F:\git\java\mar3\filemonitor\target\go-ethereum\go-ethereum-3.doc

0:F:\git\coin\ethereum\go-ethereum\eth\downloader\downloader_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package downloader

import (
"errors"
"fmt"
"math/big"
"sync"
"sync/atomic"
"testing"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/state"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/trie"
)

var (
testKey, _ =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
testAddress = crypto.PubkeyToAddress(testKey.PublicKey)
)

// Reduce some of the parameters to make the tester faster.
func init() {
MaxForkAncestry = uint64(10000)
blockCacheLimit = 1024
fsCriticalTrials = 10
}

// downloadTester is a test simulator for mocking out local block chain.

```
type downloadTester struct {
downloader *Downloader

genesis *types.Block   // Genesis blocks used by the tester and peers
stateDb ethdb.Database // Database used by the tester for syncing from peers
peerDb  ethdb.Database // Database of the peers containing all data

ownHashes   []common.Hash                 // Hash chain belonging to the tester
ownHeaders  map[common.Hash]*types.Header  // Headers belonging to the tester
ownBlocks   map[common.Hash]*types.Block   // Blocks belonging to the tester
ownReceipts map[common.Hash]types.Receipts // Receipts belonging to the tester
ownChainTd  map[common.Hash]*big.Int       // Total difficulties of the blocks in the local chain

peerHashes   map[string][]common.Hash                 // Hash chain belonging to different test peers
peerHeaders  map[string]map[common.Hash]*types.Header  // Headers belonging to different test
peers
peerBlocks   map[string]map[common.Hash]*types.Block   // Blocks belonging to different test
peers
peerReceipts map[string]map[common.Hash]types.Receipts // Receipts belonging to different test
peers
peerChainTds map[string]map[common.Hash]*big.Int       // Total difficulties of the blocks in the
peer chains

peerMissingStates map[string]map[common.Hash]bool // State entries that fast sync should not
return

lock sync.RWMutex
}

// newTester creates a new downloader test mocker.
func newTester() *downloadTester {
testdb, _ := ethdb.NewMemDatabase()
genesis := core.GenesisBlockForTesting(testdb, testAddress, big.NewInt(1000000000))

tester := &downloadTester{
genesis:           genesis,
peerDb:            testdb,
ownHashes:         []common.Hash{genesis.Hash()},
ownHeaders:        map[common.Hash]*types.Header{genesis.Hash(): genesis.Header()},
ownBlocks:         map[common.Hash]*types.Block{genesis.Hash(): genesis},
ownReceipts:       map[common.Hash]types.Receipts{genesis.Hash(): nil},
ownChainTd:        map[common.Hash]*big.Int{genesis.Hash(): genesis.Difficulty()},
```

```
peerHashes:        make(map[string][]common.Hash),
peerHeaders:       make(map[string]map[common.Hash]*types.Header),
peerBlocks:        make(map[string]map[common.Hash]*types.Block),
peerReceipts:      make(map[string]map[common.Hash]types.Receipts),
peerChainTds:      make(map[string]map[common.Hash]*big.Int),
peerMissingStates: make(map[string]map[common.Hash]bool),
}
tester.stateDb, _ = ethdb.NewMemDatabase()
tester.stateDb.Put(genesis.Root().Bytes(), []byte{0x00})

tester.downloader = New(FullSync, tester.stateDb, new(event.TypeMux), tester.hasHeader,
tester.hasBlock, tester.getHeader,
tester.getBlock, tester.headHeader, tester.headBlock, tester.headFastBlock,
tester.commitHeadBlock, tester.getTd,
tester.insertHeaders, tester.insertBlocks, tester.insertReceipts, tester.rollback, tester.dropPeer)

return tester
}

// makeChain creates a chain of n blocks starting at and including parent.
// the returned hash chain is ordered head->parent. In addition, every 3rd block
// contains a transaction and every 5th an uncle to allow testing correct block
// reassembly.
func (dl *downloadTester) makeChain(n int, seed byte, parent *types.Block, parentReceipts
types.Receipts, heavy bool) ([]common.Hash, map[common.Hash]*types.Header,
map[common.Hash]*types.Block, map[common.Hash]types.Receipts) {
// Generate the block chain
blocks, receipts := core.GenerateChain(params.TestChainConfig, parent, dl.peerDb, n, func(i int,
block *core.BlockGen) {
block.SetCoinbase(common.Address{seed})

// If a heavy chain is requested, delay blocks to raise difficulty
if heavy {
block.OffsetTime(-1)
}
// If the block number is multiple of 3, send a bonus transaction to the miner
if parent == dl.genesis && i%3 == 0 {
signer := types.MakeSigner(params.TestChainConfig, block.Number())
tx, err := types.SignTx(types.NewTransaction(block.TxNonce(testAddress),
common.Address{seed}, big.NewInt(1000), new(big.Int).SetUint64(params.TxGas), nil, nil), signer,
testKey)
if err != nil {
```

```
panic(err)
}
block.AddTx(tx)
}
// If the block number is a multiple of 5, add a bonus uncle to the block
if i > 0 && i%5 == 0 {
block.AddUncle(&types.Header{
ParentHash: block.PrevBlock(i - 1).Hash(),
Number:     big.NewInt(block.Number().Int64() - 1),
})
}
})
// Convert the block-chain into a hash-chain and header/block maps
hashes := make([]common.Hash, n+1)
hashes[len(hashes)-1] = parent.Hash()

headerm := make(map[common.Hash]*types.Header, n+1)
headerm[parent.Hash()] = parent.Header()

blockm := make(map[common.Hash]*types.Block, n+1)
blockm[parent.Hash()] = parent

receiptm := make(map[common.Hash]types.Receipts, n+1)
receiptm[parent.Hash()] = parentReceipts

for i, b := range blocks {
hashes[len(hashes)-i-2] = b.Hash()
headerm[b.Hash()] = b.Header()
blockm[b.Hash()] = b
receiptm[b.Hash()] = receipts[i]
}
return hashes, headerm, blockm, receiptm
}

// makeChainFork creates two chains of length n, such that h1[:f] and
// h2[:f] are different but have a common suffix of length n-f.
func (dl *downloadTester) makeChainFork(n, f int, parent *types.Block, parentReceipts
types.Receipts, balanced bool) ([]common.Hash, []common.Hash,
map[common.Hash]*types.Header, map[common.Hash]*types.Header,
map[common.Hash]*types.Block, map[common.Hash]*types.Block,
map[common.Hash]types.Receipts, map[common.Hash]types.Receipts) {
// Create the common suffix
```

```
hashes, headers, blocks, receipts := dl.makeChain(n-f, 0, parent, parentReceipts, false)

// Create the forks, making the second heavyer if non balanced forks were requested
hashes1, headers1, blocks1, receipts1 := dl.makeChain(f, 1, blocks[hashes[0]],
receipts[hashes[0]], false)
hashes1 = append(hashes1, hashes[1:]...)

heavy := false
if !balanced {
heavy = true
}
hashes2, headers2, blocks2, receipts2 := dl.makeChain(f, 2, blocks[hashes[0]],
receipts[hashes[0]], heavy)
hashes2 = append(hashes2, hashes[1:]...)

for hash, header := range headers {
headers1[hash] = header
headers2[hash] = header
}
for hash, block := range blocks {
blocks1[hash] = block
blocks2[hash] = block
}
for hash, receipt := range receipts {
receipts1[hash] = receipt
receipts2[hash] = receipt
}
return hashes1, hashes2, headers1, headers2, blocks1, blocks2, receipts1, receipts2
}

// terminate aborts any operations on the embedded downloader and releases all
// held resources.
func (dl *downloadTester) terminate() {
dl.downloader.Terminate()
}

// sync starts synchronizing with a remote peer, blocking until it completes.
func (dl *downloadTester) sync(id string, td *big.Int, mode SyncMode) error {
dl.lock.RLock()
hash := dl.peerHashes[id][0]
// If no particular TD was requested, load from the peer's blockchain
if td == nil {
```

```
td = big.NewInt(1)
if diff, ok := dl.peerChainTds[id][hash]; ok {
td = diff
}
}
dl.lock.RUnlock()

// Synchronise with the chosen peer and ensure proper cleanup afterwards
err := dl.downloader.synchronise(id, hash, td, mode)
select {
case <-dl.downloader.cancelCh:
// Ok, downloader fully cancelled after sync cycle
default:
// Downloader is still accepting packets, can block a peer up
panic("downloader active post sync cycle") // panic will be caught by tester
}
return err
}

// hasHeader checks if a header is present in the testers canonical chain.
func (dl *downloadTester) hasHeader(hash common.Hash) bool {
return dl.getHeader(hash) != nil
}

// hasBlock checks if a block and associated state is present in the testers canonical chain.
func (dl *downloadTester) hasBlock(hash common.Hash) bool {
block := dl.getBlock(hash)
if block == nil {
return false
}
_, err := dl.stateDb.Get(block.Root().Bytes())
return err == nil
}

// getHeader retrieves a header from the testers canonical chain.
func (dl *downloadTester) getHeader(hash common.Hash) *types.Header {
dl.lock.RLock()
defer dl.lock.RUnlock()

return dl.ownHeaders[hash]
}
```

```
// getBlock retrieves a block from the testers canonical chain.
func (dl *downloadTester) getBlock(hash common.Hash) *types.Block {
dl.lock.RLock()
defer dl.lock.RUnlock()

return dl.ownBlocks[hash]
}

// headHeader retrieves the current head header from the canonical chain.
func (dl *downloadTester) headHeader() *types.Header {
dl.lock.RLock()
defer dl.lock.RUnlock()

for i := len(dl.ownHashes) - 1; i >= 0; i-- {
if header := dl.ownHeaders[dl.ownHashes[i]]; header != nil {
return header
}
}
return dl.genesis.Header()
}

// headBlock retrieves the current head block from the canonical chain.
