ChatGPT

Idena-Go Hard Fork Implementation (Consensus V13) – Reported Flips Penalty

Code Modifications

- `blockchain/types/types.go` – *Extend BadAuthorReason enum:*
  Add a new reason code for authors with multiple reported flips. This will flag identities penalized for producing bad flips (>=2 reported by majority):

```
type BadAuthorReason byte

const (
    NoQualifiedFlipsBadAuthor    BadAuthorReason = 0
    QualifiedByNoneBadAuthor     BadAuthorReason = 1
    WrongWordsBadAuthor          BadAuthorReason = 2
    // New reason: flips reported by majority (>=2 flips)
    ReportedFlipsBadAuthor       BadAuthorReason = 3
)
```

- `config/consensus.go` – *Introduce ConsensusV13:*
  Define the new consensus version constant and initialize its config. Also add a feature flag for the fork's rules:

```
const (
    ConsensusV9  ConsensusVerson = 9
    ...
    ConsensusV12 ConsensusVerson = 12
    ConsensusV13 ConsensusVerson = 13    // Added new consensus version
)

var (
    v9, v10, v11, v12, v13 ConsensusConf
    ConsensusVersions map[ConsensusVerson]*ConsensusConf
)

func init() {
    ...
    v12 = v11
    ApplyConsensusVersion(ConsensusV12, &v12)
    ConsensusVersions[ConsensusV12] = &v12

    // Initialize config for new fork (V13)
    v13 = v12
```

```
    ApplyConsensusVersion(ConsensusV13, &v13)
    ConsensusVersions[ConsensusV13] = &v13
}

func ApplyConsensusVersion(ver ConsensusVerson, cfg *ConsensusConf) {
    switch ver {
    …
    case ConsensusV13:
        cfg.Version = ConsensusV13
        cfg.EnableUpgrade13 = true            // Enable hard fork rules
        cfg.StartActivationDate = time.Date(2025, 7, 1, 0, 0, 0, 0,
time.UTC).Unix()
        cfg.EndActivationDate   = time.Date(2025, 7, 10, 0, 0, 0, 0,
time.UTC).Unix()
    }
}
```

The activation window (dates/epoch) should be set as appropriate for the network fork.

- `core/ceremony/ceremony.go` – *Flip qualification analysis:*
Adjust flip analysis to count reported flips per author and assign the new bad author reason when applicable (under Consensus V13). Changes are made in the `analyzeAuthors` function:

```
func (vc *ValidationCeremony) analyzeAuthors(qualifications
[]FlipQualification, …, cfg *config.ConsensusConf) (…) {
    …
    badAuthors = make(map[common.Address]types.BadAuthorReason)
    goodAuthors = make(map[common.Address]*types.ValidationResult)
    authorResults = make(map[common.Address]*types.AuthorResults)
    reportedFlipsCount := make(map[common.Address]int)    // Track
number of reported flips per author
    …
    for flipIdx, item := range qualifications {
        cid := shard.flips[flipIdx]
        author := shard.flipAuthorMap[string(cid)]
        …
        if item.grade == types.GradeReported || item.status ==
QualifiedByNone {
            if item.status == QualifiedByNone {
                badAuthorsWithoutReport[author] = struct{}{}
            }
            if item.grade == types.GradeReported {
                badAuthors[author] = types.WrongWordsBadAuthor
                reportedFlipsCount[author]++                  //
Increment count of reported flips
                if !rewardAnyReport {
                    if item.status != Qualified && item.status !=
WeaklyQualified {
                        reportersToReward.deleteFlip(flipIdx)
                    }
```

```
                }
            } else if _, ok := badAuthors[author]; !ok {
                badAuthors[author] = types.QualifiedByNoneBadAuthor
            }
            authorResults[author].HasOneReportedFlip = true
        }
        if item.status == NotQualified {
            nonQualifiedFlips[author] += 1
            authorResults[author].HasOneNotQualifiedFlip = true
        }
        if item.status == Qualified || item.status == WeaklyQualified {
            vr, ok := goodAuthors[author]; if !ok {
                vr = new(types.ValidationResult)
                goodAuthors[author] = vr
            }
            vr.FlipsToReward = append(vr.FlipsToReward,
&types.FlipToReward{
                Cid: cid, Grade: item.grade, GradeScore:
item.gradeScore,
            })
        }
    }
    // After evaluating all flips, apply fork rule for multiple reported
flips
    if cfg.EnableUpgrade13 {
        for author, count := range reportedFlipsCount {
            if count >= 2 {
                badAuthors[author] = types.ReportedFlipsBadAuthor  //
Flag author for 2+ reported flips
            }
        }
    }
    ...
    for author := range badAuthors {
        delete(goodAuthors, author)
        reportersToReward.deleteReporter(author)
        if !rewardAnyReport {
            if _, ok := badAuthorsWithoutReport[author]; ok {
                for _, flipIdx := range madeFlips[author] {
                    reportersToReward.deleteFlip(flipIdx)
                }
            }
        }
    }
    return badAuthors, goodAuthors, authorResults, madeFlips,
reportersToReward
}
```

