finalCommitHeight have been processed
+       // commit the trie
+       if err := a.commit(finalCommitHeight); err != nil {
+               return err
+       }
+       // Flush any remaining changes that have not been committed yet in the versiondb.
+       if err := a.db.Commit(); err != nil {
+               return err
+       }
+
+       // process uncommitted ops for heights > finalCommitHeight
+       for height, ops := range uncommittedOpsMap {
+               if err := a.updateTrie(height, ops); err != nil {
+                       return fmt.Errorf("failed to update trie at height %d: %w", height, err)
+               }
+
+               postCommitTxIndexed++
+               if time.Since(lastUpdate) > progressLogUpdate {
+                       a.log.Info("imported entries into atomic trie post-commit", "entriesIndexed", postCommitTxIndexed)
+                       lastUpdate = time.Now()
+               }
+       }
+
+       a.log.Info(
+               "finished initializing atomic trie",
+               "lastAcceptedBlockNumber", lastAcceptedBlockNumber,
+               "preCommitEntriesIndexed", preCommitBlockIndexed, "postCommitEntriesIndexed", postCommitTxIndexed,
+               "time", time.Since(start),
+       )
+       return nil
+}
+
+// Index updates the trie with entries in atomicOps
+// This function updates the following:
+// - heightBytes => trie root hash (if the trie was committed)
+// - lastCommittedBlock => height (if the trie was committed)
+func (a *atomicTrie) Index(height uint64, atomicOps map[ids.ID]*atomic.Requests) error {
+       if err := a.validateIndexHeight(height); err != nil {
+               return err
+       }
+
+       if err := a.updateTrie(height, atomicOps); err != nil {
+               return err
+       }
+
+       if height%a.commitHeightInterval == 0 {
+               return a.commit(height)
+       }
+
+       return nil
+}
+
+// validateIndexHeight returns an error if [height] is not currently valid to be indexed.
+func (a *atomicTrie) validateIndexHeight(height uint64) error {
+       // Do not allow a height that we have already passed to be indexed
+       if height < a.lastCommittedHeight {
+               return fmt.Errorf("height %d must be after last committed height %d", height, a.lastCommittedHeight)
+       }
+
+       // Do not allow a height that is more than a commit interval ahead
+       // of the current index
+       nextCommitHeight := a.lastCommittedHeight + a.commitHeightInterval
+       if height > nextCommitHeight {
+               return fmt.Errorf("height %d not within the next commit height %d", height, nextCommitHeight)
+       }
+
+       return nil
+}
+
+// commit calls commit on the trie to generate a root, commits the underlying trieDB, and updates the
+// metadata pointers.
+// assumes that the caller is aware of the commit rules i.e. the height being within commitInterval.
+// returns the trie root from the commit
+func (a *atomicTrie) commit(height uint64) error {
+       hash, _, err := a.trie.Commit(nil)
+       if err != nil {
+               return err
+       }
+
+       a.log.Info("committed atomic trie", "hash", hash.String(), "height", height)
+       if err := a.trieDB.Commit(hash, false, nil); err != nil {
+               return err
+       }
+
+       // all good here, update the heightBytes
+       heightBytes := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(heightBytes, height)
+
+       // now save the trie hash against the height it was committed at
+       if err := a.metadataDB.Put(heightBytes, hash[:]); err != nil {
+               return err
+       }
+
+       // update lastCommittedKey with the current height
+       if err := a.metadataDB.Put(lastCommittedKey, heightBytes); err != nil {
+               return err
+       }
+
+       a.lastCommittedHash = hash
+       a.lastCommittedHeight = height
+       return nil
+}
+
+func (a *atomicTrie) updateTrie(height uint64, atomicOps map[ids.ID]*atomic.Requests) error {
+       for blockchainID, requests := range atomicOps {
+               valueBytes, err := a.codec.Marshal(codecVersion, requests)
+               if err != nil {
+                       // highly unlikely but possible if atomic.Element
+                       // has a change that is unsupported by the codec
+                       return err
+               }
+
+               // key is [height]+[blockchainID]
+               keyPacker := wrappers.Packer{Bytes: make([]byte, wrappers.LongLen+common.HashLength)}
+               keyPacker.PackLong(height)
+               keyPacker.PackFixedBytes(blockchainID[:])
+               if err := a.trie.TryUpdate(keyPacker.Bytes, valueBytes); err != nil {
+                       return err
+               }
+       }
+
+       return nil
+}
+
+// LastCommitted returns the last committed trie hash and last committed height
+func (a *atomicTrie) LastCommitted() (common.Hash, uint64) {
+       return a.lastCommittedHash, a.lastCommittedHeight
+}
+
+// Iterator returns a types.AtomicTrieIterator that iterates the trie from the given
+// atomic trie root, starting at the specified height
+func (a *atomicTrie) Iterator(root common.Hash, startHeight uint64) (AtomicTrieIterator, error) {
+       startKey := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(startKey, startHeight)
+
+       t, err := trie.New(root, a.trieDB)
+       if err != nil {
+               return nil, err
+       }
+
+       iter := trie.NewIterator(t.NodeIterator(startKey))
+       return NewAtomicTrieIterator(iter, a.codec), iter.Err
+}
+
+func (a *atomicTrie) TrieDB() *trie.Database {
+       return a.trieDB
+}
+
+// Root returns hash if it exists at specified height
+// if trie was not committed at provided height, it returns
+// common.Hash{} instead
+func (a *atomicTrie) Root(height uint64) (common.Hash, error) {
+       heightBytes := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(heightBytes, height)
+
+       hash, err := a.metadataDB.Get(heightBytes)
+       switch {
+       case err == database.ErrNotFound:
+               return common.Hash{}, nil
+       case err != nil:
+               return common.Hash{}, err
+       }
+       return common.BytesToHash(hash), nil
+}
diff --git a/plugin/evm/atomic_trie_iterator.go b/plugin/evm/atomic_trie_iterator.go
new file mode 100644
index 00000000..2c64c44f
--- /dev/null
+++ b/plugin/evm/atomic_trie_iterator.go
@@ -0,0 +1,110 @@
+// (c) 2019-2020, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+       "encoding/binary"
+       "fmt"
+
+       "github.com/flare-foundation/flare/codec"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/trie"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/wrappers"
+)
+
+const atomicTrieKeyLen = wrappers.LongLen + common.HashLength
+
+// atomicTrieIterator is an implementation of types.AtomicTrieIterator that serves
+// parsed data with each iteration
+type atomicTrieIterator struct {
+       trieIterator *trie.Iterator // underlying trie.Iterator
+       codec        codec.Manager
+       atomicOps    *atomic.Requests // atomic operation entries at this iteration
+       blockchainID ids.ID           // blockchain ID
+       blockNumber  uint64           // block number at this iteration
+       err          error            // error if any has occurred
+}
+
+func NewAtomicTrieIterator(trieIterator *trie.Iterator, codec codec.Manager) AtomicTrieIterator {
+       return &atomicTrieIterator{trieIterator: trieIterator, codec: codec}
+}
+
+// Error returns error, if any encountered during this iteration
+func (a *atomicTrieIterator) Error() error {
+       return a.err
+}
+
+// Next returns whether there are more nodes to iterate over
+// On success, this function sets the blockNumber and atomicOps fields
+// In case of an error during this iteration, it sets the error value and resets the above fields.
+// It is the responsibility of the caller to check the result of Error() after an iterator reports
+// having no more elements to iterate.
+func (a *atomicTrieIterator) Next() bool {
+       hasNext := a.trieIterator.Next()
+
+       if a.trieIterator.Err != nil {
+               a.resetFields(a.trieIterator.Err)
+               return false
+       }
+       if !hasNext {
+               a.resetFields(nil)
+               return false
+       }
+
+       // if the underlying iterator has data to iterate over, parse and set the fields
+       // key is [blockNumberBytes]+[blockchainIDBytes] = 8+32=40 bytes
+       keyLen := len(a.trieIterator.Key)
+       // If the key has an unexpected length, set the error and stop the iteration since the data is
+       // no longer reliable.
+       if keyLen != atomicTrieKeyLen {
+               a.resetFields(fmt.Errorf("expected atomic trie key length to be %d but was %d", atomicTrieKeyLen, keyLen))
+               return false
+       }
+
+       blockNumber := binary.BigEndian.Uint64(a.trieIterator.Key[:wrappers.LongLen])
+       blockchainID, err := ids.ToID(a.trieIterator.Key[wrappers.LongLen:])
+       if err != nil {
+               a.resetFields(err)
+               return false
+       }
+
+	// The value in the iterator should be the atomic requests serialized the the codec.
+	requests := new(atomic.Requests)
+	if _, err = a.codec.Unmarshal(a.trieIterator.Value, requests); err != nil {
+		a.resetFields(err)
+		return false
+	}
+
+	// Success, update the struct fields
+	a.blockNumber = blockNumber
+	a.blockchainID = blockchainID
+	a.atomicOps = requests
+	return true
+}
+
+// resetFields resets the value fields of the iterator to their nil values and sets the error value to [err].
+func (a *atomicTrieIterator) resetFields(err error) {
+	a.err = err
+	a.blockNumber = 0
+	a.blockchainID = ids.ID{}
+	a.atomicOps = nil
+}
+
+// BlockNumber returns the current block number
+func (a *atomicTrieIterator) BlockNumber() uint64 {
+	return a.blockNumber
+}
+
+// BlockchainID returns the current blockchain ID at the current block number
+func (a *atomicTrieIterator) BlockchainID() ids.ID {
+	return a.blockchainID
+}
+
+// AtomicOps returns atomic requests for the blockchainID at the current block number
+func (a *atomicTrieIterator) AtomicOps() *atomic.Requests {
+	return a.atomicOps
+}
diff --git a/plugin/evm/atomic_trie_iterator_test.go b/plugin/evm/atomic_trie_iterator_test.go
new file mode 100644
index 00000000..ca43ebd8
--- /dev/null
+++ b/plugin/evm/atomic_trie_iterator_test.go
@@ -0,0 +1,90 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+	"testing"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/flare/chains/atomic"
+	"github.com/flare-foundation/flare/database/memdb"
+	"github.com/flare-foundation/flare/database/versiondb"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/utils"
+	"github.com/stretchr/testify/assert"
+)
+
+func TestIteratorCanIterate(t *testing.T) {
+	lastAcceptedHeight := uint64(1000)
+	db := versiondb.New(memdb.New())
+	codec := testTxCodec()
+	repo, err := NewAtomicTxRepository(db, codec, lastAcceptedHeight)
+	assert.NoError(t, err)
+
+	// create state with multiple transactions
+	// since each test transaction generates random ID for blockchainID we should get
+	// multiple blockchain IDs per block in the overall combined atomic operation map
+	operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+	writeTxs(t, repo, 0, lastAcceptedHeight, constTxsPerHeight(3), nil, operationsMap)
+
+	// create an atomic trie
+	// on create it will initialize all the transactions from the above atomic repository
+	atomicTrie1, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 100)
+	assert.NoError(t, err)
+
+	lastCommittedHash1, lastCommittedHeight1 := atomicTrie1.LastCommitted()
+	assert.NoError(t, err)
+	assert.NotEqual(t, common.Hash{}, lastCommittedHash1)
+	assert.EqualValues(t, 1000, lastCommittedHeight1)
+
+	verifyOperations(t, atomicTrie1, codec, lastCommittedHash1, 0, 1000, operationsMap)
+
+	// iterate on a new atomic trie to make sure there is no resident state affecting the data and the
+	// iterator
+	atomicTrie2, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 100)
+	assert.NoError(t, err)
+	lastCommittedHash2, lastCommittedHeight2 := atomicTrie2.LastCommitted()
+	assert.NoError(t, err)
+	assert.NotEqual(t, common.Hash{}, lastCommittedHash2)
+	assert.EqualValues(t, 1000, lastCommittedHeight2)
+
+	verifyOperations(t, atomicTrie2, codec, lastCommittedHash1, 0, 1000, operationsMap)
+}
+
+func TestIteratorHandlesInvalidData(t *testing.T) {
+	lastAcceptedHeight := uint64(1000)
+	db := versiondb.New(memdb.New())
+	codec := testTxCodec()
+	repo, err := NewAtomicTxRepository(db, codec, lastAcceptedHeight)
+	assert.NoError(t, err)
+
+	// create state with multiple transactions
+	// since each test transaction generates random ID for blockchainID we should get
+	// multiple blockchain IDs per block in the overall combined atomic operation map
+	operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+	writeTxs(t, repo, 0, lastAcceptedHeight, constTxsPerHeight(3), nil, operationsMap)
+
+	// create an atomic trie
+	// on create it will initialize all the transactions from the above atomic repository
+	atomicTrie, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 100)
+	assert.NoError(t, err)
+
+	lastCommittedHash, lastCommittedHeight := atomicTrie.LastCommitted()
+	assert.NoError(t, err)
+	assert.NotEqual(t, common.Hash{}, lastCommittedHash)
+	assert.EqualValues(t, 1000, lastCommittedHeight)
+
+	verifyOperations(t, atomicTrie, codec, lastCommittedHash, 0, 1000, operationsMap)
+
+	// Add a random key-value pair to the atomic trie in order to test that the iterator correctly
+	// handles an error when it runs into an unexpected key-value pair in the trie.
+	assert.NoError(t, atomicTrie.trie.TryUpdate(utils.RandomBytes(50), utils.RandomBytes(50)))
+	assert.NoError(t, atomicTrie.commit(lastCommittedHeight+1))
+	corruptedHash, _ := atomicTrie.LastCommitted()
+	iter, err := atomicTrie.Iterator(corruptedHash, 0)
+	assert.NoError(t, err)
+	for iter.Next() {
+	}
+	assert.Error(t, iter.Error())
+}
diff --git a/plugin/evm/atomic_trie_test.go b/plugin/evm/atomic_trie_test.go
new file mode 100644
```

```
index 00000000..ba3fb68c
--- /dev/null
+++ b/plugin/evm/atomic_trie_test.go
@@ -0,0 +1,401 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+	"testing"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/flare/chains/atomic"
+	"github.com/flare-foundation/flare/database/memdb"
+	"github.com/flare-foundation/flare/database/versiondb"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/stretchr/testify/assert"
+)
+
+const testCommitInterval = 100
+
+func (tx *Tx) mustAtomicOps() map[ids.ID]*atomic.Requests {
+	id, reqs, err := tx.AtomicOps()
+	if err != nil {
+		panic(err)
+	}
+	return map[ids.ID]*atomic.Requests{id: reqs}
+}
+
+func TestNearestCommitHeight(t *testing.T) {
+	type test struct {
+		height, commitInterval, expectedCommitHeight uint64
+	}
+
+	for _, test := range []test{
+		{
+			height:               4500,
+			commitInterval:       4096,
+			expectedCommitHeight: 4096,
+		},
+		{
+			height:               8500,
+			commitInterval:       4096,
+			expectedCommitHeight: 8192,
+		},
+		{
+			height:               950,
+			commitInterval:       100,
+			expectedCommitHeight: 900,
+		},
+	} {
+		commitHeight := nearestCommitHeight(test.height, test.commitInterval)
+		assert.Equal(t, commitHeight, test.expectedCommitHeight)
+	}
+}
+
+func TestAtomicTrieInitialize(t *testing.T) {
+	type test struct {
+		commitInterval, lastAcceptedHeight, expectedCommitHeight uint64
+		numTxsPerBlock                                           func(uint64) int
+	}
+	for name, test := range map[string]test{
+		"genesis": {
+			commitInterval:       10,
+			lastAcceptedHeight:   0,
+			expectedCommitHeight: 0,
+			numTxsPerBlock:       constTxsPerHeight(0),
+		},
+		"before first commit": {
+			commitInterval:       10,
+			lastAcceptedHeight:   5,
+			expectedCommitHeight: 0,
+			numTxsPerBlock:       constTxsPerHeight(3),
+		},
+		"first commit": {
+			commitInterval:       10,
+			lastAcceptedHeight:   10,
+			expectedCommitHeight: 10,
+			numTxsPerBlock:       constTxsPerHeight(3),
+		},
+		"past first commit": {
+			commitInterval:       10,
+			lastAcceptedHeight:   15,
+			expectedCommitHeight: 10,
+			numTxsPerBlock:       constTxsPerHeight(3),
+		},
+		"many existing commits": {
+			commitInterval:       10,
+			lastAcceptedHeight:   1000,
+			expectedCommitHeight: 1000,
+			numTxsPerBlock:       constTxsPerHeight(3),
+		},
+		"many existing commits plus 1": {
+			commitInterval:       10,
+			lastAcceptedHeight:   1001,
+			expectedCommitHeight: 1000,
+			numTxsPerBlock:       constTxsPerHeight(3),
+		},
+		"some blocks without atomic tx": {
+			commitInterval:       10,
+			lastAcceptedHeight:   101,
+			expectedCommitHeight: 100,
+			numTxsPerBlock: func(height uint64) int {
+				if height <= 50 || height == 101 {
+					return 1
+				}
+				return 0
+			},
+		},
+	} {
+		t.Run(name, func(t *testing.T) {
+			db := versiondb.New(memdb.New())
+			codec := testTxCodec()
+			repo, err := NewAtomicTxRepository(db, codec, test.lastAcceptedHeight)
+			if err != nil {
+				t.Fatal(err)
+			}
+			operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+			writeTxs(t, repo, 0, test.lastAcceptedHeight+1, test.numTxsPerBlock, nil, operationsMap)
+
+			// Construct the atomic trie for the first time
+			atomicTrie1, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, test.lastAcceptedHeight, test.commitInterval)
+			if err != nil {
+				t.Fatal(err)
+			}
+			rootHash1, commitHeight1 := atomicTrie1.LastCommitted()
+			assert.EqualValues(t, test.expectedCommitHeight, commitHeight1)
+			if test.expectedCommitHeight != 0 {
+				assert.NotEqual(t, common.Hash{}, rootHash1)
+			}
+
+			// Verify the operations up to the expected commit height
+			verifyOperations(t, atomicTrie1, codec, rootHash1, 0, test.expectedCommitHeight, operationsMap)
+
+                       // Construct the atomic trie a second time and ensure that it produces the same hash
+                       atomicTrie2, err := newAtomicTrie(versiondb.New(memdb.New()), make(map[uint64]ids.ID), repo, codec, test.lastAcceptedHeight, test.commitInterval)
+                       if err != nil {
+                               t.Fatal(err)
+                       }
+                       rootHash2, commitHeight2 := atomicTrie2.LastCommitted()
+                       assert.EqualValues(t, commitHeight1, commitHeight2)
+                       assert.EqualValues(t, rootHash1, rootHash2)
+
+                       // We now index additional operations up the next commit interval in order to confirm that nothing
+                       // during the initialization phase will cause an invalid root when indexing continues.
+                       nextCommitHeight := nearestCommitHeight(test.lastAcceptedHeight+test.commitInterval, test.commitInterval)
+                       for i := test.lastAcceptedHeight + 1; i <= nextCommitHeight; i++ {
+                               txs := newTestTxs(test.numTxsPerBlock(i))
+                               if err := repo.Write(i, txs); err != nil {
+                                       t.Fatal(err)
+                               }
+
+                               atomicOps, err := mergeAtomicOps(txs)
+                               if err != nil {
+                                       t.Fatal(err)
+                               }
+                               if err := atomicTrie1.Index(i, atomicOps); err != nil {
+                                       t.Fatal(err)
+                               }
+                               operationsMap[i] = atomicOps
+                       }
+
+                       updatedRoot, updatedLastCommitHeight := atomicTrie1.LastCommitted()
+                       assert.EqualValues(t, nextCommitHeight, updatedLastCommitHeight)
+                       assert.NotEqual(t, common.Hash{}, updatedRoot)
+
+                       // Verify the operations up to the new expected commit height
+                       verifyOperations(t, atomicTrie1, codec, updatedRoot, 0, updatedLastCommitHeight, operationsMap)
+
+                       // Generate a new atomic trie to compare the root against.
+                       atomicTrie3, err := newAtomicTrie(versiondb.New(memdb.New()), make(map[uint64]ids.ID), repo, codec, nextCommitHeight, test.commitInterval)
+                       if err != nil {
+                               t.Fatal(err)
+                       }
+                       rootHash3, commitHeight3 := atomicTrie3.LastCommitted()
+                       assert.EqualValues(t, rootHash3, updatedRoot)
+                       assert.EqualValues(t, updatedLastCommitHeight, commitHeight3)
+               })
+       }
+}
+
+func TestIndexerInitializesOnlyOnce(t *testing.T) {
+       lastAcceptedHeight := uint64(25)
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+       repo, err := NewAtomicTxRepository(db, codec, lastAcceptedHeight)
+       assert.NoError(t, err)
+       operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+       writeTxs(t, repo, 0, lastAcceptedHeight+1, constTxsPerHeight(2), nil, operationsMap)
+
+       // Initialize atomic repository
+       atomicTrie, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 10 /*commitHeightInterval*/)
+       assert.NoError(t, err)
+
+       hash, height := atomicTrie.LastCommitted()
+       assert.NotEqual(t, common.Hash{}, hash)
+       assert.Equal(t, uint64(20), height)
+
+       // We write another tx at a height below the last committed height in the repo and then
+       // re-initialize the atomic trie since initialize is not supposed to run again the height
+       // at the trie should still be the old height with the old commit hash without any changes.
+       // This scenario is not realistic, but is used to test potential double initialization behavior.
+       err = repo.Write(15, []*Tx{testDataExportTx()})
+       assert.NoError(t, err)
+
+       // Re-initialize the atomic trie
+       atomicTrie, err = newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 10 /*commitHeightInterval*/)
+       assert.NoError(t, err)
+
+       newHash, newHeight := atomicTrie.LastCommitted()
+       assert.Equal(t, height, newHeight, "height should not have changed")
+       assert.Equal(t, hash, newHash, "hash should be the same")
+}
+
+func newTestAtomicTrieIndexer(t *testing.T) AtomicTrie {
+       db := versiondb.New(memdb.New())
+       repo, err := NewAtomicTxRepository(db, testTxCodec(), 0)
+       assert.NoError(t, err)
+       indexer, err := newAtomicTrie(db, make(map[uint64]ids.ID), repo, testTxCodec(), 0, testCommitInterval)
+       assert.NoError(t, err)
+       assert.NotNil(t, indexer)
+       return indexer
+}
+
+func TestIndexerWriteAndRead(t *testing.T) {
+       atomicTrie := newTestAtomicTrieIndexer(t)
+
+       blockRootMap := make(map[uint64]common.Hash)
+       lastCommittedBlockHeight := uint64(0)
+       var lastCommittedBlockHash common.Hash
+
+       // process 205 blocks so that we get three commits (0, 100, 200)
+       for height := uint64(0); height <= testCommitInterval*2+5; /*=205*/ height++ {
+               atomicRequests := testDataImportTx().mustAtomicOps()
+               err := atomicTrie.Index(height, atomicRequests)
+               assert.NoError(t, err)
+               if height%testCommitInterval == 0 {
+                       lastCommittedBlockHash, lastCommittedBlockHeight = atomicTrie.LastCommitted()
+                       assert.NoError(t, err)
+                       assert.NotEqual(t, common.Hash{}, lastCommittedBlockHash)
+                       blockRootMap[lastCommittedBlockHeight] = lastCommittedBlockHash
+               }
+       }
+
+       // ensure we have 3 roots
+       assert.Len(t, blockRootMap, 3)
+
+       hash, height := atomicTrie.LastCommitted()
+       assert.EqualValues(t, lastCommittedBlockHeight, height, "expected %d was %d", 200, lastCommittedBlockHeight)
+       assert.Equal(t, lastCommittedBlockHash, hash)
+
+       // Verify that [atomicTrie] can access each of the expected roots
+       for height, hash := range blockRootMap {
+               root, err := atomicTrie.Root(height)
+               assert.NoError(t, err)
+               assert.Equal(t, hash, root)
+       }
+
+       // Ensure that Index refuses to accept blocks older than the last committed height
+       err := atomicTrie.Index(10, testDataExportTx().mustAtomicOps())
+       assert.Error(t, err)
+       assert.Equal(t, "height 10 must be after last committed height 200", err.Error())
+
+       // Ensure Index does not accept blocks beyond the next commit interval
+       nextCommitHeight := lastCommittedBlockHeight + testCommitInterval + 1 // =301
+       err = atomicTrie.Index(nextCommitHeight, testDataExportTx().mustAtomicOps())
+       assert.Error(t, err)
+       assert.Equal(t, "height 301 not within the next commit height 300", err.Error())
+}
+
+func TestAtomicOpsAreNotTxOrderDependent(t *testing.T) {
+       atomicTrie1 := newTestAtomicTrieIndexer(t)
+       atomicTrie2 := newTestAtomicTrieIndexer(t)
+
+       for height := uint64(0); height <= testCommitInterval; /*=205*/ height++ {
+               tx1 := testDataImportTx()
+               tx2 := testDataImportTx()
+               atomicRequests1, err := mergeAtomicOps([]*Tx{tx1, tx2})
+               assert.NoError(t, err)
+               atomicRequests2, err := mergeAtomicOps([]*Tx{tx2, tx1})
+               assert.NoError(t, err)
+
+               err = atomicTrie1.Index(height, atomicRequests1)
+               assert.NoError(t, err)
+               err = atomicTrie2.Index(height, atomicRequests2)
+               assert.NoError(t, err)
+       }
+       root1, height1 := atomicTrie1.LastCommitted()
+       root2, height2 := atomicTrie2.LastCommitted()
+       assert.NotEqual(t, common.Hash{}, root1)
+       assert.Equal(t, uint64(testCommitInterval), height1)
+       assert.Equal(t, uint64(testCommitInterval), height2)
+       assert.Equal(t, root1, root2)
+}
+
+func TestAtomicTrieSkipsBonusBlocks(t *testing.T) {
+       lastAcceptedHeight := uint64(100)
+       numTxsPerBlock := 3
+       commitInterval := uint64(10)
+       expectedCommitHeight := uint64(100)
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+       repo, err := NewAtomicTxRepository(db, codec, lastAcceptedHeight)
+       if err != nil {
+               t.Fatal(err)
+       }
+       operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+       writeTxs(t, repo, 0, lastAcceptedHeight, constTxsPerHeight(numTxsPerBlock), nil, operationsMap)
+
+       bonusBlocks := map[uint64]ids.ID{
+               10: {},
+               13: {},
+               14: {},
+       }
+       // Construct the atomic trie for the first time
+       atomicTrie, err := newAtomicTrie(db, bonusBlocks, repo, codec, lastAcceptedHeight, commitInterval)
+       if err != nil {
+               t.Fatal(err)
+       }
+       rootHash, commitHeight := atomicTrie.LastCommitted()
+       assert.EqualValues(t, expectedCommitHeight, commitHeight)
+       assert.NotEqual(t, common.Hash{}, rootHash)
+
+       // Verify the operations are as expected with the bonus block heights removed from the operations map
+       for height := range bonusBlocks {
+               delete(operationsMap, height)
+       }
+       verifyOperations(t, atomicTrie, codec, rootHash, 0, expectedCommitHeight, operationsMap)
+}
+
+func TestIndexingNilShouldNotImpactTrie(t *testing.T) {
+       // operations to index
+       ops := make([]map[ids.ID]*atomic.Requests, 0)
+       for i := 0; i <= testCommitInterval; i++ {
+               ops = append(ops, testDataImportTx().mustAtomicOps())
+       }
+
+       // without nils
+       a1 := newTestAtomicTrieIndexer(t)
+       for i := uint64(0); i <= testCommitInterval; i++ {
+               if i%2 == 0 {
+                       if err := a1.Index(i, ops[i]); err != nil {
+                               t.Fatal(err)
+                       }
+               } else {
+                       // do nothing
+               }
+       }
+
+       root1, height1 := a1.LastCommitted()
+       assert.NotEqual(t, common.Hash{}, root1)
+       assert.Equal(t, uint64(testCommitInterval), height1)
+
+       // with nils
+       a2 := newTestAtomicTrieIndexer(t)
+       for i := uint64(0); i <= testCommitInterval; i++ {
+               if i%2 == 0 {
+                       if err := a2.Index(i, ops[i]); err != nil {
+                               t.Fatal(err)
+                       }
+               } else {
+                       if err := a2.Index(i, nil); err != nil {
+                               t.Fatal(err)
+                       }
+               }
+       }
+       root2, height2 := a2.LastCommitted()
+       assert.NotEqual(t, common.Hash{}, root2)
+       assert.Equal(t, uint64(testCommitInterval), height2)
+
+       // key assertion of the test
+       assert.Equal(t, root1, root2)
+}
+
+func BenchmarkAtomicTrieInit(b *testing.B) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+
+       operationsMap := make(map[uint64]map[ids.ID]*atomic.Requests)
+
+       lastAcceptedHeight := uint64(25000)
+       // add 25000 * 3 = 75000 transactions
+       repo, err := NewAtomicTxRepository(db, codec, lastAcceptedHeight)
+       assert.NoError(b, err)
+       writeTxs(b, repo, 0, 25000, constTxsPerHeight(3), nil, operationsMap)
+
+       var atomicTrie AtomicTrie
+       var hash common.Hash
+       var height uint64
+       b.ReportAllocs()
+       b.ResetTimer()
+       for i := 0; i < b.N; i++ {
+               atomicTrie, err = newAtomicTrie(db, make(map[uint64]ids.ID), repo, codec, lastAcceptedHeight, 5000)
+               assert.NoError(b, err)
+
+               hash, height = atomicTrie.LastCommitted()
+               assert.Equal(b, lastAcceptedHeight, height)
+               assert.NotEqual(b, common.Hash{}, hash)
+       }
+       b.StopTimer()
+
+	// Verify operations
+	verifyOperations(b, atomicTrie, codec, hash, 0, lastAcceptedHeight, operationsMap)
+}
diff --git a/plugin/evm/atomic_tx_repository.go b/plugin/evm/atomic_tx_repository.go
new file mode 100644
index 00000000..e98e2f4d
--- /dev/null
+++ b/plugin/evm/atomic_tx_repository.go
@@ -0,0 +1,363 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+	"encoding/binary"
+	"fmt"
+	"sort"
+	"time"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/ethereum/go-ethereum/log"
+
+	"github.com/flare-foundation/flare/codec"
+	"github.com/flare-foundation/flare/database"
+	"github.com/flare-foundation/flare/database/prefixdb"
+	"github.com/flare-foundation/flare/database/versiondb"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/utils/units"
+	"github.com/flare-foundation/flare/utils/wrappers"
+)
+
+const (
+	commitSizeCap = 10 * units.MiB
+)
+
+var (
+	atomicTxIDDBPrefix         = []byte("atomicTxDB")
+	atomicHeightTxDBPrefix     = []byte("atomicHeightTxDB")
+	atomicRepoMetadataDBPrefix = []byte("atomicRepoMetadataDB")
+	maxIndexedHeightKey        = []byte("maxIndexedAtomicTxHeight")
+	bonusBlocksRepairedKey     = []byte("bonusBlocksRepaired")
+)
+
+// AtomicTxRepository defines an entity that manages storage and indexing of
+// atomic transactions
+type AtomicTxRepository interface {
+	GetIndexHeight() (uint64, error)
+	GetByTxID(txID ids.ID) (*Tx, uint64, error)
+	GetByHeight(height uint64) ([]*Tx, error)
+	Write(height uint64, txs []*Tx) error
+	WriteBonus(height uint64, txs []*Tx) error
+
+	IterateByHeight([]byte) database.Iterator
+
+	IsBonusBlocksRepaired() (bool, error)
+	MarkBonusBlocksRepaired(repairedEntries uint64) error
+}
+
+// atomicTxRepository is a prefixdb implementation of the AtomicTxRepository interface
+type atomicTxRepository struct {
+	// [acceptedAtomicTxDB] maintains an index of [txID] => [height]+[atomic tx] for all accepted atomic txs.
+	acceptedAtomicTxDB database.Database
+
+	// [acceptedAtomicTxByHeightDB] maintains an index of [height] => [atomic txs] for all accepted block heights.
+	acceptedAtomicTxByHeightDB database.Database
+
+	// [atomicRepoMetadataDB] maintains a single key-value pair which tracks the height up to which the atomic repository
+	// has indexed.
+	atomicRepoMetadataDB database.Database
+
+	// This db is used to store [maxIndexedHeightKey] to avoid interfering with the iterators over the atomic transaction DBs.
+	db *versiondb.Database
+
+	// Use this codec for serializing
+	codec codec.Manager
+}
+
+func NewAtomicTxRepository(db *versiondb.Database, codec codec.Manager, lastAcceptedHeight uint64) (AtomicTxRepository, error) {
+	repo := &atomicTxRepository{
+		acceptedAtomicTxDB:         prefixdb.New(atomicTxIDDBPrefix, db),
+		acceptedAtomicTxByHeightDB: prefixdb.New(atomicHeightTxDBPrefix, db),
+		atomicRepoMetadataDB:       prefixdb.New(atomicRepoMetadataDBPrefix, db),
+		codec:                      codec,
+		db:                         db,
+	}
+	return repo, repo.initializeHeightIndex(lastAcceptedHeight)
+}
+
+// initializeHeightIndex initializes the atomic repository and takes care of any required migration from the previous database
+// format which did not have a height -> txs index.
+func (a *atomicTxRepository) initializeHeightIndex(lastAcceptedHeight uint64) error {
+	startTime := time.Now()
+	lastLogTime := startTime
+
+	// [lastTxID] will be initialized to the last transaction that we indexed
+	// if we are part way through a migration.
+	var lastTxID ids.ID
+	indexHeightBytes, err := a.atomicRepoMetadataDB.Get(maxIndexedHeightKey)
+	switch err {
+	case nil:
+		break
+	case database.ErrNotFound:
+		break
+	default: // unexpected value in the database
+		return fmt.Errorf("found invalid value at max indexed height: %v", indexHeightBytes)
+	}
+
+	switch len(indexHeightBytes) {
+	case 0:
+		log.Info("Initializing atomic transaction repository from scratch")
+	case common.HashLength: // partially initialized
+		lastTxID, err = ids.ToID(indexHeightBytes)
+		if err != nil {
+			return err
+		}
+		log.Info("Initializing atomic transaction repository from txID", "lastTxID", lastTxID)
+	case wrappers.LongLen: // already initialized
+		return nil
+	default: // unexpected value in the database
+		return fmt.Errorf("found invalid value at max indexed height: %v", indexHeightBytes)
+	}
+
+	// Iterate from [lastTxID] to complete the re-index -> generating an index
+	// from height to a slice of transactions accepted at that height
+	iter := a.acceptedAtomicTxDB.NewIteratorWithStart(lastTxID[:])
+	defer iter.Release()
+
+	indexedTxs := 0
+
+	// Keep track of the size of the currently pending writes
+	pendingBytesApproximation := 0
+	for iter.Next() {
+		// iter.Value() consists of [height packed as uint64] + [tx serialized as packed []byte]
```

```
+               iterValue := iter.Value()
+               if len(iterValue) < wrappers.LongLen {
+                       return fmt.Errorf("atomic tx DB iterator value had invalid length (%d) < (%d)", len(iterValue), wrappers.LongLen)
+               }
+               heightBytes := iterValue[:wrappers.LongLen]
+
+               // Get the tx iter is pointing to, len(txs) == 1 is expected here.
+               txBytes := iterValue[wrappers.LongLen+wrappers.IntLen:]
+               tx, err := ExtractAtomicTx(txBytes, a.codec)
+               if err != nil {
+                       return err
+               }
+
+               // Check if there are already transactions at [height], to ensure that we
+               // add [txs] to the already indexed transactions at [height] instead of
+               // overwriting them.
+               if err := a.appendTxToHeightIndex(heightBytes, tx); err != nil {
+                       return err
+               }
+               lastTxID = tx.ID()
+               pendingBytesApproximation += len(txBytes)
+
+               // call commitFn to write to underlying DB if we have reached
+               // [commitSizeCap]
+               if pendingBytesApproximation > commitSizeCap {
+                       if err := a.atomicRepoMetadataDB.Put(maxIndexedHeightKey, lastTxID[:]); err != nil {
+                               return err
+                       }
+                       if err := a.db.Commit(); err != nil {
+                               return err
+                       }
+                       log.Info("Committing work initializing the atomic repository", "lastTxID", lastTxID, "pendingBytesApprox", pendingBytesApproximation)
+                       pendingBytesApproximation = 0
+               }
+               indexedTxs++
+               // Periodically log progress
+               if time.Since(lastLogTime) > 15*time.Second {
+                       lastLogTime = time.Now()
+                       log.Info("Atomic repository initialization", "indexedTxs", indexedTxs)
+               }
+       }
+       if err := iter.Error(); err != nil {
+               return fmt.Errorf("atomic tx DB iterator errored while initializing atomic trie: %w", err)
+       }
+
+       // Updated the value stored [maxIndexedHeightKey] to be the lastAcceptedHeight
+       indexedHeight := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(indexedHeight, lastAcceptedHeight)
+       if err := a.atomicRepoMetadataDB.Put(maxIndexedHeightKey, indexedHeight); err != nil {
+               return err
+       }
+
+       log.Info("Completed atomic transaction repository migration", "lastAcceptedHeight", lastAcceptedHeight, "duration", time.Since(startTime))
+       return a.db.Commit()
+}
+
+// GetIndexHeight returns the last height that was indexed by the atomic repository
+func (a *atomicTxRepository) GetIndexHeight() (uint64, error) {
+       indexHeightBytes, err := a.atomicRepoMetadataDB.Get(maxIndexedHeightKey)
+       if err != nil {
+               return 0, err
+       }
+
+       if len(indexHeightBytes) != wrappers.LongLen {
+               return 0, fmt.Errorf("unexpected length for indexHeightBytes %d", len(indexHeightBytes))
+       }
+       indexHeight := binary.BigEndian.Uint64(indexHeightBytes)
+       return indexHeight, nil
+}
+
+// GetByTxID queries [acceptedAtomicTxDB] for the [txID], parses a [*Tx] object
+// if an entry is found, and returns it with the block height the atomic tx it
+// represents was accepted on, along with an optional error.
+func (a *atomicTxRepository) GetByTxID(txID ids.ID) (*Tx, uint64, error) {
+       indexedTxBytes, err := a.acceptedAtomicTxDB.Get(txID[:])
+       if err != nil {
+               return nil, 0, err
+       }
+
+       if len(indexedTxBytes) < wrappers.LongLen {
+               return nil, 0, fmt.Errorf("acceptedAtomicTxDB entry too short: %d", len(indexedTxBytes))
+       }
+
+       // value is stored as [height]+[tx bytes], decompose with a packer.
+       packer := wrappers.Packer{Bytes: indexedTxBytes}
+       height := packer.UnpackLong()
+       txBytes := packer.UnpackBytes()
+       tx, err := ExtractAtomicTx(txBytes, a.codec)
+       if err != nil {
+               return nil, 0, err
+       }
+
+       return tx, height, nil
+}
+
+// GetByHeight returns all atomic txs processed on block at [height].
+// Returns [database.ErrNotFound] if there are no atomic transactions indexed at [height].
+// Note: if [height] is below the last accepted height, then this means that there were
+// no atomic transactions in the block accepted at [height].
+// If [height] is greater than the last accepted height, then this will always return
+// [database.ErrNotFound]
+func (a *atomicTxRepository) GetByHeight(height uint64) ([]*Tx, error) {
+       heightBytes := make([]byte, wrappers.LongLen)
+       binary.BigEndian.PutUint64(heightBytes, height)
+
+       return a.getByHeightBytes(heightBytes)
+}
+
+func (a *atomicTxRepository) getByHeightBytes(heightBytes []byte) ([]*Tx, error) {
+       txsBytes, err := a.acceptedAtomicTxByHeightDB.Get(heightBytes)
+       if err != nil {
+               return nil, err
+       }
+       return ExtractAtomicTxsBatch(txsBytes, a.codec)
+}
+
+// Write updates indexes maintained on atomic txs, so they can be queried
+// by txID or height. This method must be called only once per height,
+// and [txs] must include all atomic txs for the block accepted at the
+// corresponding height.
+func (a *atomicTxRepository) Write(height uint64, txs []*Tx) error {
+       return a.write(height, txs, false)
+}
+
+// WriteBonus is similar to Write, except the [txID] => [height] is not
+// overwritten if already exists.
+func (a *atomicTxRepository) WriteBonus(height uint64, txs []*Tx) error {
+       return a.write(height, txs, true)
+}
+
+func (a *atomicTxRepository) write(height uint64, txs []*Tx, bonus bool) error {
+       if len(txs) > 1 {
+               // txs should be stored in order of txID to ensure consistency
+               // with txs initialized from the txID index.
```

```
+		copyTxs := make([]*Tx, len(txs))
+		copy(copyTxs, txs)
+		sort.Slice(copyTxs, func(i, j int) bool { return copyTxs[i].ID().Hex() < copyTxs[j].ID().Hex() })
+		txs = copyTxs
+	}
+	heightBytes := make([]byte, wrappers.LongLen)
+	binary.BigEndian.PutUint64(heightBytes, height)
+	// Skip adding an entry to the height index if [txs] is empty.
+	if len(txs) > 0 {
+		for _, tx := range txs {
+			if bonus {
+				switch _, _, err := a.GetByTxID(tx.ID()); err {
+				case nil:
+					// avoid overwriting existing value if [bonus] is true
+					continue
+				case database.ErrNotFound:
+					// no existing value to overwrite, proceed as normal
+				default:
+					// unexpected error
+					return err
+				}
+			}
+			if err := a.indexTxByID(heightBytes, tx); err != nil {
+				return err
+			}
+		}
+		if err := a.indexTxsAtHeight(heightBytes, txs); err != nil {
+			return err
+		}
+	}
+
+	// Update the index height regardless of if any atomic transactions
+	// were present at [height].
+	return a.atomicRepoMetadataDB.Put(maxIndexedHeightKey, heightBytes)
+}
+
+// indexTxByID writes [tx] into the [acceptedAtomicTxDB] stored as
+// [height] + [tx bytes]
+func (a *atomicTxRepository) indexTxByID(heightBytes []byte, tx *Tx) error {
+	txBytes, err := a.codec.Marshal(codecVersion, tx)
+	if err != nil {
+		return err
+	}
+
+	// map txID => [height]+[tx bytes]
+	heightTxPacker := wrappers.Packer{Bytes: make([]byte, wrappers.LongLen+wrappers.IntLen+len(txBytes))}
+	heightTxPacker.PackFixedBytes(heightBytes)
+	heightTxPacker.PackBytes(txBytes)
+	txID := tx.ID()
+
+	if err := a.acceptedAtomicTxDB.Put(txID[:], heightTxPacker.Bytes); err != nil {
+		return err
+	}
+
+	return nil
+}
+
+// indexTxsAtHeight adds [height] -> [txs] to the [acceptedAtomicTxByHeightDB]
+func (a *atomicTxRepository) indexTxsAtHeight(heightBytes []byte, txs []*Tx) error {
+	txsBytes, err := a.codec.Marshal(codecVersion, txs)
+	if err != nil {
+		return err
+	}
+	if err := a.acceptedAtomicTxByHeightDB.Put(heightBytes, txsBytes); err != nil {
+		return err
+	}
+	return nil
+}
+
+// appendTxToHeightIndex retrieves the transactions stored at [heightBytes] and appends
+// [tx] to the slice of transactions stored there.
+// This function is used while initializing the atomic repository to re-index the atomic transactions
+// by txID into the height -> txs index.
+func (a *atomicTxRepository) appendTxToHeightIndex(heightBytes []byte, tx *Tx) error {
+	txs, err := a.getByHeightBytes(heightBytes)
+	if err != nil && err != database.ErrNotFound {
+		return err
+	}
+
+	// Iterate over the existing transactions to ensure we do not add a
+	// duplicate to the index.
+	for _, existingTx := range txs {
+		if existingTx.ID() == tx.ID() {
+			return nil
+		}
+	}
+
+	txs = append(txs, tx)
+	return a.indexTxsAtHeight(heightBytes, txs)
+}
+
+func (a *atomicTxRepository) IterateByHeight(heightBytes []byte) database.Iterator {
+	return a.acceptedAtomicTxByHeightDB.NewIteratorWithStart(heightBytes)
+}
+
+func (a *atomicTxRepository) IsBonusBlocksRepaired() (bool, error) {
+	return a.atomicRepoMetadataDB.Has(bonusBlocksRepairedKey)
+}
+
+func (a *atomicTxRepository) MarkBonusBlocksRepaired(repairedEntries uint64) error {
+	val := make([]byte, wrappers.LongLen)
+	binary.BigEndian.PutUint64(val, repairedEntries)
+	return a.atomicRepoMetadataDB.Put(bonusBlocksRepairedKey, val)
+}
diff --git a/plugin/evm/atomic_tx_repository_test.go b/plugin/evm/atomic_tx_repository_test.go
new file mode 100644
index 00000000..56b0c2dc
--- /dev/null
+++ b/plugin/evm/atomic_tx_repository_test.go
@@ -0,0 +1,287 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+	"sort"
+	"testing"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/flare-foundation/flare/chains/atomic"
+	"github.com/flare-foundation/flare/database"
+	"github.com/flare-foundation/flare/database/prefixdb"
+	"github.com/flare-foundation/flare/database/versiondb"
+
+	"github.com/flare-foundation/flare/codec"
+	"github.com/flare-foundation/flare/utils/wrappers"
+
+	"github.com/stretchr/testify/assert"
+
+	"github.com/flare-foundation/flare/database/memdb"
+	"github.com/flare-foundation/flare/ids"
+)
+
+// addTxs writes [txsPerHeight] txs for heights ranging in [fromHeight, toHeight) directly to [acceptedAtomicTxDB],
+// storing the resulting transactions in [txMap] if non-nil and the resulting atomic operations in [operationsMap]
+// if non-nil.
+func addTxs(t testing.TB, codec codec.Manager, acceptedAtomicTxDB database.Database, fromHeight uint64, toHeight uint64, txsPerHeight int, txMap map[uint64][]*Tx, operationsMap map[uint64]map[ids.ID]*at
+	for height := fromHeight; height < toHeight; height++ {
+		txs := make([]*Tx, 0, txsPerHeight)
+		for i := 0; i < txsPerHeight; i++ {
+			tx := newTestTx()
+			txs = append(txs, tx)
+			txBytes, err := codec.Marshal(codecVersion, tx)
+			assert.NoError(t, err)
+
+			// Write atomic transactions to the [acceptedAtomicTxDB]
+			// in the format handled prior to the migration to the atomic
+			// tx repository.
+			packer := wrappers.Packer{Bytes: make([]byte, 1), MaxSize: 1024 * 1024}
+			packer.PackLong(height)
+			packer.PackBytes(txBytes)
+			txID := tx.ID()
+			err = acceptedAtomicTxDB.Put(txID[:], packer.Bytes)
+			assert.NoError(t, err)
+		}
+		// save this to the map (if non-nil) for verifying expected results in verifyTxs
+		if txMap != nil {
+			txMap[height] = txs
+		}
+		if operationsMap != nil {
+			atomicRequests, err := mergeAtomicOps(txs)
+			if err != nil {
+				t.Fatal(err)
+			}
+			operationsMap[height] = atomicRequests
+		}
+	}
+}
+
+// constTxsPerHeight returns a function for passing to [writeTxs], which will return a constant number
+// as the number of atomic txs per height to create.
+func constTxsPerHeight(txCount int) func(uint64) int {
+	return func(uint64) int { return txCount }
+}
+
+// writeTxs writes [txsPerHeight] txs for heights ranging in [fromHeight, toHeight) through the Write call on [repo],
+// storing the resulting transactions in [txMap] if non-nil and the resulting atomic operations in [operationsMap]
+// if non-nil.
+func writeTxs(t testing.TB, repo AtomicTxRepository, fromHeight uint64, toHeight uint64,
+	txsPerHeight func(height uint64) int, txMap map[uint64][]*Tx, operationsMap map[uint64]map[ids.ID]*atomic.Requests,
+) {
+	for height := fromHeight; height < toHeight; height++ {
+		txs := newTestTxs(txsPerHeight(height))
+		if err := repo.Write(height, txs); err != nil {
+			t.Fatal(err)
+		}
+		// save this to the map (if non-nil) for verifying expected results in verifyTxs
+		if txMap != nil {
+			txMap[height] = txs
+		}
+		if operationsMap != nil {
+			atomicRequests, err := mergeAtomicOps(txs)
+			if err != nil {
+				t.Fatal(err)
+			}
+			if len(atomicRequests) == 0 {
+				continue
+			}
+			operationsMap[height] = atomicRequests
+		}
+	}
+}
+
+// verifyTxs asserts [repo] can find all txs in [txMap] by height and txID
+func verifyTxs(t testing.TB, repo AtomicTxRepository, txMap map[uint64][]*Tx) {
+	// We should be able to fetch indexed txs by height:
+	getComparator := func(txs []*Tx) func(int, int) bool {
+		return func(i, j int) bool {
+			return txs[i].ID().Hex() < txs[j].ID().Hex()
+		}
+	}
+	for height, expectedTxs := range txMap {
+		txs, err := repo.GetByHeight(height)
+		assert.NoErrorf(t, err, "unexpected error on GetByHeight at height=%d", height)
+		assert.Lenf(t, txs, len(expectedTxs), "wrong len of txs at height=%d", height)
+		// txs should be stored in order of txID
+		sort.Slice(expectedTxs, getComparator(expectedTxs))
+
+		txIDs := ids.Set{}
+		for i := 0; i < len(txs); i++ {
+			assert.Equalf(t, expectedTxs[i].ID().Hex(), txs[i].ID().Hex(), "wrong txID at height=%d idx=%d", height, i)
+			txIDs.Add(txs[i].ID())
+		}
+		assert.Equalf(t, len(txs), txIDs.Len(), "incorrect number of unique transactions in slice at height %d, expected %d, found %d", height, len(txs), txIDs.Len())
+	}
+}
+
+// verifyOperations creates an iterator over the atomicTrie at [rootHash] and verifies that the all of the operations in the trie in the interval [from, to] are identical to
+// the atomic operations contained in [operationsMap] on the same interval.
+func verifyOperations(t testing.TB, atomicTrie AtomicTrie, codec codec.Manager, rootHash common.Hash, from, to uint64, operationsMap map[uint64]map[ids.ID]*atomic.Requests) {
+	// Start the iterator at [from]
+	iter, err := atomicTrie.Iterator(rootHash, from)
+	if err != nil {
+		t.Fatal(err)
+	}
+
+	// Generate map of the marshalled atomic operations on the interval [from, to]
+	// based on [operationsMap].
+	marshalledOperationsMap := make(map[uint64]map[ids.ID][]byte)
+	for height, blockRequests := range operationsMap {
+		if height < from || height > to {
+			continue
+		}
+		for blockchainID, atomicRequests := range blockRequests {
+			b, err := codec.Marshal(0, atomicRequests)
+			if err != nil {
+				t.Fatal(err)
+			}
+			if requestsMap, exists := marshalledOperationsMap[height]; exists {
+				requestsMap[blockchainID] = b
+			} else {
+				requestsMap = make(map[ids.ID][]byte)
+				requestsMap[blockchainID] = b
+				marshalledOperationsMap[height] = requestsMap
+			}
+		}
+	}
+
+	// Generate map of marshalled atomic operations on the interval [from, to]
+	// based on the contents of the trie.
+	iteratorMarshalledOperationsMap := make(map[uint64]map[ids.ID][]byte)
+	for iter.Next() {
+		height := iter.BlockNumber()
+		if height < from {
+			t.Fatalf("Iterator starting at (%d) found value at block height (%d)", from, height)
+		}
+		if height > to {
```

```
+                       continue
+               }
+
+               blockchainID := iter.BlockchainID()
+               b, err := codec.Marshal(0, iter.AtomicOps())
+               if err != nil {
+                       t.Fatal(err)
+               }
+               if requestsMap, exists := iteratorMarshalledOperationsMap[height]; exists {
+                       requestsMap[blockchainID] = b
+               } else {
+                       requestsMap = make(map[ids.ID][]byte)
+                       requestsMap[blockchainID] = b
+                       iteratorMarshalledOperationsMap[height] = requestsMap
+               }
+       }
+       if err := iter.Error(); err != nil {
+               t.Fatal(err)
+       }
+
+       assert.Equal(t, marshalledOperationsMap, iteratorMarshalledOperationsMap)
+}
+
+func TestAtomicRepositoryReadWriteSingleTx(t *testing.T) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+       repo, err := NewAtomicTxRepository(db, codec, 0)
+       if err != nil {
+               t.Fatal(err)
+       }
+       txMap := make(map[uint64][]*Tx)
+
+       writeTxs(t, repo, 0, 100, constTxsPerHeight(1), txMap, nil)
+       verifyTxs(t, repo, txMap)
+}
+
+func TestAtomicRepositoryReadWriteMultipleTxs(t *testing.T) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+       repo, err := NewAtomicTxRepository(db, codec, 0)
+       if err != nil {
+               t.Fatal(err)
+       }
+       txMap := make(map[uint64][]*Tx)
+
+       writeTxs(t, repo, 0, 100, constTxsPerHeight(10), txMap, nil)
+       verifyTxs(t, repo, txMap)
+}
+
+func TestAtomicRepositoryPreAP5Migration(t *testing.T) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+
+       acceptedAtomicTxDB := prefixdb.New(atomicTxIDDBPrefix, db)
+       txMap := make(map[uint64][]*Tx)
+       addTxs(t, codec, acceptedAtomicTxDB, 0, 100, 1, txMap, nil)
+       if err := db.Commit(); err != nil {
+               t.Fatal(err)
+       }
+
+       // Ensure the atomic repository can correctly migrate the transactions
+       // from the old accepted atomic tx DB to add the height index.
+       repo, err := NewAtomicTxRepository(db, codec, 100)
+       if err != nil {
+               t.Fatal(err)
+       }
+       assert.NoError(t, err)
+       verifyTxs(t, repo, txMap)
+
+       writeTxs(t, repo, 100, 150, constTxsPerHeight(1), txMap, nil)
+       writeTxs(t, repo, 150, 200, constTxsPerHeight(10), txMap, nil)
+       verifyTxs(t, repo, txMap)
+}
+
+func TestAtomicRepositoryPostAP5Migration(t *testing.T) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+
+       acceptedAtomicTxDB := prefixdb.New(atomicTxIDDBPrefix, db)
+       txMap := make(map[uint64][]*Tx)
+       addTxs(t, codec, acceptedAtomicTxDB, 0, 100, 1, txMap, nil)
+       addTxs(t, codec, acceptedAtomicTxDB, 100, 200, 10, txMap, nil)
+       if err := db.Commit(); err != nil {
+               t.Fatal(err)
+       }
+
+       // Ensure the atomic repository can correctly migrate the transactions
+       // from the old accepted atomic tx DB to add the height index.
+       repo, err := NewAtomicTxRepository(db, codec, 200)
+       if err != nil {
+               t.Fatal(err)
+       }
+       assert.NoError(t, err)
+       verifyTxs(t, repo, txMap)
+
+       writeTxs(t, repo, 200, 300, constTxsPerHeight(10), txMap, nil)
+       verifyTxs(t, repo, txMap)
+}
+
+func benchAtomicRepositoryIndex10_000(b *testing.B, maxHeight uint64, txsPerHeight int) {
+       db := versiondb.New(memdb.New())
+       codec := testTxCodec()
+
+       acceptedAtomicTxDB := prefixdb.New(atomicTxIDDBPrefix, db)
+       txMap := make(map[uint64][]*Tx)
+
+       addTxs(b, codec, acceptedAtomicTxDB, 0, maxHeight, txsPerHeight, txMap, nil)
+       if err := db.Commit(); err != nil {
+               b.Fatal(err)
+       }
+       repo, err := NewAtomicTxRepository(db, codec, maxHeight)
+       if err != nil {
+               b.Fatal(err)
+       }
+       assert.NoError(b, err)
+       verifyTxs(b, repo, txMap)
+}
+
+func BenchmarkAtomicRepositoryIndex_10kBlocks_1Tx(b *testing.B) {
+       for n := 0; n < b.N; n++ {
+               benchAtomicRepositoryIndex10_000(b, 10_000, 1)
+       }
+}
+
+func BenchmarkAtomicRepositoryIndex_10kBlocks_10Tx(b *testing.B) {
+       for n := 0; n < b.N; n++ {
+               benchAtomicRepositoryIndex10_000(b, 10_000, 10)
+       }
+}
diff --git a/plugin/evm/block.go b/plugin/evm/block.go
index 8789c2a7..53e19576 100644
--- a/plugin/evm/block.go
+++ b/plugin/evm/block.go
@@ -12,90 +12,106 @@ import (
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"

-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"

-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow/choices"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow/choices"
 )