func (dl *downloadTester) headBlock() *types.Block {
dl.lock.RLock()
defer dl.lock.RUnlock()

for i := len(dl.ownHashes) - 1; i >= 0; i-- {
if block := dl.ownBlocks[dl.ownHashes[i]]; block != nil {
if _, err := dl.stateDb.Get(block.Root().Bytes()); err == nil {
return block
}
}
}
return dl.genesis
}

// headFastBlock retrieves the current head fast-sync block from the canonical chain.
func (dl *downloadTester) headFastBlock() *types.Block {
dl.lock.RLock()
defer dl.lock.RUnlock()

for i := len(dl.ownHashes) - 1; i >= 0; i-- {
```

```
if block := dl.ownBlocks[dl.ownHashes[i]]; block != nil {
return block
}
}
return dl.genesis
}

// commitHeadBlock manually sets the head block to a given hash.
func (dl *downloadTester) commitHeadBlock(hash common.Hash) error {
// For now only check that the state trie is correct
if block := dl.getBlock(hash); block != nil {
_, err := trie.NewSecure(block.Root(), dl.stateDb, 0)
return err
}
return fmt.Errorf("non existent block: %x", hash[:4])
}

// getTd retrieves the block's total difficulty from the canonical chain.
func (dl *downloadTester) getTd(hash common.Hash) *big.Int {
dl.lock.RLock()
defer dl.lock.RUnlock()

return dl.ownChainTd[hash]
}

// insertHeaders injects a new batch of headers into the simulated chain.
func (dl *downloadTester) insertHeaders(headers []*types.Header, checkFreq int) (int, error) {
dl.lock.Lock()
defer dl.lock.Unlock()

// Do a quick check, as the blockchain.InsertHeaderChain doesn't insert anything in case of errors
if _, ok := dl.ownHeaders[headers[0].ParentHash]; !ok {
return 0, errors.New("unknown parent")
}
for i := 1; i < len(headers); i++ {
if headers[i].ParentHash != headers[i-1].Hash() {
return i, errors.New("unknown parent")
}
}
// Do a full insert if pre-checks passed
for i, header := range headers {
if _, ok := dl.ownHeaders[header.Hash()]; ok {
```

```
continue
}
if _, ok := dl.ownHeaders[header.ParentHash]; !ok {
return i, errors.New("unknown parent")
}
dl.ownHashes = append(dl.ownHashes, header.Hash())
dl.ownHeaders[header.Hash()] = header
dl.ownChainTd[header.Hash()] = new(big.Int).Add(dl.ownChainTd[header.ParentHash],
header.Difficulty)
}
return len(headers), nil
}

// insertBlocks injects a new batch of blocks into the simulated chain.
func (dl *downloadTester) insertBlocks(blocks types.Blocks) (int, error) {
dl.lock.Lock()
defer dl.lock.Unlock()

for i, block := range blocks {
if parent, ok := dl.ownBlocks[block.ParentHash()]; !ok {
return i, errors.New("unknown parent")
} else if _, err := dl.stateDb.Get(parent.Root().Bytes()); err != nil {
return i, fmt.Errorf("unknown parent state %x: %v", parent.Root(), err)
}
if _, ok := dl.ownHeaders[block.Hash()]; !ok {
dl.ownHashes = append(dl.ownHashes, block.Hash())
dl.ownHeaders[block.Hash()] = block.Header()
}
dl.ownBlocks[block.Hash()] = block
dl.stateDb.Put(block.Root().Bytes(), []byte{0x00})
dl.ownChainTd[block.Hash()] = new(big.Int).Add(dl.ownChainTd[block.ParentHash()],
block.Difficulty())
}
return len(blocks), nil
}

// insertReceipts injects a new batch of receipts into the simulated chain.
func (dl *downloadTester) insertReceipts(blocks types.Blocks, receipts []types.Receipts) (int, error)
{
dl.lock.Lock()
defer dl.lock.Unlock()
```

```
for i := 0; i < len(blocks) && i < len(receipts); i++ {
if _, ok := dl.ownHeaders[blocks[i].Hash()]; !ok {
return i, errors.New("unknown owner")
}
if _, ok := dl.ownBlocks[blocks[i].ParentHash()]; !ok {
return i, errors.New("unknown parent")
}
dl.ownBlocks[blocks[i].Hash()] = blocks[i]
dl.ownReceipts[blocks[i].Hash()] = receipts[i]
}
return len(blocks), nil
}

// rollback removes some recently added elements from the chain.
func (dl *downloadTester) rollback(hashes []common.Hash) {
dl.lock.Lock()
defer dl.lock.Unlock()

for i := len(hashes) - 1; i >= 0; i-- {
if dl.ownHashes[len(dl.ownHashes)-1] == hashes[i] {
dl.ownHashes = dl.ownHashes[:len(dl.ownHashes)-1]
}
delete(dl.ownChainTd, hashes[i])
delete(dl.ownHeaders, hashes[i])
delete(dl.ownReceipts, hashes[i])
delete(dl.ownBlocks, hashes[i])
}
}

// newPeer registers a new block download source into the downloader.
func (dl *downloadTester) newPeer(id string, version int, hashes []common.Hash, headers
map[common.Hash]*types.Header, blocks map[common.Hash]*types.Block, receipts
map[common.Hash]types.Receipts) error {
return dl.newSlowPeer(id, version, hashes, headers, blocks, receipts, 0)
}

// newSlowPeer registers a new block download source into the downloader, with a
// specific delay time on processing the network packets sent to it, simulating
// potentially slow network IO.
func (dl *downloadTester) newSlowPeer(id string, version int, hashes []common.Hash, headers
map[common.Hash]*types.Header, blocks map[common.Hash]*types.Block, receipts
map[common.Hash]types.Receipts, delay time.Duration) error {
```

```
dl.lock.Lock()
defer dl.lock.Unlock()

var err error
switch version {
case 62:
err = dl.downloader.RegisterPeer(id, version, dl.peerCurrentHeadFn(id),
dl.peerGetRelHeadersFn(id, delay), dl.peerGetAbsHeadersFn(id, delay), dl.peerGetBodiesFn(id,
delay), nil, nil)
case 63:
err = dl.downloader.RegisterPeer(id, version, dl.peerCurrentHeadFn(id),
dl.peerGetRelHeadersFn(id, delay), dl.peerGetAbsHeadersFn(id, delay), dl.peerGetBodiesFn(id,
delay), dl.peerGetReceiptsFn(id, delay), dl.peerGetNodeDataFn(id, delay))
case 64:
err = dl.downloader.RegisterPeer(id, version, dl.peerCurrentHeadFn(id),
dl.peerGetRelHeadersFn(id, delay), dl.peerGetAbsHeadersFn(id, delay), dl.peerGetBodiesFn(id,
delay), dl.peerGetReceiptsFn(id, delay), dl.peerGetNodeDataFn(id, delay))
}
if err == nil {
// Assign the owned hashes, headers and blocks to the peer (deep copy)
dl.peerHashes[id] = make([]common.Hash, len(hashes))
copy(dl.peerHashes[id], hashes)

dl.peerHeaders[id] = make(map[common.Hash]*types.Header)
dl.peerBlocks[id] = make(map[common.Hash]*types.Block)
dl.peerReceipts[id] = make(map[common.Hash]types.Receipts)
dl.peerChainTds[id] = make(map[common.Hash]*big.Int)
dl.peerMissingStates[id] = make(map[common.Hash]bool)

genesis := hashes[len(hashes)-1]
if header := headers[genesis]; header != nil {
dl.peerHeaders[id][genesis] = header
dl.peerChainTds[id][genesis] = header.Difficulty
}
if block := blocks[genesis]; block != nil {
dl.peerBlocks[id][genesis] = block
dl.peerChainTds[id][genesis] = block.Difficulty()
}

for i := len(hashes) - 2; i >= 0; i-- {
hash := hashes[i]
```

```
if header, ok := headers[hash]; ok {
dl.peerHeaders[id][hash] = header
if _, ok := dl.peerHeaders[id][header.ParentHash]; ok {
dl.peerChainTds[id][hash] = new(big.Int).Add(header.Difficulty,
dl.peerChainTds[id][header.ParentHash])
}
}
if block, ok := blocks[hash]; ok {
dl.peerBlocks[id][hash] = block
if _, ok := dl.peerBlocks[id][block.ParentHash()]; ok {
dl.peerChainTds[id][hash] = new(big.Int).Add(block.Difficulty(),
dl.peerChainTds[id][block.ParentHash()])
}
}
if receipt, ok := receipts[hash]; ok {
dl.peerReceipts[id][hash] = receipt
}
}
}
return err
}

// dropPeer simulates a hard peer removal from the connection pool.
func (dl *downloadTester) dropPeer(id string) {
dl.lock.Lock()
defer dl.lock.Unlock()

delete(dl.peerHashes, id)
delete(dl.peerHeaders, id)
delete(dl.peerBlocks, id)
delete(dl.peerChainTds, id)

dl.downloader.UnregisterPeer(id)
}

// peerCurrentHeadFn constructs a function to retrieve a peer's current head hash
// and total difficulty.
func (dl *downloadTester) peerCurrentHeadFn(id string) func() (common.Hash, *big.Int) {
return func() (common.Hash, *big.Int) {
dl.lock.RLock()
defer dl.lock.RUnlock()
```

```
return dl.peerHashes[id][0], nil
}
}

// peerGetRelHeadersFn constructs a GetBlockHeaders function based on a hashed
// origin; associated with a particular peer in the download tester. The returned
// function can be used to retrieve batches of headers from the particular peer.