- `core/ceremony/ceremony.go` – *Identity state updates after validation:*
  In the loop that determines each identity's new state during the After Long Session, override the

state for penalized authors. If an identity was flagged with `ReportedFlipsBadAuthor`, set the new identity status to **Killed** (for Newbie) or **Suspended** (for Verified/Human). This logic is gated by the new consensus version flag:

```
...
identity := appState.State.GetIdentity(addr)
newIdentityState := determineNewIdentityState(identity, shortScore,
longScore, totalScore, totalFlips,
                                              missed, noQualShort,
noQualLong,
                                              vc.epoch >= 93,
vc.config.Consensus.EnableUpgrade10,
                                              shortQualifiedFlipsCount,
vc.config.Consensus.EnableUpgrade12)

// Hard fork rule (ConsensusV13): override state for multiple bad flips
if vc.config.Consensus.EnableUpgrade13 {
    if reason, penalized := shardValidationResults.BadAuthors[addr];
penalized && reason == types.ReportedFlipsBadAuthor {
        if identity.State == state.Newbie {
            newIdentityState = state.Killed                    // Kill
Newbie with >=2 bad flips
        } else if identity.State == state.Verified || identity.State ==
state.Human {
            newIdentityState = state.Suspended                 //
Suspend Verified/Human with >=2 bad flips
        }
    }
}

identityBirthday := determineIdentityBirthday(vc.epoch, identity,
newIdentityState)
incSuccessfulInvites(shardValidationResults, god, addr, identity,
identityBirthday, newIdentityState, vc.epoch, allGoodInviters)
setValidationResultToGoodAuthor(addr, newIdentityState, missed,
shardValidationResults)
...
```

- `blockchain/blockchain.go` – *Consensus upgrade validation:*
  Allow blocks to signal the new consensus upgrade (version 13) without being rejected. Extend the `ValidateHeader` check for known upgrade numbers to include V13:

```
func (chain *Blockchain) ValidateHeader(header, prevBlock
*types.Header) error {
    ...
    if header.ProposedHeader != nil && header.ProposedHeader.Upgrade >
0 {
        if header.ProposedHeader.Upgrade !=
uint32(chain.upgrader.Target()) &&
```

```
            (header.ProposedHeader.Upgrade !=
uint32(config.ConsensusV11) || chain.config.Consensus.Version >
config.ConsensusV11) &&
            (header.ProposedHeader.Upgrade !=
uint32(config.ConsensusV13) || chain.config.Consensus.Version >
config.ConsensusV13) {
                return errors.New("unknown consensus upgrade")
            }
        }
        ...
}
```

This ensures that a block proposing upgrade 13 is recognized (if the node is running the new software or if it's the special fork block), preventing an "unknown upgrade" error.