-var bonusBlocks = ids.Set{}
+var (
+       bonusBlocks              = ids.Set{}
+       bonusBlockMainnetHeights = make(map[uint64]ids.ID)
+       // first height that processed a TX included on a
+       // bonus block is the canonical height for that TX.
+       canonicalBonusBlocks = []uint64{
+               102928, 103035, 103038, 103114, 103193,
+               103234, 103338, 103444, 103480, 103491,
+               103513, 103533, 103535, 103538, 103541,
+               103546, 103571, 103572, 103619,
+               103287, 103624, 103591,
+       }
+)

 func init() {
-       blockIDStrs := []string{
-               "XMoEsew2DhSgQaydcJFJUQAQYP8BTNTYbEJZvtbrV2QsX7iE3",
-               "2QiHZwLhQ3xLuyyfcdo5yCUfoSqWDvRZox5ECU19HiswfroCGp",
-               "tLLijh7oKfvWT1yk9zRv4FQvuQ5DAiuvb5kHCNN9zh4mqkFMG",
-               "2db2wMbVAoCc5EUJrsBYWvNZDekqyY8uNpaaVapdBAQZ5oRaou",
-               "2rAsBj3emqQa13CV8r5fTtHogs4sXnjvbbXVzcKPi3WmzhpK9D",
-               "amgH2C1s9H3Av7vSW4y7n7TXb9tKyKHENvrDXutgNN6nsejgc",
-               "dWBsRYRwFrcyi3DPdLoHsL67QkZ5h86hwtVfP94ZBaY18EkmF",
-               "PgaRk1UAoUvRybhnXsrLq5t6imWhEa6ksNjbN6hWgs4qPrSzm",
-               "b7XfDDLgwB12DfL7UTWZoxwBpkLPL5mdHtXngD94Y2RoeWXSh",
-               "2i2FP6nJyvhX9FR15qN2D9AVoK5XKgBD2i2AQ7FoSpfowxvQDX",
-               "2J8z7HNv4nwh82wqRGyEHqQeuw4wJ6mCDCSvUgusBu35asnshK",
-               "2cUPPHy1hspr2nAKpQrrAEisLKkaWSS9iF2wjNFyFRs8vnSkKK",
-               "2gTygYckZgFZfN5QQWPaPBD3nabqjidV55mwy1x1Nd4JmJAwaM",
-               "5MptSdP6dBMPSwk9GJjeVe39deZJTRh9i82cgNibjeDffrrTf",
-               "2v3smb35s4GLACsK4Zkd2RcLBLdWA4huqrvq8Y3VP4CVe8kfTM",
-               "7KCZKBpxovtX9opb7rMRie9WmW5YbZ8A4HwBBokJ9eSHpZPqx",
-               "2oueNTj4dUE2FFtGyPpawnmCCsy6EUQeVHVLZy8NHeQmkAciP4",
-               "Nzs93kFTvcXanFUp9Y8VQkKYnzmH8xykxVNFJTkdyAEeuxWbP",
-               "2YHZ1KymFjiBhpXzgt6HXJhLSt5SV9UQ4tJuUNjfN1nQQdm5zz",
-               "Qv5v5Ru8ArfnWKB1w6s4G5EYPh7TybHJtF6UsVwAkfvZFoqmj",
-               "z3BgePPpCXq1mRBRvUi28rYYxnEtJizkUEHnDBrcZeVA7MFVk",
-               "Ry2sfjFfGEnJxRkUGFSyZNn7GR3m4aKAf1scDW2uXSNQB568Y",
-               "2YgxGHns7Z2hMMHJsPCgVXuJaL7x1b3gnHbmSCfCdyAcYGr6mx",
-               "cwJusfmn98TW3DjAbfLRN9utYR24KAQ82qpAXmVSvjHyJZuM2",
-               "2JbuExUGKW5mYz5KfXATwq1ibRDimgks9wEdYGNSC6Ttey1R4U",
-               "21Jys8UNURmtckKSV89S2hntEWymJszrLQbdLaNcbXcxDAsQSa",
-               "MjExz2z1qhwugc1tAyiGxRsCq4GvJwKfyyS29nr4tRVB8ooic",
-               "9oZh4qyBCcVwSGyDoUzRAuausvPJN3xH6nopKS6bwYzMfLoQ2",
-               "uK5Ff9iBfDtREpVv9NgCQ1STD1nzLJG3yrfibHG4mGvmybw6f",
-               "22ck2Z7cC38hmBfX2v3jMWxun8eD8psNaicfYeokS67DxwmPTx",
-               "2AfTQ2FXNj9bkSUQnud9pFXULx6EbF7cbbw6i3ayvc2QNhgxfF",
-               "pTf7gfk1ksj7bqMrLyMCij8FBKth1uRqQrtfykMFeXhx5xnrL",
-               "2AXxT3PSEnaYHNtBTnYrVTf24TtKDWjky9sqoFEhydrGXE9iKH",
-               "PJTkRrHvKZ1m4AQdPND1MBpUXpCrGN4DDmXmJQAiUrsxPoLQX",
-               "fV8k1U8oQDmfVwK66kAwN73aSsWiWhm8quNpVnKmSznBycV2W",
-               "sg6wAwFBsPQiS5Yfyh41cVkCRQbrrXsxXmeNyQ1xkunf2sdyv",
-               "soPweZ8DGaoUMjrnzjH3V2bypa7ZvvfqBan4UCsMUxMP759gw",
-               "2dNkpQF4mooveyUDfBYQTBfsGDV4wkncQPpEw4kHKfSTSTo5x",
-               "63YLdYXfXc5tY3mwWLaDsbXzQHYmwWVxMP7HKbRh4Du3C2iM1",
-               "2tCe88ur6MLQcVgwE5XxoaHiTGtSrthwKN3SdbHE4kWiQ7MSTV",
-               "2nG4exd9eUoAGzELfksmBR8XDCKhohY1uDKRFzEXJG4M8p3qA7",
-               "2F5tSQbdTfhZxvkxZqdFp7KR3FrJPKEsDLQK7KtPhNXj1EZAh4",
-               "21o2fVTnzzmtgXqkV1yuQeze7YEQhR5JB31jVVD9oVUnaaV8qm",
-               "2pSjfo7rkFCfZ2CqAxqfw8vqM2CU2nVLHrFZe3rwxz43gkVuGo",
-               "2QBNMMFJmhVHaGF45GAPszKyj1gK6ToBERRxYvXtM7yfrdUGPK",
-               "2ez4CA7w4HHr8SSobHQUAwFgj2giRNjNFUZK9JvrZFa1AuRj6X",
-               "2DpCuBaH94zKKFNY2XTs4GeJcwsEv6qT2DHc59S8tdg97GZpcJ",
-               "i1HoerJ1axognkUKKL58FvF9aLrbZKtv7TdKLkT5kgzoeU1vB",
-               "2SiSziHHqPjb1qkw7CdGYupokiYpd2b7mMqRiyszurctcA5AKr",
-               "esx5J962LtYm2aSrskpLai5e4CMMsaS1dsu9iuLGJ3KWgSu2M",
-               "2czmtnBS44VCWNRFUM89h4Fe9m3ZeZVYyh7Pe3FhNqjRNgPXhZ",
-               "DK9NqAJGry1wAo767uuYc1dYXAjUhzwka6vi8d9tNheqzGUTd",
-               "pE93VXY3N5QKfwsEFcM9i59UpPFgeZ8nxpJNaGaDQyDgsscNf",
-               "AfWvJH3rB2fdHuPWQp6qYNCFVT29MooQPRigD88rKKwUDEDhq",
-               "2KPW9G5tiNF14tZNfG4SqHuQrtUYVZyxuof37aZ7AnTKrQdsHn",
-               "BYqLB6xpqy7HsAgP2XNfGE8Ubg1uEzse5mBPTSJH9z5s8pvMa",
-               "Njm9TcLUXRojZk8YhEM6ksvfiPdC1TME4zJvGaDXgzMCyB6oB",
+       mainnetBonusBlocks := map[uint64]string{
+               102972: "Njm9TcLUXRojZk8YhEM6ksvfiPdC1TME4zJvGaDXgzMCyB6oB",
+               103105: "BYqLB6xpqy7HsAgP2XNfGE8Ubg1uEzse5mBPTSJH9z5s8pvMa",
+               103143: "AfWvJH3rB2fdHuPWQp6qYNCFVT29MooQPRigD88rKKwUDEDhq",
+               103183: "2KPW9G5tiNF14tZNfG4SqHuQrtUYVZyxuof37aZ7AnTKrQdsHn",
+               103197: "pE93VXY3N5QKfwsEFcM9i59UpPFgeZ8nxpJNaGaDQyDgsscNf",
+               103203: "2czmtnBS44VCWNRFUM89h4Fe9m3ZeZVYyh7Pe3FhNqjRNgPXhZ",
+               103208: "esx5J962LtYm2aSrskpLai5e4CMMsaS1dsu9iuLGJ3KWgSu2M",
+               103209: "DK9NqAJGry1wAo767uuYc1dYXAjUhzwka6vi8d9tNheqzGUTd",
+               103259: "i1HoerJ1axognkUKKL58FvF9aLrbZKtv7TdKLkT5kgzoeU1vB",
+               103261: "2DpCuBaH94zKKFNY2XTs4GeJcwsEv6qT2DHc59S8tdg97GZpcJ",
+               103266: "2ez4CA7w4HHr8SSobHQUAwFgj2giRNjNFUZK9JvrZFa1AuRj6X",
+               103287: "2QBNMMFJmhVHaGF45GAPszKyj1gK6ToBERRxYvXtM7yfrdUGPK",
+               103339: "2pSjfo7rkFCfZ2CqAxqfw8vqM2CU2nVLHrFZe3rwxz43gkVuGo",
+               103346: "2SiSziHHqPjb1qkw7CdGYupokiYpd2b7mMqRiyszurctcA5AKr",
+               103350: "2F5tSQbdTfhZxvkxZqdFp7KR3FrJPKEsDLQK7KtPhNXj1EZAh4",
+               103358: "2tCe88ur6MLQcVgwE5XxoaHiTGtSrthwKN3SdbHE4kWiQ7MSTV",
+               103437: "21o2fVTnzzmtgXqkV1yuQeze7YEQhR5JB31jVVD9oVUnaaV8qm",
+               103472: "2nG4exd9eUoAGzELfksmBR8XDCKhohY1uDKRFzEXJG4M8p3qA7",
+               103478: "63YLdYXfXc5tY3mwWLaDsbXzQHYmwWVxMP7HKbRh4Du3C2iM1",
+               103493: "soPweZ8DGaoUMjrnzjH3V2bypa7ZvvfqBan4UCsMUxMP759gw",
+               103514: "2dNkpQF4mooveyUDfBYQTBfsGDV4wkncQPpEw4kHKfSTSTo5x",
+               103536: "PJTkRrHvKZ1m4AQdPND1MBpUXpCrGN4DDmXmJQAiUrsxPoLQX",
+               103545: "22ck2Z7cC38hmBfX2v3jMWxun8eD8psNaicfYeokS67DxwmPTx",
+               103547: "pTf7gfk1ksj7bqMrLyMCij8FBKth1uRqQrtfykMFeXhx5xnrL",
+               103554: "9oZh4qyBCcVwSGyDoUzRAuausvPJN3xH6nopKS6bwYzMfLoQ2",
+               103555: "MjExz2z1qhwugc1tAyiGxRsCq4GvJwKfyyS29nr4tRVB8ooic",
+               103559: "cwJusfmn98TW3DjAbfLRN9utYR24KAQ82qpAXmVSvjHyJZuM2",
+               103561: "2YgxGHns7Z2hMMHJsPCgVXuJaL7x1b3gnHbmSCfCdyAcYGr6mx",
+               103563: "2AXxT3PSEnaYHNtBTnYrVTf24TtKDWjky9sqoFEhydrGXE9iKH",
+               103564: "Ry2sfjFfGEnJxRkUGFSyZNn7GR3m4aKAf1scDW2uXSNQB568Y",
+               103569: "21Jys8UNURmtckKSV89S2hntEWymJszrLQbdLaNcbXcxDAsQSa",
+               103570: "sg6wAwFBsPQiS5Yfyh41cVkCRQbrrXsxXmeNyQ1xkunf2sdyv",
+               103575: "z3BgePPpCXq1mRBRvUi28rYYxnEtJizkUEHnDBrcZeVA7MFVk",
+               103577: "uK5Ff9iBfDtREpVv9NgCQ1STD1nzLJG3yrfibHG4mGvmybw6f",
+               103578: "Qv5v5Ru8ArfnWKB1w6s4G5EYPh7TybHJtF6UsVwAkfvZFoqmj",
+               103582: "7KCZKBpxovtX9opb7rMRie9WmW5YbZ8A4HwBBokJ9eSHpZPqx",
+               103587: "2AfTQ2FXNj9bkSUQnud9pFXULx6EbF7cbbw6i3ayvc2QNhgxfF",
+               103590: "2gTygYckZgFZfN5QQWPaPBD3nabqjidV55mwy1x1Nd4JmJAwaM",
+               103591: "2cUPPHy1hspr2nAKpQrrAEisLKkaWSS9iF2wjNFyFRs8vnSkKK",
+               103594: "5MptSdP6dBMPSwk9GJjeVe39deZJTRh9i82cgNibjeDffrrTf",
+               103597: "2J8z7HNv4nwh82wqRGyEHqQeuw4wJ6mCDCSvUgusBu35asnshK",
+               103598: "2i2FP6nJyvhX9FR15qN2D9AVoK5XKgBD2i2AQ7FoSpfowxvQDX",
+               103603: "2v3smb35s4GLACsK4Zkd2RcLBLdWA4huqrvq8Y3VP4CVe8kfTM",
+               103604: "b7XfDDLgwB12DfL7UTWZoxwBpkLPL5mdHtXngD94Y2RoeWXSh",
+               103607: "PgaRk1UAoUvRybhnXsrLq5t6imWhEa6ksNjbN6hWgs4qPrSzm",
+               103612: "2oueNTj4dUE2FFtGyPpawnmCCsy6EUQeVHVLZy8NHeQmkAciP4",
+               103614: "2YHZ1KymFjiBhpXzgt6HXJhLSt5SV9UQ4tJuUNjfN1nQQdm5zz",
+               103617: "amgH2C1s9H3Av7vSW4y7n7TXb9tKyKHENvrDXutgNN6nsejgc",
+               103618: "fV8k1U8oQDmfVwK66kAwN73aSsWiWhm8quNpVnKmSznBycV2W",
+               103621: "Nzs93kFTvcXanFUp9Y8VQkKYnzmH8xykxVNFJTkdyAEeuxWbP",
+               103623: "2rAsBj3emqQa13CV8r5fTtHogs4sXnjvbbXVzcKPi3WmzhpK9D",
+               103624: "2JbuExUGKW5mYz5KfXATwq1ibRDimgks9wEdYGNSC6Ttey1R4U",
+               103627: "tLLijh7oKfvWT1yk9zRv4FQvuQ5DAiuvb5kHCNN9zh4mqkFMG",
+               103628: "dWBsRYRwFrcyi3DPdLoHsL67QkZ5h86hwtVfP94ZBaY18EkmF",
+               103629: "XMoEsew2DhSgQaydcJFJUQAQYP8BTNTYbEJZvtbrV2QsX7iE3",
+               103630: "2db2wMbVAoCc5EUJrsBYWvNZDekqyY8uNpaaVapdBAQZ5oRaou",
+               103633: "2QiHZwLhQ3xLuyyfcdo5yCUfoSqWDvRZox5ECU19HiswfroCGp",
        }
-       for _, blkIDStr := range blockIDStrs {
+
+       for height, blkIDStr := range mainnetBonusBlocks {
                blkID, err := ids.FromString(blkIDStr)
                if err != nil {
                        panic(err)
                }
                bonusBlocks.Add(blkID)
+               bonusBlockMainnetHeights[height] = blkID
        }
 }

 // Block implements the snowman.Block interface
 type Block struct {
-       id       ids.ID
-       ethBlock *types.Block
-       vm       *VM
-       status   choices.Status
+       id        ids.ID
+       ethBlock  *types.Block
+       vm        *VM
+       status    choices.Status
+       atomicTxs []*Tx
 }

 // ID implements the snowman.Block interface
@@ -118,23 +134,29 @@ func (b *Block) Accept() error {
                return fmt.Errorf("failed to put %s as the last accepted block: %w", b.ID(), err)
        }

-       tx, err := vm.extractAtomicTx(b.ethBlock)
+       if len(b.atomicTxs) == 0 {
+               if err := b.vm.atomicTrie.Index(b.Height(), nil); err != nil {
+                       return err
+               }
+               return vm.db.Commit()
+       }
+
+       batchChainsAndInputs, err := mergeAtomicOps(b.atomicTxs)
        if err != nil {
                return err
        }
-       if tx == nil {
-               return vm.db.Commit()
+       for _, tx := range b.atomicTxs {
+               // Remove the accepted transaction from the mempool
+               vm.mempool.RemoveTx(tx.ID())
        }

-       // Remove the accepted transaction from the mempool
-       vm.mempool.RemoveTx(tx.ID())
-
-       // Save the accepted atomic transaction
-       if err := vm.writeAtomicTx(b, tx); err != nil {
+       isBonus := bonusBlocks.Contains(b.id)
+       if err := b.indexAtomics(vm, b.Height(), b.atomicTxs, batchChainsAndInputs, isBonus); err != nil {
                return err
        }
-
-       if bonusBlocks.Contains(b.id) {
+       // If [b] is a bonus block, then we commit the database without applying the requests from
+       // the atmoic transactions to shared memory.
+       if isBonus {
                log.Info("skipping atomic tx acceptance on bonus block", "block", b.id)
                return vm.db.Commit()
        }
@@ -143,8 +165,22 @@ func (b *Block) Accept() error {
        if err != nil {
                return fmt.Errorf("failed to create commit batch due to: %w", err)
        }
+       return vm.ctx.SharedMemory.Apply(batchChainsAndInputs, batch)
+}

-       return tx.UnsignedAtomicTx.Accept(vm.ctx, batch)
+// indexAtomics writes given list of atomic transactions and atomic operations to atomic repository
+// and atomic trie respectively
+func (b *Block) indexAtomics(vm *VM, height uint64, atomicTxs []*Tx, batchChainsAndInputs map[ids.ID]*atomic.Requests, isBonus bool) error {
+       if isBonus {
+               // avoid indexing atomic operations of txs on bonus blocks in the trie
+               // so we do not re-execute them the second time that they appear
+               return vm.atomicTxRepository.WriteBonus(height, atomicTxs)
+       }
+
+       if err := vm.atomicTxRepository.Write(height, atomicTxs); err != nil {
+               return err
+       }
+       return b.vm.atomicTrie.Index(height, batchChainsAndInputs)
 }

 // Reject implements the snowman.Block interface
@@ -152,14 +188,12 @@ func (b *Block) Accept() error {
 func (b *Block) Reject() error {
        b.status = choices.Rejected
        log.Debug(fmt.Sprintf("Rejecting block %s (%s) at height %d", b.ID().Hex(), b.ID(), b.Height()))
-       tx, _ := b.vm.extractAtomicTx(b.ethBlock)
-       if tx != nil {
+       for _, tx := range b.atomicTxs {
                b.vm.mempool.RemoveTx(tx.ID())
                if err := b.vm.issueTx(tx, false /* set local to false when re-issuing */); err != nil {
                        log.Debug("Failed to re-issue transaction in rejected block", "txID", tx.ID(), "err", err)
                }
        }
-
        return b.vm.chain.Reject(b.ethBlock)
 }

@@ -179,7 +213,7 @@ func (b *Block) Parent() ids.ID {

 // Height implements the snowman.Block interface
 func (b *Block) Height() uint64 {
-       return b.ethBlock.Number().Uint64()
+       return b.ethBlock.NumberU64()
 }

 // Timestamp implements the snowman.Block interface
@@ -209,31 +243,33 @@ func (b *Block) verify(writes bool) error {
                return fmt.Errorf("syntactic block verification failed: %w", err)
        }

-       vm := b.vm
```

```
+       if err := b.verifyAtomicTxs(rules); err != nil {
+               return err
+       }
+
+       return b.vm.chain.BlockChain().InsertBlockManual(b.ethBlock, writes)
+}

+func (b *Block) verifyAtomicTxs(rules params.Rules) error {
        // Ensure that the parent was verified and inserted correctly.
        ancestorID := b.Parent()
        ancestorHash := common.Hash(ancestorID)
-       if !vm.chain.BlockChain().HasBlock(ancestorHash, b.Height()-1) {
+       if !b.vm.chain.BlockChain().HasBlock(ancestorHash, b.Height()-1) {
                return errRejectedParent
        }

        // If the tx is an atomic tx, ensure that it doesn't conflict with any of
        // its processing ancestry.
-       atomicTx, err := vm.extractAtomicTx(b.ethBlock)
-       if err != nil {
-               return err
-       }
-       if atomicTx != nil {
+       inputs := &ids.Set{}
+       for _, atomicTx := range b.atomicTxs {
                // If the ancestor is unknown, then the parent failed verification when
                // it was called.
                // If the ancestor is rejected, then this block shouldn't be inserted
-               // into the canonical chain because the parent is will be missing.
-               ancestorInf, err := vm.GetBlockInternal(ancestorID)
+               // into the canonical chain because the parent will be missing.
+               ancestorInf, err := b.vm.GetBlockInternal(ancestorID)
                if err != nil {
                        return errRejectedParent
                }
-
                if blkStatus := ancestorInf.Status(); blkStatus == choices.Unknown || blkStatus == choices.Rejected {
                        return errRejectedParent
                }
@@ -241,19 +277,22 @@ func (b *Block) verify(writes bool) error {
                if !ok {
                        return fmt.Errorf("expected %s, parent of %s, to be *Block but is %T", ancestor.ID(), b.ID(), ancestorInf)
                }
-
                if bonusBlocks.Contains(b.id) {
                        log.Info("skipping atomic tx verification on bonus block", "block", b.id)
                } else {
                        utx := atomicTx.UnsignedAtomicTx
-                       if err := utx.SemanticVerify(vm, atomicTx, ancestor, b.ethBlock.BaseFee(), rules); err != nil {
+                       if err := utx.SemanticVerify(b.vm, atomicTx, ancestor, b.ethBlock.BaseFee(), rules); err != nil {
                                return fmt.Errorf("invalid block due to failed semanatic verify: %w at height %d", err, b.Height())
                        }
+                       txInputs := utx.InputUTXOs()
+                       if inputs.Overlaps(txInputs) {
+                               return errConflictingAtomicInputs
+                       }
+                       inputs.Union(txInputs)
                }
        }

-       bc := vm.chain.BlockChain()
-       return bc.InsertBlockManual(b.ethBlock, writes)
+       return nil
 }

 // Bytes implements the snowman.Block interface
diff --git a/plugin/evm/block_builder.go b/plugin/evm/block_builder.go
index d86bccd4..49b32b78 100644
--- a/plugin/evm/block_builder.go
+++ b/plugin/evm/block_builder.go
@@ -8,13 +8,13 @@ import (
        "sync"
        "time"

-       coreth "github.com/ava-labs/coreth/chain"
-       "github.com/ava-labs/coreth/params"
+       coreth "github.com/flare-foundation/coreth/chain"
+       "github.com/flare-foundation/coreth/params"

-       "github.com/ava-labs/avalanchego/snow"
-       commonEng "github.com/ava-labs/avalanchego/snow/engine/common"
-       "github.com/ava-labs/avalanchego/utils/timer"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/flare/snow"
+       commonEng "github.com/flare-foundation/flare/snow/engine/common"
+       "github.com/flare-foundation/flare/utils/timer"
 )

 // buildingBlkStatus denotes the current status of the VM in block production.
@@ -52,9 +52,9 @@ type blockBuilder struct {
        ctx         *snow.Context
        chainConfig *params.ChainConfig

-       chain   *coreth.ETHChain
-       mempool *Mempool
-       network Network
+       chain    *coreth.ETHChain
+       mempool  *Mempool
+       gossiper Gossiper

        shutdownChan <-chan struct{}
        shutdownWg   *sync.WaitGroup
@@ -90,7 +90,7 @@ func (vm *VM) NewBlockBuilder(notifyBuildBlockChan chan<- commonEng.Message) *bl
                chainConfig:          vm.chainConfig,
                chain:                vm.chain,
                mempool:              vm.mempool,
-               network:              vm.network,
+               gossiper:             vm.gossiper,
                shutdownChan:         vm.shutdownChan,
                shutdownWg:           &vm.shutdownWg,
                notifyBuildBlockChan: notifyBuildBlockChan,
@@ -279,13 +279,13 @@ func (b *blockBuilder) awaitSubmittedTxs() {
                                b.signalTxsReady()

                                // We only attempt to invoke [GossipEthTxs] once AP4 is activated
-                               if b.isAP4 && b.network != nil && len(ethTxsEvent.Txs) > 0 {
+                               if b.isAP4 && b.gossiper != nil && len(ethTxsEvent.Txs) > 0 {
                                        // Give time for this node to build a block before attempting to
                                        // gossip
                                        time.Sleep(waitBlockTime)
-                                       // [GossipEthTxs] will block unless [pushNetwork.ethTxsToGossipChan] (an
+                                       // [GossipEthTxs] will block unless [gossiper.ethTxsToGossipChan] (an
                                        // unbuffered channel) is listened on
-                                       if err := b.network.GossipEthTxs(ethTxsEvent.Txs); err != nil {
+                                       if err := b.gossiper.GossipEthTxs(ethTxsEvent.Txs); err != nil {
                                                log.Warn(
                                                        "failed to gossip new eth transactions",
                                                        "err", err,
@@ -298,11 +298,11 @@ func (b *blockBuilder) awaitSubmittedTxs() {

                                // We only attempt to invoke [GossipAtomicTxs] once AP4 is activated
                                newTxs := b.mempool.GetNewTxs()
-                               if b.isAP4 && b.network != nil && len(newTxs) > 0 {
+                               if b.isAP4 && b.gossiper != nil && len(newTxs) > 0 {
					// Give time for this node to build a block before attempting to
					// gossip
					time.Sleep(waitBlockTime)
-					if err := b.network.GossipAtomicTxs(newTxs); err != nil {
+					if err := b.gossiper.GossipAtomicTxs(newTxs); err != nil {
						log.Warn(
							"failed to gossip new atomic transactions",
							"err", err,
diff --git a/plugin/evm/block_builder_test.go b/plugin/evm/block_builder_test.go
index 95b3b18f..e6e54ecd 100644
--- a/plugin/evm/block_builder_test.go
+++ b/plugin/evm/block_builder_test.go
@@ -9,9 +9,9 @@ import (
	"testing"
	"time"

-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/params"

-	"github.com/ava-labs/avalanchego/snow"
+	"github.com/flare-foundation/flare/snow"
 )

 func TestBlockBuilderShutsDown(t *testing.T) {
diff --git a/plugin/evm/block_verification.go b/plugin/evm/block_verification.go
index 6833f68c..5a0360da 100644
--- a/plugin/evm/block_verification.go
+++ b/plugin/evm/block_verification.go
@@ -7,11 +7,15 @@ import (
	"fmt"
	"math/big"

-	coreth "github.com/ava-labs/coreth/chain"
-	"github.com/ava-labs/coreth/core/types"
-	"github.com/ava-labs/coreth/params"
-	"github.com/ava-labs/coreth/trie"
	"github.com/ethereum/go-ethereum/common"
+
+	safemath "github.com/flare-foundation/flare/utils/math"
+
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/trie"
+
+	coreth "github.com/flare-foundation/coreth/chain"
 )

 var (
@@ -21,6 +25,7 @@ var (
	apricotPhase1MinGasPrice = big.NewInt(params.ApricotPhase1MinGasPrice)
	phase3BlockValidator     = blockValidatorPhase3{}
	phase4BlockValidator     = blockValidatorPhase4{}
+	phase5BlockValidator     = blockValidatorPhase5{}
 )

 type BlockValidator interface {
@@ -125,14 +130,8 @@ func (v blockValidatorPhase0) SyntacticVerify(b *Block) error {
		return errUnclesUnsupported
	}
	// Block must not be empty
-	//
-	// Note: extractAtomicTx also asserts a maximum size
-	atomicTx, err := b.vm.extractAtomicTx(b.ethBlock)
-	if err != nil {
-		return err
-	}
	txs := b.ethBlock.Transactions()
-	if len(txs) == 0 && atomicTx == nil {
+	if len(txs) == 0 && len(b.atomicTxs) == 0 {
		return errEmptyBlock
	}

@@ -230,14 +229,8 @@ func (blockValidatorPhase1) SyntacticVerify(b *Block) error {
		return errUnclesUnsupported
	}
	// Block must not be empty
-	//
-	// Note: extractAtomicTx also asserts a maximum size
-	atomicTx, err := b.vm.extractAtomicTx(b.ethBlock)
-	if err != nil {
-		return err
-	}
	txs := b.ethBlock.Transactions()
-	if len(txs) == 0 && atomicTx == nil {
+	if len(txs) == 0 && len(b.atomicTxs) == 0 {
		return errEmptyBlock
	}

@@ -340,14 +333,8 @@ func (blockValidatorPhase3) SyntacticVerify(b *Block) error {
		return errUnclesUnsupported
	}
	// Block must not be empty
-	//
-	// Note: extractAtomicTx also asserts a maximum size
-	atomicTx, err := b.vm.extractAtomicTx(b.ethBlock)
-	if err != nil {
-		return err
-	}
	txs := b.ethBlock.Transactions()
-	if len(txs) == 0 && atomicTx == nil {
+	if len(txs) == 0 && len(b.atomicTxs) == 0 {
		return errEmptyBlock
	}

@@ -443,14 +430,8 @@ func (blockValidatorPhase4) SyntacticVerify(b *Block) error {
		return errUnclesUnsupported
	}
	// Block must not be empty
-	//
-	// Note: extractAtomicTx also asserts a maximum size
-	atomicTx, err := b.vm.extractAtomicTx(b.ethBlock)
-	if err != nil {
-		return err
-	}
	txs := b.ethBlock.Transactions()
-	if len(txs) == 0 && atomicTx == nil {
+	if len(txs) == 0 && len(b.atomicTxs) == 0 {
		return errEmptyBlock
	}

@@ -465,28 +446,163 @@ func (blockValidatorPhase4) SyntacticVerify(b *Block) error {
		return errNilExtDataGasUsedApricotPhase4
	}
	if !ethHeader.ExtDataGasUsed.IsUint64() {
-		return fmt.Errorf("too large extDataGasUsed : bitlen %d", ethHeader.ExtDataGasUsed.BitLen())
+		return fmt.Errorf("too large extDataGasUsed: %d", ethHeader.ExtDataGasUsed)
	}
-	if atomicTx != nil {
+	var totalGasUsed uint64
+	for _, atomicTx := range b.atomicTxs {
		// We perform this check manually here to avoid the overhead of having to
		// reparse the atomicTx in `CalcExtDataGasUsed`.
-		gasUsed, err := atomicTx.GasUsed()
+		gasUsed, err := atomicTx.GasUsed(false)
```

```
		if err != nil {
			return err
		}
-		if ethHeader.ExtDataGasUsed.Cmp(new(big.Int).SetUint64(gasUsed)) != 0 {
-			return fmt.Errorf("invalid extDataGasUsed: have %d, want %d", ethHeader.ExtDataGasUsed, gasUsed)
+		totalGasUsed, err = safemath.Add64(totalGasUsed, gasUsed)
+		if err != nil {
+			return err
		}
	}

+	switch {
+	case ethHeader.ExtDataGasUsed.Cmp(new(big.Int).SetUint64(totalGasUsed)) != 0:
+		return fmt.Errorf("invalid extDataGasUsed: have %d, want %d", ethHeader.ExtDataGasUsed, totalGasUsed)
+
	// Make sure BlockGasCost is not nil
	// NOTE: ethHeader.BlockGasCost correctness is checked in header verification
-	if ethHeader.BlockGasCost == nil {
+	case ethHeader.BlockGasCost == nil:
		return errNilBlockGasCostApricotPhase4
+	case !ethHeader.BlockGasCost.IsUint64():
+		return fmt.Errorf("too large blockGasCost: %d", ethHeader.BlockGasCost)
	}
-	if !ethHeader.BlockGasCost.IsUint64() {
-		return fmt.Errorf("too large blockGasCost: bitlen %d", ethHeader.BlockGasCost.BitLen())
+	return nil
+}
+
+type blockValidatorPhase5 struct{}
+
+func (blockValidatorPhase5) SyntacticVerify(b *Block) error {
+	if b == nil || b.ethBlock == nil {
+		return errInvalidBlock
	}

+	// Skip verification of the genesis block since it
+	// should already be marked as accepted
+	if b.ethBlock.Hash() == b.vm.genesisHash {
+		return nil
+	}
+
+	// Perform block and header sanity checks
+	ethHeader := b.ethBlock.Header()
+	if ethHeader.Number == nil || !ethHeader.Number.IsUint64() {
+		return errInvalidBlock
+	}
+	if ethHeader.Difficulty == nil || !ethHeader.Difficulty.IsUint64() ||
+		ethHeader.Difficulty.Uint64() != 1 {
+		return fmt.Errorf(
+			"expected difficulty to be 1 but got %v: %w",
+			ethHeader.Difficulty, errInvalidDifficulty,
+		)
+	}
+	if ethHeader.Nonce.Uint64() != 0 {
+		return fmt.Errorf(
+			"expected nonce to be 0 but got %d: %w",
+			ethHeader.Nonce.Uint64(), errInvalidNonce,
+		)
+	}
+	if ethHeader.GasLimit != params.ApricotPhase5GasLimit {
+		return fmt.Errorf(
+			"expected gas limit to be %d in apricot phase 5 but got %d",
+			params.ApricotPhase5GasLimit, ethHeader.GasLimit,
+		)
+	}
+	if ethHeader.MixDigest != (common.Hash{}) {
+		return fmt.Errorf(
+			"expected MixDigest to be empty but got %x: %w",
+			ethHeader.MixDigest, errInvalidMixDigest,
+		)
+	}
+	if hash := types.CalcExtDataHash(b.ethBlock.ExtData()); ethHeader.ExtDataHash != hash {
+		return fmt.Errorf("extra data hash mismatch: have %x, want %x", ethHeader.ExtDataHash, hash)
+	}
+	if headerExtraDataSize := len(ethHeader.Extra); headerExtraDataSize != params.ApricotPhase3ExtraDataSize {
+		return fmt.Errorf(
+			"expected header ExtraData to be %d but got %d: %w",
+			params.ApricotPhase3ExtraDataSize, headerExtraDataSize, errHeaderExtraDataTooBig,
+		)
+	}
+	if ethHeader.BaseFee == nil {
+		return errNilBaseFeeApricotPhase3
+	}
+	if bfLen := ethHeader.BaseFee.BitLen(); bfLen > 256 {
+		return fmt.Errorf("too large base fee: bitlen %d", bfLen)
+	}
+	if b.ethBlock.Version() != 0 {
+		return fmt.Errorf(
+			"expected block version to be 0 but got %d: %w",
+			b.ethBlock.Version(), errInvalidBlockVersion,
+		)
+	}
+
+	// Check that the tx hash in the header matches the body
+	txsHash := types.DeriveSha(b.ethBlock.Transactions(), new(trie.Trie))
+	if txsHash != ethHeader.TxHash {
+		return errTxHashMismatch
+	}
+	// Check that the uncle hash in the header matches the body
+	uncleHash := types.CalcUncleHash(b.ethBlock.Uncles())
+	if uncleHash != ethHeader.UncleHash {
+		return errUncleHashMismatch
+	}
+	// Coinbase must be zero on C-Chain
+	if b.ethBlock.Coinbase() != coreth.BlackholeAddr {
+		return errInvalidBlock
+	}
+	// Block must not have any uncles
+	if len(b.ethBlock.Uncles()) > 0 {
+		return errUnclesUnsupported
+	}
+	// Block must not be empty
+	txs := b.ethBlock.Transactions()
+	if len(txs) == 0 && len(b.atomicTxs) == 0 {
+		return errEmptyBlock
+	}
+
+	// Make sure the block isn't too far in the future
+	blockTimestamp := b.ethBlock.Time()
+	if maxBlockTime := uint64(b.vm.clock.Time().Add(maxFutureBlockTime).Unix()); blockTimestamp > maxBlockTime {
+		return fmt.Errorf("block timestamp is too far in the future: %d > allowed %d", blockTimestamp, maxBlockTime)
+	}
+
+	// Make sure ExtDataGasUsed is not nil and correct
+	if ethHeader.ExtDataGasUsed == nil {
+		return errNilExtDataGasUsedApricotPhase4
+	}
+	if ethHeader.ExtDataGasUsed.Cmp(params.AtomicGasLimit) == 1 {
+		return fmt.Errorf("too large extDataGasUsed: %d", ethHeader.ExtDataGasUsed)
+	}
+
+	var totalGasUsed uint64
+	for _, atomicTx := range b.atomicTxs {
+		// We perform this check manually here to avoid the overhead of having to
```

```
+               // reparse the atomicTx in `CalcExtDataGasUsed`.
+               gasUsed, err := atomicTx.GasUsed(true)
+               if err != nil {
+                       return err
+               }
+               totalGasUsed, err = safemath.Add64(totalGasUsed, gasUsed)
+               if err != nil {
+                       return err
+               }
+       }
+
+       switch {
+       case ethHeader.ExtDataGasUsed.Cmp(new(big.Int).SetUint64(totalGasUsed)) != 0:
+               return fmt.Errorf("invalid extDataGasUsed: have %d, want %d", ethHeader.ExtDataGasUsed, totalGasUsed)
+
+       // Make sure BlockGasCost is not nil
+       // NOTE: ethHeader.BlockGasCost correctness is checked in header verification
+       case ethHeader.BlockGasCost == nil:
+               return errNilBlockGasCostApricotPhase4
+       case !ethHeader.BlockGasCost.IsUint64():
+               return fmt.Errorf("too large blockGasCost: %d", ethHeader.BlockGasCost)
+       }
        return nil
 }
diff --git a/plugin/evm/client.go b/plugin/evm/client.go
index 4a33485d..3a37c053 100644
--- a/plugin/evm/client.go
+++ b/plugin/evm/client.go
@@ -4,44 +4,66 @@
 package evm

 import (
+       "context"
        "fmt"
-       "time"

-       "github.com/ava-labs/avalanchego/api"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/formatting"
-       cjson "github.com/ava-labs/avalanchego/utils/json"
-       "github.com/ava-labs/avalanchego/utils/rpc"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/flare/api"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/formatting"
+       cjson "github.com/flare-foundation/flare/utils/json"
+       "github.com/flare-foundation/flare/utils/rpc"
 )

-// Client ...
-type Client struct {
+// Interface compliance
+var _ Client = (*client)(nil)
+
+// Client interface for interacting with EVM [chain]
+type Client interface {
+       IssueTx(ctx context.Context, txBytes []byte) (ids.ID, error)
+       GetAtomicTxStatus(ctx context.Context, txID ids.ID) (Status, error)
+       GetAtomicTx(ctx context.Context, txID ids.ID) ([]byte, error)
+       GetAtomicUTXOs(ctx context.Context, addrs []string, sourceChain string, limit uint32, startAddress, startUTXOID string) ([][]byte, api.Index, error)
+       ListAddresses(ctx context.Context, userPass api.UserPass) ([]string, error)
+       ExportKey(ctx context.Context, userPass api.UserPass, addr string) (string, string, error)
+       ImportKey(ctx context.Context, userPass api.UserPass, privateKey string) (string, error)
+       Import(ctx context.Context, userPass api.UserPass, to string, sourceChain string) (ids.ID, error)
+       ExportAVAX(ctx context.Context, userPass api.UserPass, amount uint64, to string) (ids.ID, error)
+       Export(ctx context.Context, userPass api.UserPass, amount uint64, to string, assetID string) (ids.ID, error)
+       StartCPUProfiler(ctx context.Context) (bool, error)
+       StopCPUProfiler(ctx context.Context) (bool, error)
+       MemoryProfile(ctx context.Context) (bool, error)
+       LockProfile(ctx context.Context) (bool, error)
+       SetLogLevel(ctx context.Context, level log.Lvl) (bool, error)
+}
+
+// Client implementation for interacting with EVM [chain]
+type client struct {
        requester      rpc.EndpointRequester
        adminRequester rpc.EndpointRequester
 }

 // NewClient returns a Client for interacting with EVM [chain]
-func NewClient(uri, chain string, requestTimeout time.Duration) *Client {
-       return &Client{
-               requester:      rpc.NewEndpointRequester(uri, fmt.Sprintf("/ext/bc/%s/avax", chain), "avax", requestTimeout),
-               adminRequester: rpc.NewEndpointRequester(uri, fmt.Sprintf("/ext/bc/%s/admin", chain), "admin", requestTimeout),
+func NewClient(uri, chain string) Client {
+       return &client{
+               requester:      rpc.NewEndpointRequester(uri, fmt.Sprintf("/ext/bc/%s/avax", chain), "avax"),
+               adminRequester: rpc.NewEndpointRequester(uri, fmt.Sprintf("/ext/bc/%s/admin", chain), "admin"),
        }
 }

 // NewCChainClient returns a Client for interacting with the C Chain
-func NewCChainClient(uri string, requestTimeout time.Duration) *Client {
-       return NewClient(uri, "C", requestTimeout)
+func NewCChainClient(uri string) Client {
+       return NewClient(uri, "C")
 }

 // IssueTx issues a transaction to a node and returns the TxID
-func (c *Client) IssueTx(txBytes []byte) (ids.ID, error) {
+func (c *client) IssueTx(ctx context.Context, txBytes []byte) (ids.ID, error) {
        res := &api.JSONTxID{}
        txStr, err := formatting.EncodeWithChecksum(formatting.Hex, txBytes)
        if err != nil {
                return res.TxID, fmt.Errorf("problem hex encoding bytes: %w", err)
        }
-       err = c.requester.SendRequest("issueTx", &api.FormattedTx{
+       err = c.requester.SendRequest(ctx, "issueTx", &api.FormattedTx{
                Tx:       txStr,
                Encoding: formatting.Hex,
        }, res)
@@ -49,18 +71,18 @@ func (c *Client) IssueTx(txBytes []byte) (ids.ID, error) {
 }

 // GetAtomicTxStatus returns the status of [txID]
-func (c *Client) GetAtomicTxStatus(txID ids.ID) (Status, error) {
+func (c *client) GetAtomicTxStatus(ctx context.Context, txID ids.ID) (Status, error) {
        res := &GetAtomicTxStatusReply{}
-       err := c.requester.SendRequest("getAtomicTxStatus", &api.JSONTxID{
+       err := c.requester.SendRequest(ctx, "getAtomicTxStatus", &api.JSONTxID{
                TxID: txID,
        }, res)
        return res.Status, err
 }

 // GetAtomicTx returns the byte representation of [txID]
-func (c *Client) GetAtomicTx(txID ids.ID) ([]byte, error) {
+func (c *client) GetAtomicTx(ctx context.Context, txID ids.ID) ([]byte, error) {
        res := &api.FormattedTx{}
-       err := c.requester.SendRequest("getAtomicTx", &api.GetTxArgs{
+       err := c.requester.SendRequest(ctx, "getAtomicTx", &api.GetTxArgs{
                TxID:     txID,
                Encoding: formatting.Hex,
        }, res)
```

```
@@ -73,9 +95,9 @@ func (c *Client) GetAtomicTx(txID ids.ID) ([]byte, error) {

 // GetAtomicUTXOs returns the byte representation of the atomic UTXOs controlled by [addresses]
 // from [sourceChain]
-func (c *Client) GetAtomicUTXOs(addrs []string, sourceChain string, limit uint32, startAddress, startUTXOID string) ([][]byte, api.Index, error) {
+func (c *client) GetAtomicUTXOs(ctx context.Context, addrs []string, sourceChain string, limit uint32, startAddress, startUTXOID string) ([][]byte, api.Index, error) {
        res := &api.GetUTXOsReply{}
-       err := c.requester.SendRequest("getUTXOs", &api.GetUTXOsArgs{
+       err := c.requester.SendRequest(ctx, "getUTXOs", &api.GetUTXOsArgs{
                Addresses:   addrs,
                SourceChain: sourceChain,
                Limit:       cjson.Uint32(limit),
@@ -101,17 +123,17 @@ func (c *Client) GetAtomicUTXOs(addrs []string, sourceChain string, limit uint32
 }

 // ListAddresses returns all addresses on this chain controlled by [user]
-func (c *Client) ListAddresses(user api.UserPass) ([]string, error) {
+func (c *client) ListAddresses(ctx context.Context, user api.UserPass) ([]string, error) {
        res := &api.JSONAddresses{}
-       err := c.requester.SendRequest("listAddresses", &user, res)
+       err := c.requester.SendRequest(ctx, "listAddresses", &user, res)
        return res.Addresses, err
 }

 // ExportKey returns the private key corresponding to [addr] controlled by [user]
 // in both Avalanche standard format and hex format
-func (c *Client) ExportKey(user api.UserPass, addr string) (string, string, error) {
+func (c *client) ExportKey(ctx context.Context, user api.UserPass, addr string) (string, string, error) {
        res := &ExportKeyReply{}
-       err := c.requester.SendRequest("exportKey", &ExportKeyArgs{
+       err := c.requester.SendRequest(ctx, "exportKey", &ExportKeyArgs{
                UserPass: user,
                Address:  addr,
        }, res)
@@ -119,9 +141,9 @@ func (c *Client) ExportKey(user api.UserPass, addr string) (string, string, erro
 }

 // ImportKey imports [privateKey] to [user]
-func (c *Client) ImportKey(user api.UserPass, privateKey string) (string, error) {
+func (c *client) ImportKey(ctx context.Context, user api.UserPass, privateKey string) (string, error) {
        res := &api.JSONAddress{}
-       err := c.requester.SendRequest("importKey", &ImportKeyArgs{
+       err := c.requester.SendRequest(ctx, "importKey", &ImportKeyArgs{
                UserPass:   user,
                PrivateKey: privateKey,
        }, res)
@@ -130,9 +152,9 @@ func (c *Client) ImportKey(user api.UserPass, privateKey string) (string, error)

 // Import sends an import transaction to import funds from [sourceChain] and
 // returns the ID of the newly created transaction
-func (c *Client) Import(user api.UserPass, to, sourceChain string) (ids.ID, error) {
+func (c *client) Import(ctx context.Context, user api.UserPass, to, sourceChain string) (ids.ID, error) {
        res := &api.JSONTxID{}
-       err := c.requester.SendRequest("import", &ImportArgs{
+       err := c.requester.SendRequest(ctx, "import", &ImportArgs{
                UserPass:    user,
                To:          to,
                SourceChain: sourceChain,
@@ -142,25 +164,27 @@ func (c *Client) Import(user api.UserPass, to, sourceChain string) (ids.ID, erro

 // ExportAVAX sends AVAX from this chain to the address specified by [to].
 // Returns the ID of the newly created atomic transaction
-func (c *Client) ExportAVAX(
+func (c *client) ExportAVAX(
+       ctx context.Context,
        user api.UserPass,
        amount uint64,
        to string,
 ) (ids.ID, error) {
-       return c.Export(user, amount, to, "AVAX")
+       return c.Export(ctx, user, amount, to, "AVAX")
 }

 // Export sends an asset from this chain to the P/C-Chain.
 // After this tx is accepted, the AVAX must be imported to the P/C-chain with an importTx.
 // Returns the ID of the newly created atomic transaction
-func (c *Client) Export(
+func (c *client) Export(
+       ctx context.Context,
        user api.UserPass,
        amount uint64,
        to string,
        assetID string,
 ) (ids.ID, error) {
        res := &api.JSONTxID{}
-       err := c.requester.SendRequest("export", &ExportArgs{
+       err := c.requester.SendRequest(ctx, "export", &ExportArgs{
                ExportAVAXArgs: ExportAVAXArgs{
                        UserPass: user,
                        Amount:   cjson.Uint64(amount),
@@ -171,34 +195,34 @@ func (c *Client) Export(
        return res.TxID, err
 }

-func (c *Client) StartCPUProfiler() (bool, error) {
+func (c *client) StartCPUProfiler(ctx context.Context) (bool, error) {
        res := &api.SuccessResponse{}
-       err := c.adminRequester.SendRequest("startCPUProfiler", struct{}{}, res)
+       err := c.adminRequester.SendRequest(ctx, "startCPUProfiler", struct{}{}, res)
        return res.Success, err
 }

-func (c *Client) StopCPUProfiler() (bool, error) {
+func (c *client) StopCPUProfiler(ctx context.Context) (bool, error) {
        res := &api.SuccessResponse{}
-       err := c.adminRequester.SendRequest("stopCPUProfiler", struct{}{}, res)
+       err := c.adminRequester.SendRequest(ctx, "stopCPUProfiler", struct{}{}, res)
        return res.Success, err
 }

-func (c *Client) MemoryProfile() (bool, error) {
+func (c *client) MemoryProfile(ctx context.Context) (bool, error) {
        res := &api.SuccessResponse{}
-       err := c.adminRequester.SendRequest("memoryProfile", struct{}{}, res)
+       err := c.adminRequester.SendRequest(ctx, "memoryProfile", struct{}{}, res)
        return res.Success, err
 }

-func (c *Client) LockProfile() (bool, error) {
+func (c *client) LockProfile(ctx context.Context) (bool, error) {
        res := &api.SuccessResponse{}
-       err := c.adminRequester.SendRequest("lockProfile", struct{}{}, res)
+       err := c.adminRequester.SendRequest(ctx, "lockProfile", struct{}{}, res)
        return res.Success, err
 }

 // SetLogLevel dynamically sets the log level for the C Chain
-func (c *Client) SetLogLevel(level log.Lvl) (bool, error) {
+func (c *client) SetLogLevel(ctx context.Context, level log.Lvl) (bool, error) {
        res := &api.SuccessResponse{}
-       err := c.adminRequester.SendRequest("setLogLevel", &SetLogLevelArgs{
+       err := c.adminRequester.SendRequest(ctx, "setLogLevel", &SetLogLevelArgs{
                Level: level.String(),
        }, res)
        return res.Success, err
diff --git a/plugin/evm/client_interface_test.go b/plugin/evm/client_interface_test.go
new file mode 100644
index 00000000..d88c4926
--- /dev/null
+++ b/plugin/evm/client_interface_test.go
@@ -0,0 +1,17 @@
+package evm
+
+import (
+       "reflect"
+       "testing"
+)
+
+func TestInterfaceStructOneToOne(t *testing.T) {
+       // checks struct provides at least the methods signatures in the interface
+       var _ Client = (*client)(nil)
+       // checks interface and struct have the same number of methods
+       clientType := reflect.TypeOf(&client{})
+       ClientType := reflect.TypeOf((*Client)(nil)).Elem()
+       if clientType.NumMethod() != ClientType.NumMethod() {
+               t.Fatalf("no 1 to 1 compliance between struct methods (%v) and interface methods (%v)", clientType.NumMethod(), ClientType.NumMethod())
+       }
+}
diff --git a/plugin/evm/codec.go b/plugin/evm/codec.go
index 3a49eb8e..cbc5365c 100644
--- a/plugin/evm/codec.go
+++ b/plugin/evm/codec.go
@@ -4,10 +4,12 @@
 package evm
 
 import (
-       "github.com/ava-labs/avalanchego/codec"
-       "github.com/ava-labs/avalanchego/codec/linearcodec"
-       "github.com/ava-labs/avalanchego/utils/wrappers"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
+       "fmt"
+
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/codec/linearcodec"
+       "github.com/flare-foundation/flare/utils/wrappers"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"
 )
 
 // Codec does serialization and deserialization
@@ -38,3 +40,60 @@ func init() {
                panic(errs.Err)
        }
 }
+
+// extractAtomicTxs returns the atomic transactions in [atomicTxBytes] if
+// they exist.
+// if [batch] is true, it attempts to unmarshal [atomicTxBytes] as a slice of
+// transactions (post-ApricotPhase5), and if it is false, then it unmarshals
+// it as a single atomic transaction.
+func ExtractAtomicTxs(atomicTxBytes []byte, batch bool, codec codec.Manager) ([]*Tx, error) {
+       if len(atomicTxBytes) == 0 {
+               return nil, nil
+       }
+
+       if !batch {
+               tx, err := ExtractAtomicTx(atomicTxBytes, codec)
+               if err != nil {
+                       return nil, err
+               }
+               return []*Tx{tx}, err
+       }
+       return ExtractAtomicTxsBatch(atomicTxBytes, codec)
+}
+
+// [ExtractAtomicTx] extracts a singular atomic transaction from [atomicTxBytes]
+// and returns a slice of atomic transactions for compatibility with the type returned post
+// ApricotPhase5.
+// Note: this function assumes [atomicTxBytes] is non-empty.
+func ExtractAtomicTx(atomicTxBytes []byte, codec codec.Manager) (*Tx, error) {
+       atomicTx := new(Tx)
+       if _, err := codec.Unmarshal(atomicTxBytes, atomicTx); err != nil {
+               return nil, fmt.Errorf("failed to unmarshal atomic transaction (pre-AP5): %w", err)
+       }
+       if err := atomicTx.Sign(codec, nil); err != nil {
+               return nil, fmt.Errorf("failed to initialize singleton atomic tx due to: %w", err)
+       }
+       return atomicTx, nil
+}
+
+// [ExtractAtomicTxsBatch] extracts a slice of atomic transactions from [atomicTxBytes].
+// Note: this function assumes [atomicTxBytes] is non-empty.
+func ExtractAtomicTxsBatch(atomicTxBytes []byte, codec codec.Manager) ([]*Tx, error) {
+       var atomicTxs []*Tx
+       if _, err := codec.Unmarshal(atomicTxBytes, &atomicTxs); err != nil {
+               return nil, fmt.Errorf("failed to unmarshal atomic tx (AP5) due to %w", err)
+       }
+
+       // Do not allow non-empty extra data field to contain zero atomic transactions. This would allow
+       // people to construct a block that contains useless data.
+       if len(atomicTxs) == 0 {
+               return nil, errMissingAtomicTxs
+       }
+
+       for index, atx := range atomicTxs {
+               if err := atx.Sign(codec, nil); err != nil {
+                       return nil, fmt.Errorf("failed to initialize atomic tx at index %d: %w", index, err)
+               }
+       }
+       return atomicTxs, nil
+}
diff --git a/plugin/evm/config.go b/plugin/evm/config.go
index 9e3eea4c..4bb98ff8 100644
--- a/plugin/evm/config.go
+++ b/plugin/evm/config.go
@@ -7,28 +7,40 @@ import (
        "encoding/json"
        "time"
 
-       "github.com/ava-labs/coreth/eth"
+       "github.com/flare-foundation/coreth/eth"
        "github.com/spf13/cast"
 )
 
 const (
-       defaultEthApiEnabled               = true
-       defaultNetApiEnabled               = true
-       defaultWeb3ApiEnabled              = true
-       defaultPruningEnabled              = true
-       defaultSnapshotAsync               = true
-       defaultRpcGasCap                   = 2500000000 // 25000000 X 100
-       defaultRpcTxFeeCap                 = 100        // 100 AVAX
-       defaultApiMaxDuration              = 0          // Default to no maximum API call duration
-       defaultWsCpuRefillRate             = 0          // Default to no maximum WS CPU usage
-       defaultWsCpuMaxStored              = 0          // Default to no maximum WS CPU usage
-       defaultMaxBlocksPerRequest         = 0          // Default to no maximum on the number of blocks per getLogs request
-       defaultContinuousProfilerFrequency = 15 * time.Minute
-       defaultContinuousProfilerMaxFiles  = 5
-       defaultTxRegossipFrequency         = 1 * time.Minute
```

```diff
-       defaultTxRegossipMaxSize           = 15
+       defaultPruningEnabled                       = true
+       defaultSnapshotAsync                        = true
+       defaultRpcGasCap                            = 120_000_000 // Default to 120M Gas Limit
+       defaultRpcTxFeeCap                          = 100         // 100 AVAX
+       defaultMetricsEnabled                       = true
+       defaultMetricsExpensiveEnabled              = false
+       defaultApiMaxDuration                       = 0 // Default to no maximum API call duration
+       defaultWsCpuRefillRate                      = 0 // Default to no maximum WS CPU usage
+       defaultWsCpuMaxStored                       = 0 // Default to no maximum WS CPU usage
+       defaultMaxBlocksPerRequest                  = 0 // Default to no maximum on the number of blocks per getLogs request
+       defaultContinuousProfilerFrequency          = 15 * time.Minute
+       defaultContinuousProfilerMaxFiles           = 5
+       defaultTxRegossipFrequency                  = 1 * time.Minute
+       defaultTxRegossipMaxSize                    = 15
+       defaultOfflinePruningBloomFilterSize uint64 = 512 // Default size (MB) for the offline pruner to use
+       defaultLogLevel                             = "info"
+       defaultMaxOutboundActiveRequests            = 8
 )

+var defaultEnabledAPIs = []string{
+       "public-eth",
+       "public-eth-filter",
+       "net",
+       "web3",
+       "internal-public-eth",
+       "internal-public-blockchain",
+       "internal-public-transaction-pool",
+}
+
 type Duration struct {
        time.Duration
 }
@@ -36,9 +48,13 @@ type Duration struct {
 // Config ...
 type Config struct {
        // Coreth APIs
-       SnowmanAPIEnabled     bool `json:"snowman-api-enabled"`
-       CorethAdminAPIEnabled bool `json:"coreth-admin-api-enabled"`
-       NetAPIEnabled         bool `json:"net-api-enabled"`
+       SnowmanAPIEnabled     bool   `json:"snowman-api-enabled"`
+       CorethAdminAPIEnabled bool   `json:"coreth-admin-api-enabled"`
+       CorethAdminAPIDir     string `json:"coreth-admin-api-dir"`
+
+       // EnabledEthAPIs is a list of Ethereum services that should be enabled
+       // If none is specified, then we use the default list [defaultEnabledAPIs]
+       EnabledEthAPIs []string `json:"eth-apis"`

        // Continuous Profiler
        ContinuousProfilerDir       string   `json:"continuous-profiler-dir"`       // If set to non-empty string creates a continuous profiler
@@ -49,19 +65,16 @@ type Config struct {
        RPCGasCap   uint64  `json:"rpc-gas-cap"`
        RPCTxFeeCap float64 `json:"rpc-tx-fee-cap"`

-       // Eth APIs
-       EthAPIEnabled      bool `json:"eth-api-enabled"`
-       PersonalAPIEnabled bool `json:"personal-api-enabled"`
-       TxPoolAPIEnabled   bool `json:"tx-pool-api-enabled"`
-       DebugAPIEnabled    bool `json:"debug-api-enabled"`
-       Web3APIEnabled     bool `json:"web3-api-enabled"`
-
        // Eth Settings
        Preimages      bool `json:"preimages-enabled"`
        Pruning        bool `json:"pruning-enabled"`
        SnapshotAsync  bool `json:"snapshot-async"`
        SnapshotVerify bool `json:"snapshot-verification-enabled"`

+       // Metric Settings
+       MetricsEnabled          bool `json:"metrics-enabled"`
+       MetricsExpensiveEnabled bool `json:"metrics-expensive-enabled"`
+
        // API Settings
        LocalTxsEnabled         bool     `json:"local-txs-enabled"`
        APIMaxDuration          Duration `json:"api-max-duration"`
@@ -83,26 +96,19 @@ type Config struct {

        // Log level
        LogLevel string `json:"log-level"`
+
+       // Offline Pruning Settings
+       OfflinePruning                bool   `json:"offline-pruning-enabled"`
+       OfflinePruningBloomFilterSize uint64 `json:"offline-pruning-bloom-filter-size"`
+       OfflinePruningDataDirectory   string `json:"offline-pruning-data-directory"`
+
+       // VM2VM network
+       MaxOutboundActiveRequests int64 `json:"max-outbound-active-requests"`
 }

 // EthAPIs returns an array of strings representing the Eth APIs that should be enabled
 func (c Config) EthAPIs() []string {
-       ethAPIs := make([]string, 0)
-
-       if c.EthAPIEnabled {
-               ethAPIs = append(ethAPIs, "eth")
-       }
-       if c.PersonalAPIEnabled {
-               ethAPIs = append(ethAPIs, "personal")
-       }
-       if c.TxPoolAPIEnabled {
-               ethAPIs = append(ethAPIs, "txpool")
-       }
-       if c.DebugAPIEnabled {
-               ethAPIs = append(ethAPIs, "debug")
-       }
-
-       return ethAPIs
+       return c.EnabledEthAPIs
 }

 func (c Config) EthBackendSettings() eth.Settings {
@@ -110,11 +116,11 @@ func (c Config) EthBackendSettings() eth.Settings {
 }

 func (c *Config) SetDefaults() {
-       c.EthAPIEnabled = defaultEthApiEnabled
-       c.NetAPIEnabled = defaultNetApiEnabled
-       c.Web3APIEnabled = defaultWeb3ApiEnabled
+       c.EnabledEthAPIs = defaultEnabledAPIs
        c.RPCGasCap = defaultRpcGasCap
        c.RPCTxFeeCap = defaultRpcTxFeeCap
+       c.MetricsEnabled = defaultMetricsEnabled
+       c.MetricsExpensiveEnabled = defaultMetricsExpensiveEnabled
        c.APIMaxDuration.Duration = defaultApiMaxDuration
        c.WSCPURefillRate.Duration = defaultWsCpuRefillRate
        c.WSCPUMaxStored.Duration = defaultWsCpuMaxStored
@@ -125,6 +131,9 @@ func (c *Config) SetDefaults() {
        c.SnapshotAsync = defaultSnapshotAsync
        c.TxRegossipFrequency.Duration = defaultTxRegossipFrequency
        c.TxRegossipMaxSize = defaultTxRegossipMaxSize
+       c.OfflinePruningBloomFilterSize = defaultOfflinePruningBloomFilterSize
+       c.LogLevel = defaultLogLevel
+       c.MaxOutboundActiveRequests = defaultMaxOutboundActiveRequests
 }
 func (d *Duration) UnmarshalJSON(data []byte) (err error) {
diff --git a/plugin/evm/database.go b/plugin/evm/database.go
index c86ae5ee..53441e06 100644
--- a/plugin/evm/database.go
+++ b/plugin/evm/database.go
@@ -4,9 +4,9 @@
 package evm
 
 import (
-	"github.com/ava-labs/coreth/ethdb"
+	"github.com/flare-foundation/coreth/ethdb"
 
-	"github.com/ava-labs/avalanchego/database"
+	"github.com/flare-foundation/flare/database"
 )
 
 // Database implements ethdb.Database
diff --git a/plugin/evm/export_tx.go b/plugin/evm/export_tx.go
index fcc8c38f..7b241f7b 100644
--- a/plugin/evm/export_tx.go
+++ b/plugin/evm/export_tx.go
@@ -7,20 +7,21 @@ import (
 	"fmt"
 	"math/big"
 
-	"github.com/ava-labs/coreth/core/state"
-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/params"
 