func (dl *downloadTester) peerGetRelHeadersFn(id string, delay time.Duration)
func(common.Hash, int, int, bool) error {
return func(origin common.Hash, amount int, skip int, reverse bool) error {
// Find the canonical number of the hash
dl.lock.RLock()
number := uint64(0)
for num, hash := range dl.peerHashes[id] {
if hash == origin {
number = uint64(len(dl.peerHashes[id]) - num - 1)
break
}
}
dl.lock.RUnlock()

// Use the absolute header fetcher to satisfy the query
return dl.peerGetAbsHeadersFn(id, delay)(number, amount, skip, reverse)
}
}

// peerGetAbsHeadersFn constructs a GetBlockHeaders function based on a numbered
// origin; associated with a particular peer in the download tester. The returned
// function can be used to retrieve batches of headers from the particular peer.
func (dl *downloadTester) peerGetAbsHeadersFn(id string, delay time.Duration) func(uint64, int,
int, bool) error {
return func(origin uint64, amount int, skip int, reverse bool) error {
time.Sleep(delay)

dl.lock.RLock()
defer dl.lock.RUnlock()

// Gather the next batch of headers
hashes := dl.peerHashes[id]
headers := dl.peerHeaders[id]
result := make([]*types.Header, 0, amount)
for i := 0; i < amount && len(hashes)-int(origin)-1-i*(skip+1) >= 0; i++ {
```

```
if header, ok := headers[hashes[len(hashes)-int(origin)-1-i*(skip+1)]]; ok {
result = append(result, header)
}
}
// Delay delivery a bit to allow attacks to unfold
go func() {
time.Sleep(time.Millisecond)
dl.downloader.DeliverHeaders(id, result)
}()
return nil
}
}

// peerGetBodiesFn constructs a getBlockBodies method associated with a particular
// peer in the download tester. The returned function can be used to retrieve
// batches of block bodies from the particularly requested peer.
func (dl *downloadTester) peerGetBodiesFn(id string, delay time.Duration) func([]common.Hash)
error {
return func(hashes []common.Hash) error {
time.Sleep(delay)

dl.lock.RLock()
defer dl.lock.RUnlock()

blocks := dl.peerBlocks[id]

transactions := make([][]*types.Transaction, 0, len(hashes))
uncles := make([][]*types.Header, 0, len(hashes))

for _, hash := range hashes {
if block, ok := blocks[hash]; ok {
transactions = append(transactions, block.Transactions())
uncles = append(uncles, block.Uncles())
}
}
go dl.downloader.DeliverBodies(id, transactions, uncles)

return nil
}
}

// peerGetReceiptsFn constructs a getReceipts method associated with a particular
```

```
// peer in the download tester. The returned function can be used to retrieve
// batches of block receipts from the particularly requested peer.
func (dl *downloadTester) peerGetReceiptsFn(id string, delay time.Duration) func([]common.Hash)
error {
return func(hashes []common.Hash) error {
time.Sleep(delay)

dl.lock.RLock()
defer dl.lock.RUnlock()

receipts := dl.peerReceipts[id]

results := make([][]*types.Receipt, 0, len(hashes))
for _, hash := range hashes {
if receipt, ok := receipts[hash]; ok {
results = append(results, receipt)
}
}
go dl.downloader.DeliverReceipts(id, results)

return nil
}
}

// peerGetNodeDataFn constructs a getNodeData method associated with a particular
// peer in the download tester. The returned function can be used to retrieve
// batches of node state data from the particularly requested peer.
func (dl *downloadTester) peerGetNodeDataFn(id string, delay time.Duration)
func([]common.Hash) error {
return func(hashes []common.Hash) error {
time.Sleep(delay)

dl.lock.RLock()
defer dl.lock.RUnlock()

results := make([][]byte, 0, len(hashes))
for _, hash := range hashes {
if data, err := dl.peerDb.Get(hash.Bytes()); err == nil {
if !dl.peerMissingStates[id][hash] {
results = append(results, data)
}
}
```

```
}
go dl.downloader.DeliverNodeData(id, results)

return nil
}
}

// assertOwnChain checks if the local chain contains the correct number of items
// of the various chain components.
func assertOwnChain(t *testing.T, tester *downloadTester, length int) {
assertOwnForkedChain(t, tester, 1, []int{length})
}

// assertOwnForkedChain checks if the local forked chain contains the correct
// number of items of the various chain components.
func assertOwnForkedChain(t *testing.T, tester *downloadTester, common int, lengths []int) {
// Initialize the counters for the first fork
headers, blocks := lengths[0], lengths[0]

minReceipts, maxReceipts := lengths[0]-fsMinFullBlocks-fsPivotInterval, lengths[0]-
fsMinFullBlocks
if minReceipts < 0 {
minReceipts = 1
}
if maxReceipts < 0 {
maxReceipts = 1
}
// Update the counters for each subsequent fork
for _, length := range lengths[1:] {
headers += length - common
blocks += length - common

minReceipts += length - common - fsMinFullBlocks - fsPivotInterval
maxReceipts += length - common - fsMinFullBlocks
}
switch tester.downloader.mode {
case FullSync:
minReceipts, maxReceipts = 1, 1
case LightSync:
blocks, minReceipts, maxReceipts = 1, 1, 1
}
if hs := len(tester.ownHeaders); hs != headers {
```

```
t.Fatalf("synchronised headers mismatch: have %v, want %v", hs, headers)
}
if bs := len(tester.ownBlocks); bs != blocks {
t.Fatalf("synchronised blocks mismatch: have %v, want %v", bs, blocks)
}
if rs := len(tester.ownReceipts); rs < minReceipts || rs > maxReceipts {
t.Fatalf("synchronised receipts mismatch: have %v, want between [%v, %v]", rs, minReceipts,
maxReceipts)
}
// Verify the state trie too for fast syncs
if tester.downloader.mode == FastSync {
var index int
if pivot := int(tester.downloader.queue.fastSyncPivot); pivot < common {
index = pivot
} else {
index = len(tester.ownHashes) - lengths[len(lengths)-1] +
int(tester.downloader.queue.fastSyncPivot)
}
if index > 0 {
if statedb, err := state.New(tester.ownHeaders[tester.ownHashes[index]].Root,
state.NewDatabase(tester.stateDb)); statedb == nil || err != nil {
t.Fatalf("state reconstruction failed: %v", err)
}
}
}
}

// Tests that simple synchronization against a canonical chain works correctly.
// In this test common ancestor lookup should be short circuited and not require
// binary searching.
func TestCanonicalSynchronisation62(t *testing.T)      { testCanonicalSynchronisation(t, 62,
FullSync) }
func TestCanonicalSynchronisation63Full(t *testing.T)  { testCanonicalSynchronisation(t, 63,
FullSync) }
func TestCanonicalSynchronisation63Fast(t *testing.T)  { testCanonicalSynchronisation(t, 63,
FastSync) }
func TestCanonicalSynchronisation64Full(t *testing.T)  { testCanonicalSynchronisation(t, 64,
FullSync) }
func TestCanonicalSynchronisation64Fast(t *testing.T)  { testCanonicalSynchronisation(t, 64,
FastSync) }
func TestCanonicalSynchronisation64Light(t *testing.T) { testCanonicalSynchronisation(t, 64,
LightSync) }
```

```
func testCanonicalSynchronisation(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)

// Synchronise with the peer and make sure all relevant data was retrieved
if err := tester.sync("peer", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)
}

// Tests that if a large batch of blocks are being downloaded, it is throttled
// until the cached blocks are retrieved.
func TestThrottling62(t *testing.T)     { testThrottling(t, 62, FullSync) }
func TestThrottling63Full(t *testing.T) { testThrottling(t, 63, FullSync) }
func TestThrottling63Fast(t *testing.T) { testThrottling(t, 63, FastSync) }
func TestThrottling64Full(t *testing.T) { testThrottling(t, 64, FullSync) }
func TestThrottling64Fast(t *testing.T) { testThrottling(t, 64, FastSync) }

func testThrottling(t *testing.T, protocol int, mode SyncMode) {
tester := newTester()
defer tester.terminate()

// Create a long block chain to download and the tester
targetBlocks := 8 * blockCacheLimit
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)

// Wrap the importer to allow stepping
blocked, proceed := uint32(0), make(chan struct{})
tester.downloader.chainInsertHook = func(results []*fetchResult) {
atomic.StoreUint32(&blocked, uint32(len(results)))
```

```
<-proceed
}
// Start a synchronisation concurrently
errc := make(chan error)
go func() {
errc <- tester.sync("peer", nil, mode)
}()
// Iteratively take some blocks, always checking the retrieval count
for {
// Check the retrieval count synchronously (! reason for this ugly block)
tester.lock.RLock()
retrieved := len(tester.ownBlocks)
tester.lock.RUnlock()
if retrieved >= targetBlocks+1 {
break
}
// Wait a bit for sync to throttle itself
var cached, frozen int
for start := time.Now(); time.Since(start) < 3*time.Second; {
time.Sleep(25 * time.Millisecond)

tester.lock.Lock()
tester.downloader.queue.lock.Lock()
cached = len(tester.downloader.queue.blockDonePool)
if mode == FastSync {
if receipts := len(tester.downloader.queue.receiptDonePool); receipts < cached {
if tester.downloader.queue.resultCache[receipts].Header.Number.Uint64() <
tester.downloader.queue.fastSyncPivot {
cached = receipts
}
}
}
frozen = int(atomic.LoadUint32(&blocked))
retrieved = len(tester.ownBlocks)
tester.downloader.queue.lock.Unlock()
tester.lock.Unlock()

if cached == blockCacheLimit || retrieved+cached+frozen == targetBlocks+1 {
break
}
}
// Make sure we filled up the cache, then exhaust it
```

```
time.Sleep(25 * time.Millisecond) // give it a chance to screw up

tester.lock.RLock()
retrieved = len(tester.ownBlocks)
tester.lock.RUnlock()
if cached != blockCacheLimit && retrieved+cached+frozen != targetBlocks+1 {
t.Fatalf("block count mismatch: have %v, want %v (owned %v, blocked %v, target %v)", cached,
blockCacheLimit, retrieved, frozen, targetBlocks+1)
}
// Permit the blocked blocks to import
if atomic.LoadUint32(&blocked) > 0 {
atomic.StoreUint32(&blocked, uint32(0))
proceed <- struct{}{}
}
}
// Check that we haven't pulled more blocks than available
assertOwnChain(t, tester, targetBlocks+1)
if err := <-errc; err != nil {
t.Fatalf("block synchronization failed: %v", err)
}
}

// Tests that simple synchronization against a forked chain works correctly. In
// this test common ancestor lookup should *not* be short circuited, and a full
// binary search should be executed.