Testing

Unit tests are added to verify the new fork logic for various scenarios of reported flips. The tests simulate identities with different statuses and numbers of reported flips and then assert the expected outcomes:

```
import (
    "testing"
    "github.com/idena-network/idena-go/blockchain/types"
    "github.com/idena-network/idena-go/core/state"
)

func TestReportedFlipPenalties(t *testing.T) {
    addr := common.Address{} // dummy address for testing

    // Helper to simulate an identity's outcome after validation
    checkOutcome := func(initState, expectedNewState state.IdentityState,
flipsReported int) {
        // Set up identity and initial validation result
        identity := state.Identity{State: initState}
        newState := state.Verified  // assume the identity would stay/become
Verified without penalty

        // Simulate analyzeAuthors outcome: assign bad author reason based on
reported flips count
        badAuthors := make(map[common.Address]types.BadAuthorReason)
        if flipsReported >= 1 {
            // One or more flips reported triggers bad author (WrongWords for
1, updated to ReportedFlips for 2+)
            reason := types.WrongWordsBadAuthor
            if flipsReported >= 2 {
                reason = types.ReportedFlipsBadAuthor
            }
            badAuthors[addr] = reason
        }
```

```
        // Apply the fork penalty override as in ceremony logic
        if badReason, penalized := badAuthors[addr]; penalized && badReason
== types.ReportedFlipsBadAuthor {
            if identity.State == state.Newbie {
                newState = state.Killed      // Newbie with >=2 bad flips ->
Killed
            } else if identity.State == state.Verified || identity.State ==
state.Human {
                newState = state.Suspended   // Verified/Human with >=2 bad
flips -> Suspended
            }
        }

        if newState != expectedNewState {
            t.Errorf("Identity %v with %d reported flips: expected new state
%v, got %v",
                    initState, flipsReported, expectedNewState, newState)
        }
    }

    // 1. Newbie with 2 reported flips -> should be Killed
    checkOutcome(state.Newbie, state.Killed, flipsReported=2)

    // 2. Verified with 2 reported flips -> should be Suspended
    checkOutcome(state.Verified, state.Suspended, flipsReported=2)

    // 3. Human with 2 reported flips -> should be Suspended
    checkOutcome(state.Human, state.Suspended, flipsReported=2)

    // 4. Verified with 1 reported flip -> should remain Verified (no
suspension)
    checkOutcome(state.Verified, state.Verified, flipsReported=1)

    // 5. Newbie with 1 reported flip -> should **not** be killed (should
remain Newbie or become Verified if passed)
    checkOutcome(state.Newbie, state.Verified, flipsReported=1)
}
```

Each test scenario sets the initial identity status, simulates the number of flips reported, and then checks that the resulting `NewIdentityState` matches the expected outcome. For example, a Newbie with 2 bad flips should be killed (`NewIdentityState == Killed`), whereas a Verified with only 1 bad flip remains verified (no suspension).

📦 Build and Package

After implementing the above changes and updating tests, follow these steps to build and package the new fork release:

1. **Update Version and Tag Release:** Bump the version in project files if applicable (e.g. in `app/version.go` if exists) and create a Git tag for the fork. For example:

   ```
   git commit -am "Implement consensus V13 (flip report penalties hard fork)"
   git tag -a v1.2.0-fork14 -m "Idena Go v1.2.0 with hard fork #14 (Reported flips penalties)"
   git push origin master --tags
   ```

2. **Install Dependencies:** Ensure modules are up to date. Run `go mod tidy` to clean up `go.mod` and fetch any new dependencies if added.

3. **Build Executables:** Compile the binary for each target platform. For example:

4. **Linux (64-bit):** `GOOS=linux GOARCH=amd64 go build -o idena-go-linux ./`
5. **macOS (64-bit):** `GOOS=darwin GOARCH=amd64 go build -o idena-go-macos ./`
6. **Windows (64-bit):** `GOOS=windows GOARCH=amd64 go build -o idena-go-windows.exe ./`

You can also build the default binary for your current OS with:

```
go build -o idena-go .
```

This produces the `idena-go` executable with the new consensus rules.

1. **Run and Verify:** Launch a node with the new binary to ensure it starts correctly. For example:

   ```
   ./idena-go --config=config.json
   ```

   The node should report consensus `Version 13` in its logs or startup info.

2. **Package Releases:** Prepare distribution archives (zip/tar.gz) for each platform containing the binary and updated config/README. Name them appropriately (e.g., `idena-go-v1.2.0-fork14-windows.zip`, etc.).

3. **Update Documentation:** In the README and release notes, document the changes introduced in this fork:

4. Explain the new **Reported Flips** penalty: Identities producing ≥2 flips reported by the majority in one validation will be penalized. Newbies will **be killed**, and Verified/Human identities will be **suspended**.

5. Mention the activation epoch/block for **ConsensusV13** (the network will start enforcing these rules at the predetermined block height or validation ceremony date).
6. Include any other relevant details (e.g., new config flag `EnableUpgrade13`, any changes in flip reward distribution due to these penalties, etc.).

By following these steps, you will have a built and tagged release `v1.2.0-fork14` of **idena-go** with the new hard fork logic. This update enforces stricter consequences for creating poor-quality flips starting from the specified fork activation point.