-	"github.com/ava-labs/avalanchego/chains/atomic"
-	"github.com/ava-labs/avalanchego/database"
-	"github.com/ava-labs/avalanchego/ids"
-	"github.com/ava-labs/avalanchego/snow"
-	"github.com/ava-labs/avalanchego/utils/crypto"
-	"github.com/ava-labs/avalanchego/utils/math"
-	"github.com/ava-labs/avalanchego/utils/wrappers"
-	"github.com/ava-labs/avalanchego/vms/components/avax"
-	"github.com/ava-labs/avalanchego/vms/secp256k1fx"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/flare/chains/atomic"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/snow"
+	"github.com/flare-foundation/flare/utils/constants"
+	"github.com/flare-foundation/flare/utils/crypto"
+	"github.com/flare-foundation/flare/utils/math"
+	"github.com/flare-foundation/flare/utils/wrappers"
+	"github.com/flare-foundation/flare/vms/components/avax"
+	"github.com/flare-foundation/flare/vms/components/verify"
+	"github.com/flare-foundation/flare/vms/secp256k1fx"
 )
 
 // UnsignedExportTx is an unsigned ExportTx
@@ -40,94 +41,24 @@ type UnsignedExportTx struct {
 
 // InputUTXOs returns a set of all the hash(address:nonce) exporting funds.
 func (tx *UnsignedExportTx) InputUTXOs() ids.Set {
-	set := ids.NewSet(len(tx.Ins))
-	for _, in := range tx.Ins {
-		// Total populated bytes is 20 (Address) + 8 (Nonce), however, we allocate
-		// 32 bytes to make ids.ID casting easier.
-		var rawID [32]byte
-		packer := wrappers.Packer{Bytes: rawID[:]}
-		packer.PackLong(in.Nonce)
-		packer.PackBytes(in.Address.Bytes())
-		set.Add(ids.ID(rawID))
-	}
-	return set
+	return ids.Set{}
 }
 
 // Verify this transaction is well-formed
 func (tx *UnsignedExportTx) Verify(
-	xChainID ids.ID,
 	ctx *snow.Context,
 	rules params.Rules,
 ) error {
-	switch {
-	case tx == nil:
-		return errNilTx
-	case tx.DestinationChain != xChainID:
-		return errWrongChainID
-	case len(tx.ExportedOutputs) == 0:
-		return errNoExportOutputs
-	case tx.NetworkID != ctx.NetworkID:
-		return errWrongNetworkID
-	case ctx.ChainID != tx.BlockchainID:
-		return errWrongBlockchainID
-	}
-
-	for _, in := range tx.Ins {
-		if err := in.Verify(); err != nil {
-			return err
-		}
-	}
-
-	for _, out := range tx.ExportedOutputs {
-		if err := out.Verify(); err != nil {
-			return err
-		}
-	}
-	if !avax.IsSortedTransferableOutputs(tx.ExportedOutputs, Codec) {
-		return errOutputsNotSorted
-	}
-	if rules.IsApricotPhase1 && !IsSortedAndUniqueEVMInputs(tx.Ins) {
-		return errInputsNotSortedUnique
-	}
-
-	return nil
+	return errExportTxsDisabled
 }
 
-func (tx *UnsignedExportTx) GasUsed() (uint64, error) {
-	byteCost := calcBytesCost(len(tx.UnsignedBytes()))
-	numSigs := uint64(len(tx.Ins))
-	sigCost, err := math.Mul64(numSigs, secp256k1fx.CostPerSignature)
-	if err != nil {
-		return 0, err
-	}
-	return math.Add64(byteCost, sigCost)
+func (tx *UnsignedExportTx) GasUsed(fixedFee bool) (uint64, error) {
+	return 0, errExportTxsDisabled
 }
 
 // Amount of [assetID] burned by this transaction
 func (tx *UnsignedExportTx) Burned(assetID ids.ID) (uint64, error) {
-	var (
-		spent uint64
-		input uint64
-		err   error
-	)
-	for _, out := range tx.ExportedOutputs {
-		if out.AssetID() == assetID {
```

```
-                       spent, err = math.Add64(spent, out.Output().Amount())
-                       if err != nil {
-                               return 0, err
-                       }
-               }
-       }
-       for _, in := range tx.Ins {
-               if in.AssetID == assetID {
-                       input, err = math.Add64(input, in.Amount)
-                       if err != nil {
-                               return 0, err
-                       }
-               }
-       }
-
-       return math.Sub64(input, spent)
+       return 0, errExportTxsDisabled
 }
 
 // SemanticVerify this transaction is valid.
@@ -138,105 +69,12 @@ func (tx *UnsignedExportTx) SemanticVerify(
        baseFee *big.Int,
        rules params.Rules,
 ) error {
-       if err := tx.Verify(vm.ctx.XChainID, vm.ctx, rules); err != nil {
-               return err
-       }
-
-       // Check the transaction consumes and produces the right amounts
-       fc := avax.NewFlowChecker()
-       switch {
-       // Apply dynamic fees to export transactions as of Apricot Phase 3
-       case rules.IsApricotPhase3:
-               gasUsed, err := stx.GasUsed()
-               if err != nil {
-                       return err
-               }
-               txFee, err := calculateDynamicFee(gasUsed, baseFee)
-               if err != nil {
-                       return err
-               }
-               fc.Produce(vm.ctx.AVAXAssetID, txFee)
-
-       // Apply fees to export transactions before Apricot Phase 3
-       default:
-               fc.Produce(vm.ctx.AVAXAssetID, params.AvalancheAtomicTxFee)
-       }
-       for _, out := range tx.ExportedOutputs {
-               fc.Produce(out.AssetID(), out.Output().Amount())
-       }
-       for _, in := range tx.Ins {
-               fc.Consume(in.AssetID, in.Amount)
-       }
-
-       if err := fc.Verify(); err != nil {
-               return fmt.Errorf("export tx flow check failed due to: %w", err)
-       }
-
-       if len(tx.Ins) != len(stx.Creds) {
-               return fmt.Errorf("export tx contained mismatched number of inputs/credentials (%d vs. %d)", len(tx.Ins), len(stx.Creds))
-       }
-
-       for i, input := range tx.Ins {
-               cred, ok := stx.Creds[i].(*secp256k1fx.Credential)
-               if !ok {
-                       return fmt.Errorf("expected *secp256k1fx.Credential but got %T", cred)
-               }
-               if err := cred.Verify(); err != nil {
-                       return err
-               }
-
-               if len(cred.Sigs) != 1 {
-                       return fmt.Errorf("expected one signature for EVM Input Credential, but found: %d", len(cred.Sigs))
-               }
-               pubKeyIntf, err := vm.secpFactory.RecoverPublicKey(tx.UnsignedBytes(), cred.Sigs[0][:])
-               if err != nil {
-                       return err
-               }
-               pubKey, ok := pubKeyIntf.(*crypto.PublicKeySECP256K1R)
-               if !ok {
-                       // This should never happen
-                       return fmt.Errorf("expected *crypto.PublicKeySECP256K1R but got %T", pubKeyIntf)
-               }
-               if input.Address != PublicKeyToEthAddress(pubKey) {
-                       return errPublicKeySignatureMismatch
-               }
-       }
-
-       return nil
+       return errExportTxsDisabled
 }
 
 // Accept this transaction.
-func (tx *UnsignedExportTx) Accept(ctx *snow.Context, batch database.Batch) error {
-       txID := tx.ID()
-
-       elems := make([]*atomic.Element, len(tx.ExportedOutputs))
-       for i, out := range tx.ExportedOutputs {
-               utxo := &avax.UTXO{
-                       UTXOID: avax.UTXOID{
-                               TxID:        txID,
-                               OutputIndex: uint32(i),
-                       },
-                       Asset: avax.Asset{ID: out.AssetID()},
-                       Out:   out.Out,
-               }
-
-               utxoBytes, err := Codec.Marshal(codecVersion, utxo)
-               if err != nil {
-                       return err
-               }
-               utxoID := utxo.InputID()
-               elem := &atomic.Element{
-                       Key:   utxoID[:],
-                       Value: utxoBytes,
-               }
-               if out, ok := utxo.Out.(avax.Addressable); ok {
-                       elem.Traits = out.Addresses()
-               }
-
-               elems[i] = elem
-       }
-
-       return ctx.SharedMemory.Apply(map[ids.ID]*atomic.Requests{tx.DestinationChain: {PutRequests: elems}}, batch)
+func (tx *UnsignedExportTx) Accept() (ids.ID, *atomic.Requests, error) {
+       return ids.ID{}, nil, errExportTxsDisabled
 }
 
 // newExportTx returns a new ExportTx
@@ -248,122 +86,10 @@ func (vm *VM) newExportTx(
        baseFee *big.Int, // fee to use post-AP3
        keys []*crypto.PrivateKeySECP256K1R, // Pay the fee and provide the tokens
 ) (*Tx, error) {
-       if vm.ctx.XChainID != chainID {
```

```diff
-               return nil, errWrongChainID
-       }
-
-       outs := []*avax.TransferableOutput{{ // Exported to X-Chain
-               Asset: avax.Asset{ID: assetID},
-               Out: &secp256k1fx.TransferOutput{
-                       Amt: amount,
-                       OutputOwners: secp256k1fx.OutputOwners{
-                               Locktime:  0,
-                               Threshold: 1,
-                               Addrs:     []ids.ShortID{to},
-                       },
-               },
-       }}
-
-       var (
-               avaxNeeded           uint64 = 0
-               ins, avaxIns         []EVMInput
-               signers, avaxSigners [][]*crypto.PrivateKeySECP256K1R
-               err                  error
-       )
-
-       // consume non-AVAX
-       if assetID != vm.ctx.AVAXAssetID {
-               ins, signers, err = vm.GetSpendableFunds(keys, assetID, amount)
-               if err != nil {
-                       return nil, fmt.Errorf("couldn't generate tx inputs/signers: %w", err)
-               }
-       } else {
-               avaxNeeded = amount
-       }
-
-       rules := vm.currentRules()
-       switch {
-       case rules.IsApricotPhase3:
-               utx := &UnsignedExportTx{
-                       NetworkID:        vm.ctx.NetworkID,
-                       BlockchainID:     vm.ctx.ChainID,
-                       DestinationChain: chainID,
-                       Ins:              ins,
-                       ExportedOutputs:  outs,
-               }
-               tx := &Tx{UnsignedAtomicTx: utx}
-               if err := tx.Sign(vm.codec, nil); err != nil {
-                       return nil, err
-               }
-
-               var cost uint64
-               cost, err = tx.GasUsed()
-               if err != nil {
-                       return nil, err
-               }
-
-               avaxIns, avaxSigners, err = vm.GetSpendableAVAXWithFee(keys, avaxNeeded, cost, baseFee)
-       default:
-               var newAvaxNeeded uint64
-               newAvaxNeeded, err = math.Add64(avaxNeeded, params.AvalancheAtomicTxFee)
-               if err != nil {
-                       return nil, errOverflowExport
-               }
-               avaxIns, avaxSigners, err = vm.GetSpendableFunds(keys, vm.ctx.AVAXAssetID, newAvaxNeeded)
-       }
-       if err != nil {
-               return nil, fmt.Errorf("couldn't generate tx inputs/signers: %w", err)
-       }
-       ins = append(ins, avaxIns...)
-       signers = append(signers, avaxSigners...)
-
-       avax.SortTransferableOutputs(outs, vm.codec)
-       SortEVMInputsAndSigners(ins, signers)
-
-       // Create the transaction
-       utx := &UnsignedExportTx{
-               NetworkID:        vm.ctx.NetworkID,
-               BlockchainID:     vm.ctx.ChainID,
-               DestinationChain: chainID,
-               Ins:              ins,
-               ExportedOutputs:  outs,
-       }
-       tx := &Tx{UnsignedAtomicTx: utx}
-       if err := tx.Sign(vm.codec, signers); err != nil {
-               return nil, err
-       }
-       return tx, utx.Verify(vm.ctx.XChainID, vm.ctx, vm.currentRules())
+       return nil, errExportTxsDisabled
 }
 
 // EVMStateTransfer executes the state update from the atomic export transaction
 func (tx *UnsignedExportTx) EVMStateTransfer(ctx *snow.Context, state *state.StateDB) error {
-       addrs := map[[20]byte]uint64{}
-       for _, from := range tx.Ins {
-               if from.AssetID == ctx.AVAXAssetID {
-                       log.Debug("crosschain C->X", "addr", from.Address, "amount", from.Amount, "assetID", "AVAX")
-                       // We multiply the input amount by x2cRate to convert AVAX back to the appropriate
-                       // denomination before export.
-                       amount := new(big.Int).Mul(
-                               new(big.Int).SetUint64(from.Amount), x2cRate)
-                       if state.GetBalance(from.Address).Cmp(amount) < 0 {
-                               return errInsufficientFunds
-                       }
-                       state.SubBalance(from.Address, amount)
-               } else {
-                       log.Debug("crosschain C->X", "addr", from.Address, "amount", from.Amount, "assetID", from.AssetID)
-                       amount := new(big.Int).SetUint64(from.Amount)
-                       if state.GetBalanceMultiCoin(from.Address, common.Hash(from.AssetID)).Cmp(amount) < 0 {
-                               return errInsufficientFunds
-                       }
-                       state.SubBalanceMultiCoin(from.Address, common.Hash(from.AssetID), amount)
-               }
-               if state.GetNonce(from.Address) != from.Nonce {
-                       return errInvalidNonce
-               }
-               addrs[from.Address] = from.Nonce
-       }
-       for addr, nonce := range addrs {
-               state.SetNonce(addr, nonce+1)
-       }
-       return nil
+       return errExportTxsDisabled
 }
diff --git a/plugin/evm/export_tx_test.go b/plugin/evm/export_tx_test.go
deleted file mode 100644
index 2861af00..00000000
--- a/plugin/evm/export_tx_test.go
+++ /dev/null
@@ -1,1685 +0,0 @@
-// (c) 2019-2020, Ava Labs, Inc. All rights reserved.
-// See the file LICENSE for licensing terms.
-
-package evm
-
-import (
-       "bytes"
-       "math/big"
-       "testing"
-
-       "github.com/ava-labs/avalanchego/chains/atomic"
-       "github.com/ava-labs/avalanchego/ids"
-       engCommon "github.com/ava-labs/avalanchego/snow/engine/common"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/units"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ethereum/go-ethereum/common"
-)
-
-// createExportTxOptions adds funds to shared memory, imports them, and returns a list of export transactions
-// that attempt to send the funds to each of the test keys (list of length 3).
-func createExportTxOptions(t *testing.T, vm *VM, issuer chan engCommon.Message, sharedMemory *atomic.Memory) []*Tx {
-       // Add a UTXO to shared memory
-       utxo := &avax.UTXO{
-               UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-               Out: &secp256k1fx.TransferOutput{
-                       Amt: uint64(50000000),
-                       OutputOwners: secp256k1fx.OutputOwners{
-                               Threshold: 1,
-                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                       },
-               },
-       }
-       utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-       inputID := utxo.InputID()
-       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-               Key:   inputID[:],
-               Value: utxoBytes,
-               Traits: [][]byte{
-                       testKeys[0].PublicKey().Address().Bytes(),
-               },
-       }}}}); err != nil {
-               t.Fatal(err)
-       }
-
-       // Import the funds
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.SetPreference(blk.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       // Use the funds to create 3 conflicting export transactions sending the funds to each of the test addresses
-       exportTxs := make([]*Tx, 0, 3)
-       for _, addr := range testShortIDAddrs {
-               exportTx, err := vm.newExportTx(vm.ctx.AVAXAssetID, uint64(5000000), vm.ctx.XChainID, addr, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-               if err != nil {
-                       t.Fatal(err)
-               }
-               exportTxs = append(exportTxs, exportTx)
-       }
-
-       return exportTxs
-}
-
-func TestExportTxEVMStateTransfer(t *testing.T) {
-       key := testKeys[0]
-       addr := key.PublicKey().Address()
-       ethAddr := GetEthAddress(key)
-
-       avaxAmount := 50 * units.MilliAvax
-       avaxUTXOID := avax.UTXOID{
-               OutputIndex: 0,
-       }
-       avaxInputID := avaxUTXOID.InputID()
-
-       customAmount := uint64(100)
-       customAssetID := ids.ID{1, 2, 3, 4, 5, 7}
-       customUTXOID := avax.UTXOID{
-               OutputIndex: 1,
-       }
-       customInputID := customUTXOID.InputID()
-
-       customUTXO := &avax.UTXO{
-               UTXOID: customUTXOID,
-               Asset:  avax.Asset{ID: customAssetID},
-               Out: &secp256k1fx.TransferOutput{
-                       Amt: customAmount,
-                       OutputOwners: secp256k1fx.OutputOwners{
-                               Threshold: 1,
-                               Addrs:     []ids.ShortID{addr},
-                       },
-               },
-       }
-
-       tests := []struct {
-               name          string
-               tx            []EVMInput
-               avaxBalance   *big.Int
-               balances      map[ids.ID]*big.Int
-               expectedNonce uint64
-               shouldErr     bool
-       }{
-               {
-                       name:        "no transfers",
-                       tx:          nil,
-                       avaxBalance: big.NewInt(int64(avaxAmount) * x2cRateInt64),
-                       balances: map[ids.ID]*big.Int{
-                               customAssetID: big.NewInt(int64(customAmount)),
-                       },
-                       expectedNonce: 0,
-                       shouldErr:     false,
-		},
-		{
-			name: "spend half AVAX",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  avaxAmount / 2,
-					AssetID: testAvaxAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(int64(avaxAmount/2) * x2cRateInt64),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(int64(customAmount)),
-			},
-			expectedNonce: 1,
-			shouldErr:     false,
-		},
-		{
-			name: "spend all AVAX",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  avaxAmount,
-					AssetID: testAvaxAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(0),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(int64(customAmount)),
-			},
-			expectedNonce: 1,
-			shouldErr:     false,
-		},
-		{
-			name: "spend too much AVAX",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  avaxAmount + 1,
-					AssetID: testAvaxAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(0),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(int64(customAmount)),
-			},
-			expectedNonce: 1,
-			shouldErr:     true,
-		},
-		{
-			name: "spend half custom",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  customAmount / 2,
-					AssetID: customAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(int64(avaxAmount) * x2cRateInt64),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(int64(customAmount / 2)),
-			},
-			expectedNonce: 1,
-			shouldErr:     false,
-		},
-		{
-			name: "spend all custom",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  customAmount,
-					AssetID: customAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(int64(avaxAmount) * x2cRateInt64),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(0),
-			},
-			expectedNonce: 1,
-			shouldErr:     false,
-		},
-		{
-			name: "spend too much custom",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  customAmount + 1,
-					AssetID: customAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(int64(avaxAmount) * x2cRateInt64),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(0),
-			},
-			expectedNonce: 1,
-			shouldErr:     true,
-		},
-		{
-			name: "spend everything",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  customAmount,
-					AssetID: customAssetID,
-					Nonce:   0,
-				},
-				{
-					Address: ethAddr,
-					Amount:  avaxAmount,
-					AssetID: testAvaxAssetID,
-					Nonce:   0,
-				},
-			},
-			avaxBalance: big.NewInt(0),
-			balances: map[ids.ID]*big.Int{
-				customAssetID: big.NewInt(0),
-			},
-			expectedNonce: 1,
-			shouldErr:     false,
-		},
-		{
-			name: "spend everything wrong nonce",
-			tx: []EVMInput{
-				{
-					Address: ethAddr,
-					Amount:  customAmount,
-					AssetID: customAssetID,
-					Nonce:   1,
-                               },
-                               {
-                                       Address: ethAddr,
-                                       Amount:  avaxAmount,
-                                       AssetID: testAvaxAssetID,
-                                       Nonce:   1,
-                               },
-                       },
-                       avaxBalance: big.NewInt(0),
-                       balances: map[ids.ID]*big.Int{
-                               customAssetID: big.NewInt(0),
-                       },
-                       expectedNonce: 1,
-                       shouldErr:     true,
-               },
-               {
-                       name: "spend everything changing nonces",
-                       tx: []EVMInput{
-                               {
-                                       Address: ethAddr,
-                                       Amount:  customAmount,
-                                       AssetID: customAssetID,
-                                       Nonce:   0,
-                               },
-                               {
-                                       Address: ethAddr,
-                                       Amount:  avaxAmount,
-                                       AssetID: testAvaxAssetID,
-                                       Nonce:   1,
-                               },
-                       },
-                       avaxBalance: big.NewInt(0),
-                       balances: map[ids.ID]*big.Int{
-                               customAssetID: big.NewInt(0),
-                       },
-                       expectedNonce: 1,
-                       shouldErr:     true,
-               },
-       }
-       for _, test := range tests {
-               t.Run(test.name, func(t *testing.T) {
-                       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase0, "", "")
-                       defer func() {
-                               if err := vm.Shutdown(); err != nil {
-                                       t.Fatal(err)
-                               }
-                       }()
-
-                       avaxUTXO := &avax.UTXO{
-                               UTXOID: avaxUTXOID,
-                               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: avaxAmount,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{addr},
-                                       },
-                               },
-                       }
-
-                       avaxUTXOBytes, err := vm.codec.Marshal(codecVersion, avaxUTXO)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       customUTXOBytes, err := vm.codec.Marshal(codecVersion, customUTXO)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-                       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{
-                               {
-                                       Key:   avaxInputID[:],
-                                       Value: avaxUTXOBytes,
-                                       Traits: [][]byte{
-                                               addr.Bytes(),
-                                       },
-                               },
-                               {
-                                       Key:   customInputID[:],
-                                       Value: customUTXOBytes,
-                                       Traits: [][]byte{
-                                               addr.Bytes(),
-                                       },
-                               },
-                       }}}); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       tx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.issueTx(tx, true /*=local*/); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       <-issuer
-
-                       blk, err := vm.BuildBlock()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Verify(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.SetPreference(blk.ID()); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Accept(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       newTx := UnsignedExportTx{
-                               Ins: test.tx,
-                       }
-
-                       stateDB, err := vm.chain.CurrentState()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       err = newTx.EVMStateTransfer(vm.ctx, stateDB)
-                       if test.shouldErr {
-                               if err == nil {
-                                       t.Fatal("expected EVMStateTransfer to fail")
-                               }
-                               return
-                       }
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       avaxBalance := stateDB.GetBalance(ethAddr)
-                       if avaxBalance.Cmp(test.avaxBalance) != 0 {
-                               t.Fatalf("address balance %s equal %s not %s", addr.String(), avaxBalance, test.avaxBalance)
-                       }
-
-                       for assetID, expectedBalance := range test.balances {
-                               balance := stateDB.GetBalanceMultiCoin(ethAddr, common.Hash(assetID))
-                               if avaxBalance.Cmp(test.avaxBalance) != 0 {
-                                       t.Fatalf("%s address balance %s equal %s not %s", assetID, addr.String(), balance, expectedBalance)
-                               }
-                       }
-
-                       if stateDB.GetNonce(ethAddr) != test.expectedNonce {
-                               t.Fatalf("failed to set nonce to %d", test.expectedNonce)
-                       }
-               })
-       }
-}
-
-func TestExportTxSemanticVerify(t *testing.T) {
-       _, vm, _, _, _ := GenesisVM(t, true, genesisJSONApricotPhase0, "", "")
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       parent := vm.LastAcceptedBlockInternal().(*Block)
-
-       key := testKeys[0]
-       addr := key.PublicKey().Address()
-       ethAddr := testEthAddrs[0]
-
-       var (
-               avaxBalance           = 10 * units.Avax
-               custom0Balance uint64 = 100
-               custom0AssetID        = ids.ID{1, 2, 3, 4, 5}
-               custom1Balance uint64 = 1000
-               custom1AssetID        = ids.ID{1, 2, 3, 4, 5, 6}
-       )
-
-       validExportTx := &UnsignedExportTx{
-               NetworkID:        vm.ctx.NetworkID,
-               BlockchainID:     vm.ctx.ChainID,
-               DestinationChain: vm.ctx.XChainID,
-               Ins: []EVMInput{
-                       {
-                               Address: ethAddr,
-                               Amount:  avaxBalance,
-                               AssetID: vm.ctx.AVAXAssetID,
-                               Nonce:   0,
-                       },
-                       {
-                               Address: ethAddr,
-                               Amount:  custom0Balance,
-                               AssetID: custom0AssetID,
-                               Nonce:   0,
-                       },
-                       {
-                               Address: ethAddr,
-                               Amount:  custom1Balance,
-                               AssetID: custom1AssetID,
-                               Nonce:   0,
-                       },
-               },
-               ExportedOutputs: []*avax.TransferableOutput{
-                       {
-                               Asset: avax.Asset{ID: custom0AssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: custom0Balance,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{addr},
-                                       },
-                               },
-                       },
-               },
-       }
-
-       tests := []struct {
-               name      string
-               tx        *Tx
-               signers   [][]*crypto.PrivateKeySECP256K1R
-               baseFee   *big.Int
-               rules     params.Rules
-               shouldErr bool
-       }{
-               {
-                       name: "valid",
-                       tx:   &Tx{UnsignedAtomicTx: validExportTx},
-                       signers: [][]*crypto.PrivateKeySECP256K1R{
-                               {key},
-                               {key},
-                               {key},
-                       },
-                       baseFee:   initialBaseFee,
-                       rules:     apricotRulesPhase3,
-                       shouldErr: false,
-               },
-               {
-                       name: "wrong destination",
-                       tx: func() *Tx {
-                               validExportTx := *validExportTx
-                               validExportTx.DestinationChain = ids.GenerateTestID()
-                               return &Tx{UnsignedAtomicTx: &validExportTx}
-                       }(),
-                       signers: [][]*crypto.PrivateKeySECP256K1R{
-                               {key},
-                               {key},
-                               {key},
-                       },
-                       baseFee:   initialBaseFee,
-                       rules:     apricotRulesPhase3,
-                       shouldErr: true,
-               },
-               {
-                       name: "no outputs",
-                       tx: func() *Tx {
-                               validExportTx := *validExportTx
-                               validExportTx.ExportedOutputs = nil
-                               return &Tx{UnsignedAtomicTx: &validExportTx}
-                       }(),
-                       signers: [][]*crypto.PrivateKeySECP256K1R{
-                               {key},
-                               {key},
-                               {key},
-                       },
-                       baseFee:   initialBaseFee,
-                       rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "wrong networkID",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.NetworkID++
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "wrong chainID",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.BlockchainID = ids.GenerateTestID()
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "invalid input",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.Ins = append([]EVMInput{}, validExportTx.Ins...)
-				validExportTx.Ins[2].Amount = 0
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "invalid output",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.ExportedOutputs = []*avax.TransferableOutput{{
-					Asset: avax.Asset{ID: custom0AssetID},
-					Out: &secp256k1fx.TransferOutput{
-						Amt: custom0Balance,
-						OutputOwners: secp256k1fx.OutputOwners{
-							Threshold: 0,
-							Addrs:     []ids.ShortID{addr},
-						},
-					},
-				}}
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "unsorted outputs",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				exportOutputs := []*avax.TransferableOutput{
-					{
-						Asset: avax.Asset{ID: custom0AssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: custom0Balance/2 + 1,
-							OutputOwners: secp256k1fx.OutputOwners{
-								Threshold: 1,
-								Addrs:     []ids.ShortID{addr},
-							},
-						},
-					},
-					{
-						Asset: avax.Asset{ID: custom0AssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: custom0Balance/2 - 1,
-							OutputOwners: secp256k1fx.OutputOwners{
-								Threshold: 1,
-								Addrs:     []ids.ShortID{addr},
-							},
-						},
-					},
-				}
-				// Sort the outputs and then swap the ordering to ensure that they are ordered incorrectly
-				avax.SortTransferableOutputs(exportOutputs, Codec)
-				exportOutputs[0], exportOutputs[1] = exportOutputs[1], exportOutputs[0]
-				validExportTx.ExportedOutputs = exportOutputs
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "not unique inputs",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.Ins = append([]EVMInput{}, validExportTx.Ins...)
-				validExportTx.Ins[2] = validExportTx.Ins[1]
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "custom asset insufficient funds",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.ExportedOutputs = []*avax.TransferableOutput{
-					{
-						Asset: avax.Asset{ID: custom0AssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: custom0Balance + 1,
-							OutputOwners: secp256k1fx.OutputOwners{
-								Threshold: 1,
-								Addrs:     []ids.ShortID{addr},
-							},
-						},
-					},
-				}
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "avax insufficient funds",
-			tx: func() *Tx {
-				validExportTx := *validExportTx
-				validExportTx.ExportedOutputs = []*avax.TransferableOutput{
-					{
-						Asset: avax.Asset{ID: vm.ctx.AVAXAssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: avaxBalance, // after fees this should be too much
-							OutputOwners: secp256k1fx.OutputOwners{
-								Threshold: 1,
-								Addrs:     []ids.ShortID{addr},
-							},
-						},
-					},
-				}
-				return &Tx{UnsignedAtomicTx: &validExportTx}
-			}(),
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "too many signatures",
-			tx:   &Tx{UnsignedAtomicTx: validExportTx},
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "too few signatures",
-			tx:   &Tx{UnsignedAtomicTx: validExportTx},
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "too many signatures on credential",
-			tx:   &Tx{UnsignedAtomicTx: validExportTx},
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{key, testKeys[1]},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "too few signatures on credential",
-			tx:   &Tx{UnsignedAtomicTx: validExportTx},
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name: "wrong signature on credential",
-			tx:   &Tx{UnsignedAtomicTx: validExportTx},
-			signers: [][]*crypto.PrivateKeySECP256K1R{
-				{testKeys[1]},
-				{key},
-				{key},
-			},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-		{
-			name:      "no signatures",
-			tx:        &Tx{UnsignedAtomicTx: validExportTx},
-			signers:   [][]*crypto.PrivateKeySECP256K1R{},
-			baseFee:   initialBaseFee,
-			rules:     apricotRulesPhase3,
-			shouldErr: true,
-		},
-	}
-	for _, test := range tests {
-		if err := test.tx.Sign(vm.codec, test.signers); err != nil {
-			t.Fatal(err)
-		}
-
-		t.Run(test.name, func(t *testing.T) {
-			tx := test.tx
-			exportTx := tx.UnsignedAtomicTx
-
-			err := exportTx.SemanticVerify(vm, tx, parent, test.baseFee, test.rules)
-			if test.shouldErr && err == nil {
```

```
-                               t.Fatalf("should have errored but returned valid")
-                       }
-                       if !test.shouldErr && err != nil {
-                               t.Fatalf("shouldn't have errored but returned %s", err)
-                       }
-               })
-       }
-}
-
-func TestExportTxAccept(t *testing.T) {
-       _, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase0, "", "")
-
-       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       key := testKeys[0]
-       addr := key.PublicKey().Address()
-       ethAddr := testEthAddrs[0]
-
-       var (
-               avaxBalance           = 10 * units.Avax
-               custom0Balance uint64 = 100
-               custom0AssetID        = ids.ID{1, 2, 3, 4, 5}
-       )
-
-       exportTx := &UnsignedExportTx{
-               NetworkID:        vm.ctx.NetworkID,
-               BlockchainID:     vm.ctx.ChainID,
-               DestinationChain: vm.ctx.XChainID,
-               Ins: []EVMInput{
-                       {
-                               Address: ethAddr,
-                               Amount:  avaxBalance,
-                               AssetID: vm.ctx.AVAXAssetID,
-                               Nonce:   0,
-                       },
-                       {
-                               Address: ethAddr,
-                               Amount:  custom0Balance,
-                               AssetID: custom0AssetID,
-                               Nonce:   0,
-                       },
-               },
-               ExportedOutputs: []*avax.TransferableOutput{
-                       {
-                               Asset: avax.Asset{ID: vm.ctx.AVAXAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: avaxBalance,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{addr},
-                                       },
-                               },
-                       },
-                       {
-                               Asset: avax.Asset{ID: custom0AssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: custom0Balance,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{addr},
-                                       },
-                               },
-                       },
-               },
-       }
-
-       tx := &Tx{UnsignedAtomicTx: exportTx}
-
-       signers := [][]*crypto.PrivateKeySECP256K1R{
-               {key},
-               {key},
-               {key},
-       }
-
-       if err := tx.Sign(vm.codec, signers); err != nil {
-               t.Fatal(err)
-       }
-
-       commitBatch, err := vm.db.CommitBatch()
-       if err != nil {
-               t.Fatalf("Failed to create commit batch for VM due to %s", err)
-       }
-       if err := tx.Accept(vm.ctx, commitBatch); err != nil {
-               t.Fatalf("Failed to accept export transaction due to: %s", err)
-       }
-
-       indexedValues, _, _, err := xChainSharedMemory.Indexed(vm.ctx.ChainID, [][]byte{addr.Bytes()}, nil, nil, 3)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if len(indexedValues) != 2 {
-               t.Fatalf("expected 2 values but got %d", len(indexedValues))
-       }
-
-       avaxUTXOID := avax.UTXOID{
-               TxID:        tx.ID(),
-               OutputIndex: 0,
-       }
-       avaxInputID := avaxUTXOID.InputID()
-
-       customUTXOID := avax.UTXOID{
-               TxID:        tx.ID(),
-               OutputIndex: 1,
-       }
-       customInputID := customUTXOID.InputID()
-
-       fetchedValues, err := xChainSharedMemory.Get(vm.ctx.ChainID, [][]byte{
-               customInputID[:],
-               avaxInputID[:],
-       })
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if !bytes.Equal(fetchedValues[0], indexedValues[0]) {
-               t.Fatalf("inconsistent values returned fetched %x indexed %x", fetchedValues[0], indexedValues[0])
-       }
-       if !bytes.Equal(fetchedValues[1], indexedValues[1]) {
-               t.Fatalf("inconsistent values returned fetched %x indexed %x", fetchedValues[1], indexedValues[1])
-       }
-
-       customUTXOBytes, err := Codec.Marshal(codecVersion, &avax.UTXO{
-               UTXOID: customUTXOID,
-               Asset:  avax.Asset{ID: custom0AssetID},
-               Out:    exportTx.ExportedOutputs[1].Out,
-       })
-       if err != nil {
```

```
-               t.Fatal(err)
-       }
-
-       avaxUTXOBytes, err := Codec.Marshal(codecVersion, &avax.UTXO{
-               UTXOID: avaxUTXOID,
-               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-               Out:    exportTx.ExportedOutputs[0].Out,
-       })
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if !bytes.Equal(fetchedValues[0], customUTXOBytes) {
-               t.Fatalf("incorrect values returned expected %x got %x", customUTXOBytes, fetchedValues[0])
-       }
-       if !bytes.Equal(fetchedValues[1], avaxUTXOBytes) {
-               t.Fatalf("incorrect values returned expected %x got %x", avaxUTXOBytes, fetchedValues[1])
-       }
-}
-
-func TestExportTxVerifyNil(t *testing.T) {
-       var exportTx *UnsignedExportTx
-       if err := exportTx.Verify(testXChainID, NewContext(), apricotRulesPhase0); err == nil {
-               t.Fatal("Verify should have failed due to nil transaction")
-       }
-}
-
-func TestExportTxVerify(t *testing.T) {
-       var exportAmount uint64 = 10000000
-       exportTx := &UnsignedExportTx{
-               NetworkID:        testNetworkID,
-               BlockchainID:     testCChainID,
-               DestinationChain: testXChainID,
-               Ins: []EVMInput{
-                       {
-                               Address: testEthAddrs[0],
-                               Amount:  exportAmount,
-                               AssetID: testAvaxAssetID,
-                               Nonce:   0,
-                       },
-                       {
-                               Address: testEthAddrs[2],
-                               Amount:  exportAmount,
-                               AssetID: testAvaxAssetID,
-                               Nonce:   0,
-                       },
-               },
-               ExportedOutputs: []*avax.TransferableOutput{
-                       {
-                               Asset: avax.Asset{ID: testAvaxAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: exportAmount,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Locktime:  0,
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-                                       },
-                               },
-                       },
-                       {
-                               Asset: avax.Asset{ID: testAvaxAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: exportAmount,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Locktime:  0,
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{testShortIDAddrs[1]},
-                                       },
-                               },
-                       },
-               },
-       }
-
-       // Sort the inputs and outputs to ensure the transaction is canonical
-       avax.SortTransferableOutputs(exportTx.ExportedOutputs, Codec)
-       // Pass in a list of signers here with the appropriate length
-       // to avoid causing a nil-pointer error in the helper method
-       emptySigners := make([][]*crypto.PrivateKeySECP256K1R, 2)
-       SortEVMInputsAndSigners(exportTx.Ins, emptySigners)
-
-       ctx := NewContext()
-
-       // Test Valid Export Tx
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err != nil {
-               t.Fatalf("Failed to verify valid ExportTx: %s", err)
-       }
-
-       exportTx.NetworkID = testNetworkID + 1
-
-       // Test Incorrect Network ID Errors
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to incorrect network ID")
-       }
-
-       exportTx.NetworkID = testNetworkID
-       exportTx.BlockchainID = nonExistentID
-       // Test Incorrect Blockchain ID Errors
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to incorrect blockchain ID")
-       }
-
-       exportTx.BlockchainID = testCChainID
-       exportTx.DestinationChain = nonExistentID
-       // Test Incorrect Destination Chain ID Errors
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to incorrect destination chain")
-       }
-
-       exportTx.DestinationChain = testXChainID
-       exportedOuts := exportTx.ExportedOutputs
-       exportTx.ExportedOutputs = nil
-       evmInputs := exportTx.Ins
-       // Test No Exported Outputs Errors
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to no exported outputs")
-       }
-
-       exportTx.ExportedOutputs = []*avax.TransferableOutput{exportedOuts[1], exportedOuts[0]}
-       // Test Unsorted outputs Errors
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to no unsorted exported outputs")
-       }
-
-       exportTx.ExportedOutputs = []*avax.TransferableOutput{exportedOuts[0], nil}
-       // Test invalid exported output
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to invalid output")
-       }
-
-       exportTx.ExportedOutputs = []*avax.TransferableOutput{exportedOuts[0], exportedOuts[1]}
-       exportTx.Ins = []EVMInput{evmInputs[1], evmInputs[0]}
-       // Test unsorted EVM Inputs passes before AP1
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase0); err != nil {
-               t.Fatalf("ExportTx should have passed verification before AP1, but failed due to %s", err)
-       }
-       // Test unsorted EVM Inputs fails after AP1
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to unsorted EVM Inputs")
-       }
-       exportTx.Ins = []EVMInput{
-               {
-                       Address: testEthAddrs[0],
-                       Amount:  0,
-                       AssetID: testAvaxAssetID,
-                       Nonce:   0,
-               },
-       }
-       // Test ExportTx with invalid EVM Input amount 0 fails verification
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to 0 value amount")
-       }
-       exportTx.Ins = []EVMInput{evmInputs[0], evmInputs[0]}
-       // Test non-unique EVM Inputs passes verification before AP1
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase0); err != nil {
-               t.Fatalf("ExportTx with non-unique EVM Inputs should have passed verification prior to AP1, but failed due to %s", err)
-       }
-       exportTx.Ins = []EVMInput{evmInputs[0], evmInputs[0]}
-       // Test non-unique EVM Inputs fails verification after AP1
-       if err := exportTx.Verify(testXChainID, ctx, apricotRulesPhase1); err == nil {
-               t.Fatal("ExportTx should have failed verification due to non-unique inputs")
-       }
-}
-
-// Note: this is a brittle test to ensure that the gas cost of a transaction does
-// not change
-func TestExportTxGasCost(t *testing.T) {
-       avaxAssetID := ids.GenerateTestID()
-       chainID := ids.GenerateTestID()
-       xChainID := ids.GenerateTestID()
-       networkID := uint32(5)
-       exportAmount := uint64(5000000)
-
-       tests := map[string]struct {
-               UnsignedExportTx *UnsignedExportTx
-               Keys             [][]*crypto.PrivateKeySECP256K1R
-
-               BaseFee         *big.Int
-               ExpectedGasUsed uint64
-               ExpectedFee     uint64
-       }{
-               "simple export 1wei BaseFee": {
-                       UnsignedExportTx: &UnsignedExportTx{
-                               NetworkID:        networkID,
-                               BlockchainID:     chainID,
-                               DestinationChain: xChainID,
-                               Ins: []EVMInput{
-                                       {
-                                               Address: testEthAddrs[0],
-                                               Amount:  exportAmount,
-                                               AssetID: avaxAssetID,
-                                               Nonce:   0,
-                                       },
-                               },
-                               ExportedOutputs: []*avax.TransferableOutput{
-                                       {
-                                               Asset: avax.Asset{ID: avaxAssetID},
-                                               Out: &secp256k1fx.TransferOutput{
-                                                       Amt: exportAmount,
-                                                       OutputOwners: secp256k1fx.OutputOwners{
-                                                               Locktime:  0,
-                                                               Threshold: 1,
-                                                               Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-                                                       },
-                                               },
-                                       },
-                               },
-                       },
-                       Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}},
-                       ExpectedGasUsed: 1230,
-                       ExpectedFee:     1,
-                       BaseFee:         big.NewInt(1),
-               },
-               "simple export 25Gwei BaseFee": {
-                       UnsignedExportTx: &UnsignedExportTx{
-                               NetworkID:        networkID,
-                               BlockchainID:     chainID,
-                               DestinationChain: xChainID,
-                               Ins: []EVMInput{
-                                       {
-                                               Address: testEthAddrs[0],
-                                               Amount:  exportAmount,
-                                               AssetID: avaxAssetID,
-                                               Nonce:   0,
-                                       },
-                               },
-                               ExportedOutputs: []*avax.TransferableOutput{
-                                       {
-                                               Asset: avax.Asset{ID: avaxAssetID},
-                                               Out: &secp256k1fx.TransferOutput{
-                                                       Amt: exportAmount,
-                                                       OutputOwners: secp256k1fx.OutputOwners{
-                                                               Locktime:  0,
-                                                               Threshold: 1,
-                                                               Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-                                                       },
-                                               },
-                                       },
-                               },
-                       },
-                       Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}},
-                       ExpectedGasUsed: 1230,
-                       ExpectedFee:     30750,
-                       BaseFee:         big.NewInt(25 * params.GWei),
-               },
-               "simple export 225Gwei BaseFee": {
-                       UnsignedExportTx: &UnsignedExportTx{
-                               NetworkID:        networkID,
-                               BlockchainID:     chainID,
-                               DestinationChain: xChainID,
-                               Ins: []EVMInput{
-                                       {
-                                               Address: testEthAddrs[0],
-                                               Amount:  exportAmount,
-                                               AssetID: avaxAssetID,
-                                               Nonce:   0,
-                                       },
-                               },
-                               ExportedOutputs: []*avax.TransferableOutput{
-                                       {
-                                               Asset: avax.Asset{ID: avaxAssetID},
-                                               Out: &secp256k1fx.TransferOutput{
-                                                       Amt: exportAmount,
-                                                       OutputOwners: secp256k1fx.OutputOwners{
-                                                               Locktime:  0,
-                                                               Threshold: 1,
-                                                               Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-                                                       },
-                                               },
-					},
-				},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}},
-			ExpectedGasUsed: 1230,
-			ExpectedFee:     276750,
-			BaseFee:         big.NewInt(225 * params.GWei),
-		},
-		"complex export 25Gwei BaseFee": {
-			UnsignedExportTx: &UnsignedExportTx{
-				NetworkID:        networkID,
-				BlockchainID:     chainID,
-				DestinationChain: xChainID,
-				Ins: []EVMInput{
-					{
-						Address: testEthAddrs[0],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-					{
-						Address: testEthAddrs[1],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-					{
-						Address: testEthAddrs[2],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-				},
-				ExportedOutputs: []*avax.TransferableOutput{
-					{
-						Asset: avax.Asset{ID: avaxAssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: exportAmount * 3,
-							OutputOwners: secp256k1fx.OutputOwners{
-								Locktime:  0,
-								Threshold: 1,
-								Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-							},
-						},
-					},
-				},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0], testKeys[0], testKeys[0]}},
-			ExpectedGasUsed: 3366,
-			ExpectedFee:     84150,
-			BaseFee:         big.NewInt(25 * params.GWei),
-		},
-		"complex export 225Gwei BaseFee": {
-			UnsignedExportTx: &UnsignedExportTx{
-				NetworkID:        networkID,
-				BlockchainID:     chainID,
-				DestinationChain: xChainID,
-				Ins: []EVMInput{
-					{
-						Address: testEthAddrs[0],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-					{
-						Address: testEthAddrs[1],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-					{
-						Address: testEthAddrs[2],
-						Amount:  exportAmount,
-						AssetID: avaxAssetID,
-						Nonce:   0,
-					},
-				},
-				ExportedOutputs: []*avax.TransferableOutput{
-					{
-						Asset: avax.Asset{ID: avaxAssetID},
-						Out: &secp256k1fx.TransferOutput{
-							Amt: exportAmount * 3,
-							OutputOwners: secp256k1fx.OutputOwners{
-								Locktime:  0,
-								Threshold: 1,
-								Addrs:     []ids.ShortID{testShortIDAddrs[0]},
-							},
-						},
-					},
-				},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0], testKeys[0], testKeys[0]}},
-			ExpectedGasUsed: 3366,
-			ExpectedFee:     757350,
-			BaseFee:         big.NewInt(225 * params.GWei),
-		},
-	}
-
-	for name, test := range tests {
-		t.Run(name, func(t *testing.T) {
-			tx := &Tx{UnsignedAtomicTx: test.UnsignedExportTx}
-
-			// Sign with the correct key
-			if err := tx.Sign(Codec, test.Keys); err != nil {
-				t.Fatal(err)
-			}
-
-			gasUsed, err := tx.GasUsed()
-			if err != nil {
-				t.Fatal(err)
-			}
-			if gasUsed != test.ExpectedGasUsed {
-				t.Fatalf("Expected gasUsed to be %d, but found %d", test.ExpectedGasUsed, gasUsed)
-			}
-
-			fee, err := calculateDynamicFee(gasUsed, test.BaseFee)
-			if err != nil {
-				t.Fatal(err)
-			}
-			if fee != test.ExpectedFee {
-				t.Fatalf("Expected fee to be %d, but found %d", test.ExpectedFee, fee)
-			}
-		})
-	}
-}
-
-func TestNewExportTx(t *testing.T) {
-	tests := []struct {
-		name               string
-		genesis            string
-		rules              params.Rules
-		bal                uint64
-		expectedBurnedAVAX uint64
-	}{
```

```
-               {
-                       name:               "apricot phase 0",
-                       genesis:            genesisJSONApricotPhase0,
-                       rules:              apricotRulesPhase0,
-                       bal:                44000000,
-                       expectedBurnedAVAX: 1000000,
-               },
-               {
-                       name:               "apricot phase 1",
-                       genesis:            genesisJSONApricotPhase1,
-                       rules:              apricotRulesPhase1,
-                       bal:                44000000,
-                       expectedBurnedAVAX: 1000000,
-               },
-               {
-                       name:               "apricot phase 2",
-                       genesis:            genesisJSONApricotPhase2,
-                       rules:              apricotRulesPhase2,
-                       bal:                43000000,
-                       expectedBurnedAVAX: 1000000,
-               },
-               {
-                       name:               "apricot phase 3",
-                       genesis:            genesisJSONApricotPhase3,
-                       rules:              apricotRulesPhase3,
-                       bal:                44446500,
-                       expectedBurnedAVAX: 276750,
-               },
-               {
-                       name:               "apricot phase 4",
-                       genesis:            genesisJSONApricotPhase4,
-                       rules:              apricotRulesPhase4,
-                       bal:                44446500,
-                       expectedBurnedAVAX: 276750,
-               },
-       }
-       for _, test := range tests {
-               t.Run(test.name, func(t *testing.T) {
-                       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, test.genesis, "", "")
-
-                       defer func() {
-                               if err := vm.Shutdown(); err != nil {
-                                       t.Fatal(err)
-                               }
-                       }()
-
-                       parent := vm.LastAcceptedBlockInternal().(*Block)
-                       importAmount := uint64(50000000)
-                       utxoID := avax.UTXOID{TxID: ids.GenerateTestID()}
-
-                       utxo := &avax.UTXO{
-                               UTXOID: utxoID,
-                               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: importAmount,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                                       },
-                               },
-                       }
-                       utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-                       inputID := utxo.InputID()
-                       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-                               Key:   inputID[:],
-                               Value: utxoBytes,
-                               Traits: [][]byte{
-                                       testKeys[0].PublicKey().Address().Bytes(),
-                               },
-                       }}}}); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       tx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.issueTx(tx, true /*=local*/); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       <-issuer
-
-                       blk, err := vm.BuildBlock()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Verify(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.SetPreference(blk.ID()); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Accept(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       parent = vm.LastAcceptedBlockInternal().(*Block)
-                       exportAmount := uint64(5000000)
-
-                       tx, err = vm.newExportTx(vm.ctx.AVAXAssetID, exportAmount, vm.ctx.XChainID, testShortIDAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       exportTx := tx.UnsignedAtomicTx
-
-                       if err := exportTx.SemanticVerify(vm, tx, parent, parent.ethBlock.BaseFee(), test.rules); err != nil {
-                               t.Fatal("newExportTx created an invalid transaction", err)
-                       }
-
-                       burnedAVAX, err := exportTx.Burned(vm.ctx.AVAXAssetID)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       if burnedAVAX != test.expectedBurnedAVAX {
-                               t.Fatalf("burned wrong amount of AVAX - expected %d burned %d", test.expectedBurnedAVAX, burnedAVAX)
-                       }
-
-                       commitBatch, err := vm.db.CommitBatch()
-                       if err != nil {
-                               t.Fatalf("Failed to create commit batch for VM due to %s", err)
-                       }
-                       if err := exportTx.Accept(vm.ctx, commitBatch); err != nil {
-                               t.Fatalf("Failed to accept export transaction due to: %s", err)
-                       }
-
-                       sdb, err := vm.chain.CurrentState()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       err = exportTx.EVMStateTransfer(vm.ctx, sdb)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       addr := GetEthAddress(testKeys[0])
-                       if sdb.GetBalance(addr).Cmp(new(big.Int).SetUint64(test.bal*units.Avax)) != 0 {
-                               t.Fatalf("address balance %s equal %s not %s", addr.String(), sdb.GetBalance(addr), new(big.Int).SetUint64(test.bal*units.Avax))
-                       }
-               })
-       }
-}
-
-func TestNewExportTxMulticoin(t *testing.T) {
-       tests := []struct {
-               name    string
-               genesis string
-               rules   params.Rules
-               bal     uint64
-               balmc   uint64
-       }{
-               {
-                       name:    "apricot phase 0",
-                       genesis: genesisJSONApricotPhase0,
-                       rules:   apricotRulesPhase0,
-                       bal:     49000000,
-                       balmc:   25000000,
-               },
-               {
-                       name:    "apricot phase 1",
-                       genesis: genesisJSONApricotPhase1,
-                       rules:   apricotRulesPhase1,
-                       bal:     49000000,
-                       balmc:   25000000,
-               },
-               {
-                       name:    "apricot phase 2",
-                       genesis: genesisJSONApricotPhase2,
-                       rules:   apricotRulesPhase2,
-                       bal:     48000000,
-                       balmc:   25000000,
-               },
-               {
-                       name:    "apricot phase 3",
-                       genesis: genesisJSONApricotPhase3,
-                       rules:   apricotRulesPhase3,
-                       bal:     48947900,
-                       balmc:   25000000,
-               },
-       }
-       for _, test := range tests {
-               t.Run(test.name, func(t *testing.T) {
-                       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, test.genesis, "", "")
-
-                       defer func() {
-                               if err := vm.Shutdown(); err != nil {
-                                       t.Fatal(err)
-                               }
-                       }()
-
-                       parent := vm.LastAcceptedBlockInternal().(*Block)
-                       importAmount := uint64(50000000)
-                       utxoID := avax.UTXOID{TxID: ids.GenerateTestID()}
-
-                       utxo := &avax.UTXO{
-                               UTXOID: utxoID,
-                               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: importAmount,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                                       },
-                               },
-                       }
-                       utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       inputID := utxo.InputID()
-
-                       tid := ids.GenerateTestID()
-                       importAmount2 := uint64(30000000)
-                       utxoID2 := avax.UTXOID{TxID: ids.GenerateTestID()}
-                       utxo2 := &avax.UTXO{
-                               UTXOID: utxoID2,
-                               Asset:  avax.Asset{ID: tid},
-                               Out: &secp256k1fx.TransferOutput{
-                                       Amt: importAmount2,
-                                       OutputOwners: secp256k1fx.OutputOwners{
-                                               Threshold: 1,
-                                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                                       },
-                               },
-                       }
-                       utxoBytes2, err := vm.codec.Marshal(codecVersion, utxo2)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-                       inputID2 := utxo2.InputID()
-                       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{
-                               {
-                                       Key:   inputID[:],
-                                       Value: utxoBytes,
-                                       Traits: [][]byte{
-                                               testKeys[0].PublicKey().Address().Bytes(),
-                                       },
-                               },
-                               {
-                                       Key:   inputID2[:],
-                                       Value: utxoBytes2,
-                                       Traits: [][]byte{
-                                               testKeys[0].PublicKey().Address().Bytes(),
-                                       },
-                               },
-                       }}}); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       tx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.issueTx(tx, false); err != nil {
```

```
-                               t.Fatal(err)
-                       }
-
-                       <-issuer
-
-                       blk, err := vm.BuildBlock()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Verify(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := vm.SetPreference(blk.ID()); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       if err := blk.Accept(); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       parent = vm.LastAcceptedBlockInternal().(*Block)
-                       exportAmount := uint64(5000000)
-
-                       testKeys0Addr := GetEthAddress(testKeys[0])
-                       exportId, err := ids.ToShortID(testKeys0Addr[:])
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       tx, err = vm.newExportTx(tid, exportAmount, vm.ctx.XChainID, exportId, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       exportTx := tx.UnsignedAtomicTx
-
-                       if err := exportTx.SemanticVerify(vm, tx, parent, parent.ethBlock.BaseFee(), test.rules); err != nil {
-                               t.Fatal("newExportTx created an invalid transaction", err)
-                       }
-
-                       commitBatch, err := vm.db.CommitBatch()
-                       if err != nil {
-                               t.Fatalf("Failed to create commit batch for VM due to %s", err)
-                       }
-                       if err := exportTx.Accept(vm.ctx, commitBatch); err != nil {
-                               t.Fatalf("Failed to accept export transaction due to: %s", err)
-                       }
-
-                       stdb, err := vm.chain.CurrentState()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       err = exportTx.EVMStateTransfer(vm.ctx, stdb)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-
-                       addr := GetEthAddress(testKeys[0])
-                       if stdb.GetBalance(addr).Cmp(new(big.Int).SetUint64(test.bal*units.Avax)) != 0 {
-                               t.Fatalf("address balance %s equal %s not %s", addr.String(), stdb.GetBalance(addr), new(big.Int).SetUint64(test.bal*units.Avax))
-                       }
-                       if stdb.GetBalanceMultiCoin(addr, common.BytesToHash(tid[:])).Cmp(new(big.Int).SetUint64(test.balmc)) != 0 {
-                               t.Fatalf("address balance multicoin %s equal %s not %s", addr.String(), stdb.GetBalanceMultiCoin(addr, common.BytesToHash(tid[:])), new(big.Int).SetUint64(test.balmc))
-                       }
-               })
-       }
-}
diff --git a/plugin/evm/ext_data_hashes.go b/plugin/evm/ext_data_hashes.go
index 7648c3bc..1c231aaa 100644
--- a/plugin/evm/ext_data_hashes.go
+++ b/plugin/evm/ext_data_hashes.go
@@ -8,22 +8,22 @@ import (
 )
 
 var (
-       //go:embed fuji_ext_data_hashes.json
-       rawFujiExtDataHashes []byte
-       fujiExtDataHashes    map[common.Hash]common.Hash
+       //go:embed songbird_ext_data_hashes.json
+       rawSongbirdExtDataHashes []byte
+       songbirdExtDataHashes    map[common.Hash]common.Hash
 
-       //go:embed mainnet_ext_data_hashes.json
-       rawMainnetExtDataHashes []byte
-       mainnetExtDataHashes    map[common.Hash]common.Hash
+       //go:embed flare_ext_data_hashes.json
+       rawFlareExtDataHashes []byte
+       flareExtDataHashes    map[common.Hash]common.Hash
 )
 
 func init() {
-       if err := json.Unmarshal(rawFujiExtDataHashes, &fujiExtDataHashes); err != nil {
+       if err := json.Unmarshal(rawSongbirdExtDataHashes, &songbirdExtDataHashes); err != nil {
                panic(err)
        }
-       rawFujiExtDataHashes = nil
-       if err := json.Unmarshal(rawMainnetExtDataHashes, &mainnetExtDataHashes); err != nil {
+       rawSongbirdExtDataHashes = nil
+       if err := json.Unmarshal(rawFlareExtDataHashes, &flareExtDataHashes); err != nil {
                panic(err)
        }
-       rawMainnetExtDataHashes = nil
+       rawFlareExtDataHashes = nil
 }
diff --git a/plugin/evm/factory.go b/plugin/evm/factory.go
index 34ecc893..79388fc9 100644
--- a/plugin/evm/factory.go
+++ b/plugin/evm/factory.go
@@ -4,9 +4,9 @@
 package evm
 
 import (
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/vms"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/vms"
 )
 
 var (
diff --git a/plugin/evm/mainnet_ext_data_hashes.json b/plugin/evm/flare_ext_data_hashes.json
similarity index 100%
rename from plugin/evm/mainnet_ext_data_hashes.json
rename to plugin/evm/flare_ext_data_hashes.json
diff --git a/plugin/evm/formatting.go b/plugin/evm/formatting.go
index d2194ea3..cf6c6682 100644
--- a/plugin/evm/formatting.go
+++ b/plugin/evm/formatting.go
@@ -6,12 +6,12 @@ package evm
 import (
        "fmt"
```

```
-	"github.com/ava-labs/avalanchego/ids"
-	"github.com/ava-labs/avalanchego/utils/constants"
-	"github.com/ava-labs/avalanchego/utils/crypto"
-	"github.com/ava-labs/avalanchego/utils/formatting"
 	"github.com/ethereum/go-ethereum/common"
 	ethcrypto "github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/utils/constants"
+	"github.com/flare-foundation/flare/utils/crypto"
+	"github.com/flare-foundation/flare/utils/formatting"
 )
 
 // ParseLocalAddress takes in an address for this chain and produces the ID
diff --git a/plugin/evm/gasprice_update.go b/plugin/evm/gasprice_update.go
index 71a4ea1a..177be08a 100644
--- a/plugin/evm/gasprice_update.go
+++ b/plugin/evm/gasprice_update.go
@@ -8,7 +8,7 @@ import (
 	"sync"
 	"time"
 
-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/params"
 )
 
 type gasPriceUpdater struct {
diff --git a/plugin/evm/gasprice_update_test.go b/plugin/evm/gasprice_update_test.go
index 24d8337f..5a720fb0 100644
--- a/plugin/evm/gasprice_update_test.go
+++ b/plugin/evm/gasprice_update_test.go
@@ -9,7 +9,7 @@ import (
 	"testing"
 	"time"
 