func TestForkedSync62(t *testing.T)      { testForkedSync(t, 62, FullSync) }
func TestForkedSync63Full(t *testing.T)  { testForkedSync(t, 63, FullSync) }
func TestForkedSync63Fast(t *testing.T)  { testForkedSync(t, 63, FastSync) }
func TestForkedSync64Full(t *testing.T)  { testForkedSync(t, 64, FullSync) }
func TestForkedSync64Fast(t *testing.T)  { testForkedSync(t, 64, FastSync) }
func TestForkedSync64Light(t *testing.T) { testForkedSync(t, 64, LightSync) }

func testForkedSync(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a long enough forked chain
common, fork := MaxHashFetch, 2*MaxHashFetch
hashesA, hashesB, headersA, headersB, blocksA, blocksB, receiptsA, receiptsB :=
tester.makeChainFork(common+fork, fork, tester.genesis, nil, true)
```

```
tester.newPeer("fork A", protocol, hashesA, headersA, blocksA, receiptsA)
tester.newPeer("fork B", protocol, hashesB, headersB, blocksB, receiptsB)

// Synchronise with the peer and make sure all blocks were retrieved
if err := tester.sync("fork A", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, common+fork+1)

// Synchronise with the second peer and make sure that fork is pulled too
if err := tester.sync("fork B", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnForkedChain(t, tester, common+1, []int{common + fork + 1, common + fork + 1})
}

// Tests that synchronising against a much shorter but much heavyer fork works
// corrently and is not dropped.
func TestHeavyForkedSync62(t *testing.T)      { testHeavyForkedSync(t, 62, FullSync) }
func TestHeavyForkedSync63Full(t *testing.T)  { testHeavyForkedSync(t, 63, FullSync) }
func TestHeavyForkedSync63Fast(t *testing.T)  { testHeavyForkedSync(t, 63, FastSync) }
func TestHeavyForkedSync64Full(t *testing.T)  { testHeavyForkedSync(t, 64, FullSync) }
func TestHeavyForkedSync64Fast(t *testing.T)  { testHeavyForkedSync(t, 64, FastSync) }
func TestHeavyForkedSync64Light(t *testing.T) { testHeavyForkedSync(t, 64, LightSync) }

func testHeavyForkedSync(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a long enough forked chain
common, fork := MaxHashFetch, 4*MaxHashFetch
hashesA, hashesB, headersA, headersB, blocksA, blocksB, receiptsA, receiptsB :=
tester.makeChainFork(common+fork, fork, tester.genesis, nil, false)

tester.newPeer("light", protocol, hashesA, headersA, blocksA, receiptsA)
tester.newPeer("heavy", protocol, hashesB[fork/2:], headersB, blocksB, receiptsB)

// Synchronise with the peer and make sure all blocks were retrieved
if err := tester.sync("light", nil, mode); err != nil {
```

```
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, common+fork+1)

// Synchronise with the second peer and make sure that fork is pulled too
if err := tester.sync("heavy", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnForkedChain(t, tester, common+1, []int{common + fork + 1, common + fork/2 + 1})
}

// Tests that chain forks are contained within a certain interval of the current
// chain head, ensuring that malicious peers cannot waste resources by feeding
// long dead chains.
func TestBoundedForkedSync62(t *testing.T)      { testBoundedForkedSync(t, 62, FullSync) }
func TestBoundedForkedSync63Full(t *testing.T)  { testBoundedForkedSync(t, 63, FullSync) }
func TestBoundedForkedSync63Fast(t *testing.T)  { testBoundedForkedSync(t, 63, FastSync) }
func TestBoundedForkedSync64Full(t *testing.T)  { testBoundedForkedSync(t, 64, FullSync) }
func TestBoundedForkedSync64Fast(t *testing.T)  { testBoundedForkedSync(t, 64, FastSync) }
func TestBoundedForkedSync64Light(t *testing.T) { testBoundedForkedSync(t, 64, LightSync) }

func testBoundedForkedSync(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a long enough forked chain
common, fork := 13, int(MaxForkAncestry+17)
hashesA, hashesB, headersA, headersB, blocksA, blocksB, receiptsA, receiptsB :=
tester.makeChainFork(common+fork, fork, tester.genesis, nil, true)

tester.newPeer("original", protocol, hashesA, headersA, blocksA, receiptsA)
tester.newPeer("rewriter", protocol, hashesB, headersB, blocksB, receiptsB)

// Synchronise with the peer and make sure all blocks were retrieved
if err := tester.sync("original", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, common+fork+1)

// Synchronise with the second peer and ensure that the fork is rejected to being too old
```

```
if err := tester.sync("rewriter", nil, mode); err != errInvalidAncestor {
t.Fatalf("sync failure mismatch: have %v, want %v", err, errInvalidAncestor)
}
}

// Tests that chain forks are contained within a certain interval of the current
// chain head for short but heavy forks too. These are a bit special because they
// take different ancestor lookup paths.
func TestBoundedHeavyForkedSync62(t *testing.T)      { testBoundedHeavyForkedSync(t, 62, FullSync) }
func TestBoundedHeavyForkedSync63Full(t *testing.T)  { testBoundedHeavyForkedSync(t, 63, FullSync) }
func TestBoundedHeavyForkedSync63Fast(t *testing.T)  { testBoundedHeavyForkedSync(t, 63, FastSync) }
func TestBoundedHeavyForkedSync64Full(t *testing.T)  { testBoundedHeavyForkedSync(t, 64, FullSync) }
func TestBoundedHeavyForkedSync64Fast(t *testing.T)  { testBoundedHeavyForkedSync(t, 64, FastSync) }
func TestBoundedHeavyForkedSync64Light(t *testing.T) { testBoundedHeavyForkedSync(t, 64, LightSync) }

func testBoundedHeavyForkedSync(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a long enough forked chain
common, fork := 13, int(MaxForkAncestry+17)
hashesA, hashesB, headersA, headersB, blocksA, blocksB, receiptsA, receiptsB :=
tester.makeChainFork(common+fork, fork, tester.genesis, nil, false)

tester.newPeer("original", protocol, hashesA, headersA, blocksA, receiptsA)
tester.newPeer("heavy-rewriter", protocol, hashesB[MaxForkAncestry-17:], headersB, blocksB,
receiptsB) // Root the fork below the ancestor limit

// Synchronise with the peer and make sure all blocks were retrieved
if err := tester.sync("original", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, common+fork+1)
```

```
// Synchronise with the second peer and ensure that the fork is rejected to being too old
if err := tester.sync("heavy-rewriter", nil, mode); err != errInvalidAncestor {
t.Fatalf("sync failure mismatch: have %v, want %v", err, errInvalidAncestor)
}
}

// Tests that an inactive downloader will not accept incoming block headers and
// bodies.
func TestInactiveDownloader62(t *testing.T) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Check that neither block headers nor bodies are accepted
if err := tester.downloader.DeliverHeaders("bad peer", []*types.Header{}); err != errNoSyncActive {
t.Errorf("error mismatch: have %v, want %v", err, errNoSyncActive)
}
if err := tester.downloader.DeliverBodies("bad peer", [][]*types.Transaction{}, [][]*types.Header{});
err != errNoSyncActive {
t.Errorf("error mismatch: have %v, want %v", err, errNoSyncActive)
}
}

// Tests that an inactive downloader will not accept incoming block headers,
// bodies and receipts.
func TestInactiveDownloader63(t *testing.T) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Check that neither block headers nor bodies are accepted
if err := tester.downloader.DeliverHeaders("bad peer", []*types.Header{}); err != errNoSyncActive {
t.Errorf("error mismatch: have %v, want %v", err, errNoSyncActive)
}
if err := tester.downloader.DeliverBodies("bad peer", [][]*types.Transaction{}, [][]*types.Header{});
err != errNoSyncActive {
t.Errorf("error mismatch: have %v, want %v", err, errNoSyncActive)
}
if err := tester.downloader.DeliverReceipts("bad peer", [][]*types.Receipt{}); err != errNoSyncActive
{
```

```
t.Errorf("error mismatch: have %v, want %v", err, errNoSyncActive)
}
}

// Tests that a canceled download wipes all previously accumulated state.
func TestCancel62(t *testing.T)      { testCancel(t, 62, FullSync) }
func TestCancel63Full(t *testing.T)  { testCancel(t, 63, FullSync) }
func TestCancel63Fast(t *testing.T)  { testCancel(t, 63, FastSync) }
func TestCancel64Full(t *testing.T)  { testCancel(t, 64, FullSync) }
func TestCancel64Fast(t *testing.T)  { testCancel(t, 64, FastSync) }
func TestCancel64Light(t *testing.T) { testCancel(t, 64, LightSync) }

func testCancel(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download and the tester
targetBlocks := blockCacheLimit - 15
if targetBlocks >= MaxHashFetch {
targetBlocks = MaxHashFetch - 15
}
if targetBlocks >= MaxHeaderFetch {
targetBlocks = MaxHeaderFetch - 15
}
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)

// Make sure canceling works with a pristine downloader
tester.downloader.Cancel()
if !tester.downloader.queue.Idle() {
t.Errorf("download queue not idle")
}
// Synchronise with the peer, but cancel afterwards
if err := tester.sync("peer", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
tester.downloader.Cancel()
if !tester.downloader.queue.Idle() {
t.Errorf("download queue not idle")
```

```
}
}

// Tests that synchronisation from multiple peers works as intended (multi thread sanity test).