-	"github.com/ava-labs/coreth/params"
+	"github.com/flare-foundation/coreth/params"
 )
 
 type mockGasPriceSetter struct {
diff --git a/plugin/evm/gossiper.go b/plugin/evm/gossiper.go
new file mode 100644
index 00000000..2a73873c
--- /dev/null
+++ b/plugin/evm/gossiper.go
@@ -0,0 +1,516 @@
+// (c) 2019-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+	"container/heap"
+	"math/big"
+	"sync"
+	"time"
+
+	"github.com/flare-foundation/flare/codec"
+
+	"github.com/flare-foundation/coreth/peer"
+
+	"github.com/flare-foundation/flare/cache"
+	"github.com/flare-foundation/flare/ids"
+	"github.com/flare-foundation/flare/snow"
+	"github.com/flare-foundation/flare/utils/wrappers"
+
+	"github.com/ethereum/go-ethereum/common"
+	"github.com/ethereum/go-ethereum/log"
+	"github.com/ethereum/go-ethereum/rlp"
+
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/state"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/plugin/evm/message"
+)
+
+const (
+	// We allow [recentCacheSize] to be fairly large because we only store hashes
+	// in the cache, not entire transactions.
+	recentCacheSize = 512
+
+	// [ethTxsGossipInterval] is how often we attempt to gossip newly seen
+	// transactions to other nodes.
+	ethTxsGossipInterval = 500 * time.Millisecond
+)
+
+// Gossiper handles outgoing gossip of transactions
+type Gossiper interface {
+	// GossipAtomicTxs sends AppGossip message containing the given [txs]
+	GossipAtomicTxs(txs []*Tx) error
+	// GossipEthTxs sends AppGossip message containing the given [txs]
+	GossipEthTxs(txs []*types.Transaction) error
+}
+
+// pushGossiper is used to gossip transactions to the network
+type pushGossiper struct {
+	ctx                  *snow.Context
+	gossipActivationTime time.Time
+	config               Config
+
+	client        peer.Client
+	blockchain    *core.BlockChain
+	txPool        *core.TxPool
+	atomicMempool *Mempool
+
+	// We attempt to batch transactions we need to gossip to avoid runaway
+	// amplification of mempol chatter.
+	ethTxsToGossipChan chan []*types.Transaction
+	ethTxsToGossip     map[common.Hash]*types.Transaction
+	lastGossiped       time.Time
+	shutdownChan       chan struct{}
+	shutdownWg         *sync.WaitGroup
+
+	// [recentAtomicTxs] and [recentEthTxs] prevent us from over-gossiping the
+	// same transaction in a short period of time.
+	recentAtomicTxs *cache.LRU
+	recentEthTxs    *cache.LRU
+
+	codec codec.Manager
+}
+
+// newPushGossiper constructs and returns a pushGossiper
+// assumes vm.chainConfig.ApricotPhase4BlockTimestamp is set
+func (vm *VM) newPushGossiper() Gossiper {
+	net := &pushGossiper{
+		ctx:                  vm.ctx,
+		gossipActivationTime: time.Unix(vm.chainConfig.ApricotPhase4BlockTimestamp.Int64(), 0),
+		config:               vm.config,
+		client:               vm.client,
+		blockchain:           vm.chain.BlockChain(),
+		txPool:               vm.chain.GetTxPool(),
+		atomicMempool:        vm.mempool,
+		ethTxsToGossipChan:   make(chan []*types.Transaction),
+		ethTxsToGossip:       make(map[common.Hash]*types.Transaction),
+               shutdownChan:         vm.shutdownChan,
+               shutdownWg:           &vm.shutdownWg,
+               recentAtomicTxs:      &cache.LRU{Size: recentCacheSize},
+               recentEthTxs:         &cache.LRU{Size: recentCacheSize},
+               codec:                vm.networkCodec,
+       }
+       net.awaitEthTxGossip()
+       return net
+}
+
+// queueExecutableTxs attempts to select up to [maxTxs] from the tx pool for
+// regossiping.
+//
+// We assume that [txs] contains an array of nonce-ordered transactions for a given
+// account. This array of transactions can have gaps and start at a nonce lower
+// than the current state of an account.
+func (n *pushGossiper) queueExecutableTxs(state *state.StateDB, baseFee *big.Int, txs map[common.Address]types.Transactions, maxTxs int) types.Transactions {
+       // Setup heap for transactions
+       heads := make(types.TxByPriceAndTime, 0, len(txs))
+       for addr, accountTxs := range txs {
+               // Short-circuit here to avoid performing an unnecessary state lookup
+               if len(accountTxs) == 0 {
+                       continue
+               }
+
+               // Ensure any transactions regossiped are immediately executable
+               var (
+                       currentNonce = state.GetNonce(addr)
+                       tx           *types.Transaction
+               )
+               for _, accountTx := range accountTxs {
+                       // The tx pool may be out of sync with current state, so we iterate
+                       // through the account transactions until we get to one that is
+                       // executable.
+                       if accountTx.Nonce() == currentNonce {
+                               tx = accountTx
+                               break
+                       }
+                       // There may be gaps in the tx pool and we could jump past the nonce we'd
+                       // like to execute.
+                       if accountTx.Nonce() > currentNonce {
+                               break
+                       }
+               }
+               if tx == nil {
+                       continue
+               }
+
+               // Don't try to regossip a transaction too frequently
+               if time.Since(tx.FirstSeen()) < n.config.TxRegossipFrequency.Duration {
+                       continue
+               }
+
+               // Ensure the fee the transaction pays is valid at tip
+               wrapped, err := types.NewTxWithMinerFee(tx, baseFee)
+               if err != nil {
+                       log.Debug(
+                               "not queuing tx for regossip",
+                               "tx", tx.Hash(),
+                               "err", err,
+                       )
+                       continue
+               }
+
+               heads = append(heads, wrapped)
+       }
+       heap.Init(&heads)
+
+       // Add up to [maxTxs] transactions to be gossiped
+       queued := make([]*types.Transaction, 0, maxTxs)
+       for len(heads) > 0 && len(queued) < maxTxs {
+               tx := heads[0].Tx
+               queued = append(queued, tx)
+               heap.Pop(&heads)
+       }
+
+       return queued
+}
+
+// queueRegossipTxs finds the best transactions in the mempool and adds up to
+// [TxRegossipMaxSize] of them to [ethTxsToGossip].
+func (n *pushGossiper) queueRegossipTxs() types.Transactions {
+       // Fetch all pending transactions
+       pending := n.txPool.Pending(true)
+
+       // Split the pending transactions into locals and remotes
+       localTxs := make(map[common.Address]types.Transactions)
+       remoteTxs := pending
+       for _, account := range n.txPool.Locals() {
+               if txs := remoteTxs[account]; len(txs) > 0 {
+                       delete(remoteTxs, account)
+                       localTxs[account] = txs
+               }
+       }
+
+       // Add best transactions to be gossiped (preferring local txs)
+       tip := n.blockchain.CurrentBlock()
+       state, err := n.blockchain.StateAt(tip.Root())
+       if err != nil || state == nil {
+               log.Debug(
+                       "could not get state at tip",
+                       "tip", tip.Hash(),
+                       "err", err,
+               )
+               return nil
+       }
+       localQueued := n.queueExecutableTxs(state, tip.BaseFee(), localTxs, n.config.TxRegossipMaxSize)
+       localCount := len(localQueued)
+       if localCount >= n.config.TxRegossipMaxSize {
+               return localQueued
+       }
+       remoteQueued := n.queueExecutableTxs(state, tip.BaseFee(), remoteTxs, n.config.TxRegossipMaxSize-localCount)
+       return append(localQueued, remoteQueued...)
+}
+
+// awaitEthTxGossip periodically gossips transactions that have been queued for
+// gossip at least once every [ethTxsGossipInterval].
+func (n *pushGossiper) awaitEthTxGossip() {
+       n.shutdownWg.Add(1)
+       go n.ctx.Log.RecoverAndPanic(func() {
+               defer n.shutdownWg.Done()
+
+               var (
+                       gossipTicker   = time.NewTicker(ethTxsGossipInterval)
+                       regossipTicker = time.NewTicker(n.config.TxRegossipFrequency.Duration)
+               )
+
+               for {
+                       select {
+                       case <-gossipTicker.C:
+                               if attempted, err := n.gossipEthTxs(false); err != nil {
+                                       log.Warn(
+                                               "failed to send eth transactions",
+                                               "len(txs)", attempted,
+                                               "err", err,
+                                       )
+                               }
+                       case <-regossipTicker.C:
+                               for _, tx := range n.queueRegossipTxs() {
+                                       n.ethTxsToGossip[tx.Hash()] = tx
+                               }
+                               if attempted, err := n.gossipEthTxs(true); err != nil {
+                                       log.Warn(
+                                               "failed to send eth transactions",
+                                               "len(txs)", attempted,
+                                               "err", err,
+                                       )
+                               }
+                       case txs := <-n.ethTxsToGossipChan:
+                               for _, tx := range txs {
+                                       n.ethTxsToGossip[tx.Hash()] = tx
+                               }
+                               if attempted, err := n.gossipEthTxs(false); err != nil {
+                                       log.Warn(
+                                               "failed to send eth transactions",
+                                               "len(txs)", attempted,
+                                               "err", err,
+                                       )
+                               }
+                       case <-n.shutdownChan:
+                               return
+                       }
+               }
+       })
+}
+
+func (n *pushGossiper) GossipAtomicTxs(txs []*Tx) error {
+       if time.Now().Before(n.gossipActivationTime) {
+               log.Trace(
+                       "not gossiping atomic tx before the gossiping activation time",
+                       "txs", txs,
+               )
+               return nil
+       }
+
+       errs := wrappers.Errs{}
+       for _, tx := range txs {
+               errs.Add(n.gossipAtomicTx(tx))
+       }
+       return errs.Err
+}
+
+func (n *pushGossiper) gossipAtomicTx(tx *Tx) error {
+       txID := tx.ID()
+       // Don't gossip transaction if it has been recently gossiped.
+       if _, has := n.recentAtomicTxs.Get(txID); has {
+               return nil
+       }
+       // If the transaction is not pending according to the mempool
+       // then there is no need to gossip it further.
+       if _, pending := n.atomicMempool.GetPendingTx(txID); !pending {
+               return nil
+       }
+       n.recentAtomicTxs.Put(txID, nil)
+
+       msg := message.AtomicTx{
+               Tx: tx.Bytes(),
+       }
+       msgBytes, err := message.BuildMessage(n.codec, &msg)
+       if err != nil {
+               return err
+       }
+
+       log.Trace(
+               "gossiping atomic tx",
+               "txID", txID,
+       )
+       return n.client.Gossip(msgBytes)
+}
+
+func (n *pushGossiper) sendEthTxs(txs []*types.Transaction) error {
+       if len(txs) == 0 {
+               return nil
+       }
+
+       txBytes, err := rlp.EncodeToBytes(txs)
+       if err != nil {
+               return err
+       }
+       msg := message.EthTxs{
+               Txs: txBytes,
+       }
+       msgBytes, err := message.BuildMessage(n.codec, &msg)
+       if err != nil {
+               return err
+       }
+
+       log.Trace(
+               "gossiping eth txs",
+               "len(txs)", len(txs),
+               "size(txs)", len(msg.Txs),
+       )
+       return n.client.Gossip(msgBytes)
+}
+
+func (n *pushGossiper) gossipEthTxs(force bool) (int, error) {
+       if (!force && time.Since(n.lastGossiped) < ethTxsGossipInterval) || len(n.ethTxsToGossip) == 0 {
+               return 0, nil
+       }
+       n.lastGossiped = time.Now()
+       txs := make([]*types.Transaction, 0, len(n.ethTxsToGossip))
+       for _, tx := range n.ethTxsToGossip {
+               txs = append(txs, tx)
+               delete(n.ethTxsToGossip, tx.Hash())
+       }
+
+       selectedTxs := make([]*types.Transaction, 0)
+       for _, tx := range txs {
+               txHash := tx.Hash()
+               txStatus := n.txPool.Status([]common.Hash{txHash})[0]
+               if txStatus != core.TxStatusPending {
+                       continue
+               }
+
+               if n.config.RemoteTxGossipOnlyEnabled && n.txPool.HasLocal(txHash) {
+                       continue
+               }
+
+               // We check [force] outside of the if statement to avoid an unnecessary
+               // cache lookup.
+               if !force {
+                       if _, has := n.recentEthTxs.Get(txHash); has {
+                               continue
+                       }
+               }
+               n.recentEthTxs.Put(txHash, nil)
+
+               selectedTxs = append(selectedTxs, tx)
+       }
+
+       if len(selectedTxs) == 0 {
+               return 0, nil
+       }
+
+       // Attempt to gossip [selectedTxs]
+       msgTxs := make([]*types.Transaction, 0)
+       msgTxsSize := common.StorageSize(0)
+       for _, tx := range selectedTxs {
+               size := tx.Size()
+               if msgTxsSize+size > message.EthMsgSoftCapSize {
+                       if err := n.sendEthTxs(msgTxs); err != nil {
+                               return len(selectedTxs), err
+                       }
+                       msgTxs = msgTxs[:0]
+                       msgTxsSize = 0
+               }
+               msgTxs = append(msgTxs, tx)
+               msgTxsSize += size
+       }
+
+       // Send any remaining [msgTxs]
+       return len(selectedTxs), n.sendEthTxs(msgTxs)
+}
+
+// GossipEthTxs enqueues the provided [txs] for gossiping. At some point, the
+// [pushGossiper] will attempt to gossip the provided txs to other nodes
+// (usually right away if not under load).
+//
+// NOTE: We never return a non-nil error from this function but retain the
+// option to do so in case it becomes useful.
+func (n *pushGossiper) GossipEthTxs(txs []*types.Transaction) error {
+       if time.Now().Before(n.gossipActivationTime) {
+               log.Trace(
+                       "not gossiping eth txs before the gossiping activation time",
+                       "len(txs)", len(txs),
+               )
+               return nil
+       }
+
+       select {
+       case n.ethTxsToGossipChan <- txs:
+       case <-n.shutdownChan:
+       }
+       return nil
+}
+
+// GossipHandler handles incoming gossip messages
+type GossipHandler struct {
+       vm            *VM
+       atomicMempool *Mempool
+       txPool        *core.TxPool
+}
+
+func NewGossipHandler(vm *VM) *GossipHandler {
+       return &GossipHandler{
+               vm:            vm,
+               atomicMempool: vm.mempool,
+               txPool:        vm.chain.GetTxPool(),
+       }
+}
+
+func (h *GossipHandler) HandleAtomicTx(nodeID ids.ShortID, msg *message.AtomicTx) error {
+       log.Trace(
+               "AppGossip called with AtomicTx",
+               "peerID", nodeID,
+       )
+
+       if len(msg.Tx) == 0 {
+               log.Trace(
+                       "AppGossip received empty AtomicTx Message",
+                       "peerID", nodeID,
+               )
+               return nil
+       }
+
+       // In the case that the gossip message contains a transaction,
+       // attempt to parse it and add it as a remote.
+       tx := Tx{}
+       if _, err := Codec.Unmarshal(msg.Tx, &tx); err != nil {
+               log.Trace(
+                       "AppGossip provided invalid tx",
+                       "err", err,
+               )
+               return nil
+       }
+       unsignedBytes, err := Codec.Marshal(codecVersion, &tx.UnsignedAtomicTx)
+       if err != nil {
+               log.Trace(
+                       "AppGossip failed to marshal unsigned tx",
+                       "err", err,
+               )
+               return nil
+       }
+       tx.Initialize(unsignedBytes, msg.Tx)
+
+       txID := tx.ID()
+       if _, dropped, found := h.atomicMempool.GetTx(txID); found || dropped {
+               return nil
+       }
+
+       if err := h.vm.issueTx(&tx, false /*=local*/); err != nil {
+               log.Trace(
+                       "AppGossip provided invalid transaction",
+                       "peerID", nodeID,
+                       "err", err,
+               )
+       }
+
+       return nil
+}
+
+func (h *GossipHandler) HandleEthTxs(nodeID ids.ShortID, msg *message.EthTxs) error {
+       log.Trace(
+               "AppGossip called with EthTxs",
+               "peerID", nodeID,
+               "size(txs)", len(msg.Txs),
+       )
+
+       if len(msg.Txs) == 0 {
+               log.Trace(
+                       "AppGossip received empty EthTxs Message",
+                       "peerID", nodeID,
+               )
+               return nil
+       }
+
+       // The maximum size of this encoded object is enforced by the codec.
+       txs := make([]*types.Transaction, 0)
+       if err := rlp.DecodeBytes(msg.Txs, &txs); err != nil {
+               log.Trace(
+                       "AppGossip provided invalid txs",
+                       "peerID", nodeID,
+                       "err", err,
+               )
+               return nil
+       }
+       errs := h.txPool.AddRemotes(txs)
+       for i, err := range errs {
+               if err != nil {
+                       log.Trace(
+                               "AppGossip failed to add to mempool",
+                               "err", err,
+                               "tx", txs[i].Hash(),
+                       )
+               }
+       }
+       return nil
+}
+
+// noopGossiper should be used when gossip communication is not supported
+type noopGossiper struct{}
+
+func (n *noopGossiper) GossipAtomicTxs([]*Tx) error {
+       return nil
+}
+func (n *noopGossiper) GossipEthTxs([]*types.Transaction) error {
+       return nil
+}
diff --git a/plugin/evm/network_atomic_gossiping_test.go b/plugin/evm/gossiper_atomic_gossiping_test.go
similarity index 91%
rename from plugin/evm/network_atomic_gossiping_test.go
rename to plugin/evm/gossiper_atomic_gossiping_test.go
index 501bc5c8..15bc3322 100644
--- a/plugin/evm/network_atomic_gossiping_test.go
+++ b/plugin/evm/gossiper_atomic_gossiping_test.go
@@ -8,11 +8,11 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/avalanchego/ids"
+       "github.com/flare-foundation/flare/ids"

        "github.com/stretchr/testify/assert"

-       "github.com/ava-labs/coreth/plugin/evm/message"
+       "github.com/flare-foundation/coreth/plugin/evm/message"
 )

 // locally issued txs should be gossiped
@@ -35,7 +35,7 @@ func TestMempoolAtmTxsIssueTxAndGossiping(t *testing.T) {
                gossipedLock.Lock()
                defer gossipedLock.Unlock()

-               notifyMsgIntf, err := message.Parse(gossipedBytes)
+               notifyMsgIntf, err := message.ParseMessage(vm.networkCodec, gossipedBytes)
                assert.NoError(err)

                requestMsg, ok := notifyMsgIntf.(*message.AtomicTx)
@@ -61,7 +61,7 @@ func TestMempoolAtmTxsIssueTxAndGossiping(t *testing.T) {
        gossipedLock.Unlock()

        // Test hash on retry
-       assert.NoError(vm.network.GossipAtomicTxs([]*Tx{tx}))
+       assert.NoError(vm.gossiper.GossipAtomicTxs([]*Tx{tx}))
        assert.Equal(1, gossiped)
        gossipedLock.Lock()
        gossipedLock.Unlock()
@@ -111,7 +111,7 @@ func TestMempoolAtmTxsAppGossipHandling(t *testing.T) {
        msg := message.AtomicTx{
                Tx: tx.Bytes(),
        }
-       msgBytes, err := message.Build(&msg)
+       msgBytes, err := message.BuildMessage(vm.networkCodec, &msg)
        assert.NoError(err)

        // show that no txID is requested
@@ -134,7 +134,7 @@ func TestMempoolAtmTxsAppGossipHandling(t *testing.T) {
        msg = message.AtomicTx{
                Tx: conflictingTx.Bytes(),
        }
-       msgBytes, err = message.Build(&msg)
+       msgBytes, err = message.BuildMessage(vm.networkCodec, &msg)
        assert.NoError(err)
        assert.NoError(vm.AppGossip(nodeID, msgBytes))
        assert.False(txRequested, "tx should not have been requested")
@@ -179,7 +179,7 @@ func TestMempoolAtmTxsAppGossipHandlingDiscardedTx(t *testing.T) {

        mempool.AddTx(tx)
        mempool.NextTx()
-       mempool.DiscardCurrentTx()
+       mempool.DiscardCurrentTx(txID)

        // Check the mempool does not contain the discarded transaction
        assert.False(mempool.has(txID))
@@ -190,7 +190,7 @@ func TestMempoolAtmTxsAppGossipHandlingDiscardedTx(t *testing.T) {
        msg := message.AtomicTx{
                Tx: tx.Bytes(),
        }
-       msgBytes, err := message.Build(&msg)
+       msgBytes, err := message.BuildMessage(vm.networkCodec, &msg)
        assert.NoError(err)

        assert.NoError(vm.AppGossip(nodeID, msgBytes))
@@ -208,7 +208,7 @@ func TestMempoolAtmTxsAppGossipHandlingDiscardedTx(t *testing.T) {
        msg = message.AtomicTx{
                Tx: conflictingTx.Bytes(),
        }
-       msgBytes, err = message.Build(&msg)
+       msgBytes, err = message.BuildMessage(vm.networkCodec, &msg)
        assert.NoError(err)

        assert.NoError(vm.AppGossip(nodeID, msgBytes))
diff --git a/plugin/evm/gossiper_eth_gossiping_test.go b/plugin/evm/gossiper_eth_gossiping_test.go
new file mode 100644
index 00000000..af38466e
--- /dev/null
+++ b/plugin/evm/gossiper_eth_gossiping_test.go
@@ -0,0 +1,378 @@
+// (c) 2019-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+       "crypto/ecdsa"
+       "encoding/json"
+       "math/big"
+       "strings"
+       "sync"
+       "testing"
+       "time"
+
+       "github.com/flare-foundation/flare/ids"
+
+       "github.com/ethereum/go-ethereum/common"
+       "github.com/ethereum/go-ethereum/crypto"
```

```
+	"github.com/ethereum/go-ethereum/rlp"
+
+	"github.com/stretchr/testify/assert"
+
+	"github.com/flare-foundation/coreth/core"
+	"github.com/flare-foundation/coreth/core/types"
+	"github.com/flare-foundation/coreth/params"
+	"github.com/flare-foundation/coreth/plugin/evm/message"
+)
+
+func fundAddressByGenesis(addrs []common.Address) (string, error) {
+	balance := big.NewInt(0xffffffffffffff)
+	genesis := &core.Genesis{
+		Difficulty: common.Big0,
+		GasLimit:   uint64(5000000),
+	}
+	funds := make(map[common.Address]core.GenesisAccount)
+	for _, addr := range addrs {
+		funds[addr] = core.GenesisAccount{
+			Balance: balance,
+		}
+	}
+	genesis.Alloc = funds
+
+	genesis.Config = &params.ChainConfig{
+		ChainID:                     params.AvalancheLocalChainID,
+		ApricotPhase1BlockTimestamp: big.NewInt(0),
+		ApricotPhase2BlockTimestamp: big.NewInt(0),
+		ApricotPhase3BlockTimestamp: big.NewInt(0),
+		ApricotPhase4BlockTimestamp: big.NewInt(0),
+	}
+
+	bytes, err := json.Marshal(genesis)
+	return string(bytes), err
+}
+
+func getValidEthTxs(key *ecdsa.PrivateKey, count int, gasPrice *big.Int) []*types.Transaction {
+	res := make([]*types.Transaction, count)
+
+	to := common.Address{}
+	amount := big.NewInt(10000)
+	gasLimit := uint64(100000)
+
+	for i := 0; i < count; i++ {
+		tx, _ := types.SignTx(
+			types.NewTransaction(
+				uint64(i),
+				to,
+				amount,
+				gasLimit,
+				gasPrice,
+				[]byte(strings.Repeat("aaaaaaaaaa", 100))),
+			types.HomesteadSigner{}, key)
+		tx.SetFirstSeen(time.Now().Add(-1 * time.Minute))
+		res[i] = tx
+	}
+	return res
+}
+
+// show that locally issued eth txs are gossiped
+// Note: channel through which coreth mempool push txs to vm is injected here
+// to ease up UT, which target only VM behaviors in response to coreth mempool
+// signals
+func TestMempoolEthTxsAddedTxsGossipedAfterActivation(t *testing.T) {
+	assert := assert.New(t)
+
+	key, err := crypto.GenerateKey()
+	assert.NoError(err)
+
+	addr := crypto.PubkeyToAddress(key.PublicKey)
+
+	cfgJson, err := fundAddressByGenesis([]common.Address{addr})
+	assert.NoError(err)
+
+	_, vm, _, _, sender := GenesisVM(t, true, cfgJson, "", "")
+	defer func() {
+		err := vm.Shutdown()
+		assert.NoError(err)
+	}()
+	vm.chain.GetTxPool().SetGasPrice(common.Big1)
+	vm.chain.GetTxPool().SetMinFee(common.Big0)
+
+	// create eth txes
+	ethTxs := getValidEthTxs(key, 3, common.Big1)
+
+	var wg sync.WaitGroup
+	wg.Add(2)
+	sender.CantSendAppGossip = false
+	signal1 := make(chan struct{})
+	seen := 0
+	sender.SendAppGossipF = func(gossipedBytes []byte) error {
+		if seen == 0 {
+			notifyMsgIntf, err := message.ParseMessage(vm.networkCodec, gossipedBytes)
+			assert.NoError(err)
+
+			requestMsg, ok := notifyMsgIntf.(*message.EthTxs)
+			assert.True(ok)
+			assert.NotEmpty(requestMsg.Txs)
+
+			txs := make([]*types.Transaction, 0)
+			assert.NoError(rlp.DecodeBytes(requestMsg.Txs, &txs))
+			assert.Len(txs, 2)
+			assert.ElementsMatch(
+				[]common.Hash{ethTxs[0].Hash(), ethTxs[1].Hash()},
+				[]common.Hash{txs[0].Hash(), txs[1].Hash()},
+			)
+			seen++
+			close(signal1)
+		} else if seen == 1 {
+			notifyMsgIntf, err := message.ParseMessage(vm.networkCodec, gossipedBytes)
+			assert.NoError(err)
+
+			requestMsg, ok := notifyMsgIntf.(*message.EthTxs)
+			assert.True(ok)
+			assert.NotEmpty(requestMsg.Txs)
+
+			txs := make([]*types.Transaction, 0)
+			assert.NoError(rlp.DecodeBytes(requestMsg.Txs, &txs))
+			assert.Len(txs, 1)
+			assert.Equal(ethTxs[2].Hash(), txs[0].Hash())
+
+			seen++
+		} else {
+			t.Fatal("should not be seen 3 times")
+		}
+		wg.Done()
+		return nil
+	}
+
+	// Notify VM about eth txs
+	errs := vm.chain.GetTxPool().AddRemotesSync(ethTxs[:2])
+	for _, err := range errs {
+		assert.NoError(err, "failed adding coreth tx to mempool")
+	}
```

```
+
+       // Gossip txs again (shouldn't gossip hashes)
+       <-signal1 // wait until reorg processed
+       assert.NoError(vm.gossiper.GossipEthTxs(ethTxs[:2]))
+
+       errs = vm.chain.GetTxPool().AddRemotesSync(ethTxs)
+       assert.Contains(errs[0].Error(), "already known")
+       assert.Contains(errs[1].Error(), "already known")
+       assert.NoError(errs[2], "failed adding coreth tx to mempool")
+
+       attemptAwait(t, &wg, 5*time.Second)
+}
+
+// show that locally issued eth txs are chunked correctly
+func TestMempoolEthTxsAddedTxsGossipedAfterActivationChunking(t *testing.T) {
+       assert := assert.New(t)
+
+       key, err := crypto.GenerateKey()
+       assert.NoError(err)
+
+       addr := crypto.PubkeyToAddress(key.PublicKey)
+
+       cfgJson, err := fundAddressByGenesis([]common.Address{addr})
+       assert.NoError(err)
+
+       _, vm, _, _, sender := GenesisVM(t, true, cfgJson, "", "")
+       defer func() {
+               err := vm.Shutdown()
+               assert.NoError(err)
+       }()
+       vm.chain.GetTxPool().SetGasPrice(common.Big1)
+       vm.chain.GetTxPool().SetMinFee(common.Big0)
+
+       // create eth txes
+       ethTxs := getValidEthTxs(key, 100, common.Big1)
+
+       var wg sync.WaitGroup
+       wg.Add(2)
+       sender.CantSendAppGossip = false
+       seen := map[common.Hash]struct{}{}
+       sender.SendAppGossipF = func(gossipedBytes []byte) error {
+               notifyMsgIntf, err := message.ParseMessage(vm.networkCodec, gossipedBytes)
+               assert.NoError(err)
+
+               requestMsg, ok := notifyMsgIntf.(*message.EthTxs)
+               assert.True(ok)
+               assert.NotEmpty(requestMsg.Txs)
+
+               txs := make([]*types.Transaction, 0)
+               assert.NoError(rlp.DecodeBytes(requestMsg.Txs, &txs))
+               for _, tx := range txs {
+                       seen[tx.Hash()] = struct{}{}
+               }
+               wg.Done()
+               return nil
+       }
+
+       // Notify VM about eth txs
+       errs := vm.chain.GetTxPool().AddRemotesSync(ethTxs)
+       for _, err := range errs {
+               assert.NoError(err, "failed adding coreth tx to mempool")
+       }
+
+       attemptAwait(t, &wg, 5*time.Second)
+
+       for _, tx := range ethTxs {
+               _, ok := seen[tx.Hash()]
+               assert.True(ok, "missing hash: %v", tx.Hash())
+       }
+}
+
+// show that a geth tx discovered from gossip is requested to the same node that
+// gossiped it
+func TestMempoolEthTxsAppGossipHandling(t *testing.T) {
+       assert := assert.New(t)
+
+       key, err := crypto.GenerateKey()
+       assert.NoError(err)
+
+       addr := crypto.PubkeyToAddress(key.PublicKey)
+
+       cfgJson, err := fundAddressByGenesis([]common.Address{addr})
+       assert.NoError(err)
+
+       _, vm, _, _, sender := GenesisVM(t, true, cfgJson, "", "")
+       defer func() {
+               err := vm.Shutdown()
+               assert.NoError(err)
+       }()
+       vm.chain.GetTxPool().SetGasPrice(common.Big1)
+       vm.chain.GetTxPool().SetMinFee(common.Big0)
+
+       var (
+               wg          sync.WaitGroup
+               txRequested bool
+       )
+       sender.CantSendAppGossip = false
+       sender.SendAppRequestF = func(_ ids.ShortSet, _ uint32, _ []byte) error {
+               txRequested = true
+               return nil
+       }
+       wg.Add(1)
+       sender.SendAppGossipF = func(_ []byte) error {
+               wg.Done()
+               return nil
+       }
+
+       // prepare a tx
+       tx := getValidEthTxs(key, 1, common.Big1)[0]
+
+       // show that unknown coreth hashes is requested
+       txBytes, err := rlp.EncodeToBytes([]*types.Transaction{tx})
+       assert.NoError(err)
+       msg := message.EthTxs{
+               Txs: txBytes,
+       }
+       msgBytes, err := message.BuildMessage(vm.networkCodec, &msg)
+       assert.NoError(err)
+
+       nodeID := ids.GenerateTestShortID()
+       err = vm.AppGossip(nodeID, msgBytes)
+       assert.NoError(err)
+       assert.False(txRequested, "tx should not be requested")
+
+       // wait for transaction to be re-gossiped
+       attemptAwait(t, &wg, 5*time.Second)
+}
+
+func TestMempoolEthTxsRegossipSingleAccount(t *testing.T) {
+       assert := assert.New(t)
+
+       key, err := crypto.GenerateKey()
+       assert.NoError(err)
+
+       addr := crypto.PubkeyToAddress(key.PublicKey)
+
+       cfgJson, err := fundAddressByGenesis([]common.Address{addr})
+       assert.NoError(err)
+
+       _, vm, _, _, _ := GenesisVM(t, true, cfgJson, `{"local-txs-enabled":true}`, "")
+       defer func() {
+               err := vm.Shutdown()
+               assert.NoError(err)
+       }()
+       vm.chain.GetTxPool().SetGasPrice(common.Big1)
+       vm.chain.GetTxPool().SetMinFee(common.Big0)
+
+       // create eth txes
+       ethTxs := getValidEthTxs(key, 10, big.NewInt(226*params.GWei))
+
+       // Notify VM about eth txs
+       errs := vm.chain.GetTxPool().AddRemotesSync(ethTxs)
+       for _, err := range errs {
+               assert.NoError(err, "failed adding coreth tx to remote mempool")
+       }
+
+       // Only 1 transaction will be regossiped for an address (should be lowest
+       // nonce)
+       pushNetwork := vm.gossiper.(*pushGossiper)
+       queued := pushNetwork.queueRegossipTxs()
+       assert.Len(queued, 1, "unexpected length of queued txs")
+       assert.Equal(ethTxs[0].Hash(), queued[0].Hash())
+}
+
+func TestMempoolEthTxsRegossip(t *testing.T) {
+       assert := assert.New(t)
+
+       keys := make([]*ecdsa.PrivateKey, 20)
+       addrs := make([]common.Address, 20)
+       for i := 0; i < 20; i++ {
+               key, err := crypto.GenerateKey()
+               assert.NoError(err)
+               keys[i] = key
+               addrs[i] = crypto.PubkeyToAddress(key.PublicKey)
+       }
+
+       cfgJson, err := fundAddressByGenesis(addrs)
+       assert.NoError(err)
+
+       _, vm, _, _, _ := GenesisVM(t, true, cfgJson, `{"local-txs-enabled":true}`, "")
+       defer func() {
+               err := vm.Shutdown()
+               assert.NoError(err)
+       }()
+       vm.chain.GetTxPool().SetGasPrice(common.Big1)
+       vm.chain.GetTxPool().SetMinFee(common.Big0)
+
+       // create eth txes
+       ethTxs := make([]*types.Transaction, 20)
+       ethTxHashes := make([]common.Hash, 20)
+       for i := 0; i < 20; i++ {
+               txs := getValidEthTxs(keys[i], 1, big.NewInt(226*params.GWei))
+               tx := txs[0]
+               ethTxs[i] = tx
+               ethTxHashes[i] = tx.Hash()
+       }
+
+       // Notify VM about eth txs
+       errs := vm.chain.GetTxPool().AddRemotesSync(ethTxs[:10])
+       for _, err := range errs {
+               assert.NoError(err, "failed adding coreth tx to remote mempool")
+       }
+       errs = vm.chain.GetTxPool().AddLocals(ethTxs[10:])
+       for _, err := range errs {
+               assert.NoError(err, "failed adding coreth tx to local mempool")
+       }
+
+       // We expect 15 transactions (the default max number of transactions to
+       // regossip) comprised of 10 local txs and 5 remote txs (we prioritize local
+       // txs over remote).
+       pushNetwork := vm.gossiper.(*pushGossiper)
+       queued := pushNetwork.queueRegossipTxs()
+       assert.Len(queued, 15, "unexpected length of queued txs")
+
+       // Confirm queued transactions (should be ordered based on
+       // timestamp submitted, with local priorized over remote)
+       queuedTxHashes := make([]common.Hash, 15)
+       for i, tx := range queued {
+               queuedTxHashes[i] = tx.Hash()
+       }
+       assert.ElementsMatch(queuedTxHashes[:10], ethTxHashes[10:], "missing local transactions")
+
+       // NOTE: We don't care which remote transactions are included in this test
+       // (due to the non-deterministic way pending transactions are surfaced, this can be difficult
+       // to assert as well).
+}
diff --git a/plugin/evm/import_tx.go b/plugin/evm/import_tx.go
index 6ae63526..6d74bcd7 100644
--- a/plugin/evm/import_tx.go
+++ b/plugin/evm/import_tx.go
@@ -7,19 +7,19 @@ import (
        "fmt"
        "math/big"

-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"

-       "github.com/ava-labs/avalanchego/chains/atomic"
-       "github.com/ava-labs/avalanchego/database"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/math"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/utils/math"
+       "github.com/flare-foundation/flare/vms/components/avax"
+       "github.com/flare-foundation/flare/vms/components/verify"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"
 )

 // UnsignedImportTx is an unsigned ImportTx
@@ -39,102 +39,24 @@ type UnsignedImportTx struct {

 // InputUTXOs returns the UTXOIDs of the imported funds
 func (tx *UnsignedImportTx) InputUTXOs() ids.Set {
-       set := ids.NewSet(len(tx.ImportedInputs))
-       for _, in := range tx.ImportedInputs {
-               set.Add(in.InputID())
-       }
-       return set
+       return ids.Set{}
 }

 // Verify this transaction is well-formed
 func (tx *UnsignedImportTx) Verify(
-       xChainID ids.ID,
        ctx *snow.Context,
        rules params.Rules,
 ) error {
-       switch {
-       case tx == nil:
-               return errNilTx
-       case tx.SourceChain != xChainID:
-               return errWrongChainID
-       case len(tx.ImportedInputs) == 0:
-               return errNoImportInputs
-       case tx.NetworkID != ctx.NetworkID:
-               return errWrongNetworkID
-       case ctx.ChainID != tx.BlockchainID:
-               return errWrongBlockchainID
-       case rules.IsApricotPhase3 && len(tx.Outs) == 0:
-               return errNoEVMOutputs
-       }
-
-       for _, out := range tx.Outs {
-               if err := out.Verify(); err != nil {
-                       return fmt.Errorf("EVM Output failed verification: %w", err)
-               }
-       }
-
-       for _, in := range tx.ImportedInputs {
-               if err := in.Verify(); err != nil {
-                       return fmt.Errorf("atomic input failed verification: %w", err)
-               }
-       }
-       if !avax.IsSortedAndUniqueTransferableInputs(tx.ImportedInputs) {
-               return errInputsNotSortedUnique
-       }
-
-       if rules.IsApricotPhase2 {
-               if !IsSortedAndUniqueEVMOutputs(tx.Outs) {
-                       return errOutputsNotSortedUnique
-               }
-       } else if rules.IsApricotPhase1 {
-               if !IsSortedEVMOutputs(tx.Outs) {
-                       return errOutputsNotSorted
-               }
-       }
-
-       return nil
+       return errImportTxsDisabled
 }

-func (tx *UnsignedImportTx) GasUsed() (uint64, error) {
-       cost := calcBytesCost(len(tx.UnsignedBytes()))
-       for _, in := range tx.ImportedInputs {
-               inCost, err := in.In.Cost()
-               if err != nil {
-                       return 0, err
-               }
-               cost, err = math.Add64(cost, inCost)
-               if err != nil {
-                       return 0, err
-               }
-       }
-       return cost, nil
+func (tx *UnsignedImportTx) GasUsed(fixedFee bool) (uint64, error) {
+       return 0, errImportTxsDisabled
 }

 // Amount of [assetID] burned by this transaction
 func (tx *UnsignedImportTx) Burned(assetID ids.ID) (uint64, error) {
-       var (
-               spent uint64
-               input uint64
-               err   error
-       )
-       for _, out := range tx.Outs {
-               if out.AssetID == assetID {
-                       spent, err = math.Add64(spent, out.Amount)
-                       if err != nil {
-                               return 0, err
-                       }
-               }
-       }
-       for _, in := range tx.ImportedInputs {
-               if in.AssetID() == assetID {
-                       input, err = math.Add64(input, in.Input().Amount())
-                       if err != nil {
-                               return 0, err
-                       }
-               }
-       }
-
-       return math.Sub64(input, spent)
+       return 0, errImportTxsDisabled
 }

 // SemanticVerify this transaction is valid.
@@ -145,82 +67,7 @@ func (tx *UnsignedImportTx) SemanticVerify(
        baseFee *big.Int,
        rules params.Rules,
 ) error {
-       if err := tx.Verify(vm.ctx.XChainID, vm.ctx, rules); err != nil {
-               return err
-       }
-
-       // Check the transaction consumes and produces the right amounts
-       fc := avax.NewFlowChecker()
-       switch {
-       // Apply dynamic fees to import transactions as of Apricot Phase 3
-       case rules.IsApricotPhase3:
-               gasUsed, err := stx.GasUsed()
-               if err != nil {
-                       return err
-               }
-               txFee, err := calculateDynamicFee(gasUsed, baseFee)
-               if err != nil {
-                       return err
-               }
-               fc.Produce(vm.ctx.AVAXAssetID, txFee)
-
-       // Apply fees to import transactions as of Apricot Phase 2
-       case rules.IsApricotPhase2:
-               fc.Produce(vm.ctx.AVAXAssetID, params.AvalancheAtomicTxFee)
-       }
-       for _, out := range tx.Outs {
-               fc.Produce(out.AssetID, out.Amount)
-       }
-       for _, in := range tx.ImportedInputs {
-               fc.Consume(in.AssetID(), in.Input().Amount())
-       }
-
```

```
-       if err := fc.Verify(); err != nil {
-               return fmt.Errorf("import tx flow check failed due to: %w", err)
-       }
-
-       if len(stx.Creds) != len(tx.ImportedInputs) {
-               return fmt.Errorf("import tx contained mismatched number of inputs/credentials (%d vs. %d)", len(tx.ImportedInputs), len(stx.Creds))
-       }
-
-       if !vm.ctx.IsBootstrapped() {
-               // Allow for force committing during bootstrapping
-               return nil
-       }
-
-       utxoIDs := make([][]byte, len(tx.ImportedInputs))
-       for i, in := range tx.ImportedInputs {
-               inputID := in.UTXOID.InputID()
-               utxoIDs[i] = inputID[:]
-       }
-       // allUTXOBytes is guaranteed to be the same length as utxoIDs
-       allUTXOBytes, err := vm.ctx.SharedMemory.Get(tx.SourceChain, utxoIDs)
-       if err != nil {
-               return fmt.Errorf("failed to fetch import UTXOs from %s due to: %w", tx.SourceChain, err)
-       }
-
-       for i, in := range tx.ImportedInputs {
-               utxoBytes := allUTXOBytes[i]
-
-               utxo := &avax.UTXO{}
-               if _, err := vm.codec.Unmarshal(utxoBytes, utxo); err != nil {
-                       return fmt.Errorf("failed to unmarshal UTXO: %w", err)
-               }
-
-               cred := stx.Creds[i]
-
-               utxoAssetID := utxo.AssetID()
-               inAssetID := in.AssetID()
-               if utxoAssetID != inAssetID {
-                       return errAssetIDMismatch
-               }
-
-               if err := vm.fx.VerifyTransfer(tx, in.In, cred, utxo.Out); err != nil {
-                       return fmt.Errorf("import tx transfer failed verification: %w", err)
-               }
-       }
-
-       return vm.conflicts(tx.InputUTXOs(), parent)
+       return errImportTxsDisabled
 }

 // Accept this transaction and spend imported inputs
@@ -228,13 +75,8 @@ func (tx *UnsignedImportTx) SemanticVerify(
 // we don't want to remove an imported UTXO in semanticVerify
 // only to have the transaction not be Accepted. This would be inconsistent.
 // Recall that imported UTXOs are not kept in a versionDB.
-func (tx *UnsignedImportTx) Accept(ctx *snow.Context, batch database.Batch) error {
-       utxoIDs := make([][]byte, len(tx.ImportedInputs))
-       for i, in := range tx.ImportedInputs {
-               inputID := in.InputID()
-               utxoIDs[i] = inputID[:]
-       }
-       return ctx.SharedMemory.Apply(map[ids.ID]*atomic.Requests{tx.SourceChain: {RemoveRequests: utxoIDs}}, batch)
+func (tx *UnsignedImportTx) Accept() (ids.ID, *atomic.Requests, error) {
+       return ids.ID{}, nil, errImportTxsDisabled
 }

 // newImportTx returns a new ImportTx
@@ -244,160 +86,22 @@ func (vm *VM) newImportTx(
        baseFee *big.Int, // fee to use post-AP3
        keys []*crypto.PrivateKeySECP256K1R, // Keys to import the funds
 ) (*Tx, error) {
-       if vm.ctx.XChainID != chainID {
-               return nil, errWrongChainID
-       }
-
-       kc := secp256k1fx.NewKeychain()
-       for _, key := range keys {
-               kc.Add(key)
-       }
-
-       atomicUTXOs, _, _, err := vm.GetAtomicUTXOs(chainID, kc.Addresses(), ids.ShortEmpty, ids.Empty, -1)
-       if err != nil {
-               return nil, fmt.Errorf("problem retrieving atomic UTXOs: %w", err)
-       }
-
-       importedInputs := []*avax.TransferableInput{}
-       signers := [][]*crypto.PrivateKeySECP256K1R{}
-
-       importedAmount := make(map[ids.ID]uint64)
-       now := vm.clock.Unix()
-       for _, utxo := range atomicUTXOs {
-               inputIntf, utxoSigners, err := kc.Spend(utxo.Out, now)
-               if err != nil {
-                       continue
-               }
-               input, ok := inputIntf.(avax.TransferableIn)
-               if !ok {
-                       continue
-               }
-               aid := utxo.AssetID()
-               importedAmount[aid], err = math.Add64(importedAmount[aid], input.Amount())
-               if err != nil {
-                       return nil, err
-               }
-               importedInputs = append(importedInputs, &avax.TransferableInput{
-                       UTXOID: utxo.UTXOID,
-                       Asset:  utxo.Asset,
-                       In:     input,
-               })
-               signers = append(signers, utxoSigners)
-       }
-       avax.SortTransferableInputsWithSigners(importedInputs, signers)
-       importedAVAXAmount := importedAmount[vm.ctx.AVAXAssetID]
-
-       outs := make([]EVMOutput, 0, len(importedAmount))
-       // This will create unique outputs (in the context of sorting)
-       // since each output will have a unique assetID
-       for assetID, amount := range importedAmount {
-               // Skip the AVAX amount since it is included separately to account for
-               // the fee
-               if assetID == vm.ctx.AVAXAssetID || amount == 0 {
-                       continue
-               }
-               outs = append(outs, EVMOutput{
-                       Address: to,
-                       Amount:  amount,
-                       AssetID: assetID,
-               })
-       }
-
-       rules := vm.currentRules()
-
-       var (
-               txFeeWithoutChange uint64
-               txFeeWithChange    uint64
-       )
-       switch {
-       case rules.IsApricotPhase3:
-               if baseFee == nil {
-                       return nil, errNilBaseFeeApricotPhase3
-               }
-               utx := &UnsignedImportTx{
-                       NetworkID:      vm.ctx.NetworkID,
-                       BlockchainID:   vm.ctx.ChainID,
-                       Outs:           outs,
-                       ImportedInputs: importedInputs,
-                       SourceChain:    chainID,
-               }
-               tx := &Tx{UnsignedAtomicTx: utx}
-               if err := tx.Sign(vm.codec, nil); err != nil {
-                       return nil, err
-               }
-
-               gasUsedWithoutChange, err := tx.GasUsed()
-               if err != nil {
-                       return nil, err
-               }
-               gasUsedWithChange := gasUsedWithoutChange + EVMOutputGas
-
-               txFeeWithoutChange, err = calculateDynamicFee(gasUsedWithoutChange, baseFee)
-               if err != nil {
-                       return nil, err
-               }
-               txFeeWithChange, err = calculateDynamicFee(gasUsedWithChange, baseFee)
-               if err != nil {
-                       return nil, err
-               }
-       case rules.IsApricotPhase2:
-               txFeeWithoutChange = params.AvalancheAtomicTxFee
-               txFeeWithChange = params.AvalancheAtomicTxFee
-       }
-
-       // AVAX output
-       if importedAVAXAmount < txFeeWithoutChange { // imported amount goes toward paying tx fee
-               return nil, errInsufficientFundsForFee
-       }
-
-       if importedAVAXAmount > txFeeWithChange {
-               outs = append(outs, EVMOutput{
-                       Address: to,
-                       Amount:  importedAVAXAmount - txFeeWithChange,
-                       AssetID: vm.ctx.AVAXAssetID,
-               })
-       }
-
-       // If no outputs are produced, return an error.
-       // Note: this can happen if there is exactly enough AVAX to pay the
-       // transaction fee, but no other funds to be imported.
-       if len(outs) == 0 {
-               return nil, errNoEVMOutputs
-       }
-
-       SortEVMOutputs(outs)
+       return nil, errImportTxsDisabled
+}

-       // Create the transaction
-       utx := &UnsignedImportTx{
-               NetworkID:      vm.ctx.NetworkID,
-               BlockchainID:   vm.ctx.ChainID,
-               Outs:           outs,
-               ImportedInputs: importedInputs,
-               SourceChain:    chainID,
-       }
-       tx := &Tx{UnsignedAtomicTx: utx}
-       if err := tx.Sign(vm.codec, signers); err != nil {
-               return nil, err
-       }
-       return tx, utx.Verify(vm.ctx.XChainID, vm.ctx, vm.currentRules())
+// newImportTx returns a new ImportTx
+func (vm *VM) newImportTxWithUTXOs(
+       chainID ids.ID, // chain to import from
+       to common.Address, // Address of recipient
+       baseFee *big.Int, // fee to use post-AP3
+       kc *secp256k1fx.Keychain, // Keychain to use for signing the atomic UTXOs
+       atomicUTXOs []*avax.UTXO, // UTXOs to spend
+) (*Tx, error) {
+       return nil, errImportTxsDisabled
 }

 // EVMStateTransfer performs the state transfer to increase the balances of
 // accounts accordingly with the imported EVMOutputs
 func (tx *UnsignedImportTx) EVMStateTransfer(ctx *snow.Context, state *state.StateDB) error {
-       for _, to := range tx.Outs {
-               if to.AssetID == ctx.AVAXAssetID {
-                       log.Debug("crosschain X->C", "addr", to.Address, "amount", to.Amount, "assetID", "AVAX")
-                       // If the asset is AVAX, convert the input amount in nAVAX to gWei by
-                       // multiplying by the x2c rate.
-                       amount := new(big.Int).Mul(
-                               new(big.Int).SetUint64(to.Amount), x2cRate)
-                       state.AddBalance(to.Address, amount)
-               } else {
-                       log.Debug("crosschain X->C", "addr", to.Address, "amount", to.Amount, "assetID", to.AssetID)
-                       amount := new(big.Int).SetUint64(to.Amount)
-                       state.AddBalanceMultiCoin(to.Address, common.Hash(to.AssetID), amount)
-               }
-       }
-       return nil
+       return errImportTxsDisabled
 }
diff --git a/plugin/evm/import_tx_test.go b/plugin/evm/import_tx_test.go
deleted file mode 100644
index b3604e1a..00000000
--- a/plugin/evm/import_tx_test.go
+++ /dev/null
@@ -1,1153 +0,0 @@
-// (c) 2019-2020, Ava Labs, Inc. All rights reserved.
-// See the file LICENSE for licensing terms.
-
-package evm
-
-import (
-       "math/big"
-       "testing"
-
-       "github.com/ava-labs/coreth/params"
-       "github.com/ethereum/go-ethereum/common"
-
-       "github.com/ava-labs/avalanchego/chains/atomic"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
-)
-
-// createImportTxOptions adds a UTXO to shared memory and generates a list of import transactions sending this UTXO
-// to each of the three test keys (conflicting transactions)
-func createImportTxOptions(t *testing.T, vm *VM, sharedMemory *atomic.Memory) []*Tx {
-       utxo := &avax.UTXO{
```

```
-               UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-               Out: &secp256k1fx.TransferOutput{
-                       Amt: uint64(50000000),
-                       OutputOwners: secp256k1fx.OutputOwners{
-                               Threshold: 1,
-                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                       },
-               },
-       }
-       utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-       inputID := utxo.InputID()
-       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-               Key:   inputID[:],
-               Value: utxoBytes,
-               Traits: [][]byte{
-                       testKeys[0].PublicKey().Address().Bytes(),
-               },
-       }}}}); err != nil {
-               t.Fatal(err)
-       }
-
-       importTxs := make([]*Tx, 0, 3)
-       for _, ethAddr := range testEthAddrs {
-               importTx, err := vm.newImportTx(vm.ctx.XChainID, ethAddr, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-               if err != nil {
-                       t.Fatal(err)
-               }
-               importTxs = append(importTxs, importTx)
-       }
-
-       return importTxs
-}
-
-func TestImportTxVerify(t *testing.T) {
-       ctx := NewContext()
-
-       var importAmount uint64 = 10000000
-       txID := ids.GenerateTestID()
-       importTx := &UnsignedImportTx{
-               NetworkID:    ctx.NetworkID,
-               BlockchainID: ctx.ChainID,
-               SourceChain:  ctx.XChainID,
-               ImportedInputs: []*avax.TransferableInput{
-                       {
-                               UTXOID: avax.UTXOID{
-                                       TxID:        txID,
-                                       OutputIndex: uint32(0),
-                               },
-                               Asset: avax.Asset{ID: ctx.AVAXAssetID},
-                               In: &secp256k1fx.TransferInput{
-                                       Amt: importAmount,
-                                       Input: secp256k1fx.Input{
-                                               SigIndices: []uint32{0},
-                                       },
-                               },
-                       },
-                       {
-                               UTXOID: avax.UTXOID{
-                                       TxID:        txID,
-                                       OutputIndex: uint32(1),
-                               },
-                               Asset: avax.Asset{ID: ctx.AVAXAssetID},
-                               In: &secp256k1fx.TransferInput{
-                                       Amt: importAmount,
-                                       Input: secp256k1fx.Input{
-                                               SigIndices: []uint32{0},
-                                       },
-                               },
-                       },
-               },
-               Outs: []EVMOutput{
-                       {
-                               Address: testEthAddrs[0],
-                               Amount:  importAmount - params.AvalancheAtomicTxFee,
-                               AssetID: ctx.AVAXAssetID,
-                       },
-                       {
-                               Address: testEthAddrs[1],
-                               Amount:  importAmount,
-                               AssetID: ctx.AVAXAssetID,
-                       },
-               },
-       }
-
-       // // Sort the inputs and outputs to ensure the transaction is canonical
-       avax.SortTransferableInputs(importTx.ImportedInputs)
-       SortEVMOutputs(importTx.Outs)
-
-       tests := map[string]atomicTxVerifyTest{
-               "nil tx": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
-                               var importTx *UnsignedImportTx
-                               return importTx
-                       },
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase0,
-                       expectedErr: errNilTx.Error(),
-               },
-               "valid import tx": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
-                               return importTx
-                       },
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase0,
-                       expectedErr: "", // Expect this transaction to be valid
-               },
-               "invalid network ID": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
-                               tx := *importTx
-                               tx.NetworkID++
-                               return &tx
-                       },
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase0,
-                       expectedErr: errWrongNetworkID.Error(),
-               },
-               "invalid blockchain ID": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
-                               tx := *importTx
-                               tx.BlockchainID = ids.GenerateTestID()
-                               return &tx
-                       },
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase0,
-                       expectedErr: errWrongBlockchainID.Error(),
-               },
-               "invalid source chain ID": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
```

```
-					tx := *importTx
-					tx.SourceChain = ids.GenerateTestID()
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: errWrongChainID.Error(),
-			},
-			"no inputs": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.ImportedInputs = nil
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: errNoImportInputs.Error(),
-			},
-			"inputs sorted incorrectly": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.ImportedInputs = []*avax.TransferableInput{
-						tx.ImportedInputs[1],
-						tx.ImportedInputs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: errInputsNotSortedUnique.Error(),
-			},
-			"invalid input": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.ImportedInputs = []*avax.TransferableInput{
-						tx.ImportedInputs[0],
-						nil,
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: "atomic input failed verification",
-			},
-			"unsorted outputs phase 0 passes verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[1],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: "",
-			},
-			"non-unique outputs phase 0 passes verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[0],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase0,
-				expectedErr: "",
-			},
-			"unsorted outputs phase 1 fails verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[1],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase1,
-				expectedErr: errOutputsNotSorted.Error(),
-			},
-			"non-unique outputs phase 1 passes verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[0],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase1,
-				expectedErr: "",
-			},
-			"outputs not sorted and unique phase 2 fails verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[0],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase2,
-				expectedErr: errOutputsNotSortedUnique.Error(),
-			},
-			"outputs not sorted phase 2 fails verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						tx.Outs[1],
-						tx.Outs[0],
-					}
-					return &tx
-				},
-				ctx:         ctx,
-				rules:       apricotRulesPhase2,
-				expectedErr: errOutputsNotSortedUnique.Error(),
-			},
-			"invalid EVMOutput fails verification": {
-				generate: func(t *testing.T) UnsignedAtomicTx {
-					tx := *importTx
-					tx.Outs = []EVMOutput{
-						{
-							Address: testEthAddrs[0],
-							Amount:  0,
-							AssetID: testAvaxAssetID,
-						},
-					}
-					return &tx
-				},
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase0,
-                       expectedErr: "EVM Output failed verification",
-               },
-               "no outputs apricot phase 3": {
-                       generate: func(t *testing.T) UnsignedAtomicTx {
-                               tx := *importTx
-                               tx.Outs = nil
-                               return &tx
-                       },
-                       ctx:         ctx,
-                       rules:       apricotRulesPhase3,
-                       expectedErr: errNoEVMOutputs.Error(),
-               },
-       }
-       for name, test := range tests {
-               t.Run(name, func(t *testing.T) {
-                       executeTxVerifyTest(t, test)
-               })
-       }
-}
-
-func TestNewImportTx(t *testing.T) {
-       importAmount := uint64(5000000)
-       // createNewImportAVAXTx adds a UTXO to shared memory and then constructs a new import transaction
-       // and checks that it has the correct fee for the base fee that has been used
-       createNewImportAVAXTx := func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-               txID := ids.GenerateTestID()
-               _, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, importAmount, testShortIDAddrs[0])
-               if err != nil {
-                       t.Fatal(err)
-               }
-
-               tx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-               if err != nil {
-                       t.Fatal(err)
-               }
-               importTx := tx.UnsignedAtomicTx
-               var actualFee uint64
-               actualAVAXBurned, err := importTx.Burned(vm.ctx.AVAXAssetID)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               rules := vm.currentRules()
-               switch {
-               case rules.IsApricotPhase3:
-                       actualCost, err := importTx.GasUsed()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       actualFee, err = calculateDynamicFee(actualCost, initialBaseFee)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-               case rules.IsApricotPhase2:
-                       actualFee = 1000000
-               default:
-                       actualFee = 0
-               }
-
-               if actualAVAXBurned != actualFee {
-                       t.Fatalf("AVAX burned (%d) != actual fee (%d)", actualAVAXBurned, actualFee)
-               }
-
-               return tx
-       }
-       checkState := func(t *testing.T, vm *VM) {
-               tx, err := vm.extractAtomicTx(vm.LastAcceptedBlockInternal().(*Block).ethBlock)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               if tx == nil {
-                       t.Fatal("Expected import tx to be in the last accepted block, but found nil")
-               }
-               actualAVAXBurned, err := tx.UnsignedAtomicTx.Burned(vm.ctx.AVAXAssetID)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               // Ensure that the UTXO has been removed from shared memory within Accept
-               addrSet := ids.ShortSet{}
-               addrSet.Add(testShortIDAddrs[0])
-               utxos, _, _, err := vm.GetAtomicUTXOs(vm.ctx.XChainID, addrSet, ids.ShortEmpty, ids.Empty, -1)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               if len(utxos) != 0 {
-                       t.Fatalf("Expected to find 0 UTXOs after accepting import transaction, but found %d", len(utxos))
-               }
-
-               // Ensure that the call to EVMStateTransfer correctly updates the balance of [addr]
-               sdb, err := vm.chain.CurrentState()
-               if err != nil {
-                       t.Fatal(err)
-               }
-
-               expectedRemainingBalance := new(big.Int).Mul(new(big.Int).SetUint64(importAmount-actualAVAXBurned), x2cRate)
-               addr := GetEthAddress(testKeys[0])
-               if actualBalance := sdb.GetBalance(addr); actualBalance.Cmp(expectedRemainingBalance) != 0 {
-                       t.Fatalf("address remaining balance %s equal %s not %s", addr.String(), actualBalance, expectedRemainingBalance)
-               }
-       }
-       tests2 := map[string]atomicTxTest{
-               "apricot phase 0": {
-                       setup:       createNewImportAVAXTx,
-                       checkState:  checkState,
-                       genesisJSON: genesisJSONApricotPhase0,
-               },
-               "apricot phase 1": {
-                       setup:       createNewImportAVAXTx,
-                       checkState:  checkState,
-                       genesisJSON: genesisJSONApricotPhase1,
-               },
-               "apricot phase 2": {
-                       setup:       createNewImportAVAXTx,
-                       checkState:  checkState,
-                       genesisJSON: genesisJSONApricotPhase2,
-               },
-               "apricot phase 3": {
-                       setup:       createNewImportAVAXTx,
-                       checkState:  checkState,
-                       genesisJSON: genesisJSONApricotPhase3,
-               },
-       }
-
-       for name, test := range tests2 {
-               t.Run(name, func(t *testing.T) {
-                       executeTxTest(t, test)
-               })
-       }
-}
-
-// Note: this is a brittle test to ensure that the gas cost of a transaction does
-// not change
-func TestImportTxGasCost(t *testing.T) {
```