func TestMultiSynchronisation62(t *testing.T)      { testMultiSynchronisation(t, 62, FullSync) }
func TestMultiSynchronisation63Full(t *testing.T)  { testMultiSynchronisation(t, 63, FullSync) }
func TestMultiSynchronisation63Fast(t *testing.T)  { testMultiSynchronisation(t, 63, FastSync) }
func TestMultiSynchronisation64Full(t *testing.T)  { testMultiSynchronisation(t, 64, FullSync) }
func TestMultiSynchronisation64Fast(t *testing.T)  { testMultiSynchronisation(t, 64, FastSync) }
func TestMultiSynchronisation64Light(t *testing.T) { testMultiSynchronisation(t, 64, LightSync) }

func testMultiSynchronisation(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create various peers with various parts of the chain
targetPeers := 8
targetBlocks := targetPeers*blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

for i := 0; i < targetPeers; i++ {
id := fmt.Sprintf("peer #%d", i)
tester.newPeer(id, protocol, hashes[i*blockCacheLimit:], headers, blocks, receipts)
}
if err := tester.sync("peer #0", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)
}

// Tests that synchronisations behave well in multi-version protocol environments
// and not wreak havoc on other nodes in the network.
func TestMultiProtoSynchronisation62(t *testing.T)      { testMultiProtoSync(t, 62, FullSync) }
func TestMultiProtoSynchronisation63Full(t *testing.T)  { testMultiProtoSync(t, 63, FullSync) }
func TestMultiProtoSynchronisation63Fast(t *testing.T)  { testMultiProtoSync(t, 63, FastSync) }
func TestMultiProtoSynchronisation64Full(t *testing.T)  { testMultiProtoSync(t, 64, FullSync) }
func TestMultiProtoSynchronisation64Fast(t *testing.T)  { testMultiProtoSync(t, 64, FastSync) }
func TestMultiProtoSynchronisation64Light(t *testing.T) { testMultiProtoSync(t, 64, LightSync) }

func testMultiProtoSync(t *testing.T, protocol int, mode SyncMode) {
```

```
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Create peers of every type
tester.newPeer("peer 62", 62, hashes, headers, blocks, nil)
tester.newPeer("peer 63", 63, hashes, headers, blocks, receipts)
tester.newPeer("peer 64", 64, hashes, headers, blocks, receipts)

// Synchronise with the requested peer and make sure all blocks were retrieved
if err := tester.sync(fmt.Sprintf("peer %d", protocol), nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)

// Check that no peers have been dropped off
for _, version := range []int{62, 63, 64} {
peer := fmt.Sprintf("peer %d", version)
if _, ok := tester.peerHashes[peer]; !ok {
t.Errorf("%s dropped", peer)
}
}
}

// Tests that if a block is empty (e.g. header only), no body request should be
// made, and instead the header should be assembled into a whole block in itself.
func TestEmptyShortCircuit62(t *testing.T)      { testEmptyShortCircuit(t, 62, FullSync) }
func TestEmptyShortCircuit63Full(t *testing.T)  { testEmptyShortCircuit(t, 63, FullSync) }
func TestEmptyShortCircuit63Fast(t *testing.T)  { testEmptyShortCircuit(t, 63, FastSync) }
func TestEmptyShortCircuit64Full(t *testing.T)  { testEmptyShortCircuit(t, 64, FullSync) }
func TestEmptyShortCircuit64Fast(t *testing.T)  { testEmptyShortCircuit(t, 64, FastSync) }
func TestEmptyShortCircuit64Light(t *testing.T) { testEmptyShortCircuit(t, 64, LightSync) }

func testEmptyShortCircuit(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
```

```
defer tester.terminate()

// Create a block chain to download
targetBlocks := 2*blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)

// Instrument the downloader to signal body requests
bodiesHave, receiptsHave := int32(0), int32(0)
tester.downloader.bodyFetchHook = func(headers []*types.Header) {
atomic.AddInt32(&bodiesHave, int32(len(headers)))
}
tester.downloader.receiptFetchHook = func(headers []*types.Header) {
atomic.AddInt32(&receiptsHave, int32(len(headers)))
}
// Synchronise with the peer and make sure all blocks were retrieved
if err := tester.sync("peer", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)

// Validate the number of block bodies that should have been requested
bodiesNeeded, receiptsNeeded := 0, 0
for _, block := range blocks {
if mode != LightSync && block != tester.genesis && (len(block.Transactions()) > 0 ||
len(block.Uncles()) > 0) {
bodiesNeeded++
}
}
for hash, receipt := range receipts {
if mode == FastSync && len(receipt) > 0 && headers[hash].Number.Uint64() <=
tester.downloader.queue.fastSyncPivot {
receiptsNeeded++
}
}
if int(bodiesHave) != bodiesNeeded {
t.Errorf("body retrieval count mismatch: have %v, want %v", bodiesHave, bodiesNeeded)
}
if int(receiptsHave) != receiptsNeeded {
t.Errorf("receipt retrieval count mismatch: have %v, want %v", receiptsHave, receiptsNeeded)
}
```

```
}

// Tests that headers are enqueued continuously, preventing malicious nodes from
// stalling the downloader by feeding gapped header chains.
func TestMissingHeaderAttack62(t *testing.T)      { testMissingHeaderAttack(t, 62, FullSync) }
func TestMissingHeaderAttack63Full(t *testing.T)  { testMissingHeaderAttack(t, 63, FullSync) }
func TestMissingHeaderAttack63Fast(t *testing.T)  { testMissingHeaderAttack(t, 63, FastSync) }
func TestMissingHeaderAttack64Full(t *testing.T)  { testMissingHeaderAttack(t, 64, FullSync) }
func TestMissingHeaderAttack64Fast(t *testing.T)  { testMissingHeaderAttack(t, 64, FastSync) }
func TestMissingHeaderAttack64Light(t *testing.T) { testMissingHeaderAttack(t, 64, LightSync) }

func testMissingHeaderAttack(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Attempt a full sync with an attacker feeding gapped headers
tester.newPeer("attack", protocol, hashes, headers, blocks, receipts)
missing := targetBlocks / 2
delete(tester.peerHeaders["attack"], hashes[missing])

if err := tester.sync("attack", nil, mode); err == nil {
t.Fatalf("succeeded attacker synchronisation")
}
// Synchronise with the valid peer and make sure sync succeeds
tester.newPeer("valid", protocol, hashes, headers, blocks, receipts)
if err := tester.sync("valid", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)
}

// Tests that if requested headers are shifted (i.e. first is missing), the queue
// detects the invalid numbering.
func TestShiftedHeaderAttack62(t *testing.T)      { testShiftedHeaderAttack(t, 62, FullSync) }
func TestShiftedHeaderAttack63Full(t *testing.T)  { testShiftedHeaderAttack(t, 63, FullSync) }
func TestShiftedHeaderAttack63Fast(t *testing.T)  { testShiftedHeaderAttack(t, 63, FastSync) }
```

```
func TestShiftedHeaderAttack64Full(t *testing.T)  { testShiftedHeaderAttack(t, 64, FullSync) }
func TestShiftedHeaderAttack64Fast(t *testing.T)  { testShiftedHeaderAttack(t, 64, FastSync) }
func TestShiftedHeaderAttack64Light(t *testing.T) { testShiftedHeaderAttack(t, 64, LightSync) }

func testShiftedHeaderAttack(t *testing.T, protocol int, mode SyncMode) {
tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Attempt a full sync with an attacker feeding shifted headers
tester.newPeer("attack", protocol, hashes, headers, blocks, receipts)
delete(tester.peerHeaders["attack"], hashes[len(hashes)-2])
delete(tester.peerBlocks["attack"], hashes[len(hashes)-2])
delete(tester.peerReceipts["attack"], hashes[len(hashes)-2])

if err := tester.sync("attack", nil, mode); err == nil {
t.Fatalf("succeeded attacker synchronisation")
}
// Synchronise with the valid peer and make sure sync succeeds
tester.newPeer("valid", protocol, hashes, headers, blocks, receipts)
if err := tester.sync("valid", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
assertOwnChain(t, tester, targetBlocks+1)
}

// Tests that upon detecting an invalid header, the recent ones are rolled back
// for various failure scenarios. Afterwards a full sync is attempted to make
// sure no state was corrupted.
func TestInvalidHeaderRollback63Fast(t *testing.T)  { testInvalidHeaderRollback(t, 63, FastSync) }
func TestInvalidHeaderRollback64Fast(t *testing.T)  { testInvalidHeaderRollback(t, 64, FastSync) }
func TestInvalidHeaderRollback64Light(t *testing.T) { testInvalidHeaderRollback(t, 64, LightSync) }

func testInvalidHeaderRollback(t *testing.T, protocol int, mode SyncMode) {
tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := 3*fsHeaderSafetyNet + fsPivotInterval + fsMinFullBlocks
```

```
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Attempt to sync with an attacker that feeds junk during the fast sync phase.
// This should result in the last fsHeaderSafetyNet headers being rolled back.
tester.newPeer("fast-attack", protocol, hashes, headers, blocks, receipts)
missing := fsHeaderSafetyNet + MaxHeaderFetch + 1
delete(tester.peerHeaders["fast-attack"], hashes[len(hashes)-missing])

if err := tester.sync("fast-attack", nil, mode); err == nil {
t.Fatalf("succeeded fast attacker synchronisation")
}
if head := tester.headHeader().Number.Int64(); int(head) > MaxHeaderFetch {
t.Errorf("rollback head mismatch: have %v, want at most %v", head, MaxHeaderFetch)
}
// Attempt to sync with an attacker that feeds junk during the block import phase.