```
-	avaxAssetID := ids.GenerateTestID()
-	antAssetID := ids.GenerateTestID()
-	chainID := ids.GenerateTestID()
-	xChainID := ids.GenerateTestID()
-	networkID := uint32(5)
-	importAmount := uint64(5000000)
-
-	tests := map[string]struct {
-		UnsignedImportTx *UnsignedImportTx
-		Keys             [][]*crypto.PrivateKeySECP256K1R
-
-		ExpectedGasUsed uint64
-		ExpectedFee     uint64
-		BaseFee         *big.Int
-	}{
-		"simple import": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{{
-					UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-					Asset:  avax.Asset{ID: avaxAssetID},
-					In: &secp256k1fx.TransferInput{
-						Amt:   importAmount,
-						Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-					},
-				}},
-				Outs: []EVMOutput{{
-					Address: testEthAddrs[0],
-					Amount:  importAmount,
-					AssetID: avaxAssetID,
-				}},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}},
-			ExpectedGasUsed: 1230,
-			ExpectedFee:     30750,
-			BaseFee:         big.NewInt(25 * params.GWei),
-		},
-		"simple import 1wei": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{{
-					UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-					Asset:  avax.Asset{ID: avaxAssetID},
-					In: &secp256k1fx.TransferInput{
-						Amt:   importAmount,
-						Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-					},
-				}},
-				Outs: []EVMOutput{{
-					Address: testEthAddrs[0],
-					Amount:  importAmount,
-					AssetID: avaxAssetID,
-				}},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}},
-			ExpectedGasUsed: 1230,
-			ExpectedFee:     1,
-			BaseFee:         big.NewInt(1),
-		},
-		"simple ANT import": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: antAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-				},
-				Outs: []EVMOutput{
-					{
-						Address: testEthAddrs[0],
-						Amount:  importAmount,
-						AssetID: antAssetID,
-					},
-				},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}, {testKeys[0]}},
-			ExpectedGasUsed: 2318,
-			ExpectedFee:     57950,
-			BaseFee:         big.NewInt(25 * params.GWei),
-		},
-		"complex ANT import": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: antAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-				},
-				Outs: []EVMOutput{
-					{
-						Address: testEthAddrs[0],
-						Amount:  importAmount,
-						AssetID: avaxAssetID,
-					},
-					{
-						Address: testEthAddrs[0],
-						Amount:  importAmount,
-						AssetID: antAssetID,
-					},
-				},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}, {testKeys[0]}},
-			ExpectedGasUsed: 2378,
-			ExpectedFee:     59450,
-			BaseFee:         big.NewInt(25 * params.GWei),
-		},
-		"multisig import": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{{
-					UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-					Asset:  avax.Asset{ID: avaxAssetID},
-					In: &secp256k1fx.TransferInput{
-						Amt:   importAmount,
-						Input: secp256k1fx.Input{SigIndices: []uint32{0, 1}},
-					},
-				}},
-				Outs: []EVMOutput{{
-					Address: testEthAddrs[0],
-					Amount:  importAmount,
-					AssetID: avaxAssetID,
-				}},
-			},
-			Keys:            [][]*crypto.PrivateKeySECP256K1R{{testKeys[0], testKeys[1]}},
-			ExpectedGasUsed: 2234,
-			ExpectedFee:     55850,
-			BaseFee:         big.NewInt(25 * params.GWei),
-		},
-		"large import": {
-			UnsignedImportTx: &UnsignedImportTx{
-				NetworkID:    networkID,
-				BlockchainID: chainID,
-				SourceChain:  xChainID,
-				ImportedInputs: []*avax.TransferableInput{
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-					{
-						UTXOID: avax.UTXOID{TxID: ids.GenerateTestID()},
-						Asset:  avax.Asset{ID: avaxAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   importAmount,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					},
-				},
-				Outs: []EVMOutput{
-					{
-						Address: testEthAddrs[0],
-						Amount:  importAmount * 10,
-						AssetID: avaxAssetID,
-					},
-				},
-			},
-			Keys: [][]*crypto.PrivateKeySECP256K1R{
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-				{testKeys[0]},
-                               {testKeys[0]},
-                               {testKeys[0]},
-                       },
-                       ExpectedGasUsed: 11022,
-                       ExpectedFee:     275550,
-                       BaseFee:         big.NewInt(25 * params.GWei),
-               },
-       }
-
-       for name, test := range tests {
-               t.Run(name, func(t *testing.T) {
-                       tx := &Tx{UnsignedAtomicTx: test.UnsignedImportTx}
-
-                       // Sign with the correct key
-                       if err := tx.Sign(Codec, test.Keys); err != nil {
-                               t.Fatal(err)
-                       }
-
-                       gasUsed, err := tx.GasUsed()
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       if gasUsed != test.ExpectedGasUsed {
-                               t.Fatalf("Expected gasUsed to be %d, but found %d", test.ExpectedGasUsed, gasUsed)
-                       }
-
-                       fee, err := calculateDynamicFee(gasUsed, test.BaseFee)
-                       if err != nil {
-                               t.Fatal(err)
-                       }
-                       if fee != test.ExpectedFee {
-                               t.Fatalf("Expected fee to be %d, but found %d", test.ExpectedFee, fee)
-                       }
-               })
-       }
-}
-
-func TestImportTxSemanticVerify(t *testing.T) {
-       tests := map[string]atomicTxTest{
-               "UTXO not present during bootstrapping": {
-                       setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-                               tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-                                       NetworkID:    vm.ctx.NetworkID,
-                                       BlockchainID: vm.ctx.ChainID,
-                                       SourceChain:  vm.ctx.XChainID,
-                                       ImportedInputs: []*avax.TransferableInput{{
-                                               UTXOID: avax.UTXOID{
-                                                       TxID: ids.GenerateTestID(),
-                                               },
-                                               Asset: avax.Asset{ID: vm.ctx.AVAXAssetID},
-                                               In: &secp256k1fx.TransferInput{
-                                                       Amt:   1,
-                                                       Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-                                               },
-                                       }},
-                                       Outs: []EVMOutput{{
-                                               Address: testEthAddrs[0],
-                                               Amount:  1,
-                                               AssetID: vm.ctx.AVAXAssetID,
-                                       }},
-                               }}
-                               if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-                                       t.Fatal(err)
-                               }
-                               return tx
-                       },
-                       bootstrapping: true,
-               },
-               "UTXO not present": {
-                       setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-                               tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-                                       NetworkID:    vm.ctx.NetworkID,
-                                       BlockchainID: vm.ctx.ChainID,
-                                       SourceChain:  vm.ctx.XChainID,
-                                       ImportedInputs: []*avax.TransferableInput{{
-                                               UTXOID: avax.UTXOID{
-                                                       TxID: ids.GenerateTestID(),
-                                               },
-                                               Asset: avax.Asset{ID: vm.ctx.AVAXAssetID},
-                                               In: &secp256k1fx.TransferInput{
-                                                       Amt:   1,
-                                                       Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-                                               },
-                                       }},
-                                       Outs: []EVMOutput{{
-                                               Address: testEthAddrs[0],
-                                               Amount:  1,
-                                               AssetID: vm.ctx.AVAXAssetID,
-                                       }},
-                               }}
-                               if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-                                       t.Fatal(err)
-                               }
-                               return tx
-                       },
-                       semanticVerifyErr: "failed to fetch import UTXOs from",
-               },
-               "garbage UTXO": {
-                       setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-                               utxoID := avax.UTXOID{TxID: ids.GenerateTestID()}
-                               xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-                               inputID := utxoID.InputID()
-                               if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-                                       Key:   inputID[:],
-                                       Value: []byte("hey there"),
-                                       Traits: [][]byte{
-                                               testShortIDAddrs[0].Bytes(),
-                                       },
-                               }}}}); err != nil {
-                                       t.Fatal(err)
-                               }
-
-                               tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-                                       NetworkID:    vm.ctx.NetworkID,
-                                       BlockchainID: vm.ctx.ChainID,
-                                       SourceChain:  vm.ctx.XChainID,
-                                       ImportedInputs: []*avax.TransferableInput{{
-                                               UTXOID: utxoID,
-                                               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-                                               In: &secp256k1fx.TransferInput{
-                                                       Amt:   1,
-                                                       Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-                                               },
-                                       }},
-                                       Outs: []EVMOutput{{
-                                               Address: testEthAddrs[0],
-                                               Amount:  1,
-                                               AssetID: vm.ctx.AVAXAssetID,
-                                       }},
-                               }}
-                               if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-                                       t.Fatal(err)
-                               }
-                               return tx
```

```
-			},
-			semanticVerifyErr: "failed to unmarshal UTXO",
-		},
-		"UTXO AssetID mismatch": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				expectedAssetID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, expectedAssetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID}, // Use a different assetID then the actual UTXO
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  1,
-						AssetID: vm.ctx.AVAXAssetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			semanticVerifyErr: errAssetIDMismatch.Error(),
-		},
-		"insufficient AVAX funds": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  2, // Produce more output than is consumed by the transaction
-						AssetID: vm.ctx.AVAXAssetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			semanticVerifyErr: "import tx flow check failed due to",
-		},
-		"insufficient non-AVAX funds": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				assetID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, assetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: assetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  2, // Produce more output than is consumed by the transaction
-						AssetID: assetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			semanticVerifyErr: "import tx flow check failed due to",
-		},
-		"no signatures": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  1,
-						AssetID: vm.ctx.AVAXAssetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, nil); err != nil {
-					t.Fatal(err)
-				}
-				return tx
```

```
-			},
-			semanticVerifyErr: "import tx contained mismatched number of inputs/credentials",
-		},
-		"incorrect signature": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  1,
-						AssetID: vm.ctx.AVAXAssetID,
-					}},
-				}}
-				// Sign the transaction with the incorrect key
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[1]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			semanticVerifyErr: "import tx transfer failed verification",
-		},
-		"non-unique EVM Outputs": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, 2, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   2,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{
-						{
-							Address: testEthAddrs[0],
-							Amount:  1,
-							AssetID: vm.ctx.AVAXAssetID,
-						},
-						{
-							Address: testEthAddrs[0],
-							Amount:  1,
-							AssetID: vm.ctx.AVAXAssetID,
-						},
-					},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			genesisJSON:       genesisJSONApricotPhase3,
-			semanticVerifyErr: errOutputsNotSortedUnique.Error(),
-		},
-	}
-
-	for name, test := range tests {
-		t.Run(name, func(t *testing.T) {
-			executeTxTest(t, test)
-		})
-	}
-}
-
-func TestImportTxEVMStateTransfer(t *testing.T) {
-	assetID := ids.GenerateTestID()
-	tests := map[string]atomicTxTest{
-		"AVAX UTXO": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  1,
-						AssetID: vm.ctx.AVAXAssetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			checkState: func(t *testing.T, vm *VM) {
-				lastAcceptedBlock := vm.LastAcceptedBlockInternal().(*Block)
-
-				sdb, err := vm.chain.BlockState(lastAcceptedBlock.ethBlock)
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				avaxBalance := sdb.GetBalance(testEthAddrs[0])
-				if avaxBalance.Cmp(x2cRate) != 0 {
-					t.Fatalf("Expected AVAX balance to be %d, found balance: %d", x2cRate, avaxBalance)
-				}
-			},
-		},
-		"non-AVAX UTXO": {
-			setup: func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx {
-				txID := ids.GenerateTestID()
-				utxo, err := addUTXO(sharedMemory, vm.ctx, txID, assetID, 1, testShortIDAddrs[0])
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				tx := &Tx{UnsignedAtomicTx: &UnsignedImportTx{
-					NetworkID:    vm.ctx.NetworkID,
-					BlockchainID: vm.ctx.ChainID,
-					SourceChain:  vm.ctx.XChainID,
-					ImportedInputs: []*avax.TransferableInput{{
-						UTXOID: utxo.UTXOID,
-						Asset:  avax.Asset{ID: assetID},
-						In: &secp256k1fx.TransferInput{
-							Amt:   1,
-							Input: secp256k1fx.Input{SigIndices: []uint32{0}},
-						},
-					}},
-					Outs: []EVMOutput{{
-						Address: testEthAddrs[0],
-						Amount:  1,
-						AssetID: assetID,
-					}},
-				}}
-				if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{testKeys[0]}}); err != nil {
-					t.Fatal(err)
-				}
-				return tx
-			},
-			checkState: func(t *testing.T, vm *VM) {
-				lastAcceptedBlock := vm.LastAcceptedBlockInternal().(*Block)
-
-				sdb, err := vm.chain.BlockState(lastAcceptedBlock.ethBlock)
-				if err != nil {
-					t.Fatal(err)
-				}
-
-				assetBalance := sdb.GetBalanceMultiCoin(testEthAddrs[0], common.Hash(assetID))
-				if assetBalance.Cmp(common.Big1) != 0 {
-					t.Fatalf("Expected asset balance to be %d, found balance: %d", common.Big1, assetBalance)
-				}
-				avaxBalance := sdb.GetBalance(testEthAddrs[0])
-				if avaxBalance.Cmp(common.Big0) != 0 {
-					t.Fatalf("Expected AVAX balance to be 0, found balance: %d", avaxBalance)
-				}
-			},
-		},
-	}
-
-	for name, test := range tests {
-		t.Run(name, func(t *testing.T) {
-			executeTxTest(t, test)
-		})
-	}
-}
diff --git a/plugin/evm/mempool.go b/plugin/evm/mempool.go
index 53858819..9947e85e 100644
--- a/plugin/evm/mempool.go
+++ b/plugin/evm/mempool.go
@@ -8,9 +8,9 @@ import (
 	"fmt"
 	"sync"
 
-	"github.com/ava-labs/avalanchego/cache"
-	"github.com/ava-labs/avalanchego/ids"
 	"github.com/ethereum/go-ethereum/log"
+	"github.com/flare-foundation/flare/cache"
+	"github.com/flare-foundation/flare/ids"
 )
 
 const (
@@ -27,8 +27,8 @@ type Mempool struct {
 	AVAXAssetID ids.ID
 	// maxSize is the maximum number of transactions allowed to be kept in mempool
 	maxSize int
-	// currentTx is the transaction about to be added to a block.
-	currentTx *Tx
+	// currentTxs is the set of transactions about to be added to a block.
+	currentTxs map[ids.ID]*Tx
 	// issuedTxs is the set of transactions that have been issued into a new block
 	issuedTxs map[ids.ID]*Tx
 	// discardedTxs is an LRU Cache of transactions that have been discarded after failing
@@ -52,6 +52,7 @@ func NewMempool(AVAXAssetID ids.ID, maxSize int) *Mempool {
 		AVAXAssetID:  AVAXAssetID,
 		issuedTxs:    make(map[ids.ID]*Tx),
 		discardedTxs: &cache.LRU{Size: discardedTxsCacheSize},
+		currentTxs:   make(map[ids.ID]*Tx),
 		Pending:      make(chan struct{}, 1),
 		utxoSet:      ids.NewSet(maxSize),
 		txHeap:       newTxHeap(maxSize),
@@ -82,7 +83,7 @@ func (m *Mempool) has(txID ids.ID) bool {
 // atomicTxGasPrice is the [gasPrice] paid by a transaction to burn a given
 // amount of [AVAXAssetID] given the value of [gasUsed].
 func (m *Mempool) atomicTxGasPrice(tx *Tx) (uint64, error) {
-	gasUsed, err := tx.GasUsed()
+	gasUsed, err := tx.GasUsed(true)
 	if err != nil {
 		return 0, err
 	}
@@ -117,12 +118,12 @@ func (m *Mempool) ForceAddTx(tx *Tx) error {
 // If [force], skips conflict checks within the mempool.
 func (m *Mempool) addTx(tx *Tx, force bool) error {
 	txID := tx.ID()
-	// If [txID] has already been issued or is the currentTx
+	// If [txID] has already been issued or is in the currentTxs map
 	// there's no need to add it.
 	if _, exists := m.issuedTxs[txID]; exists {
 		return nil
 	}
-	if m.currentTx != nil && m.currentTx.ID() == txID {
+	if _, exists := m.currentTxs[txID]; exists {
 		return nil
 	}
 	if _, exists := m.txHeap.Get(txID); exists {
@@ -201,7 +202,7 @@ func (m *Mempool) NextTx() (*Tx, bool) {
 	// pay.
 	if m.txHeap.Len() > 0 {
 		tx := m.txHeap.PopMax()
-		m.currentTx = tx
+		m.currentTxs[tx.ID()] = tx
 		return tx, true
 	}
 
@@ -231,8 +232,8 @@ func (m *Mempool) GetTx(txID ids.ID) (*Tx, bool, bool) {
 	if tx, ok := m.issuedTxs[txID]; ok {
 		return tx, false, true
 	}
-	if m.currentTx != nil && m.currentTx.ID() == txID {
-		return m.currentTx, false, true
+	if tx, ok := m.currentTxs[txID]; ok {
+		return tx, false, true
```

```
        }
        if tx, exists := m.discardedTxs.Get(txID); exists {
                return tx.(*Tx), true, true
@@ -242,13 +243,13 @@ func (m *Mempool) GetTx(txID ids.ID) (*Tx, bool, bool) {
 }

 // IssueCurrentTx marks [currentTx] as issued if there is one
-func (m *Mempool) IssueCurrentTx() {
+func (m *Mempool) IssueCurrentTxs() {
        m.lock.Lock()
        defer m.lock.Unlock()

-       if m.currentTx != nil {
-               m.issuedTxs[m.currentTx.ID()] = m.currentTx
-               m.currentTx = nil
+       for txID := range m.currentTxs {
+               m.issuedTxs[txID] = m.currentTxs[txID]
+               delete(m.currentTxs, txID)
        }

        // If there are more transactions to be issued, add an item
@@ -258,30 +259,32 @@ func (m *Mempool) IssueCurrentTx() {
        }
 }

-// CancelCurrentTx marks the attempt to issue [currentTx]
+// CancelCurrentTx marks the attempt to issue [txID]
+// as being aborted. This should be called after NextTx returns [txID]
+// and the transaction [txID] cannot be included in the block, but should
+// not be discarded. For example, CancelCurrentTx should be called if including
+// the transaction will put the block above the atomic tx gas limit.
+func (m *Mempool) CancelCurrentTx(txID ids.ID) {
+       m.lock.Lock()
+       defer m.lock.Unlock()
+
+       if tx, ok := m.currentTxs[txID]; ok {
+               m.cancelTx(tx)
+       }
+}
+
+// [CancelCurrentTxs] marks the attempt to issue [currentTxs]
 // as being aborted. If this is called after a buildBlock error
 // caused by the atomic transaction, then DiscardCurrentTx should have been called
 // such that this call will have no effect and should not re-issue the invalid tx.
-func (m *Mempool) CancelCurrentTx() {
+func (m *Mempool) CancelCurrentTxs() {
        m.lock.Lock()
        defer m.lock.Unlock()

        // If building a block failed, put the currentTx back in [txs]
        // if it exists.
-       if m.currentTx != nil {
-               // Add tx to heap sorted by gasPrice
-               tx := m.currentTx
-               gasPrice, err := m.atomicTxGasPrice(tx)
-               if err == nil {
-                       m.txHeap.Push(tx, gasPrice)
-               } else {
-                       log.Error("failed to calculate atomic tx gas price while canceling current tx", "err", err)
-                       m.utxoSet.Remove(tx.InputUTXOs().List()...)
-                       m.discardedTxs.Put(tx.ID(), tx)
-               }
-               // If the err is not nil, we simply discard the transaction because it is
-               // invalid. This should never happen but we guard against the case it does.
-               m.currentTx = nil
+       for _, tx := range m.currentTxs {
+               m.cancelTx(tx)
        }

        // If there are more transactions to be issued, add an item
@@ -291,20 +294,52 @@ func (m *Mempool) CancelCurrentTx() {
        }
 }

-// DiscardCurrentTx marks [currentTx] as invalid and aborts the attempt
+// cancelTx removes [tx] from current transactions and moves it back into the
+// tx heap.
+// assumes the lock is held.
+func (m *Mempool) cancelTx(tx *Tx) {
+       // Add tx to heap sorted by gasPrice
+       gasPrice, err := m.atomicTxGasPrice(tx)
+       if err == nil {
+               m.txHeap.Push(tx, gasPrice)
+       } else {
+               // If the err is not nil, we simply discard the transaction because it is
+               // invalid. This should never happen but we guard against the case it does.
+               log.Error("failed to calculate atomic tx gas price while canceling current tx", "err", err)
+               m.utxoSet.Remove(tx.InputUTXOs().List()...)
+               m.discardedTxs.Put(tx.ID(), tx)
+       }
+
+       delete(m.currentTxs, tx.ID())
+}
+
+// DiscardCurrentTx marks a [tx] in the [currentTxs] map as invalid and aborts the attempt
 // to issue it since it failed verification.
-// Adding to Pending should be handled by CancelCurrentTx in this case.
-func (m *Mempool) DiscardCurrentTx() {
+func (m *Mempool) DiscardCurrentTx(txID ids.ID) {
+       m.lock.Lock()
+       defer m.lock.Unlock()
+
+       if tx, ok := m.currentTxs[txID]; ok {
+               m.discardCurrentTx(tx)
+       }
+}
+
+// DiscardCurrentTxs marks all txs in [currentTxs] as discarded.
+func (m *Mempool) DiscardCurrentTxs() {
        m.lock.Lock()
        defer m.lock.Unlock()

-       if m.currentTx == nil {
-               return
+       for _, tx := range m.currentTxs {
+               m.discardCurrentTx(tx)
        }
+}

-       m.utxoSet.Remove(m.currentTx.InputUTXOs().List()...)
-       m.discardedTxs.Put(m.currentTx.ID(), m.currentTx)
-       m.currentTx = nil
+// discardCurrentTx discards [tx] from the set of current transactions.
+// Assumes the lock is held.
+func (m *Mempool) discardCurrentTx(tx *Tx) {
+       m.utxoSet.Remove(tx.InputUTXOs().List()...)
+       m.discardedTxs.Put(tx.ID(), tx)
+       delete(m.currentTxs, tx.ID())
 }

 // RemoveTx removes [txID] from the mempool completely.
@@ -313,9 +348,9 @@ func (m *Mempool) RemoveTx(txID ids.ID) {
        defer m.lock.Unlock()
        var removedTx *Tx
-       if m.currentTx != nil && m.currentTx.ID() == txID {
-               removedTx = m.currentTx
-               m.currentTx = nil
+       if tx, ok := m.currentTxs[txID]; ok {
+               removedTx = tx
+               delete(m.currentTxs, txID)
        }
        if tx, ok := m.txHeap.Get(txID); ok {
                removedTx = tx
diff --git a/plugin/evm/mempool_atomic_gossiping_test.go b/plugin/evm/mempool_atomic_gossiping_test.go
index 43c0f9b7..392f64b9 100644
--- a/plugin/evm/mempool_atomic_gossiping_test.go
+++ b/plugin/evm/mempool_atomic_gossiping_test.go
@@ -6,13 +6,13 @@ package evm
 import (
        "testing"

-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"

-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/components/chain"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/vms/components/avax"
+       "github.com/flare-foundation/flare/vms/components/chain"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"

        "github.com/stretchr/testify/assert"
 )
@@ -71,9 +71,7 @@ func TestMempoolAddLocallyCreateAtomicTx(t *testing.T) {
                        evmBlk, ok := blk.(*chain.BlockWrapper).Block.(*Block)
                        assert.True(ok, "unknown block type")

-                       retrievedTx, err := vm.extractAtomicTx(evmBlk.ethBlock)
-                       assert.NoError(err, "could not extract atomic tx")
-                       assert.Equal(txID, retrievedTx.ID(), "block does not include expected transaction")
+                       assert.Equal(txID, evmBlk.atomicTxs[0].ID(), "block does not include expected transaction")

                        has = mempool.has(txID)
                        assert.True(has, "tx should stay in mempool until block is accepted")
diff --git a/plugin/evm/message/codec.go b/plugin/evm/message/codec.go
index 8c172740..0e3f395d 100644
--- a/plugin/evm/message/codec.go
+++ b/plugin/evm/message/codec.go
@@ -1,14 +1,19 @@
+<<<<<<< HEAD
 // (c) 2019-2021, Ava Labs, Inc. All rights reserved.
+=======
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+>>>>>>> upstream-v0.8.5-rc.2
 // See the file LICENSE for licensing terms.

 package message

 import (
-       "github.com/ava-labs/avalanchego/codec"
-       "github.com/ava-labs/avalanchego/codec/linearcodec"
-       "github.com/ava-labs/avalanchego/codec/reflectcodec"
-       "github.com/ava-labs/avalanchego/utils/units"
-       "github.com/ava-labs/avalanchego/utils/wrappers"
+<<<<<<< HEAD
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/codec/linearcodec"
+       "github.com/flare-foundation/flare/codec/reflectcodec"
+       "github.com/flare-foundation/flare/utils/units"
+       "github.com/flare-foundation/flare/utils/wrappers"
 )

 const (
@@ -33,4 +38,25 @@ func init() {
        if errs.Errored() {
                panic(errs.Err)
        }
+=======
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/codec/linearcodec"
+       "github.com/flare-foundation/flare/utils/units"
+       "github.com/flare-foundation/flare/utils/wrappers"
+)
+
+const Version = uint16(0)
+const maxMessageSize = 1 * units.MiB
+
+func BuildCodec() (codec.Manager, error) {
+       codecManager := codec.NewManager(maxMessageSize)
+       c := linearcodec.NewDefault()
+       errs := wrappers.Errs{}
+       errs.Add(
+               c.RegisterType(&AtomicTx{}),
+               c.RegisterType(&EthTxs{}),
+       )
+       errs.Add(codecManager.RegisterCodec(Version, c))
+       return codecManager, errs.Err
+>>>>>>> upstream-v0.8.5-rc.2
 }
diff --git a/plugin/evm/message/handler.go b/plugin/evm/message/handler.go
index bc883200..845dcd4a 100644
--- a/plugin/evm/message/handler.go
+++ b/plugin/evm/message/handler.go
@@ -6,7 +6,8 @@ package message
 import (
        "github.com/ethereum/go-ethereum/log"

-       "github.com/ava-labs/avalanchego/ids"
+<<<<<<< HEAD
+       "github.com/flare-foundation/flare/ids"
 )

 var _ Handler = NoopHandler{}
@@ -27,3 +28,44 @@ func (NoopHandler) HandleEthTxs(nodeID ids.ShortID, requestID uint32, _ *EthTxs)
        log.Debug("dropping unexpected EthTxs message", "peerID", nodeID, "requestID", requestID)
        return nil
 }
+=======
+       "github.com/flare-foundation/flare/ids"
+)
+
+var _ GossipHandler = NoopMempoolGossipHandler{}
+
+// GossipHandler handles incoming gossip messages
+type GossipHandler interface {
+       HandleAtomicTx(nodeID ids.ShortID, msg *AtomicTx) error
+       HandleEthTxs(nodeID ids.ShortID, msg *EthTxs) error
+}
+
+type NoopMempoolGossipHandler struct{}
+
+func (NoopMempoolGossipHandler) HandleAtomicTx(nodeID ids.ShortID, _ *AtomicTx) error {
+       log.Debug("dropping unexpected AtomicTx message", "peerID", nodeID)
+       return nil
+}
+
+func (NoopMempoolGossipHandler) HandleEthTxs(nodeID ids.ShortID, _ *EthTxs) error {
+       log.Debug("dropping unexpected EthTxs message", "peerID", nodeID)
+       return nil
+}
+
+// RequestHandler interface handles incoming requests from peers
+// Must have methods in format of handleType(context.Context, ids.ShortID, uint32, request Type) error
+// so that the Request object of relevant Type can invoke its respective handle method
+// on this struct.
+// Also see GossipHandler for implementation style.
+type RequestHandler interface{}
+
+// ResponseHandler handles response for a sent request
+// Only one of OnResponse or OnFailure is called for a given requestID, not both
+type ResponseHandler interface {
+       // OnResponse is invoked when the peer responded to a request
+       OnResponse(nodeID ids.ShortID, requestID uint32, response []byte) error
+       // OnFailure is invoked when there was a failure in processing a request
+       // The FailureReason outlines the underlying cause.
+       OnFailure(nodeID ids.ShortID, requestID uint32) error
+}
+>>>>>>> upstream-v0.8.5-rc.2
diff --git a/plugin/evm/message/handler_test.go b/plugin/evm/message/handler_test.go
index 179ac56e..097f037e 100644
--- a/plugin/evm/message/handler_test.go
+++ b/plugin/evm/message/handler_test.go
@@ -6,21 +6,35 @@ package message
 import (
        "testing"

-       "github.com/ava-labs/avalanchego/ids"
+<<<<<<< HEAD
+       "github.com/stretchr/testify/assert"
+
+       "github.com/flare-foundation/flare/ids"
+=======
+       "github.com/flare-foundation/flare/ids"

        "github.com/stretchr/testify/assert"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 type CounterHandler struct {
        AtomicTx, EthTxs int
 }

+<<<<<<< HEAD
 func (h *CounterHandler) HandleAtomicTx(ids.ShortID, uint32, *AtomicTx) error {
+=======
+func (h *CounterHandler) HandleAtomicTx(ids.ShortID, *AtomicTx) error {
+>>>>>>> upstream-v0.8.5-rc.2
        h.AtomicTx++
        return nil
 }

+<<<<<<< HEAD
 func (h *CounterHandler) HandleEthTxs(ids.ShortID, uint32, *EthTxs) error {
+=======
+func (h *CounterHandler) HandleEthTxs(ids.ShortID, *EthTxs) error {
+>>>>>>> upstream-v0.8.5-rc.2
        h.EthTxs++
        return nil
 }
@@ -31,7 +45,11 @@ func TestHandleAtomicTx(t *testing.T) {
        handler := CounterHandler{}
        msg := AtomicTx{}

+<<<<<<< HEAD
        err := msg.Handle(&handler, ids.ShortEmpty, 0)
+=======
+       err := msg.Handle(&handler, ids.ShortEmpty)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.NoError(err)
        assert.Equal(1, handler.AtomicTx)
        assert.Zero(handler.EthTxs)
@@ -43,7 +61,11 @@ func TestHandleEthTxs(t *testing.T) {
        handler := CounterHandler{}
        msg := EthTxs{}

+<<<<<<< HEAD
        err := msg.Handle(&handler, ids.ShortEmpty, 0)
+=======
+       err := msg.Handle(&handler, ids.ShortEmpty)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.NoError(err)
        assert.Zero(handler.AtomicTx)
        assert.Equal(1, handler.EthTxs)
@@ -52,11 +74,20 @@ func TestHandleEthTxs(t *testing.T) {
 func TestNoopHandler(t *testing.T) {
        assert := assert.New(t)

+<<<<<<< HEAD
        handler := NoopHandler{}

        err := handler.HandleAtomicTx(ids.ShortEmpty, 0, nil)
        assert.NoError(err)

        err = handler.HandleEthTxs(ids.ShortEmpty, 0, nil)
+=======
+       handler := NoopMempoolGossipHandler{}
+
+       err := handler.HandleAtomicTx(ids.ShortEmpty, nil)
+       assert.NoError(err)
+
+       err = handler.HandleEthTxs(ids.ShortEmpty, nil)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.NoError(err)
 }
diff --git a/plugin/evm/message/message.go b/plugin/evm/message/message.go
index 082adbe5..4286bd03 100644
--- a/plugin/evm/message/message.go
+++ b/plugin/evm/message/message.go
@@ -6,10 +6,19 @@ package message
 import (
        "errors"

+<<<<<<< HEAD
        "github.com/ethereum/go-ethereum/common"

-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/units"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/units"
+=======
+       "github.com/flare-foundation/flare/codec"
+
+       "github.com/ethereum/go-ethereum/common"
+
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/units"
+>>>>>>> upstream-v0.8.5-rc.2
```

```
 )

 const (
@@ -18,6 +27,11 @@ const (
        // this size, however. Max inbound message size is enforced by the codec
        // (512KB).
        EthMsgSoftCapSize = common.StorageSize(64 * units.KiB)
+<<<<<<< HEAD
+=======
+       atomicTxType      = "atomic-tx"
+       ethTxsType        = "eth-txs"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 var (
@@ -29,7 +43,11 @@ var (

 type Message interface {
        // Handle this message with the correct message handler
+<<<<<<< HEAD
        Handle(handler Handler, nodeID ids.ShortID, requestID uint32) error
+=======
+       Handle(handler GossipHandler, nodeID ids.ShortID) error
+>>>>>>> upstream-v0.8.5-rc.2

        // initialize should be called whenever a message is built or parsed
        initialize([]byte)
@@ -38,6 +56,12 @@ type Message interface {
        //
        // Bytes should only be called after being initialized
        Bytes() []byte
+<<<<<<< HEAD
+=======
+
+       // Type returns user-friendly name for this object that can be used for logging
+       Type() string
+>>>>>>> upstream-v0.8.5-rc.2
 }

 type message []byte
@@ -51,8 +75,17 @@ type AtomicTx struct {
        Tx []byte `serialize:"true"`
 }

+<<<<<<< HEAD
 func (msg *AtomicTx) Handle(handler Handler, nodeID ids.ShortID, requestID uint32) error {
        return handler.HandleAtomicTx(nodeID, requestID, msg)
+=======
+func (msg *AtomicTx) Handle(handler GossipHandler, nodeID ids.ShortID) error {
+       return handler.HandleAtomicTx(nodeID, msg)
+}
+
+func (msg *AtomicTx) Type() string {
+       return atomicTxType
+>>>>>>> upstream-v0.8.5-rc.2
 }

 type EthTxs struct {
@@ -61,6 +94,7 @@ type EthTxs struct {
        Txs []byte `serialize:"true"`
 }

+<<<<<<< HEAD
 func (msg *EthTxs) Handle(handler Handler, nodeID ids.ShortID, requestID uint32) error {
        return handler.HandleEthTxs(nodeID, requestID, msg)
 }
@@ -72,14 +106,36 @@ func Parse(bytes []byte) (Message, error) {
                return nil, err
        }
        if version != codecVersion {
+=======
+func (msg *EthTxs) Handle(handler GossipHandler, nodeID ids.ShortID) error {
+       return handler.HandleEthTxs(nodeID, msg)
+}
+
+func (msg *EthTxs) Type() string {
+       return ethTxsType
+}
+
+func ParseMessage(codec codec.Manager, bytes []byte) (Message, error) {
+       var msg Message
+       version, err := codec.Unmarshal(bytes, &msg)
+       if err != nil {
+               return nil, err
+       }
+       if version != Version {
+>>>>>>> upstream-v0.8.5-rc.2
                return nil, errUnexpectedCodecVersion
        }
        msg.initialize(bytes)
        return msg, nil
 }

+<<<<<<< HEAD
 func Build(msg Message) ([]byte, error) {
        bytes, err := c.Marshal(codecVersion, &msg)
+=======
+func BuildMessage(codec codec.Manager, msg Message) ([]byte, error) {
+       bytes, err := codec.Marshal(Version, &msg)
+>>>>>>> upstream-v0.8.5-rc.2
        msg.initialize(bytes)
        return bytes, err
 }
diff --git a/plugin/evm/message/message_test.go b/plugin/evm/message/message_test.go
index 5575e253..0c09bab0 100644
--- a/plugin/evm/message/message_test.go
+++ b/plugin/evm/message/message_test.go
@@ -6,10 +6,17 @@ package message
 import (
        "testing"

-       "github.com/ava-labs/avalanchego/utils"
-       "github.com/ava-labs/avalanchego/utils/units"
+<<<<<<< HEAD
+       "github.com/stretchr/testify/assert"
+
+       "github.com/flare-foundation/flare/utils"
+       "github.com/flare-foundation/flare/utils/units"
+=======
+       "github.com/flare-foundation/flare/utils"
+       "github.com/flare-foundation/flare/utils/units"

        "github.com/stretchr/testify/assert"
+>>>>>>> upstream-v0.8.5-rc.2
 )

 func TestAtomicTx(t *testing.T) {
@@ -19,11 +26,21 @@ func TestAtomicTx(t *testing.T) {
        builtMsg := AtomicTx{
                Tx: msg,
        }
+<<<<<<< HEAD
        builtMsgBytes, err := Build(&builtMsg)
        assert.NoError(err)
        assert.Equal(builtMsgBytes, builtMsg.Bytes())

        parsedMsgIntf, err := Parse(builtMsgBytes)
+=======
+       codec, err := BuildCodec()
+       assert.NoError(err)
+       builtMsgBytes, err := BuildMessage(codec, &builtMsg)
+       assert.NoError(err)
+       assert.Equal(builtMsgBytes, builtMsg.Bytes())
+
+       parsedMsgIntf, err := ParseMessage(codec, builtMsgBytes)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.NoError(err)
        assert.Equal(builtMsgBytes, parsedMsgIntf.Bytes())

@@ -40,11 +57,21 @@ func TestEthTxs(t *testing.T) {
        builtMsg := EthTxs{
                Txs: msg,
        }
+<<<<<<< HEAD
        builtMsgBytes, err := Build(&builtMsg)
        assert.NoError(err)
        assert.Equal(builtMsgBytes, builtMsg.Bytes())

        parsedMsgIntf, err := Parse(builtMsgBytes)
+=======
+       codec, err := BuildCodec()
+       assert.NoError(err)
+       builtMsgBytes, err := BuildMessage(codec, &builtMsg)
+       assert.NoError(err)
+       assert.Equal(builtMsgBytes, builtMsg.Bytes())
+
+       parsedMsgIntf, err := ParseMessage(codec, builtMsgBytes)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.NoError(err)
        assert.Equal(builtMsgBytes, parsedMsgIntf.Bytes())

@@ -60,14 +87,27 @@ func TestEthTxsTooLarge(t *testing.T) {
        builtMsg := EthTxs{
                Txs: utils.RandomBytes(1024 * units.KiB),
        }
+<<<<<<< HEAD
        _, err := Build(&builtMsg)
+=======
+       codec, err := BuildCodec()
+       assert.NoError(err)
+       _, err = BuildMessage(codec, &builtMsg)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.Error(err)
 }

 func TestParseGibberish(t *testing.T) {
        assert := assert.New(t)

+<<<<<<< HEAD
        randomBytes := utils.RandomBytes(256 * units.KiB)
        _, err := Parse(randomBytes)
+=======
+       codec, err := BuildCodec()
+       assert.NoError(err)
+       randomBytes := utils.RandomBytes(256 * units.KiB)
+       _, err = ParseMessage(codec, randomBytes)
+>>>>>>> upstream-v0.8.5-rc.2
        assert.Error(err)
 }
diff --git a/plugin/evm/message/request.go b/plugin/evm/message/request.go
new file mode 100644
index 00000000..929e38eb
--- /dev/null
+++ b/plugin/evm/message/request.go
@@ -0,0 +1,36 @@
+// (c) 2019-2022, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package message
+
+import (
+       "context"
+
+       "github.com/flare-foundation/flare/codec"
+
+       "github.com/flare-foundation/flare/ids"
+)
+
+// Request represents a Network request type
+type Request interface {
+       // Handle allows `Request` to call respective methods on handler to handle
+       // this particular request type
+       Handle(ctx context.Context, nodeID ids.ShortID, requestID uint32, handler RequestHandler) ([]byte, error)
+
+       // Type returns user-friendly name for this object that can be used for logging
+       Type() string
+}
+
+// BytesToRequest unmarshals the given requestBytes into Request object
+func BytesToRequest(codec codec.Manager, requestBytes []byte) (Request, error) {
+       var request Request
+       if _, err := codec.Unmarshal(requestBytes, &request); err != nil {
+               return nil, err
+       }
+       return request, nil
+}
+
+// RequestToBytes marshals the given request object into bytes
+func RequestToBytes(codec codec.Manager, request Request) ([]byte, error) {
+       return codec.Marshal(Version, &request)
+}
diff --git a/plugin/evm/network.go b/plugin/evm/network.go
index e3690d62..d5c9d60a 100644
--- a/plugin/evm/network.go
+++ b/plugin/evm/network.go
@@ -9,23 +9,20 @@ import (
        "sync"
        "time"

-       "github.com/ava-labs/avalanchego/cache"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/utils/wrappers"
-
-       commonEng "github.com/ava-labs/avalanchego/snow/engine/common"
-
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/plugin/evm/message"
-
-       coreth "github.com/ava-labs/coreth/chain"
+       coreth "github.com/flare-foundation/coreth/chain"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/plugin/evm/message"
+       "github.com/flare-foundation/flare/cache"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       commonEng "github.com/flare-foundation/flare/snow/engine/common"
+       "github.com/flare-foundation/flare/utils/wrappers"
 )

 const (
diff --git a/plugin/evm/network_eth_gossiping_test.go b/plugin/evm/network_eth_gossiping_test.go
index 54a2011e..0f1559dc 100644
--- a/plugin/evm/network_eth_gossiping_test.go
+++ b/plugin/evm/network_eth_gossiping_test.go
@@ -12,18 +12,17 @@ import (
        "testing"
        "time"

-       "github.com/ava-labs/avalanchego/ids"
+       "github.com/stretchr/testify/assert"

        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/rlp"

-       "github.com/stretchr/testify/assert"
-
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/plugin/evm/message"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/plugin/evm/message"
+       "github.com/flare-foundation/flare/ids"
 )

 func fundAddressByGenesis(addrs []common.Address) (string, error) {
@@ -41,11 +40,12 @@ func fundAddressByGenesis(addrs []common.Address) (string, error) {
        genesis.Alloc = funds

        genesis.Config = &params.ChainConfig{
-               ChainID:                     params.AvalancheLocalChainID,
+               ChainID:                     big.NewInt(31337),
                ApricotPhase1BlockTimestamp: big.NewInt(0),
                ApricotPhase2BlockTimestamp: big.NewInt(0),
                ApricotPhase3BlockTimestamp: big.NewInt(0),
                ApricotPhase4BlockTimestamp: big.NewInt(0),
+               ApricotPhase5BlockTimestamp: big.NewInt(0),
        }

        bytes, err := json.Marshal(genesis)
diff --git a/plugin/evm/service.go b/plugin/evm/service.go
index 0dcc7336..435dec91 100644
--- a/plugin/evm/service.go
+++ b/plugin/evm/service.go
@@ -11,17 +11,16 @@ import (
        "net/http"
        "strings"

-       "github.com/ava-labs/avalanchego/api"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/constants"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/formatting"
-       "github.com/ava-labs/avalanchego/utils/json"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
-       ethcrypto "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/flare/api"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/constants"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/utils/formatting"
+       "github.com/flare-foundation/flare/utils/json"
 )

 // test constants
@@ -41,27 +40,6 @@ var (
        initialBaseFee = big.NewInt(params.ApricotPhase3InitialBaseFee)
 )

-// NetAPI offers network related API methods
-type NetAPI struct{ vm *VM }
-
-// Listening returns an indication if the node is listening for network connections.
-func (s *NetAPI) Listening() bool { return true } // always listening
-
-// PeerCount returns the number of connected peers
-func (s *NetAPI) PeerCount() hexutil.Uint { return hexutil.Uint(0) }
-
-// Version returns the current ethereum protocol version.
-func (s *NetAPI) Version() string { return fmt.Sprintf("%d", s.vm.networkID) }
-
-// Web3API offers helper API methods
-type Web3API struct{}
-
-// ClientVersion returns the version of the vm running
-func (s *Web3API) ClientVersion() string { return Version }
-
-// Sha3 returns the bytes returned by hashing [input] with Keccak256
-func (s *Web3API) Sha3(input hexutil.Bytes) hexutil.Bytes { return ethcrypto.Keccak256(input) }
-
 // SnowmanAPI introduces snowman specific functionality to the evm
 type SnowmanAPI struct{ vm *VM }

diff --git a/plugin/evm/fuji_ext_data_hashes.json b/plugin/evm/songbird_ext_data_hashes.json
similarity index 100%
rename from plugin/evm/fuji_ext_data_hashes.json
rename to plugin/evm/songbird_ext_data_hashes.json
diff --git a/plugin/evm/static_service.go b/plugin/evm/static_service.go
index 9c592251..e1db747e 100644
--- a/plugin/evm/static_service.go
+++ b/plugin/evm/static_service.go
@@ -7,8 +7,8 @@ import (
        "context"
        "encoding/json"

-       "github.com/ava-labs/avalanchego/utils/formatting"
-       "github.com/ava-labs/coreth/core"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/flare/utils/formatting"
 )

 // StaticService defines the static API services exposed by the evm
diff --git a/plugin/evm/test_tx.go b/plugin/evm/test_tx.go
new file mode 100644
index 00000000..878610d4
--- /dev/null
```

```
+++ b/plugin/evm/test_tx.go
@@ -0,0 +1,152 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
+package evm
+
+import (
+       "math/big"
+       "math/rand"
+
+       "github.com/flare-foundation/flare/utils"
+
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/codec/linearcodec"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/utils/wrappers"
+)
+
+type TestTx struct {
+       GasUsedV                    uint64           `serialize:"true"`
+       AcceptRequestsBlockchainIDV ids.ID           `serialize:"true"`
+       AcceptRequestsV             *atomic.Requests `serialize:"true"`
+       VerifyV                     error
+       IDV                         ids.ID `serialize:"true" json:"id"`
+       BurnedV                     uint64 `serialize:"true"`
+       UnsignedBytesV              []byte
+       BytesV                      []byte
+       InputUTXOsV                 ids.Set
+       SemanticVerifyV             error
+       EVMStateTransferV           error
+}
+
+var _ UnsignedAtomicTx = &TestTx{}
+
+// GasUsed implements the UnsignedAtomicTx interface
+func (t *TestTx) GasUsed(fixedFee bool) (uint64, error) { return t.GasUsedV, nil }
+
+// Verify implements the UnsignedAtomicTx interface
+func (t *TestTx) Verify(ctx *snow.Context, rules params.Rules) error { return t.VerifyV }
+
+// AtomicOps implements the UnsignedAtomicTx interface
+func (t *TestTx) AtomicOps() (ids.ID, *atomic.Requests, error) {
+       return t.AcceptRequestsBlockchainIDV, t.AcceptRequestsV, nil
+}
+
+// Initialize implements the UnsignedAtomicTx interface
+func (t *TestTx) Initialize(unsignedBytes, signedBytes []byte) {}
+
+// ID implements the UnsignedAtomicTx interface
+func (t *TestTx) ID() ids.ID { return t.IDV }
+
+// Burned implements the UnsignedAtomicTx interface
+func (t *TestTx) Burned(assetID ids.ID) (uint64, error) { return t.BurnedV, nil }
+
+// UnsignedBytes implements the UnsignedAtomicTx interface
+func (t *TestTx) UnsignedBytes() []byte { return t.UnsignedBytesV }
+
+// Bytes implements the UnsignedAtomicTx interface
+func (t *TestTx) Bytes() []byte { return t.BytesV }
+
+// InputUTXOs implements the UnsignedAtomicTx interface
+func (t *TestTx) InputUTXOs() ids.Set { return t.InputUTXOsV }
+
+// SemanticVerify implements the UnsignedAtomicTx interface
+func (t *TestTx) SemanticVerify(vm *VM, stx *Tx, parent *Block, baseFee *big.Int, rules params.Rules) error {
+       return t.SemanticVerifyV
+}
+
+// EVMStateTransfer implements the UnsignedAtomicTx interface
+func (t *TestTx) EVMStateTransfer(ctx *snow.Context, state *state.StateDB) error {
+       return t.EVMStateTransferV
+}
+
+func testTxCodec() codec.Manager {
+       codec := codec.NewDefaultManager()
+       c := linearcodec.NewDefault()
+
+       errs := wrappers.Errs{}
+       errs.Add(
+               c.RegisterType(&TestTx{}),
+               c.RegisterType(&atomic.Element{}),
+               c.RegisterType(&atomic.Requests{}),
+               codec.RegisterCodec(codecVersion, c),
+       )
+
+       if errs.Errored() {
+               panic(errs.Err)
+       }
+       return codec
+}
+
+var blockChainID = ids.GenerateTestID()
+
+func testDataImportTx() *Tx {
+       return &Tx{
+               UnsignedAtomicTx: &TestTx{
+                       IDV:                         ids.GenerateTestID(),
+                       AcceptRequestsBlockchainIDV: blockChainID,
+                       AcceptRequestsV: &atomic.Requests{
+                               RemoveRequests: [][]byte{
+                                       utils.RandomBytes(32),
+                                       utils.RandomBytes(32),
+                               },
+                       },
+               },
+       }
+}
+
+func testDataExportTx() *Tx {
+       return &Tx{
+               UnsignedAtomicTx: &TestTx{
+                       IDV:                         ids.GenerateTestID(),
+                       AcceptRequestsBlockchainIDV: blockChainID,
+                       AcceptRequestsV: &atomic.Requests{
+                               PutRequests: []*atomic.Element{
+                                       {
+                                               Key:   utils.RandomBytes(16),
+                                               Value: utils.RandomBytes(24),
+                                               Traits: [][]byte{
+                                                       utils.RandomBytes(32),
+                                                       utils.RandomBytes(32),
+                                               },
+                                       },
+                               },
+                       },
+               },
+       }
+}
+
+func newTestTx() *Tx {
+       txType := rand.Intn(2)
+       switch txType {
+       case 0:
+               return testDataImportTx()
+       case 1:
+               return testDataExportTx()
+       default:
+               panic("rng generated unexpected value for tx type")
+       }
+}
+
+func newTestTxs(numTxs int) []*Tx {
+       txs := make([]*Tx, 0, numTxs)
+       for i := 0; i < numTxs; i++ {
+               txs = append(txs, newTestTx())
+       }
+
+       return txs
+}
diff --git a/plugin/evm/tx.go b/plugin/evm/tx.go
index 3f0ad923..4f080f57 100644
--- a/plugin/evm/tx.go
+++ b/plugin/evm/tx.go
@@ -12,19 +12,19 @@ import (

        "github.com/ethereum/go-ethereum/common"

-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/params"
-
-       "github.com/ava-labs/avalanchego/codec"
-       "github.com/ava-labs/avalanchego/database"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/utils"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/hashing"
-       "github.com/ava-labs/avalanchego/utils/wrappers"
-       "github.com/ava-labs/avalanchego/vms/components/verify"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/params"
+
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/utils"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/utils/hashing"
+       "github.com/flare-foundation/flare/utils/wrappers"
+       "github.com/flare-foundation/flare/vms/components/verify"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"
 )

 var (
@@ -92,7 +92,7 @@ func (in *EVMInput) Verify() error {
 type UnsignedTx interface {
        Initialize(unsignedBytes, signedBytes []byte)
        ID() ids.ID
-       GasUsed() (uint64, error)
+       GasUsed(fixedFee bool) (uint64, error)
        Burned(assetID ids.ID) (uint64, error)
        UnsignedBytes() []byte
        Bytes() []byte
@@ -102,16 +102,16 @@ type UnsignedTx interface {
 type UnsignedAtomicTx interface {
        UnsignedTx

-       // UTXOs this tx consumes
+       // InputUTXOs returns the UTXOs this tx consumes
        InputUTXOs() ids.Set
        // Verify attempts to verify that the transaction is well formed
-       // TODO: remove [xChainID] parameter since this is provided on [ctx]
-       Verify(xChainID ids.ID, ctx *snow.Context, rules params.Rules) error
+       Verify(ctx *snow.Context, rules params.Rules) error
        // Attempts to verify this transaction with the provided state.
        SemanticVerify(vm *VM, stx *Tx, parent *Block, baseFee *big.Int, rules params.Rules) error
-
-       // Accept this transaction with the additionally provided state transitions.
-       Accept(ctx *snow.Context, batch database.Batch) error
+       // AtomicOps returns the blockchainID and set of atomic requests that
+       // must be applied to shared memory for this transaction to be accepted.
+       // The set of atomic requests must be returned in a consistent order.
+       AtomicOps() (ids.ID, *atomic.Requests, error)

        EVMStateTransfer(ctx *snow.Context, state *state.StateDB) error
 }
@@ -160,14 +160,14 @@ func (tx *Tx) Sign(c codec.Manager, signers [][]*crypto.PrivateKeySECP256K1R) er
 // for via this transaction denominated in [avaxAssetID] with [baseFee] used to calculate the
 // cost of this transaction. This function also returns the [gasUsed] by the
 // transaction for inclusion in the [baseFee] algorithm.
-func (tx *Tx) BlockFeeContribution(avaxAssetID ids.ID, baseFee *big.Int) (*big.Int, *big.Int, error) {
+func (tx *Tx) BlockFeeContribution(fixedFee bool, avaxAssetID ids.ID, baseFee *big.Int) (*big.Int, *big.Int, error) {
        if baseFee == nil {
                return nil, nil, errNilBaseFee
        }
        if baseFee.Cmp(common.Big0) <= 0 {
                return nil, nil, fmt.Errorf("cannot calculate tip with base fee %d <= 0", baseFee)
        }
-       gasUsed, err := tx.GasUsed()
+       gasUsed, err := tx.GasUsed(fixedFee)
        if err != nil {
                return nil, nil, err
        }
@@ -279,3 +279,30 @@ func calculateDynamicFee(cost uint64, baseFee *big.Int) (uint64, error) {
 func calcBytesCost(len int) uint64 {
        return uint64(len) * TxBytesGas
 }
+
+// mergeAtomicOps merges atomic requests represented by [txs]
+// to the [output] map, depending on whether [chainID] is present in the map.
+func mergeAtomicOps(txs []*Tx) (map[ids.ID]*atomic.Requests, error) {
+       if len(txs) > 1 {
+               // txs should be stored in order of txID to ensure consistency
+               // with txs initialized from the txID index.
+               copyTxs := make([]*Tx, len(txs))
+               copy(copyTxs, txs)
+               sort.Slice(copyTxs, func(i, j int) bool { return copyTxs[i].ID().Hex() < copyTxs[j].ID().Hex() })
+               txs = copyTxs
+       }
+       output := make(map[ids.ID]*atomic.Requests)
+       for _, tx := range txs {
+               chainID, txRequest, err := tx.UnsignedAtomicTx.AtomicOps()
+               if err != nil {
+                       return nil, err
+               }
+               if request, exists := output[chainID]; exists {
+                       request.PutRequests = append(request.PutRequests, txRequest.PutRequests...)
+                       request.RemoveRequests = append(request.RemoveRequests, txRequest.RemoveRequests...)
+               } else {
+                       output[chainID] = txRequest
+               }
+       }
+       return output, nil
+}
diff --git a/plugin/evm/tx_heap.go b/plugin/evm/tx_heap.go
index 11c1319c..d5731d1b 100644
--- a/plugin/evm/tx_heap.go
+++ b/plugin/evm/tx_heap.go
@@ -1,9 +1,12 @@
+// (c) 2020-2021, Ava Labs, Inc. All rights reserved.
+// See the file LICENSE for licensing terms.
+
 package evm
 
 import (
        "container/heap"
 
-       "github.com/ava-labs/avalanchego/ids"
+       "github.com/flare-foundation/flare/ids"
 )
 
 // txEntry is used to track the [gasPrice] transactions pay to be included in
diff --git a/plugin/evm/tx_test.go b/plugin/evm/tx_test.go
index e0482e2d..6833e567 100644
--- a/plugin/evm/tx_test.go
+++ b/plugin/evm/tx_test.go
@@ -8,9 +8,9 @@ import (
        "strings"
        "testing"
 
-       "github.com/ava-labs/avalanchego/chains/atomic"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/snow"
 )
 
 func TestCalculateDynamicFee(t *testing.T) {
@@ -49,131 +49,3 @@ func TestCalculateDynamicFee(t *testing.T) {
                }
        }
 }
-
-type atomicTxVerifyTest struct {
-       ctx         *snow.Context
-       generate    func(t *testing.T) UnsignedAtomicTx
-       rules       params.Rules
-       expectedErr string
-}
-
-// executeTxVerifyTest tests
-func executeTxVerifyTest(t *testing.T, test atomicTxVerifyTest) {
-       atomicTx := test.generate(t)
-       err := atomicTx.Verify(test.ctx.XChainID, test.ctx, test.rules)
-       if len(test.expectedErr) == 0 {
-               if err != nil {
-                       t.Fatalf("Atomic tx failed unexpectedly due to: %s", err)
-               }
-       } else {
-               if err == nil {
-                       t.Fatalf("Expected atomic tx test to fail due to: %s, but passed verification", test.expectedErr)
-               }
-               if !strings.Contains(err.Error(), test.expectedErr) {
-                       t.Fatalf("Expected Verify to fail due to %s, but failed with: %s", test.expectedErr, err)
-               }
-       }
-}
-
-type atomicTxTest struct {
-       // setup returns the atomic transaction for the test
-       setup func(t *testing.T, vm *VM, sharedMemory *atomic.Memory) *Tx
-       // define a string that should be contained in the error message if the tx fails verification
-       // at some point. If the strings are empty, then the tx should pass verification at the
-       // respective step.
-       semanticVerifyErr, evmStateTransferErr, acceptErr string
-       // checkState is called iff building and verifying a block containing the transaction is successful. Verifies
-       // the state of the VM following the block's acceptance.
-       checkState func(t *testing.T, vm *VM)
-
-       // Whether or not the VM should be considered to still be bootstrapping
-       bootstrapping bool
-       // genesisJSON to use for the VM genesis (also defines the rule set that will be used in verification)
-       // If this is left empty, [genesisJSONApricotPhase0], will be used
-       genesisJSON string
-
-       // passed directly into GenesisVM
-       configJSON, upgradeJSON string
-}
-
-func executeTxTest(t *testing.T, test atomicTxTest) {
-       genesisJSON := test.genesisJSON
-       if len(genesisJSON) == 0 {
-               genesisJSON = genesisJSONApricotPhase0
-       }
-       issuer, vm, _, sharedMemory, _ := GenesisVM(t, !test.bootstrapping, genesisJSON, test.configJSON, test.upgradeJSON)
-       rules := vm.currentRules()
-
-       tx := test.setup(t, vm, sharedMemory)
-
-       var baseFee *big.Int
-       // If ApricotPhase3 is active, use the initial base fee for the atomic transaction
-       switch {
-       case rules.IsApricotPhase3:
-               baseFee = initialBaseFee
-       }
-
-       lastAcceptedBlock := vm.LastAcceptedBlockInternal().(*Block)
-       if err := tx.UnsignedAtomicTx.SemanticVerify(vm, tx, lastAcceptedBlock, baseFee, rules); len(test.semanticVerifyErr) == 0 && err != nil {
-               t.Fatalf("SemanticVerify failed unexpectedly due to: %s", err)
-       } else if len(test.semanticVerifyErr) != 0 {
-               if err == nil {
-                       t.Fatalf("SemanticVerify unexpectedly returned a nil error. Expected err: %s", test.semanticVerifyErr)
-               }
-               if strings.Contains(err.Error(), test.semanticVerifyErr) {
-                       t.Fatalf("Expected SemanticVerify to fail due to %s, but failed with: %s", test.semanticVerifyErr, err)
-               }
-               // If SemanticVerify failed for the expected reason, return early
-               return
-       }
-
-       // Retrieve dummy state to test that EVMStateTransfer works correctly
-       sdb, err := vm.chain.BlockState(lastAcceptedBlock.ethBlock)
-       if err != nil {
-               t.Fatal(err)
-       }
-       if err := tx.UnsignedAtomicTx.EVMStateTransfer(vm.ctx, sdb); len(test.evmStateTransferErr) == 0 && err != nil {
-               t.Fatalf("EVMStateTransfer failed unexpectedly due to: %s", err)
-       } else if len(test.evmStateTransferErr) != 0 {
-               if err == nil {
-                       t.Fatalf("EVMStateTransfer unexpectedly returned a nil error. Expected err: %s", test.evmStateTransferErr)
-               }
-               if !strings.Contains(err.Error(), test.evmStateTransferErr) {
-                       t.Fatalf("Expected SemanticVerify to fail due to %s, but failed with: %s", test.evmStateTransferErr, err)
-               }
-               // If EVMStateTransfer failed for the expected reason, return early
-               return
-       }
-
-       if err := vm.issueTx(tx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-       <-issuer
-
-       // If we've reached this point, we expect to be able to build and verify the block without any errors
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Accept(); len(test.acceptErr) == 0 && err != nil {
-               t.Fatalf("Accept failed unexpectedly due to: %s", err)
-       } else if len(test.acceptErr) != 0 {
-               if err == nil {
-                       t.Fatalf("Accept unexpectedly returned a nil error. Expected err: %s", test.acceptErr)
-               }
-               if !strings.Contains(err.Error(), test.acceptErr) {
-                       t.Fatalf("Expected Accept to fail due to %s, but failed with: %s", test.acceptErr, err)
-               }
-               // If Accept failed for the expected reason, return early
-               return
-       }
-
-       if test.checkState != nil {
-               test.checkState(t, vm)
-       }
-}
diff --git a/plugin/evm/user.go b/plugin/evm/user.go
index e1902598..fea1be4e 100644
--- a/plugin/evm/user.go
+++ b/plugin/evm/user.go
@@ -7,10 +7,10 @@ import (
        "errors"
        "fmt"