// This should result in both the last fsHeaderSafetyNet number of headers being
// rolled back, and also the pivot point being reverted to a non-block status.
tester.newPeer("block-attack", protocol, hashes, headers, blocks, receipts)
missing = 3*fsHeaderSafetyNet + MaxHeaderFetch + 1
delete(tester.peerHeaders["fast-attack"], hashes[len(hashes)-missing]) // Make sure the fast-attacker doesn't fill in
delete(tester.peerHeaders["block-attack"], hashes[len(hashes)-missing])

if err := tester.sync("block-attack", nil, mode); err == nil {
t.Fatalf("succeeded block attacker synchronisation")
}
if head := tester.headHeader().Number.Int64(); int(head) >
2*fsHeaderSafetyNet+MaxHeaderFetch {
t.Errorf("rollback head mismatch: have %v, want at most %v", head,
2*fsHeaderSafetyNet+MaxHeaderFetch)
}
if mode == FastSync {
if head := tester.headBlock().NumberU64(); head != 0 {
t.Errorf("fast sync pivot block #%d not rolled back", head)
}
}
// Attempt to sync with an attacker that withholds promised blocks after the
// fast sync pivot point. This could be a trial to leave the node with a bad
// but already imported pivot block.
tester.newPeer("withhold-attack", protocol, hashes, headers, blocks, receipts)
missing = 3*fsHeaderSafetyNet + MaxHeaderFetch + 1
```

```
tester.downloader.fsPivotFails = 0
tester.downloader.syncInitHook = func(uint64, uint64) {
for i := missing; i <= len(hashes); i++ {
delete(tester.peerHeaders["withhold-attack"], hashes[len(hashes)-i])
}
tester.downloader.syncInitHook = nil
}

if err := tester.sync("withhold-attack", nil, mode); err == nil {
t.Fatalf("succeeded withholding attacker synchronisation")
}
if head := tester.headHeader().Number.Int64(); int(head) >
2*fsHeaderSafetyNet+MaxHeaderFetch {
t.Errorf("rollback head mismatch: have %v, want at most %v", head,
2*fsHeaderSafetyNet+MaxHeaderFetch)
}
if mode == FastSync {
if head := tester.headBlock().NumberU64(); head != 0 {
t.Errorf("fast sync pivot block #%d not rolled back", head)
}
}
tester.downloader.fsPivotFails = fsCriticalTrials

// Synchronise with the valid peer and make sure sync succeeds. Since the last
// rollback should also disable fast syncing for this process, verify that we
// did a fresh full sync. Note, we can't assert anything about the receipts
// since we won't purge the database of them, hence we can't use assertOwnChain.
tester.newPeer("valid", protocol, hashes, headers, blocks, receipts)
if err := tester.sync("valid", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
if hs := len(tester.ownHeaders); hs != len(headers) {
t.Fatalf("synchronised headers mismatch: have %v, want %v", hs, len(headers))
}
if mode != LightSync {
if bs := len(tester.ownBlocks); bs != len(blocks) {
t.Fatalf("synchronised blocks mismatch: have %v, want %v", bs, len(blocks))
}
}
}

// Tests that a peer advertising an high TD doesn't get to stall the downloader
```

```
// afterwards by not sending any useful hashes.
func TestHighTDStarvationAttack62(t *testing.T)      { testHighTDStarvationAttack(t, 62, FullSync) }
func TestHighTDStarvationAttack63Full(t *testing.T)  { testHighTDStarvationAttack(t, 63, FullSync)
}
func TestHighTDStarvationAttack63Fast(t *testing.T)  { testHighTDStarvationAttack(t, 63,
FastSync) }
func TestHighTDStarvationAttack64Full(t *testing.T)  { testHighTDStarvationAttack(t, 64, FullSync)
}
func TestHighTDStarvationAttack64Fast(t *testing.T)  { testHighTDStarvationAttack(t, 64,
FastSync) }
func TestHighTDStarvationAttack64Light(t *testing.T) { testHighTDStarvationAttack(t, 64,
LightSync) }

func testHighTDStarvationAttack(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

hashes, headers, blocks, receipts := tester.makeChain(0, 0, tester.genesis, nil, false)
tester.newPeer("attack", protocol, []common.Hash{hashes[0]}, headers, blocks, receipts)

if err := tester.sync("attack", big.NewInt(1000000), mode); err != errStallingPeer {
t.Fatalf("synchronisation error mismatch: have %v, want %v", err, errStallingPeer)
}
}

// Tests that misbehaving peers are disconnected, whilst behaving ones are not.
func TestBlockHeaderAttackerDropping62(t *testing.T) { testBlockHeaderAttackerDropping(t, 62) }
func TestBlockHeaderAttackerDropping63(t *testing.T) { testBlockHeaderAttackerDropping(t, 63) }
func TestBlockHeaderAttackerDropping64(t *testing.T) { testBlockHeaderAttackerDropping(t, 64) }

func testBlockHeaderAttackerDropping(t *testing.T, protocol int) {
// Define the disconnection requirement for individual hash fetch errors
tests := []struct {
result error
drop   bool
}{
{nil, false},                    // Sync succeeded, all is well
{errBusy, false},                // Sync is already in progress, no problem
{errUnknownPeer, false},         // Peer is unknown, was already dropped, don't double drop
{errBadPeer, true},              // Peer was deemed bad for some reason, drop it
```

```
{errStallingPeer, true},            // Peer was detected to be stalling, drop it
{errNoPeers, false},                // No peers to download from, soft race, no issue
{errTimeout, true},                 // No hashes received in due time, drop the peer
{errEmptyHeaderSet, true},          // No headers were returned as a response, drop as it's a dead end
{errPeersUnavailable, true},        // Nobody had the advertised blocks, drop the advertiser
{errInvalidAncestor, true},         // Agreed upon ancestor is not acceptable, drop the chain rewriter
{errInvalidChain, true},            // Hash chain was detected as invalid, definitely drop
{errInvalidBlock, false},           // A bad peer was detected, but not the sync origin
{errInvalidBody, false},            // A bad peer was detected, but not the sync origin
{errInvalidReceipt, false},         // A bad peer was detected, but not the sync origin
{errCancelBlockFetch, false},       // Synchronisation was canceled, origin may be innocent, don't drop
{errCancelHeaderFetch, false},      // Synchronisation was canceled, origin may be innocent, don't drop
{errCancelBodyFetch, false},        // Synchronisation was canceled, origin may be innocent, don't drop
{errCancelReceiptFetch, false},     // Synchronisation was canceled, origin may be innocent, don't drop
{errCancelHeaderProcessing, false},  // Synchronisation was canceled, origin may be innocent, don't drop
{errCancelContentProcessing, false}, // Synchronisation was canceled, origin may be innocent, don't drop
}
// Run the tests and check disconnection status
tester := newTester()
defer tester.terminate()

for i, tt := range tests {
// Register a new peer and ensure it's presence
id := fmt.Sprintf("test %d", i)
if err := tester.newPeer(id, protocol, []common.Hash{tester.genesis.Hash()}, nil, nil, nil); err != nil {
t.Fatalf("test %d: failed to register new peer: %v", i, err)
}
if _, ok := tester.peerHashes[id]; !ok {
t.Fatalf("test %d: registered peer not found", i)
}
// Simulate a synchronisation and check the required result
tester.downloader.synchroniseMock = func(string, common.Hash) error { return tt.result }

tester.downloader.Synchronise(id, tester.genesis.Hash(), big.NewInt(1000), FullSync)
if _, ok := tester.peerHashes[id]; !ok != tt.drop {
```

```
t.Errorf("test %d: peer drop mismatch for %v: have %v, want %v", i, tt.result, !ok, tt.drop)
}
}
}

// Tests that synchronisation progress (origin block number, current block number
// and highest block number) is tracked and updated correctly.
func TestSyncProgress62(t *testing.T)      { testSyncProgress(t, 62, FullSync) }
func TestSyncProgress63Full(t *testing.T)  { testSyncProgress(t, 63, FullSync) }
func TestSyncProgress63Fast(t *testing.T)  { testSyncProgress(t, 63, FastSync) }
func TestSyncProgress64Full(t *testing.T)  { testSyncProgress(t, 64, FullSync) }
func TestSyncProgress64Fast(t *testing.T)  { testSyncProgress(t, 64, FastSync) }
func TestSyncProgress64Light(t *testing.T) { testSyncProgress(t, 64, LightSync) }

func testSyncProgress(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Set a sync init hook to catch progress changes
starting := make(chan struct{})
progress := make(chan struct{})

tester.downloader.syncInitHook = func(origin, latest uint64) {
starting <- struct{}{}
<-progress
}
// Retrieve the sync progress and ensure they are zero (pristine sync)
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != 0 {
t.Fatalf("Pristine progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, 0)
}
// Synchronise half the blocks and check initial progress
tester.newPeer("peer-half", protocol, hashes[targetBlocks/2:], headers, blocks, receipts)
pending := new(sync.WaitGroup)
pending.Add(1)
```

```
go func() {
defer pending.Done()
if err := tester.sync("peer-half", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != uint64(targetBlocks/2+1) {
t.Fatalf("Initial progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, targetBlocks/2+1)
}
progress <- struct{}{}
pending.Wait()

// Synchronise all the blocks and check continuation progress
tester.newPeer("peer-full", protocol, hashes, headers, blocks, receipts)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("peer-full", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != uint64(targetBlocks/2+1) ||
progress.CurrentBlock != uint64(targetBlocks/2+1) || progress.HighestBlock !=
uint64(targetBlocks) {
t.Fatalf("Completing progress mismatch: have %v/%v/%v, want %v/%v/%v",
progress.StartingBlock, progress.CurrentBlock, progress.HighestBlock, targetBlocks/2+1,
targetBlocks/2+1, targetBlocks)
}
progress <- struct{}{}
pending.Wait()

// Check final progress after successful sync
if progress := tester.downloader.Progress(); progress.StartingBlock != uint64(targetBlocks/2+1) ||
progress.CurrentBlock != uint64(targetBlocks) || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Final progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, targetBlocks/2+1, targetBlocks, targetBlocks)
```

```
}
}

// Tests that synchronisation progress (origin block number and highest block
// number) is tracked and updated correctly in case of a fork (or manual head
// revertal).
func TestForkedSyncProgress62(t *testing.T)      { testForkedSyncProgress(t, 62, FullSync) }
func TestForkedSyncProgress63Full(t *testing.T)  { testForkedSyncProgress(t, 63, FullSync) }
func TestForkedSyncProgress63Fast(t *testing.T)  { testForkedSyncProgress(t, 63, FastSync) }
func TestForkedSyncProgress64Full(t *testing.T)  { testForkedSyncProgress(t, 64, FullSync) }
func TestForkedSyncProgress64Fast(t *testing.T)  { testForkedSyncProgress(t, 64, FastSync) }
func TestForkedSyncProgress64Light(t *testing.T) { testForkedSyncProgress(t, 64, LightSync) }

func testForkedSyncProgress(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a forked chain to simulate origin revertal
common, fork := MaxHashFetch, 2*MaxHashFetch
hashesA, hashesB, headersA, headersB, blocksA, blocksB, receiptsA, receiptsB :=
tester.makeChainFork(common+fork, fork, tester.genesis, nil, true)

// Set a sync init hook to catch progress changes
starting := make(chan struct{})
progress := make(chan struct{})

tester.downloader.syncInitHook = func(origin, latest uint64) {
starting <- struct{}{}
<-progress
}
// Retrieve the sync progress and ensure they are zero (pristine sync)
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != 0 {
t.Fatalf("Pristine progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, 0)
}
// Synchronise with one of the forks and check progress
tester.newPeer("fork A", protocol, hashesA, headersA, blocksA, receiptsA)
pending := new(sync.WaitGroup)
pending.Add(1)
```

```
go func() {
defer pending.Done()
if err := tester.sync("fork A", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != uint64(len(hashesA)-1) {
t.Fatalf("Initial progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, len(hashesA)-1)
}
progress <- struct{}{}
pending.Wait()

// Simulate a successful sync above the fork
tester.downloader.syncStatsChainOrigin = tester.downloader.syncStatsChainHeight

// Synchronise with the second fork and check progress resets
tester.newPeer("fork B", protocol, hashesB, headersB, blocksB, receiptsB)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("fork B", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != uint64(common) ||
progress.CurrentBlock != uint64(len(hashesA)-1) || progress.HighestBlock != uint64(len(hashesB)-
1) {
t.Fatalf("Forking progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, common, len(hashesA)-1, len(hashesB)-1)
}
progress <- struct{}{}
pending.Wait()

// Check final progress after successful sync
if progress := tester.downloader.Progress(); progress.StartingBlock != uint64(common) ||
progress.CurrentBlock != uint64(len(hashesB)-1) || progress.HighestBlock != uint64(len(hashesB)-
```

```
1) {
t.Fatalf("Final progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, common, len(hashesB)-1, len(hashesB)-1)
}
}

// Tests that if synchronisation is aborted due to some failure, then the progress
// origin is not updated in the next sync cycle, as it should be considered the
// continuation of the previous sync and not a new instance.
func TestFailedSyncProgress62(t *testing.T)      { testFailedSyncProgress(t, 62, FullSync) }
func TestFailedSyncProgress63Full(t *testing.T)  { testFailedSyncProgress(t, 63, FullSync) }
func TestFailedSyncProgress63Fast(t *testing.T)  { testFailedSyncProgress(t, 63, FastSync) }
func TestFailedSyncProgress64Full(t *testing.T)  { testFailedSyncProgress(t, 64, FullSync) }
func TestFailedSyncProgress64Fast(t *testing.T)  { testFailedSyncProgress(t, 64, FastSync) }
func TestFailedSyncProgress64Light(t *testing.T) { testFailedSyncProgress(t, 64, LightSync) }

func testFailedSyncProgress(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small enough block chain to download
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Set a sync init hook to catch progress changes
starting := make(chan struct{})
progress := make(chan struct{})

tester.downloader.syncInitHook = func(origin, latest uint64) {
starting <- struct{}{}
<-progress
}
// Retrieve the sync progress and ensure they are zero (pristine sync)
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != 0 {
t.Fatalf("Pristine progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, 0)
}
// Attempt a full sync with a faulty peer
tester.newPeer("faulty", protocol, hashes, headers, blocks, receipts)
```

```
missing := targetBlocks / 2
delete(tester.peerHeaders["faulty"], hashes[missing])
delete(tester.peerBlocks["faulty"], hashes[missing])
delete(tester.peerReceipts["faulty"], hashes[missing])

pending := new(sync.WaitGroup)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("faulty", nil, mode); err == nil {
t.Fatalf("succeeded faulty synchronisation")
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Initial progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, targetBlocks)
}
progress <- struct{}{}
pending.Wait()

// Synchronise with a good peer and check that the progress origin remind the same after a failure
tester.newPeer("valid", protocol, hashes, headers, blocks, receipts)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("valid", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock >
uint64(targetBlocks/2) || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Completing progress mismatch: have %v/%v/%v, want %v/0-%v/%v",
progress.StartingBlock, progress.CurrentBlock, progress.HighestBlock, 0, targetBlocks/2,
targetBlocks)
}
progress <- struct{}{}
pending.Wait()
```

```
// Check final progress after successful sync
if progress := tester.downloader.Progress(); progress.StartingBlock > uint64(targetBlocks/2) ||
progress.CurrentBlock != uint64(targetBlocks) || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Final progress mismatch: have %v/%v/%v, want 0-%v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, targetBlocks/2, targetBlocks, targetBlocks)
}
}

// Tests that if an attacker fakes a chain height, after the attack is detected,
// the progress height is successfully reduced at the next sync invocation.
func TestFakedSyncProgress62(t *testing.T)      { testFakedSyncProgress(t, 62, FullSync) }
func TestFakedSyncProgress63Full(t *testing.T)  { testFakedSyncProgress(t, 63, FullSync) }
func TestFakedSyncProgress63Fast(t *testing.T)  { testFakedSyncProgress(t, 63, FastSync) }
func TestFakedSyncProgress64Full(t *testing.T)  { testFakedSyncProgress(t, 64, FullSync) }
func TestFakedSyncProgress64Fast(t *testing.T)  { testFakedSyncProgress(t, 64, FastSync) }
func TestFakedSyncProgress64Light(t *testing.T) { testFakedSyncProgress(t, 64, LightSync) }

func testFakedSyncProgress(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

tester := newTester()
defer tester.terminate()

// Create a small block chain
targetBlocks := blockCacheLimit - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks+3, 0, tester.genesis, nil, false)

// Set a sync init hook to catch progress changes
starting := make(chan struct{})
progress := make(chan struct{})

tester.downloader.syncInitHook = func(origin, latest uint64) {
starting <- struct{}{}
<-progress
}
// Retrieve the sync progress and ensure they are zero (pristine sync)
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != 0 {
t.Fatalf("Pristine progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, 0)
}
```

```
//  Create and sync with an attacker that promises a higher chain than available
tester.newPeer("attack", protocol, hashes, headers, blocks, receipts)
for i := 1; i < 3; i++ {
delete(tester.peerHeaders["attack"], hashes[i])
delete(tester.peerBlocks["attack"], hashes[i])
delete(tester.peerReceipts["attack"], hashes[i])
}

pending := new(sync.WaitGroup)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("attack", nil, mode); err == nil {
t.Fatalf("succeeded attacker synchronisation")
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock
!= 0 || progress.HighestBlock != uint64(targetBlocks+3) {
t.Fatalf("Initial progress mismatch: have %v/%v/%v, want %v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, 0, 0, targetBlocks+3)
}
progress <- struct{}{}
pending.Wait()

// Synchronise with a good peer and check that the progress height has been reduced to the true
value
tester.newPeer("valid", protocol, hashes[3:], headers, blocks, receipts)
pending.Add(1)

go func() {
defer pending.Done()
if err := tester.sync("valid", nil, mode); err != nil {
t.Fatalf("failed to synchronise blocks: %v", err)
}
}()
<-starting
if progress := tester.downloader.Progress(); progress.StartingBlock != 0 || progress.CurrentBlock >
uint64(targetBlocks) || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Completing progress mismatch: have %v/%v/%v, want %v/0-%v/%v",
progress.StartingBlock, progress.CurrentBlock, progress.HighestBlock, 0, targetBlocks,
```

```
targetBlocks)
}
progress <- struct{}{}
pending.Wait()

// Check final progress after successful sync
if progress := tester.downloader.Progress(); progress.StartingBlock > uint64(targetBlocks) ||
progress.CurrentBlock != uint64(targetBlocks) || progress.HighestBlock != uint64(targetBlocks) {
t.Fatalf("Final progress mismatch: have %v/%v/%v, want 0-%v/%v/%v", progress.StartingBlock,
progress.CurrentBlock, progress.HighestBlock, targetBlocks, targetBlocks, targetBlocks)
}
}

// This test reproduces an issue where unexpected deliveries would
// block indefinitely if they arrived at the right time.