-       "github.com/ava-labs/avalanchego/database/encdb"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/utils/crypto"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/flare/database/encdb"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/utils/crypto"
 )

 // Key in the database whose corresponding value is the list of
diff --git a/plugin/evm/version.go b/plugin/evm/version.go
index 9028c4de..541dac76 100644
--- a/plugin/evm/version.go
+++ b/plugin/evm/version.go
@@ -11,7 +11,7 @@ var (
        // GitCommit is set by the build script
        GitCommit string
        // Version is the version of Coreth
-       Version string
+       Version string = "v0.5.1"
 )

 func init() {
diff --git a/plugin/evm/vm.go b/plugin/evm/vm.go
index 66b98eb3..67396386 100644
--- a/plugin/evm/vm.go
+++ b/plugin/evm/vm.go
@@ -12,53 +12,67 @@ import (
        "math/big"
        "os"
        "path/filepath"
+       "sort"
        "strings"
        "sync"
        "time"

-       "github.com/ava-labs/avalanchego/database/versiondb"
-       coreth "github.com/ava-labs/coreth/chain"
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/eth/ethconfig"
-       "github.com/ava-labs/coreth/node"
-       "github.com/ava-labs/coreth/params"
-
-       "github.com/ava-labs/coreth/rpc"
+       "github.com/flare-foundation/coreth/plugin/evm/message"
+
+       coreth "github.com/flare-foundation/coreth/chain"
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/eth/ethconfig"
+       "github.com/flare-foundation/coreth/metrics/prometheus"
+       "github.com/flare-foundation/coreth/node"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/peer"
+       "github.com/flare-foundation/coreth/rpc"
+
+       // Force-load tracer engine to trigger registration
+       //
+       // We must import this package (not referenced elsewhere) so that the native "callTracer"
+       // is added to a map of client-accessible tracers. In geth, this is done
+       // inside of cmd/geth.
+       _ "github.com/flare-foundation/coreth/eth/tracers/js"
+       _ "github.com/flare-foundation/coreth/eth/tracers/native"
+
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/ethereum/go-ethereum/metrics"
        "github.com/ethereum/go-ethereum/rlp"

        avalancheRPC "github.com/gorilla/rpc/v2"

-       "github.com/ava-labs/avalanchego/cache"
-       "github.com/ava-labs/avalanchego/codec"
-       "github.com/ava-labs/avalanchego/codec/linearcodec"
-       "github.com/ava-labs/avalanchego/database"
-       "github.com/ava-labs/avalanchego/database/manager"
-       "github.com/ava-labs/avalanchego/database/prefixdb"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/snow/choices"
-       "github.com/ava-labs/avalanchego/snow/consensus/snowman"
-       "github.com/ava-labs/avalanchego/snow/engine/snowman/block"
-       "github.com/ava-labs/avalanchego/utils/constants"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/formatting"
-       "github.com/ava-labs/avalanchego/utils/logging"
-       "github.com/ava-labs/avalanchego/utils/math"
-       "github.com/ava-labs/avalanchego/utils/profiler"
-       "github.com/ava-labs/avalanchego/utils/timer/mockable"
-       "github.com/ava-labs/avalanchego/utils/wrappers"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/components/chain"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
-
-       commonEng "github.com/ava-labs/avalanchego/snow/engine/common"
-
-       avalancheJSON "github.com/ava-labs/avalanchego/utils/json"
+       "github.com/flare-foundation/flare/cache"
+       "github.com/flare-foundation/flare/codec"
+       "github.com/flare-foundation/flare/codec/linearcodec"
+       "github.com/flare-foundation/flare/database"
+       "github.com/flare-foundation/flare/database/manager"
+       "github.com/flare-foundation/flare/database/prefixdb"
+       "github.com/flare-foundation/flare/database/versiondb"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/snow/choices"
+       "github.com/flare-foundation/flare/snow/consensus/snowman"
+       "github.com/flare-foundation/flare/snow/engine/snowman/block"
+       "github.com/flare-foundation/flare/utils/constants"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/utils/formatting"
+       "github.com/flare-foundation/flare/utils/logging"
+       "github.com/flare-foundation/flare/utils/math"
+       "github.com/flare-foundation/flare/utils/perms"
+       "github.com/flare-foundation/flare/utils/profiler"
+       "github.com/flare-foundation/flare/utils/timer/mockable"
+       "github.com/flare-foundation/flare/vms/components/avax"
+       "github.com/flare-foundation/flare/vms/components/chain"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"
+
+       commonEng "github.com/flare-foundation/flare/snow/engine/common"
+
+       avalancheJSON "github.com/flare-foundation/flare/utils/json"
 )

 const (
@@ -74,7 +88,8 @@ var (
        x2cRate       = big.NewInt(x2cRateInt64)
        x2cRateMinus1 = big.NewInt(x2cRateMinus1Int64)

-       _ block.ChainVM = &VM{}
+       _ block.ChainVM              = &VM{}
+       _ block.HeightIndexedChainVM = &VM{}
 )

 const (
@@ -101,10 +116,14 @@ const (

 var (
        // Set last accepted key to be longer than the keys used to store accepted block IDs.
-       lastAcceptedKey        = []byte("last_accepted_key")
-       acceptedPrefix         = []byte("snowman_accepted")
-       ethDBPrefix            = []byte("ethdb")
-       atomicTxPrefix         = []byte("atomicTxDB")
+       lastAcceptedKey = []byte("last_accepted_key")
+       acceptedPrefix  = []byte("snowman_accepted")
+       ethDBPrefix     = []byte("ethdb")
+
+       // Prefixes for atomic trie
+       atomicTrieDBPrefix     = []byte("atomicTrieDB")
+       atomicTrieMetaDBPrefix = []byte("atomicTrieMetaDB")
+
        pruneRejectedBlocksKey = []byte("pruned_rejected_blocks")
 )

@@ -142,7 +161,7 @@ var (
        errNilBlockGasCostApricotPhase4   = errors.New("nil blockGasCost is invalid after apricotPhase4")
        errConflictingAtomicTx            = errors.New("conflicting atomic tx present")
        errTooManyAtomicTx                = errors.New("too many atomic tx")
-       defaultLogLevel                   = log.LvlDebug
+       errMissingAtomicTxs               = errors.New("cannot build a block with non-empty extra data and zero atomic transactions")
 )

 var originalStderr *os.File
@@ -176,12 +195,18 @@ type VM struct {
        // [acceptedBlockDB] is the database to store the last accepted
        // block.
        acceptedBlockDB database.Database
-       // [acceptedAtomicTxDB] maintains an index of accepted atomic txs.
-       acceptedAtomicTxDB database.Database
+
+       // [atomicTxRepository] maintains two indexes on accepted atomic txs.
+       // - txID to accepted atomic tx
+       // - block height to list of atomic txs accepted on block at that height
+       atomicTxRepository AtomicTxRepository
+       // [atomicTrie] maintains a merkle forest of [height]=>[atomic txs].
+       //  Used to state sync clients.
+       atomicTrie AtomicTrie

        builder *blockBuilder

-       network Network
+       gossiper Gossiper

        baseCodec codec.Registry
        codec     codec.Manager
@@ -196,14 +221,12 @@ type VM struct {

        // Continuous Profiler
        profiler profiler.ContinuousProfiler
-}

-func (vm *VM) Connected(nodeID ids.ShortID) error {
-       return nil // noop
-}
+       peer.Network
+       client       peer.Client
+       networkCodec codec.Manager

-func (vm *VM) Disconnected(nodeID ids.ShortID) error {
-       return nil // noop
+       bootstrapped bool
 }

 // Codec implements the secp256k1fx interface
@@ -218,9 +241,14 @@ func (vm *VM) Clock() *mockable.Clock { return &vm.clock }
 // Logger implements the secp256k1fx interface
 func (vm *VM) Logger() logging.Logger { return vm.ctx.Log }

-// SetLogLevel sets the log level with the original [os.StdErr] interface
+// setLogLevel sets the log level with the original [os.StdErr] interface along
+// with the context logger.
 func (vm *VM) setLogLevel(logLevel log.Lvl) {
-       log.Root().SetHandler(log.LvlFilterHandler(logLevel, log.StreamHandler(originalStderr, log.TerminalFormat(false))))
+       format := log.TerminalFormat(false)
+       log.Root().SetHandler(log.LvlFilterHandler(logLevel, log.MultiHandler(
+               log.StreamHandler(originalStderr, format),
+               log.StreamHandler(vm.ctx.Log, format),
+       )))
 }

 /*
@@ -235,7 +263,6 @@ func (vm *VM) GetActivationTime() time.Time {
 }

 // Initialize implements the snowman.ChainVM interface
-
 func (vm *VM) Initialize(
        ctx *snow.Context,
        dbManager manager.Manager,
@@ -263,6 +290,9 @@ func (vm *VM) Initialize(
                return errUnsupportedFXs
        }

+       metrics.Enabled = vm.config.MetricsEnabled
+       metrics.EnabledExpensive = vm.config.MetricsExpensiveEnabled
+
        vm.shutdownChan = make(chan struct{}, 1)
        vm.ctx = ctx
        baseDB := dbManager.Current().Database
@@ -271,42 +301,40 @@ func (vm *VM) Initialize(
        vm.chaindb = Database{prefixdb.NewNested(ethDBPrefix, baseDB)}
        vm.db = versiondb.New(baseDB)
        vm.acceptedBlockDB = prefixdb.New(acceptedPrefix, vm.db)
-       vm.acceptedAtomicTxDB = prefixdb.New(atomicTxPrefix, vm.db)
        g := new(core.Genesis)
        if err := json.Unmarshal(genesisBytes, g); err != nil {
                return err
        }

-       // Set the chain config for mainnet/fuji chain IDs
+       // Set the hard-coded chain config for reference network chain IDs
        switch {
-       case g.Config.ChainID.Cmp(params.AvalancheMainnetChainID) == 0:
-               g.Config = params.AvalancheMainnetChainConfig
-               phase0BlockValidator.extDataHashes = mainnetExtDataHashes
-       case g.Config.ChainID.Cmp(params.AvalancheFujiChainID) == 0:
-               g.Config = params.AvalancheFujiChainConfig
-               phase0BlockValidator.extDataHashes = fujiExtDataHashes
-       case g.Config.ChainID.Cmp(params.AvalancheLocalChainID) == 0:
-               g.Config = params.AvalancheLocalChainConfig
-       }
-
-       // Free the memory of the extDataHash map that is not used (i.e. if mainnet
-       // config, free fuji)
-       fujiExtDataHashes = nil
-       mainnetExtDataHashes = nil
+       case g.Config.ChainID.Cmp(params.FlareChainID) == 0:
+               g.Config = params.FlareChainConfig
+               phase0BlockValidator.extDataHashes = flareExtDataHashes
+       case g.Config.ChainID.Cmp(params.CostonChainID) == 0:
+               g.Config = params.CostonChainConfig
+       case g.Config.ChainID.Cmp(params.SongbirdChainID) == 0:
+               g.Config = params.SongbirdChainConfig
+               phase0BlockValidator.extDataHashes = songbirdExtDataHashes
+       case g.Config.ChainID.Cmp(params.LocalChainID) == 0:
+               g.Config = params.LocalChainConfig
+       }
+
+       // Free the memory of the extDataHash map that is not used (i.e. if flare
+       // config, free songbird)
+       songbirdExtDataHashes = nil
+       flareExtDataHashes = nil

        vm.chainID = g.Config.ChainID

        ethConfig := ethconfig.NewDefaultConfig()
        ethConfig.Genesis = g
+       ethConfig.NetworkId = vm.chainID.Uint64()

        // Set log level
-       logLevel := defaultLogLevel
-       if vm.config.LogLevel != "" {
-               configLogLevel, err := log.LvlFromString(vm.config.LogLevel)
-               if err != nil {
-                       return fmt.Errorf("failed to initialize logger due to: %w ", err)
-               }
-               logLevel = configLogLevel
+       logLevel, err := log.LvlFromString(vm.config.LogLevel)
+       if err != nil {
+               return fmt.Errorf("failed to initialize logger due to: %w ", err)
        }

        vm.setLogLevel(logLevel)
@@ -323,6 +351,16 @@ func (vm *VM) Initialize(
        ethConfig.Pruning = vm.config.Pruning
        ethConfig.SnapshotAsync = vm.config.SnapshotAsync
        ethConfig.SnapshotVerify = vm.config.SnapshotVerify
+       ethConfig.OfflinePruning = vm.config.OfflinePruning
+       ethConfig.OfflinePruningBloomFilterSize = vm.config.OfflinePruningBloomFilterSize
+       ethConfig.OfflinePruningDataDirectory = vm.config.OfflinePruningDataDirectory
+
+       if len(ethConfig.OfflinePruningDataDirectory) != 0 {
+               if err := os.MkdirAll(ethConfig.OfflinePruningDataDirectory, perms.ReadWriteExecute); err != nil {
+                       log.Error("failed to create offline pruning data directory", "error", err)
+                       return err
+               }
+       }

        vm.chainConfig = g.Config
        vm.networkID = ethConfig.NetworkId
@@ -346,7 +384,7 @@ func (vm *VM) Initialize(
        var lastAcceptedHash common.Hash
        switch {
        case lastAcceptedErr == database.ErrNotFound:
-               // // Set [lastAcceptedHash] to the genesis block hash.
+               // Set [lastAcceptedHash] to the genesis block hash.
                lastAcceptedHash = ethConfig.Genesis.ToBlock(nil).Hash()
        case lastAcceptedErr != nil:
                return fmt.Errorf("failed to get last accepted block ID due to: %w", lastAcceptedErr)
@@ -355,20 +393,42 @@ func (vm *VM) Initialize(
        default:
                lastAcceptedHash = common.BytesToHash(lastAcceptedBytes)
        }
-       ethChain, err := coreth.NewETHChain(&ethConfig, &nodecfg, vm.chaindb, vm.config.EthBackendSettings(), vm.createConsensusCallbacks(), lastAcceptedHash)
+       ethChain, err := coreth.NewETHChain(&ethConfig, &nodecfg, vm.chaindb, vm.config.EthBackendSettings(), vm.createConsensusCallbacks(), lastAcceptedHash, &vm.clock)
        if err != nil {
                return err
        }
        vm.chain = ethChain
        lastAccepted := vm.chain.LastAcceptedBlock()

+       vm.atomicTxRepository, err = NewAtomicTxRepository(vm.db, vm.codec, lastAccepted.NumberU64())
+       if err != nil {
+               return fmt.Errorf("failed to create atomic repository: %w", err)
+       }
+
+       bonusBlockHeights := make(map[uint64]ids.ID)
+       if vm.chainID.Cmp(params.AvalancheMainnetChainID) == 0 {
+               bonusBlockHeights = bonusBlockMainnetHeights
+       }
+       if err := vm.repairAtomicRepositoryForBonusBlockTxs(getAtomicRepositoryRepairHeights(vm.chainID), vm.getAtomicTxFromPreApricot5BlockByHeight); err != nil {
```

```
+               return fmt.Errorf("failed to repair atomic repository: %w", err)
+       }
+       vm.atomicTrie, err = NewAtomicTrie(vm.db, bonusBlockHeights, vm.atomicTxRepository, vm.codec, lastAccepted.NumberU64())
+       if err != nil {
+               return fmt.Errorf("failed to create atomic trie: %w", err)
+       }
+
        // start goroutines to update the tx pool gas minimum gas price when upgrades go into effect
        vm.handleGasPriceUpdates()

-       // initialize new gossip network
-       //
-       // NOTE: This network must be initialized after the atomic mempool.
-       vm.network = vm.NewNetwork(appSender)
+       vm.networkCodec, err = message.BuildCodec()
+       if err != nil {
+               return err
+       }
+
+       // initialize peer network
+       vm.Network = peer.NewNetwork(appSender, vm.networkCodec, ctx.NodeID, vm.config.MaxOutboundActiveRequests)
+       vm.client = peer.NewClient(vm.Network)
+       vm.initGossipHandling()

        // start goroutines to manage block building
        //
@@ -381,17 +441,23 @@ func (vm *VM) Initialize(
        vm.genesisHash = vm.chain.GetGenesisBlock().Hash()
        log.Info(fmt.Sprintf("lastAccepted = %s", lastAccepted.Hash().Hex()))

+       isApricotPhase5 := vm.chainConfig.IsApricotPhase5(new(big.Int).SetUint64(lastAccepted.Time()))
+       atomicTxs, err := ExtractAtomicTxs(lastAccepted.ExtData(), isApricotPhase5, vm.codec)
+       if err != nil {
+               return err
+       }
        vm.State = chain.NewState(&chain.Config{
                DecidedCacheSize:    decidedCacheSize,
                MissingCacheSize:    missingCacheSize,
                UnverifiedCacheSize: unverifiedCacheSize,
                LastAcceptedBlock: &Block{
-                       id:       ids.ID(lastAccepted.Hash()),
-                       ethBlock: lastAccepted,
-                       vm:       vm,
-                       status:   choices.Accepted,
+                       id:        ids.ID(lastAccepted.Hash()),
+                       ethBlock:  lastAccepted,
+                       vm:        vm,
+                       status:    choices.Accepted,
+                       atomicTxs: atomicTxs,
                },
-               GetBlockIDAtHeight: vm.getBlockIDAtHeight,
+               GetBlockIDAtHeight: vm.GetBlockIDAtHeight,
                GetBlock:           vm.getBlock,
                UnmarshalBlock:     vm.parseBlock,
                BuildBlock:         vm.buildBlock,
@@ -414,9 +480,27 @@ func (vm *VM) Initialize(
        //      return err
        // }

+       // Only provide metrics if they are being populated.
+       if metrics.Enabled {
+               gatherer := prometheus.Gatherer(metrics.DefaultRegistry)
+               if err := ctx.Metrics.Register(gatherer); err != nil {
+                       return err
+               }
+       }
+
        return vm.fx.Initialize(vm)
 }

+func (vm *VM) initGossipHandling() {
+       if vm.chainConfig.ApricotPhase4BlockTimestamp != nil {
+               vm.gossiper = vm.newPushGossiper()
+               vm.Network.SetGossipHandler(NewGossipHandler(vm))
+       } else {
+               vm.gossiper = &noopGossiper{}
+               vm.Network.SetGossipHandler(message.NoopMempoolGossipHandler{})
+       }
+}
+
 func (vm *VM) createConsensusCallbacks() *dummy.ConsensusCallbacks {
        return &dummy.ConsensusCallbacks{
                OnFinalizeAndAssemble: vm.onFinalizeAndAssemble,
@@ -424,7 +508,7 @@ func (vm *VM) createConsensusCallbacks() *dummy.ConsensusCallbacks {
        }
 }

-func (vm *VM) onFinalizeAndAssemble(header *types.Header, state *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
+func (vm *VM) preBatchOnFinalizeAndAssemble(header *types.Header, state *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
        for {
                tx, exists := vm.mempool.NextTx()
                if !exists {
@@ -438,7 +522,7 @@ func (vm *VM) onFinalizeAndAssemble(header *types.Header, state *state.StateDB,
                rules := vm.chainConfig.AvalancheRules(header.Number, new(big.Int).SetUint64(header.Time))
                if err := vm.verifyTx(tx, header.ParentHash, header.BaseFee, state, rules); err != nil {
                        // Discard the transaction from the mempool on failed verification.
-                       vm.mempool.DiscardCurrentTx()
+                       vm.mempool.DiscardCurrentTx(tx.ID())
                        state.RevertToSnapshot(snapshot)
                        continue
                }
@@ -447,12 +531,12 @@ func (vm *VM) onFinalizeAndAssemble(header *types.Header, state *state.StateDB,
                if err != nil {
                        // Discard the transaction from the mempool and error if the transaction
                        // cannot be marshalled. This should never happen.
-                       vm.mempool.DiscardCurrentTx()
+                       vm.mempool.DiscardCurrentTx(tx.ID())
                        return nil, nil, nil, fmt.Errorf("failed to marshal atomic transaction %s due to %w", tx.ID(), err)
                }
                var contribution, gasUsed *big.Int
                if rules.IsApricotPhase4 {
-                       contribution, gasUsed, err = tx.BlockFeeContribution(vm.ctx.AVAXAssetID, header.BaseFee)
+                       contribution, gasUsed, err = tx.BlockFeeContribution(rules.IsApricotPhase5, vm.ctx.AVAXAssetID, header.BaseFee)
                        if err != nil {
                                return nil, nil, nil, err
                        }
@@ -468,28 +552,146 @@ func (vm *VM) onFinalizeAndAssemble(header *types.Header, state *state.StateDB,
        return nil, nil, nil, nil
 }

+// assumes that we are in at least Apricot Phase 5.
+func (vm *VM) postBatchOnFinalizeAndAssemble(header *types.Header, state *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
+       var (
+               batchAtomicTxs    []*Tx
+               batchAtomicUTXOs  ids.Set
+               batchContribution *big.Int = new(big.Int).Set(common.Big0)
+               batchGasUsed      *big.Int = new(big.Int).Set(common.Big0)
+               rules                      = vm.chainConfig.AvalancheRules(header.Number, new(big.Int).SetUint64(header.Time))
+       )
+
+       for {
+               tx, exists := vm.mempool.NextTx()
+               if !exists {
+                       break
```

```diff
+               }
+
+               var (
+                       txGasUsed, txContribution *big.Int
+                       err                       error
+               )
+
+               // Note: we do not need to check if we are in at least ApricotPhase4 here because
+               // we assume that this function will only be called when the block is in at least
+               // ApricotPhase5.
+               txContribution, txGasUsed, err = tx.BlockFeeContribution(true, vm.ctx.AVAXAssetID, header.BaseFee)
+               if err != nil {
+                       return nil, nil, nil, err
+               }
+               // ensure [gasUsed] + [batchGasUsed] doesnt exceed the [atomicGasLimit]
+               if totalGasUsed := new(big.Int).Add(batchGasUsed, txGasUsed); totalGasUsed.Cmp(params.AtomicGasLimit) > 0 {
+                       // Send [tx] back to the mempool's tx heap.
+                       vm.mempool.CancelCurrentTx(tx.ID())
+                       break
+               }
+
+               if batchAtomicUTXOs.Overlaps(tx.InputUTXOs()) {
+                       // Discard the transaction from the mempool since it will fail verification
+                       // after this block has been accepted.
+                       // Note: if the proposed block is not accepted, the transaction may still be
+                       // valid, but we discard it early here based on the assumption that the proposed
+                       // block will most likely be accepted.
+                       // Discard the transaction from the mempool on failed verification.
+                       vm.mempool.DiscardCurrentTx(tx.ID())
+                       continue
+               }
+
+               snapshot := state.Snapshot()
+               if err := vm.verifyTx(tx, header.ParentHash, header.BaseFee, state, rules); err != nil {
+                       // Discard the transaction from the mempool and reset the state to [snapshot]
+                       // if it fails verification here.
+                       // Note: prior to this point, we have not modified [state] so there is no need to
+                       // revert to a snapshot if we discard the transaction prior to this point.
+                       vm.mempool.DiscardCurrentTx(tx.ID())
+                       state.RevertToSnapshot(snapshot)
+                       continue
+               }
+
+               batchAtomicTxs = append(batchAtomicTxs, tx)
+               batchAtomicUTXOs.Union(tx.InputUTXOs())
+               // Add the [txGasUsed] to the [batchGasUsed] when the [tx] has passed verification
+               batchGasUsed.Add(batchGasUsed, txGasUsed)
+               batchContribution.Add(batchContribution, txContribution)
+       }
+
+       // If there is a non-zero number of transactions, marshal them and return the byte slice
+       // for the block's extra data along with the contribution and gas used.
+       if len(batchAtomicTxs) > 0 {
+               atomicTxBytes, err := vm.codec.Marshal(codecVersion, batchAtomicTxs)
+               if err != nil {
+                       // If we fail to marshal the batch of atomic transactions for any reason,
+                       // discard the entire set of current transactions.
+                       vm.mempool.DiscardCurrentTxs()
+                       return nil, nil, nil, fmt.Errorf("failed to marshal batch of atomic transactions due to %w", err)
+               }
+               return atomicTxBytes, batchContribution, batchGasUsed, nil
+       }
+
+       // If there are no regular transactions and there were also no atomic transactions to be included,
+       // then the block is empty and should be considered invalid.
+       if len(txs) == 0 {
+               // this could happen due to the async logic of geth tx pool
+               return nil, nil, nil, errEmptyBlock
+       }
+
+       // If there are no atomic transactions, but there is a non-zero number of regular transactions, then
+       // we return a nil slice with no contribution from the atomic transactions and a nil error.
+       return nil, nil, nil, nil
+}
+
+func (vm *VM) onFinalizeAndAssemble(header *types.Header, state *state.StateDB, txs []*types.Transaction) ([]byte, *big.Int, *big.Int, error) {
+       if !vm.chainConfig.IsApricotPhase5(new(big.Int).SetUint64(header.Time)) {
+               return vm.preBatchOnFinalizeAndAssemble(header, state, txs)
+       }
+       return vm.postBatchOnFinalizeAndAssemble(header, state, txs)
+}
+
 func (vm *VM) onExtraStateChange(block *types.Block, state *state.StateDB) (*big.Int, *big.Int, error) {
-       tx, err := vm.extractAtomicTx(block)
+       var (
+               batchContribution *big.Int = big.NewInt(0)
+               batchGasUsed      *big.Int = big.NewInt(0)
+               timestamp                  = new(big.Int).SetUint64(block.Time())
+               isApricotPhase4            = vm.chainConfig.IsApricotPhase4(timestamp)
+               isApricotPhase5            = vm.chainConfig.IsApricotPhase5(timestamp)
+       )
+
+       txs, err := ExtractAtomicTxs(block.ExtData(), isApricotPhase5, vm.codec)
        if err != nil {
                return nil, nil, err
        }
-       // If [tx] is nil, we can return nil for the extra state contribution instead of allocating
-       // a big Int for 0.
-       if tx == nil {
+
+       // If there are no transactions, we can return early
+       if len(txs) == 0 {
                return nil, nil, nil
        }
-       if err := tx.UnsignedAtomicTx.EVMStateTransfer(vm.ctx, state); err != nil {
-               return nil, nil, err
-       }

-       switch {
-       // If ApricotPahse4 is enabled, calculate the block fee contribution
-       case vm.chainConfig.IsApricotPhase4(new(big.Int).SetUint64(block.Time())):
-               return tx.BlockFeeContribution(vm.ctx.AVAXAssetID, block.BaseFee())
-       default:
-               // Otherwise, there is no contribution
-               return nil, nil, nil
+       for _, tx := range txs {
+               if err := tx.UnsignedAtomicTx.EVMStateTransfer(vm.ctx, state); err != nil {
+                       return nil, nil, err
+               }
+               // If ApricotPhase4 is enabled, calculate the block fee contribution
+               if isApricotPhase4 {
+                       contribution, gasUsed, err := tx.BlockFeeContribution(isApricotPhase5, vm.ctx.AVAXAssetID, block.BaseFee())
+                       if err != nil {
+                               return nil, nil, err
+                       }
+
+                       batchContribution.Add(batchContribution, contribution)
+                       batchGasUsed.Add(batchGasUsed, gasUsed)
+               }
+
+               // If ApricotPhase5 is enabled, enforce that the atomic gas used does not exceed the
+               // atomic gas limit.
+               if vm.chainConfig.IsApricotPhase5(timestamp) {
+                       // Ensure that [tx] does not push [block] above the atomic gas limit.
+                       if batchGasUsed.Cmp(params.AtomicGasLimit) == 1 {
+                               return nil, nil, fmt.Errorf("atomic gas used (%d) by block (%s), exceeds atomic gas limit (%d)", batchGasUsed, block.Hash().Hex(), params.AtomicGasLimit)
+                       }
+               }
        }
+       return batchContribution, batchGasUsed, nil
 }

 func (vm *VM) pruneChain() error {
@@ -513,19 +715,20 @@ func (vm *VM) pruneChain() error {
        if err := vm.db.Put(pruneRejectedBlocksKey, heightBytes); err != nil {
                return err
        }
-
        return vm.db.Commit()
 }

-// Bootstrapping notifies this VM that the consensus engine is performing
-// bootstrapping
-func (vm *VM) Bootstrapping() error { return vm.fx.Bootstrapping() }
-
-// Bootstrapped notifies this VM that the consensus engine has finished
-// bootstrapping
-func (vm *VM) Bootstrapped() error {
-       vm.ctx.Bootstrapped()
-       return vm.fx.Bootstrapped()
+func (vm *VM) SetState(state snow.State) error {
+       switch state {
+       case snow.Bootstrapping:
+               vm.bootstrapped = false
+               return vm.fx.Bootstrapping()
+       case snow.NormalOp:
+               vm.bootstrapped = true
+               return vm.fx.Bootstrapped()
+       default:
+               return snow.ErrUnknownState
+       }
 }

 // Shutdown implements the snowman.ChainVM interface
@@ -545,15 +748,22 @@ func (vm *VM) buildBlock() (snowman.Block, error) {
        block, err := vm.chain.GenerateBlock()
        vm.builder.handleGenerateBlock()
        if err != nil {
-               vm.mempool.CancelCurrentTx()
+               vm.mempool.CancelCurrentTxs()
                return nil, err
        }

+       isApricotPhase5 := vm.chainConfig.IsApricotPhase5(new(big.Int).SetUint64(block.Time()))
+       atomicTxs, err := ExtractAtomicTxs(block.ExtData(), isApricotPhase5, vm.codec)
+       if err != nil {
+               vm.mempool.DiscardCurrentTxs()
+               return nil, err
+       }
        // Note: the status of block is set by ChainState
        blk := &Block{
-               id:       ids.ID(block.Hash()),
-               ethBlock: block,
-               vm:       vm,
+               id:        ids.ID(block.Hash()),
+               ethBlock:  block,
+               vm:        vm,
+               atomicTxs: atomicTxs,
        }

        // Verify is called on a non-wrapped block here, such that this
@@ -569,14 +779,14 @@ func (vm *VM) buildBlock() (snowman.Block, error) {
        // to the blk state root in the triedb when we are going to call verify
        // again from the consensus engine with writes enabled.
        if err := blk.verify(false /*=writes*/); err != nil {
-               vm.mempool.CancelCurrentTx()
+               vm.mempool.CancelCurrentTxs()
                return nil, fmt.Errorf("block failed verification due to: %w", err)
        }

        log.Debug(fmt.Sprintf("Built block %s", blk.ID()))
-       // Marks the current tx from the mempool as being successfully issued
+       // Marks the current transactions from the mempool as being successfully issued
        // into a block.
-       vm.mempool.IssueCurrentTx()
+       vm.mempool.IssueCurrentTxs()
        return blk, nil
 }

@@ -586,11 +796,18 @@ func (vm *VM) parseBlock(b []byte) (snowman.Block, error) {
        if err := rlp.DecodeBytes(b, ethBlock); err != nil {
                return nil, err
        }
+
+       isApricotPhase5 := vm.chainConfig.IsApricotPhase5(new(big.Int).SetUint64(ethBlock.Time()))
+       atomicTxs, err := ExtractAtomicTxs(ethBlock.ExtData(), isApricotPhase5, vm.codec)
+       if err != nil {
+               return nil, err
+       }
        // Note: the status of block is set by ChainState
        block := &Block{
-               id:       ids.ID(ethBlock.Hash()),
-               ethBlock: ethBlock,
-               vm:       vm,
+               id:        ids.ID(ethBlock.Hash()),
+               ethBlock:  ethBlock,
+               vm:        vm,
+               atomicTxs: atomicTxs,
        }
        // Performing syntactic verification in ParseBlock allows for
        // short-circuiting bad blocks before they are processed by the VM.
@@ -609,11 +826,17 @@ func (vm *VM) getBlock(id ids.ID) (snowman.Block, error) {
        if ethBlock == nil {
                return nil, database.ErrNotFound
        }
+       isApricotPhase5 := vm.chainConfig.IsApricotPhase5(new(big.Int).SetUint64(ethBlock.Time()))
+       atomicTxs, err := ExtractAtomicTxs(ethBlock.ExtData(), isApricotPhase5, vm.codec)
+       if err != nil {
+               return nil, err
+       }
        // Note: the status of block is set by ChainState
        blk := &Block{
-               id:       ids.ID(ethBlock.Hash()),
-               ethBlock: ethBlock,
-               vm:       vm,
+               id:        ids.ID(ethBlock.Hash()),
+               ethBlock:  ethBlock,
+               vm:        vm,
+               atomicTxs: atomicTxs,
        }
        return blk, nil
 }
@@ -631,10 +854,15 @@ func (vm *VM) SetPreference(blkID ids.ID) error {
        return vm.chain.SetPreference(block.(*Block).ethBlock)
 }

-// getBlockIDAtHeight retrieves the blkID of the canonical block at [blkHeight]
+func (vm *VM) VerifyHeightIndex() error {
```

```
+       // our index is vm.chain.GetBlockByNumber
+       return nil
+}
+
+// GetBlockIDAtHeight retrieves the blkID of the canonical block at [blkHeight]
 // if [blkHeight] is less than the height of the last accepted block, this will return
 // a canonical block. Otherwise, it may return a blkID that has not yet been accepted.
-func (vm *VM) getBlockIDAtHeight(blkHeight uint64) (ids.ID, error) {
+func (vm *VM) GetBlockIDAtHeight(blkHeight uint64) (ids.ID, error) {
        ethBlock := vm.chain.GetBlockByNumber(blkHeight)
        if ethBlock == nil {
                return ids.ID{}, fmt.Errorf("could not find block at height: %d", blkHeight)
@@ -673,7 +901,9 @@ func newHandler(name string, service interface{}, lockOption ...commonEng.LockOp
 func (vm *VM) CreateHandlers() (map[string]*commonEng.HTTPHandler, error) {
        handler := vm.chain.NewRPCHandler(vm.config.APIMaxDuration.Duration)
        enabledAPIs := vm.config.EthAPIs()
-       vm.chain.AttachEthService(handler, enabledAPIs)
+       if err := vm.chain.AttachEthService(handler, enabledAPIs); err != nil {
+               return nil, err
+       }

        primaryAlias, err := vm.ctx.BCLookup.PrimaryAlias(vm.ctx.ChainID)
        if err != nil {
@@ -688,7 +918,7 @@ func (vm *VM) CreateHandlers() (map[string]*commonEng.HTTPHandler, error) {
        apis[avaxEndpoint] = avaxAPI

        if vm.config.CorethAdminAPIEnabled {
-               adminAPI, err := newHandler("admin", NewAdminService(vm, fmt.Sprintf("coreth_performance_%s", primaryAlias)))
+               adminAPI, err := newHandler("admin", NewAdminService(vm, os.ExpandEnv(fmt.Sprintf("%s_coreth_performance_%s", vm.config.CorethAdminAPIDir, primaryAlias))))
                if err != nil {
                        return nil, fmt.Errorf("failed to register service for admin API due to %w", err)
                }
@@ -696,22 +926,12 @@ func (vm *VM) CreateHandlers() (map[string]*commonEng.HTTPHandler, error) {
                enabledAPIs = append(enabledAPIs, "coreth-admin")
        }

-       errs := wrappers.Errs{}
        if vm.config.SnowmanAPIEnabled {
-               errs.Add(handler.RegisterName("snowman", &SnowmanAPI{vm}))
+               if err := handler.RegisterName("snowman", &SnowmanAPI{vm}); err != nil {
+                       return nil, err
+               }
                enabledAPIs = append(enabledAPIs, "snowman")
        }
-       if vm.config.NetAPIEnabled {
-               errs.Add(handler.RegisterName("net", &NetAPI{vm}))
-               enabledAPIs = append(enabledAPIs, "net")
-       }
-       if vm.config.Web3APIEnabled {
-               errs.Add(handler.RegisterName("web3", &Web3API{}))
-               enabledAPIs = append(enabledAPIs, "web3")
-       }
-       if errs.Errored() {
-               return nil, errs.Err
-       }

        log.Info(fmt.Sprintf("Enabled APIs: %s", strings.Join(enabledAPIs, ", ")))
        apis[ethRPCEndpoint] = &commonEng.HTTPHandler{
@@ -740,7 +960,6 @@ func (vm *VM) CreateStaticHandlers() (map[string]*commonEng.HTTPHandler, error)

        return map[string]*commonEng.HTTPHandler{
                "/rpc": {LockOptions: commonEng.NoLock, Handler: handler},
-               "/ws":  {LockOptions: commonEng.NoLock, Handler: handler.WebsocketHandler([]string{"*"})},
        }, nil
 }

@@ -749,35 +968,17 @@ func (vm *VM) CreateStaticHandlers() (map[string]*commonEng.HTTPHandler, error)
  ********************************** Helpers **********************************
  ******************************************************************************
  */
-// extractAtomicTx returns the atomic transaction in [block] if
-// one exists.
-func (vm *VM) extractAtomicTx(block *types.Block) (*Tx, error) {
-       extdata := block.ExtData()
-       if len(extdata) == 0 {
-               return nil, nil
-       }
-       atx := new(Tx)
-       if _, err := vm.codec.Unmarshal(extdata, atx); err != nil {
-               return nil, fmt.Errorf("failed to unmarshal atomic tx due to %w", err)
-       }
-       if err := atx.Sign(vm.codec, nil); err != nil {
-               return nil, fmt.Errorf("failed to initialize atomic tx in block %s", block.Hash().Hex())
-       }
-
-       return atx, nil
-}

+// conflicts returns an error if [inputs] conflicts with any of the atomic inputs contained in [ancestor]
+// or any of its ancestor blocks going back to the last accepted block in its ancestry. If [ancestor] is
+// accepted, then nil will be returned immediately.
+// If the ancestry of [ancestor] cannot be fetched, then [errRejectedParent] may be returned.
 func (vm *VM) conflicts(inputs ids.Set, ancestor *Block) error {
        for ancestor.Status() != choices.Accepted {
-               atx, err := vm.extractAtomicTx(ancestor.ethBlock)
-               if err != nil {
-                       return fmt.Errorf("problem parsing atomic tx of ancestor block %s: %w", ancestor.ID(), err)
-               }
-               // If the ancestor isn't an atomic block, it can't conflict with
-               // the import tx.
-               if atx != nil {
-                       ancestorInputs := atx.UnsignedAtomicTx.InputUTXOs()
-                       if inputs.Overlaps(ancestorInputs) {
+               // If any of the atomic transactions in the ancestor conflict with [inputs]
+               // return an error.
+               for _, atomicTx := range ancestor.atomicTxs {
+                       if inputs.Overlaps(atomicTx.InputUTXOs()) {
                                return errConflictingAtomicInputs
                        }
                }
@@ -809,37 +1010,14 @@ func (vm *VM) conflicts(inputs ids.Set, ancestor *Block) error {
        return nil
 }

-// getAcceptedAtomicTx attempts to get [txID] from the database.
-func (vm *VM) getAcceptedAtomicTx(txID ids.ID) (*Tx, uint64, error) {
-       indexedTxBytes, err := vm.acceptedAtomicTxDB.Get(txID[:])
-       if err != nil {
-               return nil, 0, err
-       }
-
-       packer := wrappers.Packer{Bytes: indexedTxBytes}
-       height := packer.UnpackLong()
-       txBytes := packer.UnpackBytes()
-
-       tx := &Tx{}
-       if _, err := vm.codec.Unmarshal(txBytes, tx); err != nil {
-               return nil, 0, fmt.Errorf("problem parsing atomic transaction from db: %w", err)
-       }
-       if err := tx.Sign(vm.codec, nil); err != nil {
-               return nil, 0, fmt.Errorf("problem initializing atomic transaction from db: %w", err)
-       }
-
-       return tx, height, nil
```

```diff
-}
-
 // getAtomicTx returns the requested transaction, status, and height.
 // If the status is Unknown, then the returned transaction will be nil.
 func (vm *VM) getAtomicTx(txID ids.ID) (*Tx, Status, uint64, error) {
-       if tx, height, err := vm.getAcceptedAtomicTx(txID); err == nil {
+       if tx, height, err := vm.atomicTxRepository.GetByTxID(txID); err == nil {
                return tx, Accepted, height, nil
        } else if err != database.ErrNotFound {
                return nil, Unknown, 0, err
        }
-
        tx, dropped, found := vm.mempool.GetTx(txID)
        switch {
        case found && dropped:
@@ -851,20 +1029,6 @@ func (vm *VM) getAtomicTx(txID ids.ID) (*Tx, Status, uint64, error) {
        }
 }
 
-// writeAtomicTx writes indexes [tx] in [blk]
-func (vm *VM) writeAtomicTx(blk *Block, tx *Tx) error {
-       // 8 bytes
-       height := blk.ethBlock.NumberU64()
-       // 4 + len(txBytes)
-       txBytes := tx.Bytes()
-       packer := wrappers.Packer{Bytes: make([]byte, 12+len(txBytes))}
-       packer.PackLong(height)
-       packer.PackBytes(txBytes)
-       txID := tx.ID()
-
-       return vm.acceptedAtomicTxDB.Put(txID[:], packer.Bytes)
-}
-
 // ParseAddress takes in an address and produces the ID of the chain it's for
 // the ID of the address
 func (vm *VM) ParseAddress(addrStr string) (ids.ID, ids.ShortID, error) {
@@ -924,7 +1088,6 @@ func (vm *VM) issueTx(tx *Tx, local bool) error {
                }
                return err
        }
-
        // NOTE: Gossiping of the issued [Tx] is handled in [AddTx]
        return nil
 }
@@ -939,10 +1102,10 @@ func (vm *VM) verifyTxAtTip(tx *Tx) error {
        rules := vm.currentRules()
        parentHeader := preferredBlock.Header()
        var nextBaseFee *big.Int
-       timestamp := time.Now().Unix()
+       timestamp := vm.clock.Time().Unix()
        bigTimestamp := big.NewInt(timestamp)
        if vm.chainConfig.IsApricotPhase3(bigTimestamp) {
-               _, nextBaseFee, err = dummy.CalcBaseFee(vm.chainConfig, parentHeader, uint64(timestamp))
+               _, nextBaseFee, err = dummy.EstimateNextBaseFee(vm.chainConfig, parentHeader, uint64(timestamp))
                if err != nil {
                        // Return extremely detailed error since CalcBaseFee should never encounter an issue here
                        return fmt.Errorf("failed to calculate base fee with parent timestamp (%d), parent ExtraData: (0x%x), and current timestamp (%d): %w", parentHeader.Time, parentHeader.Extra, times
@@ -1201,6 +1364,8 @@ func (vm *VM) currentRules() params.Rules {
 // follows the ruleset defined by [rules]
 func (vm *VM) getBlockValidator(rules params.Rules) BlockValidator {
        switch {
+       case rules.IsApricotPhase5:
+               return phase5BlockValidator
        case rules.IsApricotPhase4:
                return phase4BlockValidator
        case rules.IsApricotPhase3:
@@ -1255,3 +1420,84 @@ func (vm *VM) estimateBaseFee(ctx context.Context) (*big.Int, error) {
 
        return baseFee, nil
 }
+
+func getAtomicRepositoryRepairHeights(chainID *big.Int) []uint64 {
+       if chainID.Cmp(params.AvalancheMainnetChainID) != 0 {
+               return nil
+       }
+       repairHeights := make([]uint64, 0, len(bonusBlockMainnetHeights)+len(canonicalBonusBlocks))
+       for height := range bonusBlockMainnetHeights {
+               repairHeights = append(repairHeights, height)
+       }
+       for _, height := range canonicalBonusBlocks {
+               if _, exists := bonusBlockMainnetHeights[height]; !exists {
+                       repairHeights = append(repairHeights, height)
+               }
+       }
+       sort.Slice(repairHeights, func(i, j int) bool { return repairHeights[i] < repairHeights[j] })
+       return repairHeights
+}
+
+func (vm *VM) getAtomicTxFromPreApricot5BlockByHeight(height uint64) (*Tx, error) {
+       blk := vm.chain.GetBlockByNumber(height)
+       if blk == nil {
+               return nil, nil
+       }
+       return ExtractAtomicTx(blk.ExtData(), vm.codec)
+}
+
+// repairAtomicRepositoryForBonusBlockTxs ensures that atomic txs that were processed
+// on more than one block (canonical block + a number of bonus blocks) are indexed to
+// the first height they were processed on (canonical block).
+// [sortedHeights] should include all canonical block + bonus block heights in ascending
+// order, and will only be passed as non-empty on mainnet.
+func (vm *VM) repairAtomicRepositoryForBonusBlockTxs(
+       sortedHeights []uint64, getAtomicTxFromBlockByHeight func(height uint64) (*Tx, error),
+) error {
+       done, err := vm.atomicTxRepository.IsBonusBlocksRepaired()
+       if err != nil {
+               return err
+       }
+       if done {
+               return nil
+       }
+       repairedEntries := uint64(0)
+       seenTxs := make(map[ids.ID][]uint64)
+       for _, height := range sortedHeights {
+               // get atomic tx from block
+               tx, err := getAtomicTxFromBlockByHeight(height)
+               if err != nil {
+                       return err
+               }
+               if tx == nil {
+                       continue
+               }
+
+               // get the tx by txID and update it, the first time we encounter
+               // a given [txID], overwrite the previous [txID] => [height]
+               // mapping. This provides a canonical mapping across nodes.
+               heights, seen := seenTxs[tx.ID()]
+               _, foundHeight, err := vm.atomicTxRepository.GetByTxID(tx.ID())
+               if err != nil && !errors.Is(err, database.ErrNotFound) {
+                       return err
+               }
+               if !seen {
+                       if err := vm.atomicTxRepository.Write(height, []*Tx{tx}); err != nil {
+                               return err
+                       }
+               } else {
+                       if err := vm.atomicTxRepository.WriteBonus(height, []*Tx{tx}); err != nil {
+                               return err
+                       }
+               }
+               if foundHeight != height && !seen {
+                       repairedEntries++
+               }
+               seenTxs[tx.ID()] = append(heights, height)
+       }
+       if err := vm.atomicTxRepository.MarkBonusBlocksRepaired(repairedEntries); err != nil {
+               return err
+       }
+       log.Info("repairAtomicRepositoryForBonusBlockTxs complete", "repairedEntries", repairedEntries)
+       return vm.db.Commit()
+}
diff --git a/plugin/evm/vm_test.go b/plugin/evm/vm_test.go
index 2050e189..76f4d1e0 100644
--- a/plugin/evm/vm_test.go
+++ b/plugin/evm/vm_test.go
@@ -12,43 +12,48 @@ import (
        "math/big"
        "os"
        "path/filepath"
+       "sort"
        "strings"
        "testing"
        "time"
 
-       "github.com/ava-labs/coreth/trie"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
+       "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/trie"
 
        "github.com/stretchr/testify/assert"
 
-       "github.com/ava-labs/avalanchego/api/keystore"
-       "github.com/ava-labs/avalanchego/chains/atomic"
-       "github.com/ava-labs/avalanchego/database/manager"
-       "github.com/ava-labs/avalanchego/database/prefixdb"
-       "github.com/ava-labs/avalanchego/ids"
-       "github.com/ava-labs/avalanchego/snow"
-       "github.com/ava-labs/avalanchego/snow/choices"
-       "github.com/ava-labs/avalanchego/utils/crypto"
-       "github.com/ava-labs/avalanchego/utils/formatting"
-       "github.com/ava-labs/avalanchego/utils/hashing"
-       "github.com/ava-labs/avalanchego/utils/logging"
-       "github.com/ava-labs/avalanchego/utils/units"
-       "github.com/ava-labs/avalanchego/version"
-       "github.com/ava-labs/avalanchego/vms/components/avax"
-       "github.com/ava-labs/avalanchego/vms/components/chain"
-       "github.com/ava-labs/avalanchego/vms/secp256k1fx"
-
-       engCommon "github.com/ava-labs/avalanchego/snow/engine/common"
-
-       "github.com/ava-labs/coreth/consensus/dummy"
-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/eth"
-       "github.com/ava-labs/coreth/params"
-       "github.com/ava-labs/coreth/rpc"
-
-       accountKeystore "github.com/ava-labs/coreth/accounts/keystore"
+       "github.com/flare-foundation/flare/api/keystore"
+       "github.com/flare-foundation/flare/chains/atomic"
+       "github.com/flare-foundation/flare/database/manager"
+       "github.com/flare-foundation/flare/database/memdb"
+       "github.com/flare-foundation/flare/database/prefixdb"
+       "github.com/flare-foundation/flare/database/versiondb"
+       "github.com/flare-foundation/flare/ids"
+       "github.com/flare-foundation/flare/snow"
+       "github.com/flare-foundation/flare/snow/choices"
+       "github.com/flare-foundation/flare/utils/constants"
+       "github.com/flare-foundation/flare/utils/crypto"
+       "github.com/flare-foundation/flare/utils/formatting"
+       "github.com/flare-foundation/flare/utils/hashing"
+       "github.com/flare-foundation/flare/utils/logging"
+       "github.com/flare-foundation/flare/utils/units"
+       "github.com/flare-foundation/flare/version"
+       "github.com/flare-foundation/flare/vms/components/avax"
+       "github.com/flare-foundation/flare/vms/components/chain"
+       "github.com/flare-foundation/flare/vms/secp256k1fx"
+
+       engCommon "github.com/flare-foundation/flare/snow/engine/common"
+
+       "github.com/flare-foundation/coreth/consensus/dummy"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/eth"
+       "github.com/flare-foundation/coreth/params"
+       "github.com/flare-foundation/coreth/rpc"
+
+       accountKeystore "github.com/flare-foundation/coreth/accounts/keystore"
 )
 
 var (
@@ -62,19 +67,21 @@ var (
        testAvaxAssetID  = ids.ID{1, 2, 3}
        username         = "Johns"
        password         = "CjasdjhiPeirbSenfeI13" // #nosec G101
-       // Use chainId: 43111, so that it does not overlap with any Avalanche ChainIDs, which may have their
+       // Use chainId: 31337, so that it does not overlap with any Avalanche ChainIDs, which may have their
        // config overridden in vm.Initialize.
-       genesisJSONApricotPhase0 = "{\"config\":{\"chainId\":43111,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
-       genesisJSONApricotPhase1 = "{\"config\":{\"chainId\":43111,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
-       genesisJSONApricotPhase2 = "{\"config\":{\"chainId\":43111,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
-       genesisJSONApricotPhase3 = "{\"config\":{\"chainId\":43111,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
-       genesisJSONApricotPhase4 = "{\"config\":{\"chainId\":43111,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase0 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase1 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase2 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase3 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase4 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
+       genesisJSONApricotPhase5 = "{\"config\":{\"chainId\":31337,\"homesteadBlock\":0,\"daoForkBlock\":0,\"daoForkSupport\":true,\"eip150Block\":0,\"eip150Hash\":\"0x2086799aeebeae135c246c65021c82b4e15
 
        apricotRulesPhase0 = params.Rules{}
        apricotRulesPhase1 = params.Rules{IsApricotPhase1: true}
        apricotRulesPhase2 = params.Rules{IsApricotPhase1: true, IsApricotPhase2: true}
        apricotRulesPhase3 = params.Rules{IsApricotPhase1: true, IsApricotPhase2: true, IsApricotPhase3: true}
        apricotRulesPhase4 = params.Rules{IsApricotPhase1: true, IsApricotPhase2: true, IsApricotPhase3: true, IsApricotPhase4: true}
+       apricotRulesPhase5 = params.Rules{IsApricotPhase1: true, IsApricotPhase2: true, IsApricotPhase3: true, IsApricotPhase4: true, IsApricotPhase5: true}
 )
 
 func init() {
@@ -123,16 +130,34 @@ func NewContext() *snow.Context {
        ctx.NetworkID = testNetworkID
        ctx.ChainID = testCChainID
        ctx.AVAXAssetID = testAvaxAssetID
-       ctx.XChainID = ids.Empty.Prefix(0)
+       ctx.XChainID = testXChainID
        aliaser := ctx.BCLookup.(ids.Aliaser)
        _ = aliaser.Alias(testCChainID, "C")
	_ = aliaser.Alias(testCChainID, testCChainID.String())
	_ = aliaser.Alias(testXChainID, "X")
	_ = aliaser.Alias(testXChainID, testXChainID.String())
-
+	ctx.SNLookup = &snLookup{
+		chainsToSubnet: map[ids.ID]ids.ID{
+			constants.PlatformChainID: constants.PrimaryNetworkID,
+			testXChainID:              constants.PrimaryNetworkID,
+			testCChainID:              constants.PrimaryNetworkID,
+		},
+	}
	return ctx
 }
 
+type snLookup struct {
+	chainsToSubnet map[ids.ID]ids.ID
+}
+
+func (sn *snLookup) SubnetID(chainID ids.ID) (ids.ID, error) {
+	subnetID, ok := sn.chainsToSubnet[chainID]
+	if !ok {
+		return ids.ID{}, errors.New("unknown chain")
+	}
+	return subnetID, nil
+}
+
 func setupGenesis(t *testing.T,
	genesisJSON string,
 ) (*snow.Context,
@@ -194,17 +219,18 @@ func GenesisVM(t *testing.T,
	}
 
	if finishBootstrapping {
-		assert.NoError(t, vm.Bootstrapping())
-		assert.NoError(t, vm.Bootstrapped())
+		assert.NoError(t, vm.SetState(snow.Bootstrapping))
+		assert.NoError(t, vm.SetState(snow.NormalOp))
	}
 
	return issuer, vm, dbManager, m, appSender
 }
 
-func addUTXO(sharedMemory *atomic.Memory, ctx *snow.Context, txID ids.ID, assetID ids.ID, amount uint64, addr ids.ShortID) (*avax.UTXO, error) {
+func addUTXO(sharedMemory *atomic.Memory, ctx *snow.Context, txID ids.ID, index uint32, assetID ids.ID, amount uint64, addr ids.ShortID) (*avax.UTXO, error) {
	utxo := &avax.UTXO{
		UTXOID: avax.UTXOID{
-			TxID: txID,
+			TxID:        txID,
+			OutputIndex: index,
		},
		Asset: avax.Asset{ID: assetID},
		Out: &secp256k1fx.TransferOutput{
@@ -244,7 +270,7 @@ func GenesisVMWithUTXOs(t *testing.T, finishBootstrapping bool, genesisJSON stri
		if err != nil {
			t.Fatalf("Failed to generate txID from addr: %s", err)
		}
-		if _, err := addUTXO(sharedMemory, vm.ctx, txID, vm.ctx.AVAXAssetID, avaxAmount, addr); err != nil {
+		if _, err := addUTXO(sharedMemory, vm.ctx, txID, 0, vm.ctx.AVAXAssetID, avaxAmount, addr); err != nil {
			t.Fatalf("Failed to add UTXO to shared memory: %s", err)
		}
	}
@@ -254,24 +280,24 @@ func GenesisVMWithUTXOs(t *testing.T, finishBootstrapping bool, genesisJSON stri
 
 func TestVMConfig(t *testing.T) {
	txFeeCap := float64(11)
-	netApiEnabled := true
-	configJSON := fmt.Sprintf("{\"rpc-tx-fee-cap\": %g,\"net-api-enabled\": %t}", txFeeCap, netApiEnabled)
+	enabledEthAPIs := []string{"internal-private-debug"}
+	configJSON := fmt.Sprintf("{\"rpc-tx-fee-cap\": %g,\"eth-apis\": %s}", txFeeCap, fmt.Sprintf("[%q]", enabledEthAPIs[0]))
	_, vm, _, _, _ := GenesisVM(t, false, genesisJSONApricotPhase0, configJSON, "")
	assert.Equal(t, vm.config.RPCTxFeeCap, txFeeCap, "Tx Fee Cap should be set")
-	assert.Equal(t, vm.config.NetAPIEnabled, netApiEnabled, "Net API Enabled should be set")
+	assert.Equal(t, vm.config.EthAPIs(), enabledEthAPIs, "EnabledEthAPIs should be set")
	assert.NoError(t, vm.Shutdown())
 }
 
 func TestVMConfigDefaults(t *testing.T) {
	txFeeCap := float64(11)
-	netApiEnabled := true
-	configJSON := fmt.Sprintf("{\"rpc-tx-fee-cap\": %g,\"net-api-enabled\": %t}", txFeeCap, netApiEnabled)
+	enabledEthAPIs := []string{"internal-private-debug"}
+	configJSON := fmt.Sprintf("{\"rpc-tx-fee-cap\": %g,\"eth-apis\": %s}", txFeeCap, fmt.Sprintf("[%q]", enabledEthAPIs[0]))
	_, vm, _, _, _ := GenesisVM(t, false, genesisJSONApricotPhase0, configJSON, "")
 
	var vmConfig Config
	vmConfig.SetDefaults()
	vmConfig.RPCTxFeeCap = txFeeCap
-	vmConfig.NetAPIEnabled = netApiEnabled
+	vmConfig.EnabledEthAPIs = enabledEthAPIs
	assert.Equal(t, vmConfig, vm.config, "VM Config should match default with overrides")
	assert.NoError(t, vm.Shutdown())
 }
@@ -336,6 +362,11 @@ func TestVMUpgrades(t *testing.T) {
			genesis:          genesisJSONApricotPhase4,
			expectedGasPrice: big.NewInt(0),
		},
+		{
+			name:             "Apricot Phase 5",
+			genesis:          genesisJSONApricotPhase5,
+			expectedGasPrice: big.NewInt(0),
+		},
	}
	for _, test := range genesisTests {
		t.Run(test.name, func(t *testing.T) {
@@ -394,2519 +425,63 @@ func TestVMUpgrades(t *testing.T) {
	}
 }
 
-// Simple test to ensure we can issue an import transaction followed by an export transaction
-// and they will be indexed correctly when accepted.
-func TestIssueAtomicTxs(t *testing.T) {
-	importAmount := uint64(50000000)
-	issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase2, "", "", map[ids.ShortID]uint64{
-		testShortIDAddrs[0]: importAmount,
-	})
-
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	importTx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blk, err := vm.BuildBlock()
```

```
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm.SetPreference(blk.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Accept(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk.Status(); status != choices.Accepted {
-		t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-	}
-
-	if lastAcceptedID, err := vm.LastAccepted(); err != nil {
-		t.Fatal(err)
-	} else if lastAcceptedID != blk.ID() {
-		t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk.ID(), lastAcceptedID)
-	}
-
-	exportTx, err := vm.newExportTx(vm.ctx.AVAXAssetID, importAmount-(2*params.AvalancheAtomicTxFee), vm.ctx.XChainID, testShortIDAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]}
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm.issueTx(exportTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blk2, err := vm.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk2.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk2.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := blk2.Accept(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk2.Status(); status != choices.Accepted {
-		t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-	}
-
-	if lastAcceptedID, err := vm.LastAccepted(); err != nil {
-		t.Fatal(err)
-	} else if lastAcceptedID != blk2.ID() {
-		t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk2.ID(), lastAcceptedID)
-	}
-
-	// Check that both atomic transactions were indexed as expected.
-	indexedImportTx, status, height, err := vm.getAtomicTx(importTx.ID())
-	assert.NoError(t, err)
-	assert.Equal(t, Accepted, status)
-	assert.Equal(t, uint64(1), height, "expected height of indexed import tx to be 1")
-	assert.Equal(t, indexedImportTx.ID(), importTx.ID(), "expected ID of indexed import tx to match original txID")
-
-	indexedExportTx, status, height, err := vm.getAtomicTx(exportTx.ID())
-	assert.NoError(t, err)
-	assert.Equal(t, Accepted, status)
-	assert.Equal(t, uint64(2), height, "expected height of indexed export tx to be 2")
-	assert.Equal(t, indexedExportTx.ID(), exportTx.ID(), "expected ID of indexed import tx to match original txID")
-}
-
-func TestBuildEthTxBlock(t *testing.T) {
-	importAmount := uint64(20000000)
-	issuer, vm, dbManager, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase2, "{\"pruning-enabled\":true}", "", map[ids.ShortID]uint64{
-		testShortIDAddrs[0]: importAmount,
-	})
-
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	newTxPoolHeadChan := make(chan core.NewTxPoolReorgEvent, 1)
-	vm.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan)
-
-	key, err := accountKeystore.NewKey(rand.Reader)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blk1, err := vm.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk1.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk1.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm.SetPreference(blk1.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk1.Accept(); err != nil {
-		t.Fatal(err)
-	}
-
```

```
-       newHead := <-newTxPoolHeadChan
-       if newHead.Head.Hash() != common.Hash(blk1.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-       errs := vm.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add tx at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer
-
-       blk2, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk2.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk2.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := blk2.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead = <-newTxPoolHeadChan
-       if newHead.Head.Hash() != common.Hash(blk2.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       if status := blk2.Status(); status != choices.Accepted {
-               t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-       }
-
-       lastAcceptedID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-       if lastAcceptedID != blk2.ID() {
-               t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk2.ID(), lastAcceptedID)
-       }
-
-       ethBlk1 := blk1.(*chain.BlockWrapper).Block.(*Block).ethBlock
-       if ethBlk1Root := ethBlk1.Root(); !vm.chain.BlockChain().HasState(ethBlk1Root) {
-               t.Fatalf("Expected blk1 state root to not yet be pruned after blk2 was accepted because of tip buffer")
-       }
-
-       // Clear the cache and ensure that GetBlock returns internal blocks with the correct status
-       vm.State.Flush()
-       blk2Refreshed, err := vm.GetBlockInternal(blk2.ID())
-       if err != nil {
-               t.Fatal(err)
-       }
-       if status := blk2Refreshed.Status(); status != choices.Accepted {
-               t.Fatalf("Expected refreshed blk2 to be Accepted, but found status: %s", status)
-       }
-
-       blk1RefreshedID := blk2Refreshed.Parent()
-       blk1Refreshed, err := vm.GetBlockInternal(blk1RefreshedID)
-       if err != nil {
-               t.Fatal(err)
-       }
-       if status := blk1Refreshed.Status(); status != choices.Accepted {
-               t.Fatalf("Expected refreshed blk1 to be Accepted, but found status: %s", status)
-       }
-
-       if blk1Refreshed.ID() != blk1.ID() {
-               t.Fatalf("Found unexpected blkID for parent of blk2")
-       }
-
-       restartedVM := &VM{}
-       if err := restartedVM.Initialize(
-               NewContext(),
-               dbManager,
-               []byte(genesisJSONApricotPhase2),
-               []byte(""),
-               []byte("{\"pruning-enabled\":true}"),
-               issuer,
-               []*engCommon.Fx{},
-               nil,
-       ); err != nil {
-               t.Fatal(err)
-       }
-
-       // State root should not have been committed and discarded on restart
-       if ethBlk1Root := ethBlk1.Root(); restartedVM.chain.BlockChain().HasState(ethBlk1Root) {
-               t.Fatalf("Expected blk1 state root to be pruned after blk2 was accepted on top of it in pruning mode")
-       }
-
-       // State root should be committed when accepted tip on shutdown
-       ethBlk2 := blk2.(*chain.BlockWrapper).Block.(*Block).ethBlock
-       if ethBlk2Root := ethBlk2.Root(); !restartedVM.chain.BlockChain().HasState(ethBlk2Root) {
-               t.Fatalf("Expected blk2 state root to not be pruned after shutdown (last accepted tip should be committed)")
-       }
-}
-
-func TestConflictingImportTxs(t *testing.T) {
-       importAmount := uint64(10000000)
-       issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-               testShortIDAddrs[1]: importAmount,
-               testShortIDAddrs[2]: importAmount,
-       })
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       conflictKey, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTxs := make([]*Tx, 0, 3)
-       conflictTxs := make([]*Tx, 0, 3)
-       for i, key := range testKeys {
-               importTx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[i], initialBaseFee, []*crypto.PrivateKeySECP256K1R{key})
-               if err != nil {
-                       t.Fatal(err)
-               }
-               importTxs = append(importTxs, importTx)
-
-               conflictTx, err := vm.newImportTx(vm.ctx.XChainID, conflictKey.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{key})
-               if err != nil {
-                       t.Fatal(err)
-               }
-               conflictTxs = append(conflictTxs, conflictTx)
-       }
-
-       expectedParentBlkID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-       for i, tx := range importTxs {
-               if err := vm.issueTx(tx, true /*=local*/); err != nil {
-                       t.Fatal(err)
-               }
-
-               <-issuer
-
-               blk, err := vm.BuildBlock()
-               if err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := blk.Verify(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if status := blk.Status(); status != choices.Processing {
-                       t.Fatalf("Expected status of built block %d to be %s, but found %s", i, choices.Processing, status)
-               }
-
-               if parentID := blk.Parent(); parentID != expectedParentBlkID {
-                       t.Fatalf("Expected parent to have blockID %s, but found %s", expectedParentBlkID, parentID)
-               }
-
-               expectedParentBlkID = blk.ID()
-               if err := vm.SetPreference(blk.ID()); err != nil {
-                       t.Fatal(err)
-               }
-       }
-
-       for i, tx := range conflictTxs {
-               if err := vm.issueTx(tx, true /*=local*/); err == nil {
-                       t.Fatal("Expected issueTx to fail due to conflicting transaction")
-               }
-               // Force issue transaction directly to the mempool
-               if err := vm.mempool.ForceAddTx(tx); err != nil {
-                       t.Fatal(err)
-               }
-               <-issuer
-
-               _, err = vm.BuildBlock()
-               // The new block is verified in BuildBlock, so
-               // BuildBlock should fail due to an attempt to
-               // double spend an atomic UTXO.
-               if err == nil {
-                       t.Fatalf("Block verification should have failed in BuildBlock %d due to double spending atomic UTXO", i)
-               }
+func TestConfigureLogLevel(t *testing.T) {
+       configTests := []struct {
+               name                     string
+               logConfig                string
+               genesisJSON, upgradeJSON string
+               expectedErr              string
+       }{
+               {
+                       name:        "Log level info",
+                       logConfig:   "{\"log-level\": \"info\"}",
+                       genesisJSON: genesisJSONApricotPhase2,
+                       upgradeJSON: "",
+                       expectedErr: "",
+               },
+               {
+                       name:        "Invalid log level",
+                       logConfig:   "{\"log-level\": \"cchain\"}",
+                       genesisJSON: genesisJSONApricotPhase3,
+                       upgradeJSON: "",
+                       expectedErr: "failed to initialize logger due to",
+               },
        }
-}
-
-// Regression test to ensure that after accepting block A
-// then calling SetPreference on block B (when it becomes preferred)
-// and the head of a longer chain (block D) does not corrupt the
-// canonical chain.
-//  A
-// / \
-// B  C
-//    |
-//    D
-func TestSetPreferenceRace(t *testing.T) {
-       // Create two VMs which will agree on block A and then
-       // build the two distinct preferred chains above
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "{\"pruning-enabled\":true}", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "{\"pruning-enabled\":true}", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
+       for _, test := range configTests {
+               t.Run(test.name, func(t *testing.T) {
+                       vm := &VM{}
+                       ctx, dbManager, genesisBytes, issuer, _ := setupGenesis(t, test.genesisJSON)
+                       appSender := &engCommon.SenderTest{}
+                       appSender.CantSendAppGossip = true
+                       appSender.SendAppGossipF = func([]byte) error { return nil }
+                       err := vm.Initialize(
+                               ctx,
+                               dbManager,
+                               genesisBytes,
+                               []byte(""),
+                               []byte(test.logConfig),
+                               issuer,
+                               []*engCommon.Fx{},
+                               appSender,
+                       )
+                       if len(test.expectedErr) == 0 && err != nil {
+                               t.Fatal(err)
+                       } else if len(test.expectedErr) > 0 {
+                               if err == nil {
+                                       t.Fatalf("initialize should have failed due to %s", test.expectedErr)
+                               } else if !strings.Contains(err.Error(), test.expectedErr) {
+                                       t.Fatalf("Expected initialize to fail due to %s, but failed with %s", test.expectedErr, err.Error())
+                               }
+                       }

-       defer func() {
```

```
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-       newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer1
-
-       vm1BlkA, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := vm1BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if status := vm1BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       if err := vm2BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-       if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkA.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-       if err := vm2BlkA.Accept(); err != nil {
-               t.Fatalf("VM2 failed to accept block: %s", err)
-       }
-
-       newHead := <-newTxPoolHeadChan1
-       if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       // Create list of 10 successive transactions to build block A on vm1
-       // and to be split into two separate blocks on VM2
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-
-       var errs []error
-
-       // Add the remote transactions, build the block, and set VM1's preference for block A
-       errs = vm1.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer1
-
-       vm1BlkB, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := vm1BlkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       // Split the transactions over two blocks, and set VM2's preference to them in sequence
-       // after building each block
-       // Block C
-       errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkC, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkC on VM2: %s", err)
-       }
-
```

```
-	if err := vm2BlkC.Verify(); err != nil {
-		t.Fatalf("BlkC failed verification on VM2: %s", err)
-	}
-
-	if status := vm2BlkC.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block C to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm2.SetPreference(vm2BlkC.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	newHead = <-newTxPoolHeadChan2
-	if newHead.Head.Hash() != common.Hash(vm2BlkC.ID()) {
-		t.Fatalf("Expected new block to match")
-	}
-
-	// Block D
-	errs = vm2.chain.AddRemoteTxsSync(txs[5:10])
-	for i, err := range errs {
-		if err != nil {
-			t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-		}
-	}
-
-	<-issuer2
-	vm2BlkD, err := vm2.BuildBlock()
-	if err != nil {
-		t.Fatalf("Failed to build BlkD on VM2: %s", err)
-	}
-
-	if err := vm2BlkD.Verify(); err != nil {
-		t.Fatalf("BlkD failed verification on VM2: %s", err)
-	}
-
-	if status := vm2BlkD.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block D to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm2.SetPreference(vm2BlkD.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	// VM1 receives blkC and blkD from VM1
-	// and happens to call SetPreference on blkD without ever calling SetPreference
-	// on blkC
-	// Here we parse them in reverse order to simulate receiving a chain from the tip
-	// back to the last accepted block as would typically be the case in the consensus
-	// engine
-	vm1BlkD, err := vm1.ParseBlock(vm2BlkD.Bytes())
-	if err != nil {
-		t.Fatalf("VM1 errored parsing blkD: %s", err)
-	}
-	vm1BlkC, err := vm1.ParseBlock(vm2BlkC.Bytes())
-	if err != nil {
-		t.Fatalf("VM1 errored parsing blkC: %s", err)
-	}
-
-	// The blocks must be verified in order. This invariant is maintained
-	// in the consensus engine.
-	if err := vm1BlkC.Verify(); err != nil {
-		t.Fatalf("VM1 BlkC failed verification: %s", err)
-	}
-	if err := vm1BlkD.Verify(); err != nil {
-		t.Fatalf("VM1 BlkD failed verification: %s", err)
-	}
-
-	// Set VM1's preference to blockD, skipping blockC
-	if err := vm1.SetPreference(vm1BlkD.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	// Accept the longer chain on both VMs and ensure there are no errors
-	// VM1 Accepts the blocks in order
-	if err := vm1BlkC.Accept(); err != nil {
-		t.Fatalf("VM1 BlkC failed on accept: %s", err)
-	}
-	if err := vm1BlkD.Accept(); err != nil {
-		t.Fatalf("VM1 BlkC failed on accept: %s", err)
-	}
-
-	// VM2 Accepts the blocks in order
-	if err := vm2BlkC.Accept(); err != nil {
-		t.Fatalf("VM2 BlkC failed on accept: %s", err)
-	}
-	if err := vm2BlkD.Accept(); err != nil {
-		t.Fatalf("VM2 BlkC failed on accept: %s", err)
-	}
-
-	log.Info("Validating canonical chain")
-	// Verify the Canonical Chain for Both VMs
-	if err := vm2.chain.ValidateCanonicalChain(); err != nil {
-		t.Fatalf("VM2 failed canonical chain verification due to: %s", err)
-	}
-
-	if err := vm1.chain.ValidateCanonicalChain(); err != nil {
-		t.Fatalf("VM1 failed canonical chain verification due to: %s", err)
-	}
-}
-
-func TestConflictingTransitiveAncestryWithGap(t *testing.T) {
-	key, err := accountKeystore.NewKey(rand.Reader)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	key0 := testKeys[0]
-	addr0 := key0.PublicKey().Address()
-
-	key1 := testKeys[1]
-	addr1 := key1.PublicKey().Address()
-
-	importAmount := uint64(1000000000)
-
-	issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "",
-		map[ids.ShortID]uint64{
-			addr0: importAmount,
-			addr1: importAmount,
-		})
-
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	newTxPoolHeadChan := make(chan core.NewTxPoolReorgEvent, 1)
-	vm.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan)
-
-	importTx0A, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{key0})
-	if err != nil {
-		t.Fatal(err)
-	}
-	// Create a conflicting transaction
```

```
-       importTx0B, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[2], initialBaseFee, []*crypto.PrivateKeySECP256K1R{key0})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx0A, true /*=local*/); err != nil {
-               t.Fatalf("Failed to issue importTx0A: %s", err)
-       }
-
-       <-issuer
-
-       blk0, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := blk0.Verify(); err != nil {
-               t.Fatalf("Block failed verification: %s", err)
-       }
-
-       if err := vm.SetPreference(blk0.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead := <-newTxPoolHeadChan
-       if newHead.Head.Hash() != common.Hash(blk0.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       tx := types.NewTransaction(0, key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-       signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       // Add the remote transactions, build the block, and set VM1's preference for block A
-       errs := vm.chain.AddRemoteTxsSync([]*types.Transaction{signedTx})
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer
-
-       blk1, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build blk1: %s", err)
-       }
-
-       if err := blk1.Verify(); err != nil {
-               t.Fatalf("blk1 failed verification due to %s", err)
-       }
-
-       if err := vm.SetPreference(blk1.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       importTx1, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{key1})
-       if err != nil {
-               t.Fatalf("Failed to issue importTx1 due to: %s", err)
-       }
-
-       if err := vm.issueTx(importTx1, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk2, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := blk2.Verify(); err != nil {
-               t.Fatalf("Block failed verification: %s", err)
-       }
-
-       if err := vm.SetPreference(blk2.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx0B, true /*=local*/); err == nil {
-               t.Fatalf("Should not have been able to issue import tx with conflict")
-       }
-       // Force issue transaction directly into the mempool
-       if err := vm.mempool.ForceAddTx(importTx0B); err != nil {
-               t.Fatal(err)
-       }
-       <-issuer
-
-       _, err = vm.BuildBlock()
-       if err == nil {
-               t.Fatal("Shouldn't have been able to build an invalid block")
-       }
-}
-
-func TestBonusBlocksTxs(t *testing.T) {
-       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase0, "", "")
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       importAmount := uint64(10000000)
-       utxoID := avax.UTXOID{TxID: ids.GenerateTestID()}
-
-       utxo := &avax.UTXO{
-               UTXOID: utxoID,
-               Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-               Out: &secp256k1fx.TransferOutput{
-                       Amt: importAmount,
-                       OutputOwners: secp256k1fx.OutputOwners{
-                               Threshold: 1,
-                               Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-                       },
-               },
-       }
-       utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-       inputID := utxo.InputID()
-       if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-               Key:   inputID[:],
-               Value: utxoBytes,
-               Traits: [][]byte{
-                       testKeys[0].PublicKey().Address().Bytes(),
-               },
-       }}}}); err != nil {
```

```
-               t.Fatal(err)
-       }
-
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       bonusBlocks.Add(blk.ID())
-
-       // Remove the UTXOs from shared memory, so that non-bonus blocks will fail verification
-       if err := vm.ctx.SharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.XChainID: {RemoveRequests: [][]byte{inputID[:]}}}); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm.SetPreference(blk.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Accepted {
-               t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-       }
-
-       lastAcceptedID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-       if lastAcceptedID != blk.ID() {
-               t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk.ID(), lastAcceptedID)
-       }
-}
-
-// Regression test to ensure that a VM that accepts block A and B
-// will not attempt to orphan either when verifying blocks C and D
-// from another VM (which have a common ancestor under the finalized
-// frontier).
-//   A
-//  / \
-// B   C
-//
-// verifies block B and C, then Accepts block B. Then we test to ensure
-// that the VM defends against any attempt to set the preference or to
-// accept block C, which should be an orphaned block at this point and
-// get rejected.
-func TestReorgProtection(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "{\"pruning-enabled\":false}", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "{\"pruning-enabled\":false}", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-       newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer1
-
-       vm1BlkA, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := vm1BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if status := vm1BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       if err := vm2BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-       if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkA.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-       if err := vm2BlkA.Accept(); err != nil {
-               t.Fatalf("VM2 failed to accept block: %s", err)
-       }
-
-       newHead := <-newTxPoolHeadChan1
-       if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       // Create list of 10 successive transactions to build block A on vm1
-       // and to be split into two separate blocks on VM2
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-
-       var errs []error
-
-       // Add the remote transactions, build the block, and set VM1's preference for block A
-       errs = vm1.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer1
-
-       vm1BlkB, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := vm1BlkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       // Split the transactions over two blocks, and set VM2's preference to them in sequence
-       // after building each block
-       // Block C
-       errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkC, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkC on VM2: %s", err)
-       }
-
-       if err := vm2BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkC.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-
-       vm1BlkC, err := vm1.ParseBlock(vm2BlkC.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-
-       if err := vm1BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       // Accept B, such that block C should get Rejected.
-       if err := vm1BlkB.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-
-       // The below (setting preference blocks that have a common ancestor
-       // with the preferred chain lower than the last finalized block)
-       // should NEVER happen. However, the VM defends against this
-       // just in case.
-       if err := vm1.SetPreference(vm1BlkC.ID()); !strings.Contains(err.Error(), "cannot orphan finalized block") {
-               t.Fatalf("Unexpected error when setting preference that would trigger reorg: %s", err)
-       }
-
-       if err := vm1BlkC.Accept(); !strings.Contains(err.Error(), "expected accepted block to have parent") {
-               t.Fatalf("Unexpected error when setting block at finalized height: %s", err)
-       }
-}
-
-// Regression test to ensure that a VM that accepts block C while preferring
-// block B will trigger a reorg.
-//   A
-//  / \
-// B   C
-func TestNonCanonicalAccept(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
```

```
-
-	newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-	vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-	newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-	vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-	key, err := accountKeystore.NewKey(rand.Reader)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer1
-
-	vm1BlkA, err := vm1.BuildBlock()
-	if err != nil {
-		t.Fatalf("Failed to build block with import transaction: %s", err)
-	}
-
-	if err := vm1BlkA.Verify(); err != nil {
-		t.Fatalf("Block failed verification on VM1: %s", err)
-	}
-
-	if status := vm1BlkA.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-	if err != nil {
-		t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-	}
-	if err := vm2BlkA.Verify(); err != nil {
-		t.Fatalf("Block failed verification on VM2: %s", err)
-	}
-	if status := vm2BlkA.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-	}
-	if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm1BlkA.Accept(); err != nil {
-		t.Fatalf("VM1 failed to accept block: %s", err)
-	}
-	if err := vm2BlkA.Accept(); err != nil {
-		t.Fatalf("VM2 failed to accept block: %s", err)
-	}
-
-	newHead := <-newTxPoolHeadChan1
-	if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-		t.Fatalf("Expected new block to match")
-	}
-	newHead = <-newTxPoolHeadChan2
-	if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-		t.Fatalf("Expected new block to match")
-	}
-
-	// Create list of 10 successive transactions to build block A on vm1
-	// and to be split into two separate blocks on VM2
-	txs := make([]*types.Transaction, 10)
-	for i := 0; i < 10; i++ {
-		tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-		signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-		if err != nil {
-			t.Fatal(err)
-		}
-		txs[i] = signedTx
-	}
-
-	var errs []error
-
-	// Add the remote transactions, build the block, and set VM1's preference for block A
-	errs = vm1.chain.AddRemoteTxsSync(txs)
-	for i, err := range errs {
-		if err != nil {
-			t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-		}
-	}
-
-	<-issuer1
-
-	vm1BlkB, err := vm1.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm1BlkB.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := vm1BlkB.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	vm1.chain.BlockChain().GetVMConfig().AllowUnfinalizedQueries = true
-
-	blkBHeight := vm1BlkB.Height()
-	blkBHash := vm1BlkB.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-	if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkBHash {
-		t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkBHash.Hex(), b.Hash().Hex())
-	}
-
-	errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-	for i, err := range errs {
-		if err != nil {
-			t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-		}
-	}
-
-	<-issuer2
-	vm2BlkC, err := vm2.BuildBlock()
-	if err != nil {
-		t.Fatalf("Failed to build BlkC on VM2: %s", err)
-	}
-
-	vm1BlkC, err := vm1.ParseBlock(vm2BlkC.Bytes())
-	if err != nil {
-		t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-
-       if err := vm1BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if err := vm1BlkC.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-
-       blkCHash := vm1BlkC.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkCHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkCHash.Hex(), b.Hash().Hex())
-       }
-}
-
-// Regression test to ensure that a VM that verifies block B, C, then
-// D (preferring block B) does not trigger a reorg through the re-verification
-// of block C or D.
-//   A
-//  / \
-// B   C
-//     |
-//     D
-func TestStickyPreference(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-       newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer1
-
-       vm1BlkA, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := vm1BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if status := vm1BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       if err := vm2BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-       if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkA.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-       if err := vm2BlkA.Accept(); err != nil {
-               t.Fatalf("VM2 failed to accept block: %s", err)
-       }
-
-       newHead := <-newTxPoolHeadChan1
-       if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       // Create list of 10 successive transactions to build block A on vm1
-       // and to be split into two separate blocks on VM2
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-
-       var errs []error
-
-       // Add the remote transactions, build the block, and set VM1's preference for block A
-       errs = vm1.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer1
```

```
-
-       vm1BlkB, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := vm1BlkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm1.chain.BlockChain().GetVMConfig().AllowUnfinalizedQueries = true
-
-       blkBHeight := vm1BlkB.Height()
-       blkBHash := vm1BlkB.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkBHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkBHash.Hex(), b.Hash().Hex())
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkC, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkC on VM2: %s", err)
-       }
-
-       if err := vm2BlkC.Verify(); err != nil {
-               t.Fatalf("BlkC failed verification on VM2: %s", err)
-       }
-
-       if status := vm2BlkC.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block C to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm2.SetPreference(vm2BlkC.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkC.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[5:])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkD, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkD on VM2: %s", err)
-       }
-
-       // Parse blocks produced in vm2
-       vm1BlkC, err := vm1.ParseBlock(vm2BlkC.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       blkCHash := vm1BlkC.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-
-       vm1BlkD, err := vm1.ParseBlock(vm2BlkD.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       blkDHeight := vm1BlkD.Height()
-       blkDHash := vm1BlkD.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-
-       // Should be no-ops
-       if err := vm1BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if err := vm1BlkD.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkBHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkBHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.GetBlockByNumber(blkDHeight); b != nil {
-               t.Fatalf("expected block at %d to be nil but got %s", blkDHeight, b.Hash().Hex())
-       }
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkBHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkBHash.Hex(), b.Hash().Hex())
-       }
-
-       // Should still be no-ops on re-verify
-       if err := vm1BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if err := vm1BlkD.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkBHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkBHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.GetBlockByNumber(blkDHeight); b != nil {
-               t.Fatalf("expected block at %d to be nil but got %s", blkDHeight, b.Hash().Hex())
-       }
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkBHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkBHash.Hex(), b.Hash().Hex())
-       }
-
-       // Should be queryable after setting preference to side chain
-       if err := vm1.SetPreference(vm1BlkD.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkCHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkCHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.GetBlockByNumber(blkDHeight); b.Hash() != blkDHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkDHeight, blkDHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkDHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkDHash.Hex(), b.Hash().Hex())
-       }
-
-       // Attempt to accept out of order
-       if err := vm1BlkD.Accept(); !strings.Contains(err.Error(), "expected accepted block to have parent") {
```

```
-               t.Fatalf("unexpected error when accepting out of order block: %s", err)
-       }
-
-       // Accept in order
-       if err := vm1BlkC.Accept(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if err := vm1BlkD.Accept(); err != nil {
-               t.Fatalf("Block failed acceptance on VM1: %s", err)
-       }
-
-       // Ensure queryable after accepting
-       if b := vm1.chain.GetBlockByNumber(blkBHeight); b.Hash() != blkCHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkBHeight, blkCHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.GetBlockByNumber(blkDHeight); b.Hash() != blkDHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkDHeight, blkDHash.Hex(), b.Hash().Hex())
-       }
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkDHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkDHash.Hex(), b.Hash().Hex())
-       }
-}
-
-// Regression test to ensure that a VM that prefers block B is able to parse
-// block C but unable to parse block D because it names B as an uncle, which
-// are not supported.
-//   A
-//  / \
-// B   C
-//     |
-//     D
-func TestUncleBlock(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-       newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer1
-
-       vm1BlkA, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := vm1BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if status := vm1BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       if err := vm2BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-       if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkA.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-       if err := vm2BlkA.Accept(); err != nil {
-               t.Fatalf("VM2 failed to accept block: %s", err)
-       }
-
-       newHead := <-newTxPoolHeadChan1
-       if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-
-       var errs []error
-
-       errs = vm1.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer1
-
-       vm1BlkB, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := vm1BlkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkC, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkC on VM2: %s", err)
-       }
-
-       if err := vm2BlkC.Verify(); err != nil {
-               t.Fatalf("BlkC failed verification on VM2: %s", err)
-       }
-
-       if status := vm2BlkC.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block C to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm2.SetPreference(vm2BlkC.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkC.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[5:10])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-       vm2BlkD, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkD on VM2: %s", err)
-       }
-
-       // Create uncle block from blkD
-       blkDEthBlock := vm2BlkD.(*chain.BlockWrapper).Block.(*Block).ethBlock
-       uncles := []*types.Header{vm1BlkB.(*chain.BlockWrapper).Block.(*Block).ethBlock.Header()}
-       uncleBlockHeader := types.CopyHeader(blkDEthBlock.Header())
-       uncleBlockHeader.UncleHash = types.CalcUncleHash(uncles)
-
-       uncleEthBlock := types.NewBlock(
-               uncleBlockHeader,
-               blkDEthBlock.Transactions(),
-               uncles,
-               nil,
-               new(trie.Trie),
-               blkDEthBlock.ExtData(),
-               false,
-       )
-       uncleBlock := &Block{
-               vm:       vm2,
-               ethBlock: uncleEthBlock,
-               id:       ids.ID(uncleEthBlock.Hash()),
-       }
-       if err := uncleBlock.Verify(); !errors.Is(err, errUnclesUnsupported) {
-               t.Fatalf("VM2 should have failed with %q but got %q", errUnclesUnsupported, err.Error())
-       }
-       if _, err := vm1.ParseBlock(vm2BlkC.Bytes()); err != nil {
-               t.Fatalf("VM1 errored parsing blkC: %s", err)
-       }
-       if _, err := vm1.ParseBlock(uncleBlock.Bytes()); !errors.Is(err, errUnclesUnsupported) {
-               t.Fatalf("VM1 should have failed with %q but got %q", errUnclesUnsupported, err.Error())
-       }
-}
-
-// Regression test to ensure that a VM that is not able to parse a block that
-// contains no transactions.
-func TestEmptyBlock(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       // Create empty block from blkA
-       ethBlock := blk.(*chain.BlockWrapper).Block.(*Block).ethBlock
-
-       emptyEthBlock := types.NewBlock(
-               types.CopyHeader(ethBlock.Header()),
-               nil,
-               nil,
-               nil,
-               new(trie.Trie),
-               nil,
-               false,
-       )
-
-       if len(emptyEthBlock.ExtData()) != 0 || emptyEthBlock.Header().ExtDataHash != (common.Hash{}) {
-               t.Fatalf("emptyEthBlock should not have any extra data")
-       }
-
-       emptyBlock := &Block{
-               vm:       vm,
-               ethBlock: emptyEthBlock,
-               id:       ids.ID(emptyEthBlock.Hash()),
-       }
-
-       if _, err := vm.ParseBlock(emptyBlock.Bytes()); !errors.Is(err, errEmptyBlock) {
-               t.Fatalf("VM should have failed with errEmptyBlock but got %s", err.Error())
-       }
-       if err := emptyBlock.Verify(); !errors.Is(err, errEmptyBlock) {
-               t.Fatalf("block should have failed verification with errEmptyBlock but got %s", err.Error())
-       }
-}
-
-// Regression test to ensure that a VM that verifies block B, C, then
-// D (preferring block B) reorgs when C and then D are accepted.
-//   A
-//  / \
-// B   C
-//     |
-//     D
-func TestAcceptReorg(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer1, vm1, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-       issuer2, vm2, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm1.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-
-               if err := vm2.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan1 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm1.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan1)
-       newTxPoolHeadChan2 := make(chan core.NewTxPoolReorgEvent, 1)
-       vm2.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan2)
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-       importTx, err := vm1.newImportTx(vm1.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer1
-
-       vm1BlkA, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := vm1BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       if status := vm1BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       vm2BlkA, err := vm2.ParseBlock(vm1BlkA.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-       if err := vm2BlkA.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM2: %s", err)
-       }
-       if status := vm2BlkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of block on VM2 to be %s, but found %s", choices.Processing, status)
-       }
-       if err := vm2.SetPreference(vm2BlkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkA.Accept(); err != nil {
-               t.Fatalf("VM1 failed to accept block: %s", err)
-       }
-       if err := vm2BlkA.Accept(); err != nil {
-               t.Fatalf("VM2 failed to accept block: %s", err)
-       }
-
-       newHead := <-newTxPoolHeadChan1
-       if newHead.Head.Hash() != common.Hash(vm1BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkA.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       // Create list of 10 successive transactions to build block A on vm1
-       // and to be split into two separate blocks on VM2
-       txs := make([]*types.Transaction, 10)
-       for i := 0; i < 10; i++ {
-               tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-               signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm1.chainID), key.PrivateKey)
-               if err != nil {
-                       t.Fatal(err)
-               }
-               txs[i] = signedTx
-       }
-
-       // Add the remote transactions, build the block, and set VM1's preference
-       // for block B
-       errs := vm1.chain.AddRemoteTxsSync(txs)
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM1 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer1
-
-       vm1BlkB, err := vm1.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := vm1BlkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm1.SetPreference(vm1BlkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[0:5])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-
-       vm2BlkC, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkC on VM2: %s", err)
-       }
-
-       if err := vm2BlkC.Verify(); err != nil {
-               t.Fatalf("BlkC failed verification on VM2: %s", err)
-       }
-
-       if err := vm2.SetPreference(vm2BlkC.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead = <-newTxPoolHeadChan2
-       if newHead.Head.Hash() != common.Hash(vm2BlkC.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       errs = vm2.chain.AddRemoteTxsSync(txs[5:])
-       for i, err := range errs {
-               if err != nil {
-                       t.Fatalf("Failed to add transaction to VM2 at index %d: %s", i, err)
-               }
-       }
-
-       <-issuer2
-
-       vm2BlkD, err := vm2.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build BlkD on VM2: %s", err)
-       }
-
-       // Parse blocks produced in vm2
-       vm1BlkC, err := vm1.ParseBlock(vm2BlkC.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-
-       vm1BlkD, err := vm1.ParseBlock(vm2BlkD.Bytes())
-       if err != nil {
-               t.Fatalf("Unexpected error parsing block from vm2: %s", err)
-       }
-
-       if err := vm1BlkC.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-       if err := vm1BlkD.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM1: %s", err)
-       }
-
-       blkBHash := vm1BlkB.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkBHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkBHash.Hex(), b.Hash().Hex())
-       }
-
-       if err := vm1BlkC.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       blkCHash := vm1BlkC.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkCHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkCHash.Hex(), b.Hash().Hex())
-       }
-
-       if err := vm1BlkB.Reject(); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm1BlkD.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       blkDHash := vm1BlkD.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-       if b := vm1.chain.BlockChain().CurrentBlock(); b.Hash() != blkDHash {
-               t.Fatalf("expected current block to have hash %s but got %s", blkDHash.Hex(), b.Hash().Hex())
-       }
-}
-
-func TestFutureBlock(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
```

```
-
-	if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blkA, err := vm.BuildBlock()
-	if err != nil {
-		t.Fatalf("Failed to build block with import transaction: %s", err)
-	}
-
-	// Create empty block from blkA
-	blkAEthBlock := blkA.(*chain.BlockWrapper).Block.(*Block).ethBlock
-
-	modifiedHeader := types.CopyHeader(blkAEthBlock.Header())
-	// Set the VM's clock to the time of the produced block
-	vm.clock.Set(time.Unix(int64(modifiedHeader.Time), 0))
-	// Set the modified time to exceed the allowed future time
-	modifiedTime := modifiedHeader.Time + uint64(maxFutureBlockTime.Seconds()+1)
-	modifiedHeader.Time = modifiedTime
-	modifiedBlock := types.NewBlock(
-		modifiedHeader,
-		nil,
-		nil,
-		nil,
-		new(trie.Trie),
-		blkAEthBlock.ExtData(),
-		false,
-	)
-
-	futureBlock := &Block{
-		vm:       vm,
-		ethBlock: modifiedBlock,
-		id:       ids.ID(modifiedBlock.Hash()),
-	}
-
-	if err := futureBlock.Verify(); err == nil {
-		t.Fatal("Future block should have failed verification due to block timestamp too far in the future")
-	} else if !strings.Contains(err.Error(), "block timestamp is too far in the future") {
-		t.Fatalf("Expected error to be block timestamp too far in the future but found %s", err)
-	}
-}
-
-// Regression test to ensure we can build blocks if we are starting with the
-// Apricot Phase 1 ruleset in genesis.
-func TestBuildApricotPhase1Block(t *testing.T) {
-	importAmount := uint64(1000000000)
-	issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase1, "", "", map[ids.ShortID]uint64{
-		testShortIDAddrs[0]: importAmount,
-	})
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	newTxPoolHeadChan := make(chan core.NewTxPoolReorgEvent, 1)
-	vm.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan)
-
-	key, err := accountKeystore.NewKey(rand.Reader)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blk, err := vm.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm.SetPreference(blk.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Accept(); err != nil {
-		t.Fatal(err)
-	}
-
-	newHead := <-newTxPoolHeadChan
-	if newHead.Head.Hash() != common.Hash(blk.ID()) {
-		t.Fatalf("Expected new block to match")
-	}
-
-	txs := make([]*types.Transaction, 10)
-	for i := 0; i < 5; i++ {
-		tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-		signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-		if err != nil {
-			t.Fatal(err)
-		}
-		txs[i] = signedTx
-	}
-	for i := 5; i < 10; i++ {
-		tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.ApricotPhase1MinGasPrice), nil)
-		signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-		if err != nil {
-			t.Fatal(err)
-		}
-		txs[i] = signedTx
-	}
-	errs := vm.chain.AddRemoteTxsSync(txs)
-	for i, err := range errs {
-		if err != nil {
-			t.Fatalf("Failed to add tx at index %d: %s", i, err)
-		}
-	}
-
-	<-issuer
-
-	blk, err = vm.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Verify(); err != nil {
```

```
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Accepted {
-               t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-       }
-
-       lastAcceptedID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-       if lastAcceptedID != blk.ID() {
-               t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk.ID(), lastAcceptedID)
-       }
-
-       // Confirm all txs are present
-       ethBlkTxs := vm.chain.GetBlockByNumber(2).Transactions()
-       for i, tx := range txs {
-               if len(ethBlkTxs) <= i {
-                       t.Fatalf("missing transactions expected: %d but found: %d", len(txs), len(ethBlkTxs))
-               }
-               if ethBlkTxs[i].Hash() != tx.Hash() {
-                       t.Fatalf("expected tx at index %d to have hash: %x but has: %x", i, txs[i].Hash(), tx.Hash())
-               }
-       }
-}
-
-func TestLastAcceptedBlockNumberAllow(t *testing.T) {
-       importAmount := uint64(1000000000)
-       issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase0, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: importAmount,
-       })
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       key, err := accountKeystore.NewKey(rand.Reader)
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatalf("Failed to build block with import transaction: %s", err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatalf("Block failed verification on VM: %s", err)
-       }
-
-       if status := blk.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm.SetPreference(blk.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       blkHeight := blk.Height()
-       blkHash := blk.(*chain.BlockWrapper).Block.(*Block).ethBlock.Hash()
-
-       vm.chain.BlockChain().GetVMConfig().AllowUnfinalizedQueries = true
-
-       ctx := context.Background()
-       b, err := vm.chain.APIBackend().BlockByNumber(ctx, rpc.BlockNumber(blkHeight))
-       if err != nil {
-               t.Fatal(err)
-       }
-       if b.Hash() != blkHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkHeight, blkHash.Hex(), b.Hash().Hex())
-       }
-
-       vm.chain.BlockChain().GetVMConfig().AllowUnfinalizedQueries = false
-
-       _, err = vm.chain.APIBackend().BlockByNumber(ctx, rpc.BlockNumber(blkHeight))
-       if !errors.Is(err, eth.ErrUnfinalizedData) {
-               t.Fatalf("expected ErrUnfinalizedData but got %s", err.Error())
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatalf("VM failed to accept block: %s", err)
-       }
-
-       if b := vm.chain.GetBlockByNumber(blkHeight); b.Hash() != blkHash {
-               t.Fatalf("expected block at %d to have hash %s but got %s", blkHeight, blkHash.Hex(), b.Hash().Hex())
-       }
-}
-
-// Builds [blkA] with a virtuous import transaction and [blkB] with a separate import transaction
-// that does not conflict. Accepts [blkB] and rejects [blkA], then asserts that the virtuous atomic
-// transaction in [blkA] is correctly re-issued into the atomic transaction mempool.
-func TestReissueAtomicTx(t *testing.T) {
-       issuer, vm, _, _, _ := GenesisVMWithUTXOs(t, true, genesisJSONApricotPhase1, "", "", map[ids.ShortID]uint64{
-               testShortIDAddrs[0]: 10000000,
-               testShortIDAddrs[1]: 10000000,
-       })
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       genesisBlkID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       importTx, err := vm.newImportTx(vm.ctx.XChainID, testEthAddrs[0], initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-       if err != nil {
-               t.Fatal(err)
-       }
```

```
-
-       if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blkA, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blkA.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm.SetPreference(blkA.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       // SetPreference to parent before rejecting (will rollback state to genesis
-       // so that atomic transaction can be reissued, otherwise current block will
-       // conflict with UTXO to be reissued)
-       if err := vm.SetPreference(genesisBlkID); err != nil {
-               t.Fatal(err)
-       }
-
-       // Rejecting [blkA] should cause [importTx] to be re-issued into the mempool.
-       if err := blkA.Reject(); err != nil {
-               t.Fatal(err)
-       }
-
-       // Sleep for a minimum of two seconds to ensure that [blkB] will have a different timestamp
-       // than [blkA] so that the block will be unique. This is necessary since we have marked [blkA]
-       // as Rejected.
-       time.Sleep(2 * time.Second)
-       <-issuer
-       blkB, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if blkB.Height() != blkA.Height() {
-               t.Fatalf("Expected blkB (%d) to have the same height as blkA (%d)", blkB.Height(), blkA.Height())
-       }
-       if status := blkA.Status(); status != choices.Rejected {
-               t.Fatalf("Expected status of blkA to be %s, but found %s", choices.Rejected, status)
-       }
-       if status := blkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of blkB to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := blkB.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blkB.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of blkC to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm.SetPreference(blkB.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blkB.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blkB.Status(); status != choices.Accepted {
-               t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-       }
-
-       if lastAcceptedID, err := vm.LastAccepted(); err != nil {
-               t.Fatal(err)
-       } else if lastAcceptedID != blkB.ID() {
-               t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blkB.ID(), lastAcceptedID)
-       }
-
-       // Check that [importTx] has been indexed correctly after [blkB] is accepted.
-       _, height, err := vm.getAcceptedAtomicTx(importTx.ID())
-       if err != nil {
-               t.Fatal(err)
-       } else if height != blkB.Height() {
-               t.Fatalf("Expected indexed height of import tx to be %d, but found %d", blkB.Height(), height)
-       }
-}
-
-func TestAtomicTxFailsEVMStateTransferBuildBlock(t *testing.T) {
-       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase1, "", "")
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       exportTxs := createExportTxOptions(t, vm, issuer, sharedMemory)
-       exportTx1, exportTx2 := exportTxs[0], exportTxs[1]
-
-       if err := vm.issueTx(exportTx1, true /*=local*/); err != nil {
-               t.Fatal(err)
-       }
-       <-issuer
-       exportBlk1, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-       if err := exportBlk1.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.SetPreference(exportBlk1.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := vm.issueTx(exportTx2, true /*=local*/); err == nil {
-               t.Fatal("Should have failed to issue due to an invalid export tx")
-       }
-
-       if err := vm.mempool.AddTx(exportTx2); err == nil {
-               t.Fatal("Should have failed to add because conflicting")
-       }
-
-       // Manually add transaction to mempool to bypass validation
-       if err := vm.mempool.ForceAddTx(exportTx2); err != nil {
-               t.Fatal(err)
-       }
-       <-issuer
-
-       _, err = vm.BuildBlock()
-       if err == nil {
-               t.Fatal("BuildBlock should have returned an error due to invalid export transaction")
-       }
-}
```

```
-
-func TestBuildInvalidBlockHead(t *testing.T) {
-	issuer, vm, _, _, _ := GenesisVM(t, true, genesisJSONApricotPhase0, "", "")
-
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	key0 := testKeys[0]
-	addr0 := key0.PublicKey().Address()
-
-	// Create the transaction
-	utx := &UnsignedImportTx{
-		NetworkID:    vm.ctx.NetworkID,
-		BlockchainID: vm.ctx.ChainID,
-		Outs: []EVMOutput{{
-			Address: common.Address(addr0),
-			Amount:  1 * units.Avax,
-			AssetID: vm.ctx.AVAXAssetID,
-		}},
-		ImportedInputs: []*avax.TransferableInput{
-			{
-				Asset: avax.Asset{ID: vm.ctx.AVAXAssetID},
-				In: &secp256k1fx.TransferInput{
-					Amt: 1 * units.Avax,
-					Input: secp256k1fx.Input{
-						SigIndices: []uint32{0},
-					},
-				},
-			},
-		},
-		SourceChain: vm.ctx.XChainID,
-	}
-	tx := &Tx{UnsignedAtomicTx: utx}
-	if err := tx.Sign(vm.codec, [][]*crypto.PrivateKeySECP256K1R{{key0}}); err != nil {
-		t.Fatal(err)
-	}
-
-	currentBlock := vm.chain.BlockChain().CurrentBlock()
-
-	// Verify that the transaction fails verification when attempting to issue
-	// it into the atomic mempool.
-	if err := vm.issueTx(tx, true /*=local*/); err == nil {
-		t.Fatal("Should have failed to issue invalid transaction")
-	}
-	// Force issue the transaction directly to the mempool
-	if err := vm.mempool.AddTx(tx); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	if _, err := vm.BuildBlock(); err == nil {
-		t.Fatalf("Unexpectedly created a block")
-	}
-
-	newCurrentBlock := vm.chain.BlockChain().CurrentBlock()
-
-	if currentBlock.Hash() != newCurrentBlock.Hash() {
-		t.Fatal("current block changed")
-	}
-}
-
-func TestConfigureLogLevel(t *testing.T) {
-	configTests := []struct {
-		name                     string
-		logConfig                string
-		genesisJSON, upgradeJSON string
-		expectedErr              string
-	}{
-		{
-			name:        "Log level info",
-			logConfig:   "{\"log-level\": \"info\"}",
-			genesisJSON: genesisJSONApricotPhase2,
-			upgradeJSON: "",
-			expectedErr: "",
-		},
-		{
-			name:        "Invalid log level",
-			logConfig:   "{\"log-level\": \"cchain\"}",
-			genesisJSON: genesisJSONApricotPhase3,
-			upgradeJSON: "",
-			expectedErr: "failed to initialize logger due to",
-		},
-	}
-	for _, test := range configTests {
-		t.Run(test.name, func(t *testing.T) {
-			vm := &VM{}
-			ctx, dbManager, genesisBytes, issuer, _ := setupGenesis(t, test.genesisJSON)
-			appSender := &engCommon.SenderTest{}
-			appSender.CantSendAppGossip = true
-			appSender.SendAppGossipF = func([]byte) error { return nil }
-			err := vm.Initialize(
-				ctx,
-				dbManager,
-				genesisBytes,
-				[]byte(""),
-				[]byte(test.logConfig),
-				issuer,
-				[]*engCommon.Fx{},
-				appSender,
-			)
-			if len(test.expectedErr) == 0 && err != nil {
-				t.Fatal(err)
-			} else if len(test.expectedErr) > 0 {
-				if err == nil {
-					t.Fatalf("initialize should have failed due to %s", test.expectedErr)
-				} else if !strings.Contains(err.Error(), test.expectedErr) {
-					t.Fatalf("Expected initialize to fail due to %s, but failed with %s", test.expectedErr, err.Error())
-				}
-			}
-
-			// If the VM was not initialized, do not attept to shut it down
-			if err == nil {
-				shutdownChan := make(chan error, 1)
-				shutdownFunc := func() {
-					err := vm.Shutdown()
-					shutdownChan <- err
-				}
-				go shutdownFunc()
+			// If the VM was not initialized, do not attept to shut it down
+			if err == nil {
+				shutdownChan := make(chan error, 1)
+				shutdownFunc := func() {
+					err := vm.Shutdown()
+					shutdownChan <- err
+				}
+				go shutdownFunc()
 
 				shutdownTimeout := 50 * time.Millisecond
 				ticker := time.NewTicker(shutdownTimeout)
@@ -2922,248 +497,3 @@ func TestConfigureLogLevel(t *testing.T) {
		})
	}
 }
-
-// Regression test to ensure we can build blocks if we are starting with the
-// Apricot Phase 4 ruleset in genesis.
-func TestBuildApricotPhase4Block(t *testing.T) {
-	issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase4, "", "")
-
-	defer func() {
-		if err := vm.Shutdown(); err != nil {
-			t.Fatal(err)
-		}
-	}()
-
-	newTxPoolHeadChan := make(chan core.NewTxPoolReorgEvent, 1)
-	vm.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan)
-
-	key, err := accountKeystore.NewKey(rand.Reader)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	importAmount := uint64(1000000000)
-	utxoID := avax.UTXOID{TxID: ids.GenerateTestID()}
-
-	utxo := &avax.UTXO{
-		UTXOID: utxoID,
-		Asset:  avax.Asset{ID: vm.ctx.AVAXAssetID},
-		Out: &secp256k1fx.TransferOutput{
-			Amt: importAmount,
-			OutputOwners: secp256k1fx.OutputOwners{
-				Threshold: 1,
-				Addrs:     []ids.ShortID{testKeys[0].PublicKey().Address()},
-			},
-		},
-	}
-	utxoBytes, err := vm.codec.Marshal(codecVersion, utxo)
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	xChainSharedMemory := sharedMemory.NewSharedMemory(vm.ctx.XChainID)
-	inputID := utxo.InputID()
-	if err := xChainSharedMemory.Apply(map[ids.ID]*atomic.Requests{vm.ctx.ChainID: {PutRequests: []*atomic.Element{{
-		Key:   inputID[:],
-		Value: utxoBytes,
-		Traits: [][]byte{
-			testKeys[0].PublicKey().Address().Bytes(),
-		},
-	}}}}); err != nil {
-		t.Fatal(err)
-	}
-
-	importTx, err := vm.newImportTx(vm.ctx.XChainID, key.Address, initialBaseFee, []*crypto.PrivateKeySECP256K1R{testKeys[0]})
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := vm.issueTx(importTx, true /*=local*/); err != nil {
-		t.Fatal(err)
-	}
-
-	<-issuer
-
-	blk, err := vm.BuildBlock()
-	if err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Verify(); err != nil {
-		t.Fatal(err)
-	}
-
-	if status := blk.Status(); status != choices.Processing {
-		t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-	}
-
-	if err := vm.SetPreference(blk.ID()); err != nil {
-		t.Fatal(err)
-	}
-
-	if err := blk.Accept(); err != nil {
-		t.Fatal(err)
-	}
-
-	ethBlk := blk.(*chain.BlockWrapper).Block.(*Block).ethBlock
-	if eBlockGasCost := ethBlk.BlockGasCost(); eBlockGasCost == nil || eBlockGasCost.Cmp(common.Big0) != 0 {
-		t.Fatalf("expected blockGasCost to be 0 but got %d", eBlockGasCost)
-	}
-	if eExtDataGasUsed := ethBlk.ExtDataGasUsed(); eExtDataGasUsed == nil || eExtDataGasUsed.Cmp(big.NewInt(1230)) != 0 {
-		t.Fatalf("expected extDataGasUsed to be 1000 but got %d", eExtDataGasUsed)
-	}
-	minRequiredTip, err := dummy.MinRequiredTip(vm.chainConfig, ethBlk.Header())
-	if err != nil {
-		t.Fatal(err)
-	}
-	if minRequiredTip == nil || minRequiredTip.Cmp(common.Big0) != 0 {
-		t.Fatalf("expected minRequiredTip to be 0 but got %d", minRequiredTip)
-	}
-
-	newHead := <-newTxPoolHeadChan
-	if newHead.Head.Hash() != common.Hash(blk.ID()) {
-		t.Fatalf("Expected new block to match")
-	}
-
-	txs := make([]*types.Transaction, 10)
-	for i := 0; i < 5; i++ {
-		tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.LaunchMinGasPrice), nil)
-		signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-		if err != nil {
-			t.Fatal(err)
-		}
-		txs[i] = signedTx
-	}
-	for i := 5; i < 10; i++ {
-		tx := types.NewTransaction(uint64(i), key.Address, big.NewInt(10), 21000, big.NewInt(params.ApricotPhase1MinGasPrice), nil)
-		signedTx, err := types.SignTx(tx, types.NewEIP155Signer(vm.chainID), key.PrivateKey)
-		if err != nil {
-			t.Fatal(err)
-		}
-		txs[i] = signedTx
-	}
-	errs := vm.chain.AddRemoteTxs(txs)
-	for i, err := range errs {
-		if err != nil {
-			t.Fatalf("Failed to add tx at index %d: %s", i, err)
-		}
-	}
-
-	<-issuer
-
-	blk, err = vm.BuildBlock()
-	if err != nil {
```

```
-               t.Fatal(err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       ethBlk = blk.(*chain.BlockWrapper).Block.(*Block).ethBlock
-       if ethBlk.BlockGasCost() == nil || ethBlk.BlockGasCost().Cmp(big.NewInt(100)) < 0 {
-               t.Fatalf("expected blockGasCost to be at least 100 but got %d", ethBlk.BlockGasCost())
-       }
-       if ethBlk.ExtDataGasUsed() == nil || ethBlk.ExtDataGasUsed().Cmp(common.Big0) != 0 {
-               t.Fatalf("expected extDataGasUsed to be 0 but got %d", ethBlk.ExtDataGasUsed())
-       }
-       minRequiredTip, err = dummy.MinRequiredTip(vm.chainConfig, ethBlk.Header())
-       if err != nil {
-               t.Fatal(err)
-       }
-       if minRequiredTip == nil || minRequiredTip.Cmp(big.NewInt(0.05*params.GWei)) < 0 {
-               t.Fatalf("expected minRequiredTip to be at least 0.05 gwei but got %d", minRequiredTip)
-       }
-
-       if status := blk.Status(); status != choices.Accepted {
-               t.Fatalf("Expected status of accepted block to be %s, but found %s", choices.Accepted, status)
-       }
-
-       lastAcceptedID, err := vm.LastAccepted()
-       if err != nil {
-               t.Fatal(err)
-       }
-       if lastAcceptedID != blk.ID() {
-               t.Fatalf("Expected last accepted blockID to be the accepted block: %s, but found %s", blk.ID(), lastAcceptedID)
-       }
-
-       // Confirm all txs are present
-       ethBlkTxs := vm.chain.GetBlockByNumber(2).Transactions()
-       for i, tx := range txs {
-               if len(ethBlkTxs) <= i {
-                       t.Fatalf("missing transactions expected: %d but found: %d", len(txs), len(ethBlkTxs))
-               }
-               if ethBlkTxs[i].Hash() != tx.Hash() {
-                       t.Fatalf("expected tx at index %d to have hash: %x but has: %x", i, txs[i].Hash(), tx.Hash())
-               }
-       }
-}
-
-// This is a regression test to ensure that if two consecutive atomic transactions fail verification
-// in onFinalizeAndAssemble it will not cause a panic due to calling RevertToSnapshot(revID) on the
-// same revision ID twice.
-func TestConsecutiveAtomicTransactionsRevertSnapshot(t *testing.T) {
-       issuer, vm, _, sharedMemory, _ := GenesisVM(t, true, genesisJSONApricotPhase1, "", "")
-
-       defer func() {
-               if err := vm.Shutdown(); err != nil {
-                       t.Fatal(err)
-               }
-       }()
-
-       newTxPoolHeadChan := make(chan core.NewTxPoolReorgEvent, 1)
-       vm.chain.GetTxPool().SubscribeNewReorgEvent(newTxPoolHeadChan)
-
-       // Create three conflicting import transactions
-       importTxs := createImportTxOptions(t, vm, sharedMemory)
-
-       // Issue the first import transaction, build, and accept the block.
-       if err := vm.issueTx(importTxs[0], true); err != nil {
-               t.Fatal(err)
-       }
-
-       <-issuer
-
-       blk, err := vm.BuildBlock()
-       if err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Verify(); err != nil {
-               t.Fatal(err)
-       }
-
-       if status := blk.Status(); status != choices.Processing {
-               t.Fatalf("Expected status of built block to be %s, but found %s", choices.Processing, status)
-       }
-
-       if err := vm.SetPreference(blk.ID()); err != nil {
-               t.Fatal(err)
-       }
-
-       if err := blk.Accept(); err != nil {
-               t.Fatal(err)
-       }
-
-       newHead := <-newTxPoolHeadChan
-       if newHead.Head.Hash() != common.Hash(blk.ID()) {
-               t.Fatalf("Expected new block to match")
-       }
-
-       // Add the two conflicting transactions directly to the mempool, so that two consecutive transactions
-       // will fail verification when build block is called.
-       vm.mempool.AddTx(importTxs[1])
-       vm.mempool.AddTx(importTxs[2])
-
-       if _, err := vm.BuildBlock(); err == nil {
-               t.Fatal("Expected build block to fail due to empty block")
-       }
-}
diff --git a/plugin/main.go b/plugin/main.go
index 96cd428f..44588435 100644
--- a/plugin/main.go
+++ b/plugin/main.go
@@ -9,9 +9,10 @@ import (

        "github.com/hashicorp/go-plugin"

-       "github.com/ava-labs/avalanchego/vms/rpcchainvm"
+       "github.com/flare-foundation/flare/utils/ulimit"
+       "github.com/flare-foundation/flare/vms/rpcchainvm"

-       "github.com/ava-labs/coreth/plugin/evm"
+       "github.com/flare-foundation/coreth/plugin/evm"
 )

 func main() {
@@ -24,6 +25,10 @@ func main() {
                fmt.Println(evm.Version)
                os.Exit(0)
        }
+       if err := ulimit.Set(ulimit.DefaultFDLimit); err != nil {
+               fmt.Printf("failed to set fd limit correctly due to: %s", err)
+               os.Exit(1)
+       }
        plugin.Serve(&plugin.ServeConfig{
                HandshakeConfig: rpcchainvm.Handshake,
                Plugins: map[string]plugin.Plugin{
diff --git a/rpc/errors.go b/rpc/errors.go
index a43fa9bd..d8b597c1 100644
--- a/rpc/errors.go
+++ b/rpc/errors.go
@@ -64,6 +64,7 @@ var (
        _ Error = new(invalidRequestError)
        _ Error = new(invalidMessageError)
        _ Error = new(invalidParamsError)
+       _ Error = new(CustomError)
 )

 const defaultErrorCode = -32000
@@ -111,3 +112,12 @@ type invalidParamsError struct{ message string }
 func (e *invalidParamsError) ErrorCode() int { return -32602 }

 func (e *invalidParamsError) Error() string { return e.message }
+
+type CustomError struct {
+       Code            int
+       ValidationError string
+}
+
+func (e *CustomError) ErrorCode() int { return e.Code }
+
+func (e *CustomError) Error() string { return e.ValidationError }
diff --git a/rpc/handler.go b/rpc/handler.go
index abafa663..82d9d664 100644
--- a/rpc/handler.go
+++ b/rpc/handler.go
@@ -36,6 +36,7 @@ import (
        "time"

        "github.com/ethereum/go-ethereum/log"
+       "github.com/ethereum/go-ethereum/metrics"
        "golang.org/x/time/rate"
 )

@@ -83,6 +84,8 @@ type handler struct {
 type callProc struct {
        ctx       context.Context
        notifiers []*Notifier
+       callStart time.Time
+       procStart time.Time
 }

 func newHandler(connCtx context.Context, conn jsonWriter, idgen func() ID, reg *serviceRegistry) *handler {
@@ -262,15 +265,15 @@ func (h *handler) awaitLimit(ctx context.Context) {
        timer.Stop()
 }

-// consumeLimit removes the time since [startTime] from the rate limiter. It is
+// consumeLimit removes the time since [procStart] from the rate limiter. It is
 // assumed that the rate limiter is full.
-func (h *handler) consumeLimit(startTime time.Time) {
+func (h *handler) consumeLimit(procStart time.Time) {
        if h.limiter == nil {
                return
        }

        stopTime := time.Now()
-       processingTime := stopTime.Sub(startTime)
+       processingTime := stopTime.Sub(procStart)
        if processingTime > h.deadlineContext {
                processingTime = h.deadlineContext
        }
@@ -293,11 +296,17 @@ func (h *handler) startCallProc(fn func(*callProc)) {
                }
                defer h.callWG.Done()

+               // Capture the time before we await for processing
+               callStart := time.Now()
                h.awaitLimit(ctx)
-               startTime := time.Now()
+
+               // If we are not limiting CPU, [procStart] will be identical to
+               // [callStart]
+               procStart := time.Now()
                defer cancel()
-               fn(&callProc{ctx: ctx})
-               h.consumeLimit(startTime)
+
+               fn(&callProc{ctx: ctx, callStart: callStart, procStart: procStart})
+               h.consumeLimit(procStart)
        }
        if h.limiter == nil {
                go callFn()
@@ -309,7 +318,7 @@ func (h *handler) startCallProc(fn func(*callProc)) {
 // handleImmediate executes non-call messages. It returns false if the message is a
 // call or requires a reply.
 func (h *handler) handleImmediate(msg *jsonrpcMessage) bool {
-       start := time.Now()
+       execStart := time.Now()
        switch {
        case msg.isNotification():
                if strings.HasSuffix(msg.Method, notificationMethodSuffix) {
@@ -319,7 +328,7 @@ func (h *handler) handleImmediate(msg *jsonrpcMessage) bool {
                return false
        case msg.isResponse():
                h.handleResponse(msg)
-               h.log.Trace("Handled RPC response", "reqid", idForLog{msg.ID}, "t", time.Since(start))
+               h.log.Trace("Handled RPC response", "reqid", idForLog{msg.ID}, "duration", time.Since(execStart))
                return true
        default:
                return false
@@ -367,16 +376,26 @@ func (h *handler) handleResponse(msg *jsonrpcMessage) {

 // handleCallMsg executes a call message and returns the answer.
 func (h *handler) handleCallMsg(ctx *callProc, msg *jsonrpcMessage) *jsonrpcMessage {
-       start := time.Now()
+       // [callStart] is the time the message was enqueued for handler processing
+       callStart := ctx.callStart
+       // [procStart] is the time the message cleared the [limiter] and began to be
+       // processed by the handler
+       procStart := ctx.procStart
+       // [execStart] is the time the message began to be executed by the handler
+       //
+       // Note: This can be different than the executionStart in [startCallProc] as
+       // the goroutine that handles execution may not be executed right away.
+       execStart := time.Now()
+
        switch {
        case msg.isNotification():
                h.handleCall(ctx, msg)
-               h.log.Debug("Served "+msg.Method, "t", time.Since(start))
+               h.log.Debug("Served "+msg.Method, "execTime", time.Since(execStart), "procTime", time.Since(procStart), "totalTime", time.Since(callStart))
                return nil
        case msg.isCall():
                resp := h.handleCall(ctx, msg)
                var ctx []interface{}
-               ctx = append(ctx, "reqid", idForLog{msg.ID}, "t", time.Since(start))
+               ctx = append(ctx, "reqid", idForLog{msg.ID}, "execTime", time.Since(execStart), "procTime", time.Since(procStart), "totalTime", time.Since(callStart))
                if resp.Error != nil {
                        ctx = append(ctx, "err", resp.Error.Message)
                        if resp.Error.Data != nil {
@@ -425,7 +444,9 @@ func (h *handler) handleCall(cp *callProc, msg *jsonrpcMessage) *jsonrpcMessage
                        successfulRequestGauge.Inc(1)
                }
                rpcServingTimer.UpdateSince(start)
-               newRPCServingTimer(msg.Method, answer.Error == nil).UpdateSince(start)
+               if metrics.EnabledExpensive {
+                       newRPCServingTimer(msg.Method, answer.Error == nil).UpdateSince(start)
+               }
        }
        return answer
 }
diff --git a/rpc/types.go b/rpc/types.go
index 96015551..4fd6ded5 100644
--- a/rpc/types.go
+++ b/rpc/types.go
@@ -44,6 +44,7 @@ type API struct {
        Version   string      // api version for DApp's
        Service   interface{} // receiver instance which holds the methods
        Public    bool        // indication if the methods must be considered safe for public use
+       Name      string      // Name of the API
 }

 // ServerCodec implements reading, parsing and writing RPC messages for the server side of
diff --git a/scripts/build.sh b/scripts/build.sh
index 98c19678..a4250631 100755
--- a/scripts/build.sh
+++ b/scripts/build.sh
@@ -4,7 +4,11 @@ set -o errexit
 set -o nounset
 set -o pipefail

+<<<<<<< HEAD
+# Coreth root directory
+=======
 # Avalanche root directory
+>>>>>>> upstream-v0.8.5-rc.2
 CORETH_PATH=$( cd "$( dirname "${BASH_SOURCE[0]}" )"; cd .. && pwd )

 # Load the versions
@@ -27,4 +31,4 @@ coreth_commit=${CORETH_COMMIT:-$( git rev-list -1 HEAD )}

 # Build Coreth, which is run as a subprocess
 echo "Building Coreth Version: $coreth_version; GitCommit: $coreth_commit"
-go build -ldflags "-X github.com/ava-labs/coreth/plugin/evm.GitCommit=$coreth_commit -X github.com/ava-labs/coreth/plugin/evm.Version=$coreth_version" -o "$binary_path" "plugin/"*.go
+go build -ldflags "-X github.com/flare-foundation/coreth/plugin/evm.GitCommit=$coreth_commit -X github.com/flare-foundation/coreth/plugin/evm.Version=$coreth_version" -o "$binary_path" "plugin/"*.go
diff --git a/scripts/build_image.sh b/scripts/build_image.sh
deleted file mode 100755
index 6de76735..00000000
--- a/scripts/build_image.sh
+++ /dev/null
@@ -1,20 +0,0 @@
-#!/usr/bin/env bash
-
-set -o errexit
-set -o nounset
-set -o pipefail
-
-# Avalanche root directory
-CORETH_PATH=$( cd "$( dirname "${BASH_SOURCE[0]}" )"; cd .. && pwd )
-
-# Load the versions
-source "$CORETH_PATH"/scripts/versions.sh
-
-# Load the constants
-source "$CORETH_PATH"/scripts/constants.sh
-
-echo "Building Docker Image: $dockerhub_repo:$build_image_id based of $avalanche_version"
-docker build -t "$dockerhub_repo:$build_image_id" "$CORETH_PATH" -f "$CORETH_PATH/Dockerfile" \
-  --build-arg AVALANCHE_VERSION="$avalanche_version" \
-  --build-arg CORETH_COMMIT="$coreth_commit" \
-  --build-arg CURRENT_BRANCH="$current_branch"
diff --git a/scripts/constants.sh b/scripts/constants.sh
index 262f1a48..69a84370 100644
--- a/scripts/constants.sh
+++ b/scripts/constants.sh
@@ -4,20 +4,12 @@
 GOPATH="$(go env GOPATH)"

 # Set binary location
-binary_path=${CORETH_BINARY_PATH:-"$GOPATH/src/github.com/ava-labs/avalanchego/build/plugins/evm"}
-
-# Avalabs docker hub
-dockerhub_repo="avaplatform/avalanchego"
+binary_path=${CORETH_BINARY_PATH:-"$GOPATH/src/github.com/flare-foundation/flare/build/plugins/evm"}

 # Current branch
 current_branch=${CURRENT_BRANCH:-$(git describe --tags --exact-match 2> /dev/null || git symbolic-ref -q --short HEAD || git rev-parse --short HEAD)}
 echo "Using branch: ${current_branch}"

-# Image build id
-# Use an abbreviated version of the full commit to tag the image.
-
 # WARNING: this will use the most recent commit even if there are un-committed changes present
 coreth_commit="$(git --git-dir="$CORETH_PATH/.git" rev-parse HEAD)"
 coreth_commit_id="${coreth_commit::8}"
-
-build_image_id=${BUILD_IMAGE_ID:-"$avalanche_version-$coreth_commit_id"}
diff --git a/scripts/lint.sh b/scripts/lint.sh
new file mode 100755
index 00000000..f41d6b56
--- /dev/null
+++ b/scripts/lint.sh
@@ -0,0 +1,7 @@
+#!/usr/bin/env bash
+
+set -o errexit
+set -o nounset
+set -o pipefail
+
+golangci-lint run --path-prefix=. --timeout 3m
diff --git a/scripts/versions.sh b/scripts/versions.sh
index 6f2cc893..5a3f2696 100644
--- a/scripts/versions.sh
+++ b/scripts/versions.sh
@@ -1,6 +1,6 @@
 #!/usr/bin/env bash

 # Set up the versions to be used
-coreth_version=${CORETH_VERSION:-'v0.7.4'}
+coreth_version=${CORETH_VERSION:-'v0.3.1'}
 # Don't export them as they're used in the context of other calls
-avalanche_version=${AVALANCHE_VERSION:-'v1.6.4'}
+flare_version=${FLARE_VERSION:-'v0.5.1'}
diff --git a/signer/core/apitypes/types.go b/signer/core/apitypes/types.go
index cab7f9cf..60afd52f 100644
--- a/signer/core/apitypes/types.go
```

```
+++ b/signer/core/apitypes/types.go
@@ -32,9 +32,9 @@ import (
        "math/big"
        "strings"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
+       "github.com/flare-foundation/coreth/core/types"
 )

 type ValidationInfo struct {
diff --git a/tests/init.go b/tests/init.go
index d08bca7e..25e33865 100644
--- a/tests/init.go
+++ b/tests/init.go
@@ -31,7 +31,7 @@ import (
        "math/big"
        "sort"

-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"
 )

 // Forks table defines supported forks and their chain config.
@@ -208,6 +208,22 @@ var Forks = map[string]*params.ChainConfig{
                ApricotPhase3BlockTimestamp: big.NewInt(0),
                ApricotPhase4BlockTimestamp: big.NewInt(0),
        },
+       "ApricotPhase5": {
+               ChainID:                     big.NewInt(1),
+               HomesteadBlock:              big.NewInt(0),
+               EIP150Block:                 big.NewInt(0),
+               EIP155Block:                 big.NewInt(0),
+               EIP158Block:                 big.NewInt(0),
+               ByzantiumBlock:              big.NewInt(0),
+               ConstantinopleBlock:         big.NewInt(0),
+               PetersburgBlock:             big.NewInt(0),
+               IstanbulBlock:               big.NewInt(0),
+               ApricotPhase1BlockTimestamp: big.NewInt(0),
+               ApricotPhase2BlockTimestamp: big.NewInt(0),
+               ApricotPhase3BlockTimestamp: big.NewInt(0),
+               ApricotPhase4BlockTimestamp: big.NewInt(0),
+               ApricotPhase5BlockTimestamp: big.NewInt(0),
+       },
 }

 // Returns the set of defined fork names
diff --git a/tests/init_test.go b/tests/init_test.go
index 1ddbfcc6..5888be03 100644
--- a/tests/init_test.go
+++ b/tests/init_test.go
@@ -40,7 +40,7 @@ import (
        "strings"
        "testing"

-       "github.com/ava-labs/coreth/params"
+       "github.com/flare-foundation/coreth/params"
 )

 func readJSON(reader io.Reader, value interface{}) error {
diff --git a/tests/state_test_util.go b/tests/state_test_util.go
index 6e33efa5..73ae6c1f 100644
--- a/tests/state_test_util.go
+++ b/tests/state_test_util.go
@@ -34,17 +34,17 @@ import (
        "strconv"
        "strings"

-       "github.com/ava-labs/coreth/core"
-       "github.com/ava-labs/coreth/core/state"
-       "github.com/ava-labs/coreth/core/state/snapshot"
-       "github.com/ava-labs/coreth/core/types"
-       "github.com/ava-labs/coreth/core/vm"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/params"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/common/hexutil"
        "github.com/ethereum/go-ethereum/common/math"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/core"
+       "github.com/flare-foundation/coreth/core/state"
+       "github.com/flare-foundation/coreth/core/state/snapshot"
+       "github.com/flare-foundation/coreth/core/types"
+       "github.com/flare-foundation/coreth/core/vm"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/params"
 )

 // StateTest checks transaction processing without block context.
diff --git a/trie/database.go b/trie/database.go
index 4f59c68e..c5e59f49 100644
--- a/trie/database.go
+++ b/trie/database.go
@@ -36,12 +36,12 @@ import (
        "time"

        "github.com/VictoriaMetrics/fastcache"
-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/metrics"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 var (
diff --git a/trie/database_test.go b/trie/database_test.go
index bc3d3e9d..626ddfe0 100644
--- a/trie/database_test.go
+++ b/trie/database_test.go
@@ -29,8 +29,8 @@ package trie
 import (
        "testing"

-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/ethereum/go-ethereum/common"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )

 // Tests that the trie database returns a missing trie node error if attempting
diff --git a/trie/iterator.go b/trie/iterator.go
index 4ab0863c..dbba17ce 100644
--- a/trie/iterator.go
+++ b/trie/iterator.go
@@ -31,9 +31,9 @@ import (
        "container/heap"
        "errors"

-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/rlp"
```

```
+	"github.com/flare-foundation/coreth/ethdb"
 )

 // Iterator is a key-value trie iterator that traverses a Trie.
@@ -285,7 +285,7 @@ func (it *nodeIterator) seek(prefix []byte) error {
	}
 }

-// init initializes the the iterator.
+// init initializes the iterator.
 func (it *nodeIterator) init() (*nodeIteratorState, error) {
	root := it.trie.Hash()
	state := &nodeIteratorState{node: it.trie.root, index: -1}
diff --git a/trie/iterator_test.go b/trie/iterator_test.go
index 7ef711c3..fe13af8b 100644
--- a/trie/iterator_test.go
+++ b/trie/iterator_test.go
@@ -33,10 +33,10 @@ import (
	"math/rand"
	"testing"

-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/ethdb/memorydb"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
 )

 func TestIterator(t *testing.T) {
diff --git a/trie/proof.go b/trie/proof.go
index 37ce9ed3..4ded339a 100644
--- a/trie/proof.go
+++ b/trie/proof.go
@@ -31,11 +31,11 @@ import (
	"errors"
	"fmt"

-	"github.com/ava-labs/coreth/ethdb"
-	"github.com/ava-labs/coreth/ethdb/memorydb"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/log"
	"github.com/ethereum/go-ethereum/rlp"
+	"github.com/flare-foundation/coreth/ethdb"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
 )

 // Prove constructs a merkle proof for key. The result contains all encoded nodes
@@ -482,12 +482,17 @@ func VerifyRangeProof(rootHash common.Hash, firstKey []byte, lastKey []byte, key
	if len(keys) != len(values) {
		return false, fmt.Errorf("inconsistent proof data, keys: %d, values: %d", len(keys), len(values))
	}
-	// Ensure the received batch is monotonic increasing.
+	// Ensure the received batch is monotonic increasing and contains no deletions
	for i := 0; i < len(keys)-1; i++ {
		if bytes.Compare(keys[i], keys[i+1]) >= 0 {
			return false, errors.New("range is not monotonically increasing")
		}
	}
+	for _, value := range values {
+		if len(value) == 0 {
+			return false, errors.New("range contains deletion")
+		}
+	}
	// Special case, there is no edge proof at all. The given range is expected
	// to be the whole leaf-set in the trie.
	if proof == nil {
diff --git a/trie/proof_test.go b/trie/proof_test.go
index f37c2060..3471d671 100644
--- a/trie/proof_test.go
+++ b/trie/proof_test.go
@@ -35,9 +35,9 @@ import (
	"testing"
	"time"

-	"github.com/ava-labs/coreth/ethdb/memorydb"
	"github.com/ethereum/go-ethereum/common"
	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
 )

 func init() {
@@ -823,6 +823,85 @@ func TestBloatedProof(t *testing.T) {
	}
 }

+// TestEmptyValueRangeProof tests normal range proof with both edge proofs
+// as the existent proof, but with an extra empty value included, which is a
+// noop technically, but practically should be rejected.
+func TestEmptyValueRangeProof(t *testing.T) {
+	trie, values := randomTrie(512)
+	var entries entrySlice
+	for _, kv := range values {
+		entries = append(entries, kv)
+	}
+	sort.Sort(entries)
+
+	// Create a new entry with a slightly modified key
+	mid := len(entries) / 2
+	key := common.CopyBytes(entries[mid-1].k)
+	for n := len(key) - 1; n >= 0; n-- {
+		if key[n] < 0xff {
+			key[n]++
+			break
+		}
+	}
+	noop := &kv{key, []byte{}, false}
+	entries = append(append(append([]*kv{}, entries[:mid]...), noop), entries[mid:]...)
+
+	start, end := 1, len(entries)-1
+
+	proof := memorydb.New()
+	if err := trie.Prove(entries[start].k, 0, proof); err != nil {
+		t.Fatalf("Failed to prove the first node %v", err)
+	}
+	if err := trie.Prove(entries[end-1].k, 0, proof); err != nil {
+		t.Fatalf("Failed to prove the last node %v", err)
+	}
+	var keys [][]byte
+	var vals [][]byte
+	for i := start; i < end; i++ {
+		keys = append(keys, entries[i].k)
+		vals = append(vals, entries[i].v)
+	}
+	_, err := VerifyRangeProof(trie.Hash(), keys[0], keys[len(keys)-1], keys, vals, proof)
+	if err == nil {
+		t.Fatalf("Expected failure on noop entry")
+	}
+}
+
+// TestAllElementsEmptyValueRangeProof tests the range proof with all elements,
+// but with an extra empty value included, which is a noop technically, but
+// practically should be rejected.
+func TestAllElementsEmptyValueRangeProof(t *testing.T) {
```

```diff
+       trie, values := randomTrie(512)
+       var entries entrySlice
+       for _, kv := range values {
+               entries = append(entries, kv)
+       }
+       sort.Sort(entries)
+
+       // Create a new entry with a slightly modified key
+       mid := len(entries) / 2
+       key := common.CopyBytes(entries[mid-1].k)
+       for n := len(key) - 1; n >= 0; n-- {
+               if key[n] < 0xff {
+                       key[n]++
+                       break
+               }
+       }
+       noop := &kv{key, []byte{}, false}
+       entries = append(append(append([]*kv{}, entries[:mid]...), noop), entries[mid:]...)
+
+       var keys [][]byte
+       var vals [][]byte
+       for i := 0; i < len(entries); i++ {
+               keys = append(keys, entries[i].k)
+               vals = append(vals, entries[i].v)
+       }
+       _, err := VerifyRangeProof(trie.Hash(), nil, nil, keys, vals, nil)
+       if err == nil {
+               t.Fatalf("Expected failure on noop entry")
+       }
+}
+
 // mutateByte changes one byte in b.
 func mutateByte(b []byte) {
        for r := mrand.Intn(len(b)); ; {
diff --git a/trie/secure_trie.go b/trie/secure_trie.go
index 4f6a556e..0d324e15 100644
--- a/trie/secure_trie.go
+++ b/trie/secure_trie.go
@@ -29,10 +29,10 @@ package trie
 import (
        "fmt"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
 )

 // SecureTrie wraps a trie with key hashing. In a secure trie, all
diff --git a/trie/secure_trie_test.go b/trie/secure_trie_test.go
index c9c29d70..642e7d2a 100644
--- a/trie/secure_trie_test.go
+++ b/trie/secure_trie_test.go
@@ -32,9 +32,9 @@ import (
        "sync"
        "testing"

-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )

 func newEmptySecure() *SecureTrie {
diff --git a/trie/stacktrie.go b/trie/stacktrie.go
index 248b4e4e..33445f8f 100644
--- a/trie/stacktrie.go
+++ b/trie/stacktrie.go
@@ -35,10 +35,10 @@ import (
        "io"
        "sync"

-       "github.com/ava-labs/coreth/ethdb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/ethdb"
 )

 var ErrCommitDisabled = errors.New("no database for committing")
@@ -64,12 +64,11 @@ func returnToPool(st *StackTrie) {
 // in order. Once it determines that a subtree will no longer be inserted
 // into, it will hash it and free up the memory it uses.
 type StackTrie struct {
-       nodeType  uint8                // node type (as in branch, ext, leaf)
-       val       []byte               // value contained by this node if it's a leaf
-       key       []byte               // key chunk covered by this (full|ext) node
-       keyOffset int                  // offset of the key chunk inside a full key
-       children  [16]*StackTrie       // list of children (for fullnodes and exts)
-       db        ethdb.KeyValueWriter // Pointer to the commit db, can be nil
+       nodeType uint8                // node type (as in branch, ext, leaf)
+       val      []byte               // value contained by this node if it's a leaf
+       key      []byte               // key chunk covered by this (leaf|ext) node
+       children [16]*StackTrie       // list of children (for branch and exts)
+       db       ethdb.KeyValueWriter // Pointer to the commit db, can be nil
 }

 // NewStackTrie allocates and initializes an empty trie.
@@ -100,15 +99,13 @@ func (st *StackTrie) MarshalBinary() (data []byte, err error) {
                w = bufio.NewWriter(&b)
        )
        if err := gob.NewEncoder(w).Encode(struct {
-               Nodetype  uint8
-               Val       []byte
-               Key       []byte
-               KeyOffset uint8
+               Nodetype uint8
+               Val      []byte
+               Key      []byte
        }{
                st.nodeType,
                st.val,
                st.key,
-               uint8(st.keyOffset),
        }); err != nil {
                return nil, err
        }
@@ -136,16 +133,14 @@ func (st *StackTrie) UnmarshalBinary(data []byte) error {

 func (st *StackTrie) unmarshalBinary(r io.Reader) error {
        var dec struct {
-               Nodetype  uint8
-               Val       []byte
-               Key       []byte
-               KeyOffset uint8
+               Nodetype uint8
+               Val      []byte
+               Key      []byte
        }
        gob.NewDecoder(r).Decode(&dec)
        st.nodeType = dec.Nodetype
        st.val = dec.Val
        st.key = dec.Key
```

```
-       st.keyOffset = int(dec.KeyOffset)

        var hasChild = make([]byte, 1)
        for i := range st.children {
@@ -170,20 +165,18 @@ func (st *StackTrie) setDb(db ethdb.KeyValueWriter) {
        }
 }

-func newLeaf(ko int, key, val []byte, db ethdb.KeyValueWriter) *StackTrie {
+func newLeaf(key, val []byte, db ethdb.KeyValueWriter) *StackTrie {
        st := stackTrieFromPool(db)
        st.nodeType = leafNode
-       st.keyOffset = ko
-       st.key = append(st.key, key[ko:]...)
+       st.key = append(st.key, key...)
        st.val = val
        return st
 }

-func newExt(ko int, key []byte, child *StackTrie, db ethdb.KeyValueWriter) *StackTrie {
+func newExt(key []byte, child *StackTrie, db ethdb.KeyValueWriter) *StackTrie {
        st := stackTrieFromPool(db)
        st.nodeType = extNode
-       st.keyOffset = ko
-       st.key = append(st.key, key[ko:]...)
+       st.key = append(st.key, key...)
        st.children[0] = child
        return st
 }
@@ -221,17 +214,18 @@ func (st *StackTrie) Reset() {
                st.children[i] = nil
        }
        st.nodeType = emptyNode
-       st.keyOffset = 0
 }

 // Helper function that, given a full key, determines the index
 // at which the chunk pointed by st.keyOffset is different from
 // the same chunk in the full key.
 func (st *StackTrie) getDiffIndex(key []byte) int {
-       diffindex := 0
-       for ; diffindex < len(st.key) && st.key[diffindex] == key[st.keyOffset+diffindex]; diffindex++ {
+       for idx, nibble := range st.key {
+               if nibble != key[idx] {
+                       return idx
+               }
        }
-       return diffindex
+       return len(st.key)
 }

 // Helper function to that inserts a (key, value) pair into
@@ -239,7 +233,7 @@ func (st *StackTrie) getDiffIndex(key []byte) int {
 func (st *StackTrie) insert(key, value []byte) {
        switch st.nodeType {
        case branchNode: /* Branch */
-               idx := int(key[st.keyOffset])
+               idx := int(key[0])
                // Unresolve elder siblings
                for i := idx - 1; i >= 0; i-- {
                        if st.children[i] != nil {
@@ -251,10 +245,10 @@ func (st *StackTrie) insert(key, value []byte) {
                }
                // Add new child
                if st.children[idx] == nil {
-                       st.children[idx] = stackTrieFromPool(st.db)
-                       st.children[idx].keyOffset = st.keyOffset + 1
+                       st.children[idx] = newLeaf(key[1:], value, st.db)
+               } else {
+                       st.children[idx].insert(key[1:], value)
                }
-               st.children[idx].insert(key, value)
        case extNode: /* Ext */
                // Compare both key chunks and see where they differ
                diffidx := st.getDiffIndex(key)
@@ -267,7 +261,7 @@ func (st *StackTrie) insert(key, value []byte) {
                if diffidx == len(st.key) {
                        // Ext key and key segment are identical, recurse into
                        // the child node.
-                       st.children[0].insert(key, value)
+                       st.children[0].insert(key[diffidx:], value)
                        return
                }
                // Save the original part. Depending if the break is
@@ -276,7 +270,7 @@ func (st *StackTrie) insert(key, value []byte) {
                // node directly.
                var n *StackTrie
                if diffidx < len(st.key)-1 {
-                       n = newExt(diffidx+1, st.key, st.children[0], st.db)
+                       n = newExt(st.key[diffidx+1:], st.children[0], st.db)
                } else {
                        // Break on the last byte, no need to insert
                        // an extension node: reuse the current node
@@ -298,15 +292,14 @@ func (st *StackTrie) insert(key, value []byte) {
                        // node.
                        st.children[0] = stackTrieFromPool(st.db)
                        st.children[0].nodeType = branchNode
-                       st.children[0].keyOffset = st.keyOffset + diffidx
                        p = st.children[0]
                }
                // Create a leaf for the inserted part
-               o := newLeaf(st.keyOffset+diffidx+1, key, value, st.db)
+               o := newLeaf(key[diffidx+1:], value, st.db)

                // Insert both child leaves where they belong:
                origIdx := st.key[diffidx]
-               newIdx := key[diffidx+st.keyOffset]
+               newIdx := key[diffidx]
                p.children[origIdx] = n
                p.children[newIdx] = o
                st.key = st.key[:diffidx]
@@ -340,7 +333,6 @@ func (st *StackTrie) insert(key, value []byte) {
                        st.nodeType = extNode
                        st.children[0] = NewStackTrie(st.db)
                        st.children[0].nodeType = branchNode
-                       st.children[0].keyOffset = st.keyOffset + diffidx
                        p = st.children[0]
                }

@@ -349,11 +341,11 @@ func (st *StackTrie) insert(key, value []byte) {
                // The child leave will be hashed directly in order to
                // free up some memory.
                origIdx := st.key[diffidx]
-               p.children[origIdx] = newLeaf(diffidx+1, st.key, st.val, st.db)
+               p.children[origIdx] = newLeaf(st.key[diffidx+1:], st.val, st.db)
                p.children[origIdx].hash()

-               newIdx := key[diffidx+st.keyOffset]
-               p.children[newIdx] = newLeaf(p.keyOffset+1, key, value, st.db)
+               newIdx := key[diffidx]
+               p.children[newIdx] = newLeaf(key[diffidx+1:], value, st.db)

                // Finally, cut off the key part that has been passed
                // over to the children.
```

```
@@ -361,7 +353,7 @@ func (st *StackTrie) insert(key, value []byte) {
                st.val = nil
        case emptyNode: /* Empty */
                st.nodeType = leafNode
-               st.key = key[st.keyOffset:]
+               st.key = key
                st.val = value
        case hashedNode:
                panic("trying to insert into hash")
diff --git a/trie/stacktrie_test.go b/trie/stacktrie_test.go
index 189e9d41..1a5b2005 100644
--- a/trie/stacktrie_test.go
+++ b/trie/stacktrie_test.go
@@ -31,11 +31,171 @@ import (
        "math/big"
        "testing"

-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
 )

+func TestStackTrieInsertAndHash(t *testing.T) {
+       type KeyValueHash struct {
+               K string // Hex string for key.
+               V string // Value, directly converted to bytes.
+               H string // Expected root hash after insert of (K, V) to an existing trie.
+       }
+       tests := [][]KeyValueHash{
+               { // {0:0, 7:0, f:0}
+                       {"00", "v_______________________0___0", "5cb26357b95bb9af08475be00243ceb68ade0b66b5cd816b0c18a18c612d2d21"},
+                       {"70", "v_______________________0___1", "8ff64309574f7a437a7ad1628e690eb7663cfde10676f8a904a8c8291dbc1603"},
+                       {"f0", "v_______________________0___2", "9e3a01bd8d43efb8e9d4b5506648150b8e3ed1caea596f84ee28e01a72635470"},
+               },
+               { // {1:0cc, e:{1:fc, e:fc}}
+                       {"10cc", "v_______________________1___0", "233e9b257843f3dfdb1cce6676cdaf9e595ac96ee1b55031434d852bc7ac9185"},
+                       {"e1fc", "v_______________________1___1", "39c5e908ae83d0c78520c7c7bda0b3782daf594700e44546e93def8f049cca95"},
+                       {"eefc", "v_______________________1___2", "d789567559fd76fe5b7d9cc42f3750f942502ac1c7f2a466e2f690ec4b6c2a7c"},
+               },
+               { // {b:{a:ac, b:ac}, d:acc}
+                       {"baac", "v_______________________2___0", "8be1c86ba7ec4c61e14c1a9b75055e0464c2633ae66a055a24e75450156a5d42"},
+                       {"bbac", "v_______________________2___1", "8495159b9895a7d88d973171d737c0aace6fe6ac02a4769fff1bc43bcccce4cc"},
+                       {"dacc", "v_______________________2___2", "9bcfc5b220a27328deb9dc6ee2e3d46c9ebc9c69e78acda1fa2c7040602c63ca"},
+               },
+               { // {0:0cccc, 2:456{0:0, 2:2}
+                       {"00cccc", "v_______________________3___0", "e57dc2785b99ce9205080cb41b32ebea7ac3e158952b44c87d186e6d190a6530"},
+                       {"245600", "v_______________________3___1", "0335354adbd360a45c1871a842452287721b64b4234dfe08760b243523c998db"},
+                       {"245622", "v_______________________3___2", "9e6832db0dca2b5cf81c0e0727bfde6afc39d5de33e5720bccacc183c162104e"},
+               },
+               { // {1:4567{1:1c, 3:3c}, 3:0cccccc}
+                       {"1456711c", "v_______________________4___0", "f2389e78d98fed99f3e63d6d1623c1d4d9e8c91cb1d585de81fbc7c0e60d3529"},
+                       {"1456733c", "v_______________________4___1", "101189b3fab852be97a0120c03d95eefcf984d3ed639f2328527de6def55a9c0"},
+                       {"30cccccc", "v_______________________4___2", "3780ce111f98d15751dfde1eb21080efc7d3914b429e5c84c64db637c55405b3"},
+               },
+               { // 8800{1:f, 2:e, 3:d}
+                       {"88001f", "v_______________________5___0", "e817db50d84f341d443c6f6593cafda093fc85e773a762421d47daa6ac993bd5"},
+                       {"88002e", "v_______________________5___1", "d6e3e6047bdc110edd296a4d63c030aec451bee9d8075bc5a198eee8cda34f68"},
+                       {"88003d", "v_______________________5___2", "b6bdf8298c703342188e5f7f84921a402042d0e5fb059969dd53a6b6b1fb989e"},
+               },
+               { // 0{1:fc, 2:ec, 4:dc}
+                       {"01fc", "v_______________________6___0", "693268f2ca80d32b015f61cd2c4dba5a47a6b52a14c34f8e6945fad684e7a0d5"},
+                       {"02ec", "v_______________________6___1", "e24ddd44469310c2b785a2044618874bf486d2f7822603a9b8dce58d6524d5de"},
+                       {"04dc", "v_______________________6___2", "33fc259629187bbe54b92f82f0cd8083b91a12e41a9456b84fc155321e334db7"},
+               },
+               { // f{0:fccc, f:ff{0:f, f:f}}
+                       {"f0fccc", "v_______________________7___0", "b0966b5aa469a3e292bc5fcfa6c396ae7a657255eef552ea7e12f996de795b90"},
+                       {"ffff0f", "v_______________________7___1", "3b1ca154ec2a3d96d8d77bddef0abfe40a53a64eb03cecf78da9ec43799fa3d0"},
+                       {"ffffff", "v_______________________7___2", "e75463041f1be8252781be0ace579a44ea4387bf5b2739f4607af676f7719678"},
+               },
+               { // ff{0:f{0:f, f:f}, f:fcc}
+                       {"ff0f0f", "v_______________________8___0", "0928af9b14718ec8262ab89df430f1e5fbf66fac0fed037aff2b6767ae8c8684"},
+                       {"ff0fff", "v_______________________8___1", "d870f4d3ce26b0bf86912810a1960693630c20a48ba56be0ad04bc3e9ddb01e6"},
+                       {"ffffcc", "v_______________________8___2", "4239f10dd9d9915ecf2e047d6a576bdc1733ed77a30830f1bf29deaf7d8e966f"},
+               },
+               {
+                       {"123d", "x___________________________0", "fc453d88b6f128a77c448669710497380fa4588abbea9f78f4c20c80daa797d0"},
+                       {"123e", "x___________________________1", "5af48f2d8a9a015c1ff7fa8b8c7f6b676233bd320e8fb57fd7933622badd2cec"},
+                       {"123f", "x___________________________2", "1164d7299964e74ac40d761f9189b2a3987fae959800d0f7e29d3aaf3eae9e15"},
+               },
+               {
+                       {"123d", "x___________________________0", "fc453d88b6f128a77c448669710497380fa4588abbea9f78f4c20c80daa797d0"},
+                       {"123e", "x___________________________1", "5af48f2d8a9a015c1ff7fa8b8c7f6b676233bd320e8fb57fd7933622badd2cec"},
+                       {"124a", "x___________________________2", "661a96a669869d76b7231380da0649d013301425fbea9d5c5fae6405aa31cfce"},
+               },
+               {
+                       {"123d", "x___________________________0", "fc453d88b6f128a77c448669710497380fa4588abbea9f78f4c20c80daa797d0"},
+                       {"123e", "x___________________________1", "5af48f2d8a9a015c1ff7fa8b8c7f6b676233bd320e8fb57fd7933622badd2cec"},
+                       {"13aa", "x___________________________2", "6590120e1fd3ffd1a90e8de5bb10750b61079bb0776cca4414dd79a24e4d4356"},
+               },
+               {
+                       {"123d", "x___________________________0", "fc453d88b6f128a77c448669710497380fa4588abbea9f78f4c20c80daa797d0"},
+                       {"123e", "x___________________________1", "5af48f2d8a9a015c1ff7fa8b8c7f6b676233bd320e8fb57fd7933622badd2cec"},
+                       {"2aaa", "x___________________________2", "f869b40e0c55eace1918332ef91563616fbf0755e2b946119679f7ef8e44b514"},
+               },
+               {
+                       {"1234da", "x___________________________0", "1c4b4462e9f56a80ca0f5d77c0d632c41b0102290930343cf1791e971a045a79"},
+                       {"1234ea", "x___________________________1", "2f502917f3ba7d328c21c8b45ee0f160652e68450332c166d4ad02d1afe31862"},
+                       {"1234fa", "x___________________________2", "4f4e368ab367090d5bc3dbf25f7729f8bd60df84de309b4633a6b69ab66142c0"},
+               },
+               {
+                       {"1234da", "x___________________________0", "1c4b4462e9f56a80ca0f5d77c0d632c41b0102290930343cf1791e971a045a79"},
+                       {"1234ea", "x___________________________1", "2f502917f3ba7d328c21c8b45ee0f160652e68450332c166d4ad02d1afe31862"},
+                       {"1235aa", "x___________________________2", "21840121d11a91ac8bbad9a5d06af902a5c8d56a47b85600ba813814b7bfcb9b"},
+               },
+               {
+                       {"1234da", "x___________________________0", "1c4b4462e9f56a80ca0f5d77c0d632c41b0102290930343cf1791e971a045a79"},
+                       {"1234ea", "x___________________________1", "2f502917f3ba7d328c21c8b45ee0f160652e68450332c166d4ad02d1afe31862"},
+                       {"124aaa", "x___________________________2", "ea4040ddf6ae3fbd1524bdec19c0ab1581015996262006632027fa5cf21e441e"},
+               },
+               {
+                       {"1234da", "x___________________________0", "1c4b4462e9f56a80ca0f5d77c0d632c41b0102290930343cf1791e971a045a79"},
+                       {"1234ea", "x___________________________1", "2f502917f3ba7d328c21c8b45ee0f160652e68450332c166d4ad02d1afe31862"},
+                       {"13aaaa", "x___________________________2", "e4beb66c67e44f2dd8ba36036e45a44ff68f8d52942472b1911a45f886a34507"},
+               },
+               {
+                       {"1234da", "x___________________________0", "1c4b4462e9f56a80ca0f5d77c0d632c41b0102290930343cf1791e971a045a79"},
+                       {"1234ea", "x___________________________1", "2f502917f3ba7d328c21c8b45ee0f160652e68450332c166d4ad02d1afe31862"},
+                       {"2aaaaa", "x___________________________2", "5f5989b820ff5d76b7d49e77bb64f26602294f6c42a1a3becc669cd9e0dc8ec9"},
+               },
+               {
+                       {"000000", "x___________________________0", "3b32b7af0bddc7940e7364ee18b5a59702c1825e469452c8483b9c4e0218b55a"},
+                       {"1234da", "x___________________________1", "3ab152a1285dca31945566f872c1cc2f17a770440eda32aeee46a5e91033dde2"},
+                       {"1234ea", "x___________________________2", "0cccc87f96ddef55563c1b3be3c64fff6a644333c3d9cd99852cb53b6412b9b8"},
+                       {"1234fa", "x___________________________3", "65bb3aafea8121111d693ffe34881c14d27b128fd113fa120961f251fe28428d"},
+               },
+               {
+                       {"000000", "x___________________________0", "3b32b7af0bddc7940e7364ee18b5a59702c1825e469452c8483b9c4e0218b55a"},
+                       {"1234da", "x___________________________1", "3ab152a1285dca31945566f872c1cc2f17a770440eda32aeee46a5e91033dde2"},
+                       {"1234ea", "x___________________________2", "0cccc87f96ddef55563c1b3be3c64fff6a644333c3d9cd99852cb53b6412b9b8"},
+                       {"1235aa", "x___________________________3", "f670e4d2547c533c5f21e0045442e2ecb733f347ad6d29ef36e0f5ba31bb11a8"},
+               },
+               {
+                       {"000000", "x___________________________0", "3b32b7af0bddc7940e7364ee18b5a59702c1825e469452c8483b9c4e0218b55a"},
+			{"1234da", "x___________________________1", "3ab152a1285dca31945566f872c1cc2f17a770440eda32aeee46a5e91033dde2"},
+			{"1234ea", "x___________________________2", "0cccc87f96ddef55563c1b3be3c64fff6a644333c3d9cd99852cb53b6412b9b8"},
+			{"124aaa", "x___________________________3", "c17464123050a9a6f29b5574bb2f92f6d305c1794976b475b7fb0316b6335598"},
+		},
+		{
+			{"000000", "x___________________________0", "3b32b7af0bddc7940e7364ee18b5a59702c1825e469452c8483b9c4e0218b55a"},
+			{"1234da", "x___________________________1", "3ab152a1285dca31945566f872c1cc2f17a770440eda32aeee46a5e91033dde2"},
+			{"1234ea", "x___________________________2", "0cccc87f96ddef55563c1b3be3c64fff6a644333c3d9cd99852cb53b6412b9b8"},
+			{"13aaaa", "x___________________________3", "aa8301be8cb52ea5cd249f5feb79fb4315ee8de2140c604033f4b3fff78f0105"},
+		},
+		{
+			{"0000", "x___________________________0", "cb8c09ad07ae882136f602b3f21f8733a9f5a78f1d2525a8d24d1c13258000b2"},
+			{"123d", "x___________________________1", "8f09663deb02f08958136410dc48565e077f76bb6c9d8c84d35fc8913a657d31"},
+			{"123e", "x___________________________2", "0d230561e398c579e09a9f7b69ceaf7d3970f5a436fdb28b68b7a37c5bdd6b80"},
+			{"123f", "x___________________________3", "80f7bad1893ca57e3443bb3305a517723a74d3ba831bcaca22a170645eb7aafb"},
+		},
+		{
+			{"0000", "x___________________________0", "cb8c09ad07ae882136f602b3f21f8733a9f5a78f1d2525a8d24d1c13258000b2"},
+			{"123d", "x___________________________1", "8f09663deb02f08958136410dc48565e077f76bb6c9d8c84d35fc8913a657d31"},
+			{"123e", "x___________________________2", "0d230561e398c579e09a9f7b69ceaf7d3970f5a436fdb28b68b7a37c5bdd6b80"},
+			{"124a", "x___________________________3", "383bc1bb4f019e6bc4da3751509ea709b58dd1ac46081670834bae072f3e9557"},
+		},
+		{
+			{"0000", "x___________________________0", "cb8c09ad07ae882136f602b3f21f8733a9f5a78f1d2525a8d24d1c13258000b2"},
+			{"123d", "x___________________________1", "8f09663deb02f08958136410dc48565e077f76bb6c9d8c84d35fc8913a657d31"},
+			{"123e", "x___________________________2", "0d230561e398c579e09a9f7b69ceaf7d3970f5a436fdb28b68b7a37c5bdd6b80"},
+			{"13aa", "x___________________________3", "ff0dc70ce2e5db90ee42a4c2ad12139596b890e90eb4e16526ab38fa465b35cf"},
+		},
+	}
+	st := NewStackTrie(nil)
+	for i, test := range tests {
+		// The StackTrie does not allow Insert(), Hash(), Insert(), ...
+		// so we will create new trie for every sequence length of inserts.
+		for l := 1; l <= len(test); l++ {
+			st.Reset()
+			for j := 0; j < l; j++ {
+				kv := &test[j]
+				if err := st.TryUpdate(common.FromHex(kv.K), []byte(kv.V)); err != nil {
+					t.Fatal(err)
+				}
+			}
+			expected := common.HexToHash(test[l-1].H)
+			if h := st.Hash(); h != expected {
+				t.Errorf("%d(%d): root hash mismatch: %x, expected %x", i, l, h, expected)
+			}
+		}
+	}
+}
+
 func TestSizeBug(t *testing.T) {
 	st := NewStackTrie(nil)
 	nt, _ := New(common.Hash{}, NewDatabase(memorydb.New()))
diff --git a/trie/sync.go b/trie/sync.go
index 85b1f5a5..af5e2821 100644
--- a/trie/sync.go
+++ b/trie/sync.go
@@ -30,10 +30,11 @@ import (
 	"errors"
 	"fmt"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/common/prque"
+
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 // ErrNotRequested is returned by the trie sync when it's requested to process a
diff --git a/trie/sync_bloom.go b/trie/sync_bloom.go
index 51c318c4..fafc9839 100644
--- a/trie/sync_bloom.go
+++ b/trie/sync_bloom.go
@@ -33,12 +33,14 @@ import (
 	"sync/atomic"
 	"time"
 
-	"github.com/ava-labs/coreth/core/rawdb"
-	"github.com/ava-labs/coreth/ethdb"
+	bloomfilter "github.com/holiman/bloomfilter/v2"
+
 	"github.com/ethereum/go-ethereum/common"
 	"github.com/ethereum/go-ethereum/log"
 	"github.com/ethereum/go-ethereum/metrics"
-	bloomfilter "github.com/holiman/bloomfilter/v2"
+
+	"github.com/flare-foundation/coreth/core/rawdb"
+	"github.com/flare-foundation/coreth/ethdb"
 )
 
 var (
diff --git a/trie/sync_test.go b/trie/sync_test.go
index d3bbdbd9..1f3839d2 100644
--- a/trie/sync_test.go
+++ b/trie/sync_test.go
@@ -27,12 +27,8 @@
 package trie
 
 import (
-	"bytes"
-	"testing"
-
-	"github.com/ava-labs/coreth/ethdb/memorydb"
 	"github.com/ethereum/go-ethereum/common"
-	"github.com/ethereum/go-ethereum/crypto"
+	"github.com/flare-foundation/coreth/ethdb/memorydb"
 )
 
 // makeTestTrie create a sample test trie to test node-wise reconstruction.
@@ -65,424 +61,3 @@ func makeTestTrie() (*Database, *SecureTrie, map[string][]byte) {
 	// Return the generated trie
 	return triedb, trie, content
 }
-
-// checkTrieContents cross references a reconstructed trie with an expected data
-// content map.
-func checkTrieContents(t *testing.T, db *Database, root []byte, content map[string][]byte) {
-	// Check root availability and trie contents
-	trie, err := NewSecure(common.BytesToHash(root), db)
-	if err != nil {
-		t.Fatalf("failed to create trie at %x: %v", root, err)
-	}
-	if err := checkTrieConsistency(db, common.BytesToHash(root)); err != nil {
-		t.Fatalf("inconsistent trie at %x: %v", root, err)
-	}
-	for key, val := range content {
-		if have := trie.Get([]byte(key)); !bytes.Equal(have, val) {
-			t.Errorf("entry %x: content mismatch: have %x, want %x", key, have, val)
-		}
-	}
-}
-
-// checkTrieConsistency checks that all nodes in a trie are indeed present.
```

```
-func checkTrieConsistency(db *Database, root common.Hash) error {
-	// Create and iterate a trie rooted in a subnode
-	trie, err := NewSecure(root, db)
-	if err != nil {
-		return nil // Consider a non existent state consistent
-	}
-	it := trie.NodeIterator(nil)
-	for it.Next(true) {
-	}
-	return it.Error()
-}
-
-// Tests that an empty trie is not scheduled for syncing.
-func TestEmptySync(t *testing.T) {
-	dbA := NewDatabase(memorydb.New())
-	dbB := NewDatabase(memorydb.New())
-	emptyA, _ := New(common.Hash{}, dbA)
-	emptyB, _ := New(emptyRoot, dbB)
-
-	for i, trie := range []*Trie{emptyA, emptyB} {
-		sync := NewSync(trie.Hash(), memorydb.New(), nil, NewSyncBloom(1, memorydb.New()))
-		if nodes, paths, codes := sync.Missing(1); len(nodes) != 0 || len(paths) != 0 || len(codes) != 0 {
-			t.Errorf("test %d: content requested for empty trie: %v, %v, %v", i, nodes, paths, codes)
-		}
-	}
-}
-
-// Tests that given a root hash, a trie can sync iteratively on a single thread,
-// requesting retrieval tasks and returning all of them in one go.
-func TestIterativeSyncIndividual(t *testing.T)       { testIterativeSync(t, 1, false) }
-func TestIterativeSyncBatched(t *testing.T)          { testIterativeSync(t, 100, false) }
-func TestIterativeSyncIndividualByPath(t *testing.T) { testIterativeSync(t, 1, true) }
-func TestIterativeSyncBatchedByPath(t *testing.T)    { testIterativeSync(t, 100, true) }
-
-func testIterativeSync(t *testing.T, count int, bypath bool) {
-	// Create a random trie to copy
-	srcDb, srcTrie, srcData := makeTestTrie()
-
-	// Create a destination trie and sync with the scheduler
-	diskdb := memorydb.New()
-	triedb := NewDatabase(diskdb)
-	sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-	nodes, paths, codes := sched.Missing(count)
-	var (
-		hashQueue []common.Hash
-		pathQueue []SyncPath
-	)
-	if !bypath {
-		hashQueue = append(append(hashQueue[:0], nodes...), codes...)
-	} else {
-		hashQueue = append(hashQueue[:0], codes...)
-		pathQueue = append(pathQueue[:0], paths...)
-	}
-	for len(hashQueue)+len(pathQueue) > 0 {
-		results := make([]SyncResult, len(hashQueue)+len(pathQueue))
-		for i, hash := range hashQueue {
-			data, err := srcDb.Node(hash)
-			if err != nil {
-				t.Fatalf("failed to retrieve node data for hash %x: %v", hash, err)
-			}
-			results[i] = SyncResult{hash, data}
-		}
-		for i, path := range pathQueue {
-			data, _, err := srcTrie.TryGetNode(path[0])
-			if err != nil {
-				t.Fatalf("failed to retrieve node data for path %x: %v", path, err)
-			}
-			results[len(hashQueue)+i] = SyncResult{crypto.Keccak256Hash(data), data}
-		}
-		for _, result := range results {
-			if err := sched.Process(result); err != nil {
-				t.Fatalf("failed to process result %v", err)
-			}
-		}
-		batch := diskdb.NewBatch()
-		if err := sched.Commit(batch); err != nil {
-			t.Fatalf("failed to commit data: %v", err)
-		}
-		batch.Write()
-
-		nodes, paths, codes = sched.Missing(count)
-		if !bypath {
-			hashQueue = append(append(hashQueue[:0], nodes...), codes...)
-		} else {
-			hashQueue = append(hashQueue[:0], codes...)
-			pathQueue = append(pathQueue[:0], paths...)
-		}
-	}
-	// Cross check that the two tries are in sync
-	checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-}
-
-// Tests that the trie scheduler can correctly reconstruct the state even if only
-// partial results are returned, and the others sent only later.
-func TestIterativeDelayedSync(t *testing.T) {
-	// Create a random trie to copy
-	srcDb, srcTrie, srcData := makeTestTrie()
-
-	// Create a destination trie and sync with the scheduler
-	diskdb := memorydb.New()
-	triedb := NewDatabase(diskdb)
-	sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-	nodes, _, codes := sched.Missing(10000)
-	queue := append(append([]common.Hash{}, nodes...), codes...)
-
-	for len(queue) > 0 {
-		// Sync only half of the scheduled nodes
-		results := make([]SyncResult, len(queue)/2+1)
-		for i, hash := range queue[:len(results)] {
-			data, err := srcDb.Node(hash)
-			if err != nil {
-				t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-			}
-			results[i] = SyncResult{hash, data}
-		}
-		for _, result := range results {
-			if err := sched.Process(result); err != nil {
-				t.Fatalf("failed to process result %v", err)
-			}
-		}
-		batch := diskdb.NewBatch()
-		if err := sched.Commit(batch); err != nil {
-			t.Fatalf("failed to commit data: %v", err)
-		}
-		batch.Write()
-
-		nodes, _, codes = sched.Missing(10000)
-		queue = append(append(queue[len(results):], nodes...), codes...)
-	}
-	// Cross check that the two tries are in sync
-	checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-}
```

```
-
-// Tests that given a root hash, a trie can sync iteratively on a single thread,
-// requesting retrieval tasks and returning all of them in one go, however in a
-// random order.
-func TestIterativeRandomSyncIndividual(t *testing.T) { testIterativeRandomSync(t, 1) }
-func TestIterativeRandomSyncBatched(t *testing.T)    { testIterativeRandomSync(t, 100) }
-
-func testIterativeRandomSync(t *testing.T, count int) {
-       // Create a random trie to copy
-       srcDb, srcTrie, srcData := makeTestTrie()
-
-       // Create a destination trie and sync with the scheduler
-       diskdb := memorydb.New()
-       triedb := NewDatabase(diskdb)
-       sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-       queue := make(map[common.Hash]struct{})
-       nodes, _, codes := sched.Missing(count)
-       for _, hash := range append(nodes, codes...) {
-               queue[hash] = struct{}{}
-       }
-       for len(queue) > 0 {
-               // Fetch all the queued nodes in a random order
-               results := make([]SyncResult, 0, len(queue))
-               for hash := range queue {
-                       data, err := srcDb.Node(hash)
-                       if err != nil {
-                               t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-                       }
-                       results = append(results, SyncResult{hash, data})
-               }
-               // Feed the retrieved results back and queue new tasks
-               for _, result := range results {
-                       if err := sched.Process(result); err != nil {
-                               t.Fatalf("failed to process result %v", err)
-                       }
-               }
-               batch := diskdb.NewBatch()
-               if err := sched.Commit(batch); err != nil {
-                       t.Fatalf("failed to commit data: %v", err)
-               }
-               batch.Write()
-
-               queue = make(map[common.Hash]struct{})
-               nodes, _, codes = sched.Missing(count)
-               for _, hash := range append(nodes, codes...) {
-                       queue[hash] = struct{}{}
-               }
-       }
-       // Cross check that the two tries are in sync
-       checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-}
-
-// Tests that the trie scheduler can correctly reconstruct the state even if only
-// partial results are returned (Even those randomly), others sent only later.
-func TestIterativeRandomDelayedSync(t *testing.T) {
-       // Create a random trie to copy
-       srcDb, srcTrie, srcData := makeTestTrie()
-
-       // Create a destination trie and sync with the scheduler
-       diskdb := memorydb.New()
-       triedb := NewDatabase(diskdb)
-       sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-       queue := make(map[common.Hash]struct{})
-       nodes, _, codes := sched.Missing(10000)
-       for _, hash := range append(nodes, codes...) {
-               queue[hash] = struct{}{}
-       }
-       for len(queue) > 0 {
-               // Sync only half of the scheduled nodes, even those in random order
-               results := make([]SyncResult, 0, len(queue)/2+1)
-               for hash := range queue {
-                       data, err := srcDb.Node(hash)
-                       if err != nil {
-                               t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-                       }
-                       results = append(results, SyncResult{hash, data})
-
-                       if len(results) >= cap(results) {
-                               break
-                       }
-               }
-               // Feed the retrieved results back and queue new tasks
-               for _, result := range results {
-                       if err := sched.Process(result); err != nil {
-                               t.Fatalf("failed to process result %v", err)
-                       }
-               }
-               batch := diskdb.NewBatch()
-               if err := sched.Commit(batch); err != nil {
-                       t.Fatalf("failed to commit data: %v", err)
-               }
-               batch.Write()
-               for _, result := range results {
-                       delete(queue, result.Hash)
-               }
-               nodes, _, codes = sched.Missing(10000)
-               for _, hash := range append(nodes, codes...) {
-                       queue[hash] = struct{}{}
-               }
-       }
-       // Cross check that the two tries are in sync
-       checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-}
-
-// Tests that a trie sync will not request nodes multiple times, even if they
-// have such references.
-func TestDuplicateAvoidanceSync(t *testing.T) {
-       // Create a random trie to copy
-       srcDb, srcTrie, srcData := makeTestTrie()
-
-       // Create a destination trie and sync with the scheduler
-       diskdb := memorydb.New()
-       triedb := NewDatabase(diskdb)
-       sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-       nodes, _, codes := sched.Missing(0)
-       queue := append(append([]common.Hash{}, nodes...), codes...)
-       requested := make(map[common.Hash]struct{})
-
-       for len(queue) > 0 {
-               results := make([]SyncResult, len(queue))
-               for i, hash := range queue {
-                       data, err := srcDb.Node(hash)
-                       if err != nil {
-                               t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-                       }
-                       if _, ok := requested[hash]; ok {
-                               t.Errorf("hash %x already requested once", hash)
-                       }
-                       requested[hash] = struct{}{}
-
-                       results[i] = SyncResult{hash, data}
-               }
-               for _, result := range results {
-                       if err := sched.Process(result); err != nil {
-                               t.Fatalf("failed to process result %v", err)
-                       }
-               }
-               batch := diskdb.NewBatch()
-               if err := sched.Commit(batch); err != nil {
-                       t.Fatalf("failed to commit data: %v", err)
-               }
-               batch.Write()
-
-               nodes, _, codes = sched.Missing(0)
-               queue = append(append(queue[:0], nodes...), codes...)
-       }
-       // Cross check that the two tries are in sync
-       checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-}
-
-// Tests that at any point in time during a sync, only complete sub-tries are in
-// the database.
-func TestIncompleteSync(t *testing.T) {
-       // Create a random trie to copy
-       srcDb, srcTrie, _ := makeTestTrie()
-
-       // Create a destination trie and sync with the scheduler
-       diskdb := memorydb.New()
-       triedb := NewDatabase(diskdb)
-       sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-       var added []common.Hash
-
-       nodes, _, codes := sched.Missing(1)
-       queue := append(append([]common.Hash{}, nodes...), codes...)
-       for len(queue) > 0 {
-               // Fetch a batch of trie nodes
-               results := make([]SyncResult, len(queue))
-               for i, hash := range queue {
-                       data, err := srcDb.Node(hash)
-                       if err != nil {
-                               t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-                       }
-                       results[i] = SyncResult{hash, data}
-               }
-               // Process each of the trie nodes
-               for _, result := range results {
-                       if err := sched.Process(result); err != nil {
-                               t.Fatalf("failed to process result %v", err)
-                       }
-               }
-               batch := diskdb.NewBatch()
-               if err := sched.Commit(batch); err != nil {
-                       t.Fatalf("failed to commit data: %v", err)
-               }
-               batch.Write()
-               for _, result := range results {
-                       added = append(added, result.Hash)
-                       // Check that all known sub-tries in the synced trie are complete
-                       if err := checkTrieConsistency(triedb, result.Hash); err != nil {
-                               t.Fatalf("trie inconsistent: %v", err)
-                       }
-               }
-               // Fetch the next batch to retrieve
-               nodes, _, codes = sched.Missing(1)
-               queue = append(append(queue[:0], nodes...), codes...)
-       }
-       // Sanity check that removing any node from the database is detected
-       for _, node := range added[1:] {
-               key := node.Bytes()
-               value, _ := diskdb.Get(key)
-
-               diskdb.Delete(key)
-               if err := checkTrieConsistency(triedb, added[0]); err == nil {
-                       t.Fatalf("trie inconsistency not caught, missing: %x", key)
-               }
-               diskdb.Put(key, value)
-       }
-}
-
-// Tests that trie nodes get scheduled lexicographically when having the same
-// depth.
-func TestSyncOrdering(t *testing.T) {
-       // Create a random trie to copy
-       srcDb, srcTrie, srcData := makeTestTrie()
-
-       // Create a destination trie and sync with the scheduler, tracking the requests
-       diskdb := memorydb.New()
-       triedb := NewDatabase(diskdb)
-       sched := NewSync(srcTrie.Hash(), diskdb, nil, NewSyncBloom(1, diskdb))
-
-       nodes, paths, _ := sched.Missing(1)
-       queue := append([]common.Hash{}, nodes...)
-       reqs := append([]SyncPath{}, paths...)
-
-       for len(queue) > 0 {
-               results := make([]SyncResult, len(queue))
-               for i, hash := range queue {
-                       data, err := srcDb.Node(hash)
-                       if err != nil {
-                               t.Fatalf("failed to retrieve node data for %x: %v", hash, err)
-                       }
-                       results[i] = SyncResult{hash, data}
-               }
-               for _, result := range results {
-                       if err := sched.Process(result); err != nil {
-                               t.Fatalf("failed to process result %v", err)
-                       }
-               }
-               batch := diskdb.NewBatch()
-               if err := sched.Commit(batch); err != nil {
-                       t.Fatalf("failed to commit data: %v", err)
-               }
-               batch.Write()
-
-               nodes, paths, _ = sched.Missing(1)
-               queue = append(queue[:0], nodes...)
-               reqs = append(reqs, paths...)
-       }
-       // Cross check that the two tries are in sync
-       checkTrieContents(t, triedb, srcTrie.Hash().Bytes(), srcData)
-
-       // Check that the trie nodes have been requested path-ordered
-       for i := 0; i < len(reqs)-1; i++ {
-               if len(reqs[i]) > 1 || len(reqs[i+1]) > 1 {
-                       // In the case of the trie tests, there's no storage so the tuples
-                       // must always be single items. 2-tuples should be tested in state.
-                       t.Errorf("Invalid request tuples: len(%v) or len(%v) > 1", reqs[i], reqs[i+1])
-               }
-               if bytes.Compare(compactToHex(reqs[i][0]), compactToHex(reqs[i+1][0])) > 0 {
-                       t.Errorf("Invalid request order: %v before %v", compactToHex(reqs[i][0]), compactToHex(reqs[i+1][0]))
-               }
-       }
-}
diff --git a/trie/trie.go b/trie/trie.go
```

```
index ac682725..200d79e5 100644
--- a/trie/trie.go
+++ b/trie/trie.go
@@ -33,11 +33,11 @@ import (
        "fmt"
        "sync"

-       "github.com/ava-labs/coreth/core/types"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/log"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/types"
 )

 var (
diff --git a/trie/trie_test.go b/trie/trie_test.go
index 8f2447ff..40f64b97 100644
--- a/trie/trie_test.go
+++ b/trie/trie_test.go
@@ -38,13 +38,13 @@ import (
        "testing"
        "testing/quick"

-       "github.com/ava-labs/coreth/core/rawdb"
-       "github.com/ava-labs/coreth/ethdb"
-       "github.com/ava-labs/coreth/ethdb/memorydb"
        "github.com/davecgh/go-spew/spew"
        "github.com/ethereum/go-ethereum/common"
        "github.com/ethereum/go-ethereum/crypto"
        "github.com/ethereum/go-ethereum/rlp"
+       "github.com/flare-foundation/coreth/core/rawdb"
+       "github.com/flare-foundation/coreth/ethdb"
+       "github.com/flare-foundation/coreth/ethdb/memorydb"
        "golang.org/x/crypto/sha3"
 )
```