func TestDeliverHeadersHang62(t *testing.T)      { testDeliverHeadersHang(t, 62, FullSync) }
func TestDeliverHeadersHang63Full(t *testing.T)  { testDeliverHeadersHang(t, 63, FullSync) }
func TestDeliverHeadersHang63Fast(t *testing.T)  { testDeliverHeadersHang(t, 63, FastSync) }
func TestDeliverHeadersHang64Full(t *testing.T)  { testDeliverHeadersHang(t, 64, FullSync) }
func TestDeliverHeadersHang64Fast(t *testing.T)  { testDeliverHeadersHang(t, 64, FastSync) }
func TestDeliverHeadersHang64Light(t *testing.T) { testDeliverHeadersHang(t, 64, LightSync) }

func testDeliverHeadersHang(t *testing.T, protocol int, mode SyncMode) {
t.Parallel()

master := newTester()
defer master.terminate()

hashes, headers, blocks, receipts := master.makeChain(5, 0, master.genesis, nil, false)
fakeHeads := []*types.Header{{}, {}, {}, {}}
for i := 0; i < 200; i++ {
tester := newTester()
tester.peerDb = master.peerDb

tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)
// Whenever the downloader requests headers, flood it with
// a lot of unrequested header deliveries.
tester.downloader.peers.peers["peer"].getAbsHeaders = func(from uint64, count, skip int, reverse
bool) error {
deliveriesDone := make(chan struct{}, 500)
for i := 0; i < cap(deliveriesDone); i++ {
peer := fmt.Sprintf("fake-peer%d", i)
```

```
go func() {
tester.downloader.DeliverHeaders(peer, fakeHeads)
deliveriesDone <- struct{}{}
}()
}
// Deliver the actual requested headers.
impl := tester.peerGetAbsHeadersFn("peer", 0)
go impl(from, count, skip, reverse)
// None of the extra deliveries should block.
timeout := time.After(15 * time.Second)
for i := 0; i < cap(deliveriesDone); i++ {
select {
case <-deliveriesDone:
case <-timeout:
panic("blocked")
}
}
return nil
}
if err := tester.sync("peer", nil, mode); err != nil {
t.Errorf("sync failed: %v", err)
}
tester.terminate()
}
}

// Tests that if fast sync aborts in the critical section, it can restart a few
// times before giving up.
func TestFastCriticalRestartsFail63(t *testing.T) { testFastCriticalRestarts(t, 63, false) }
func TestFastCriticalRestartsFail64(t *testing.T) { testFastCriticalRestarts(t, 64, false) }
func TestFastCriticalRestartsCont63(t *testing.T) { testFastCriticalRestarts(t, 63, true) }
func TestFastCriticalRestartsCont64(t *testing.T) { testFastCriticalRestarts(t, 64, true) }

func testFastCriticalRestarts(t *testing.T, protocol int, progress bool) {
tester := newTester()
defer tester.terminate()

// Create a large enough blockchin to actually fast sync on
targetBlocks := fsMinFullBlocks + 2*fsPivotInterval - 15
hashes, headers, blocks, receipts := tester.makeChain(targetBlocks, 0, tester.genesis, nil, false)

// Create a tester peer with a critical section header missing (force failures)
```

```
tester.newPeer("peer", protocol, hashes, headers, blocks, receipts)
delete(tester.peerHeaders["peer"], hashes[fsMinFullBlocks-1])
tester.downloader.dropPeer = func(id string) {} // We reuse the same "faulty" peer throughout the test

// Remove all possible pivot state roots and slow down replies (test failure resets later)
for i := 0; i < fsPivotInterval; i++ {
tester.peerMissingStates["peer"][headers[hashes[fsMinFullBlocks+i]].Root] = true
}
tester.downloader.peers.peers["peer"].getNodeData = tester.peerGetNodeDataFn("peer", 500*time.Millisecond) // Enough to reach the critical section

// Synchronise with the peer a few times and make sure they fail until the retry limit
for i := 0; i < int(fsCriticalTrials)-1; i++ {
// Attempt a sync and ensure it fails properly
if err := tester.sync("peer", nil, FastSync); err == nil {
t.Fatalf("failing fast sync succeeded: %v", err)
}
time.Sleep(150 * time.Millisecond) // Make sure no in-flight requests remain

// If it's the first failure, pivot should be locked => reenable all others to detect pivot changes
if i == 0 {
if tester.downloader.fsPivotLock == nil {
time.Sleep(400 * time.Millisecond) // Make sure the first huge timeout expires too
t.Fatalf("pivot block not locked in after critical section failure")
}
tester.lock.Lock()
tester.peerHeaders["peer"][hashes[fsMinFullBlocks-1]] = headers[hashes[fsMinFullBlocks-1]]
tester.peerMissingStates["peer"] = map[common.Hash]bool{tester.downloader.fsPivotLock.Root: true}
tester.downloader.peers.peers["peer"].getNodeData = tester.peerGetNodeDataFn("peer", 0)
tester.lock.Unlock()
}
}
// Return all nodes if we're testing fast sync progression
if progress {
tester.lock.Lock()
tester.peerMissingStates["peer"] = map[common.Hash]bool{}
tester.lock.Unlock()

if err := tester.sync("peer", nil, FastSync); err != nil {
t.Fatalf("failed to synchronise blocks in progressed fast sync: %v", err)
```

```
}
time.Sleep(150 * time.Millisecond) // Make sure no in-flight requests remain

if fails := atomic.LoadUint32(&tester.downloader.fsPivotFails); fails != 1 {
t.Fatalf("progressed pivot trial count mismatch: have %v, want %v", fails, 1)
}
assertOwnChain(t, tester, targetBlocks+1)
} else {
if err := tester.sync("peer", nil, FastSync); err == nil {
t.Fatalf("succeeded to synchronise blocks in failed fast sync")
}
time.Sleep(150 * time.Millisecond) // Make sure no in-flight requests remain

if fails := atomic.LoadUint32(&tester.downloader.fsPivotFails); fails != fsCriticalTrials {
t.Fatalf("failed pivot trial count mismatch: have %v, want %v", fails, fsCriticalTrials)
}
}
// Retry limit exhausted, downloader will switch to full sync, should succeed
if err := tester.sync("peer", nil, FastSync); err != nil {
t.Fatalf("failed to synchronise blocks in slow sync: %v", err)
}
// Note, we can't assert the chain here because the test asserter assumes sync
// completed using a single mode of operation, whereas fast-then-slow can result
// in arbitrary intermediate state that's not cleanly verifiable.
}

1:F:\git\coin\ethereum\go-ethereum\eth\downloader\events.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package downloader

type DoneEvent struct{}
type StartEvent struct{}
type FailedEvent struct{ Err error }

2:F:\git\coin\ethereum\go-ethereum\eth\downloader\metrics.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Contains the metrics collected by the downloader.

package downloader
```

```
import (
"github.com/ethereum/go-ethereum/metrics"
)

var (
headerInMeter      = metrics.NewMeter("eth/downloader/headers/in")
headerReqTimer     = metrics.NewTimer("eth/downloader/headers/req")
headerDropMeter    = metrics.NewMeter("eth/downloader/headers/drop")
headerTimeoutMeter = metrics.NewMeter("eth/downloader/headers/timeout")

bodyInMeter      = metrics.NewMeter("eth/downloader/bodies/in")
bodyReqTimer     = metrics.NewTimer("eth/downloader/bodies/req")
bodyDropMeter    = metrics.NewMeter("eth/downloader/bodies/drop")
bodyTimeoutMeter = metrics.NewMeter("eth/downloader/bodies/timeout")

receiptInMeter      = metrics.NewMeter("eth/downloader/receipts/in")
receiptReqTimer     = metrics.NewTimer("eth/downloader/receipts/req")
receiptDropMeter    = metrics.NewMeter("eth/downloader/receipts/drop")
receiptTimeoutMeter = metrics.NewMeter("eth/downloader/receipts/timeout")

stateInMeter   = metrics.NewMeter("eth/downloader/states/in")
stateDropMeter = metrics.NewMeter("eth/downloader/states/drop")
)

3:F:\git\coin\ethereum\go-ethereum\eth\downloader\modes.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package downloader

import "fmt"

// SyncMode represents the synchronisation mode of the downloader.
type SyncMode int

const (
FullSync  SyncMode = iota // Synchronise the entire blockchain history from full blocks
FastSync                  // Quickly download the headers, full sync only at the chain head
LightSync                 // Download only the headers and terminate afterwards
)

func (mode SyncMode) IsValid() bool {
return mode >= FullSync && mode <= LightSync
```

```
}

// String implements the stringer interface.
func (mode SyncMode) String() string {
switch mode {
case FullSync:
return "full"
case FastSync:
return "fast"
case LightSync:
return "light"
default:
return "unknown"
}
}

func (mode SyncMode) MarshalText() ([]byte, error) {
switch mode {
case FullSync:
return []byte("full"), nil
case FastSync:
return []byte("fast"), nil
case LightSync:
return []byte("light"), nil
default:
return nil, fmt.Errorf("unknown sync mode %d", mode)
}
}

func (mode *SyncMode) UnmarshalText(text []byte) error {
switch string(text) {
case "full":
*mode = FullSync
case "fast":
*mode = FastSync
case "light":
*mode = LightSync
default:
return fmt.Errorf(`unknown sync mode %q, want "full", "fast" or "light"`, text)
}
return nil
}
```

4:F:\git\coin\ethereum\go-ethereum\eth\downloader\peer.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Contains the active peer-set of the downloader, maintaining both failures
// as well as reputation metrics to prioritize the block retrievals.

package downloader

import (
"errors"
"fmt"
"math"
"math/big"
"sort"
"sync"
"sync/atomic"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/log"
)

const (
maxLackingHashes = 4096 // Maximum number of entries allowed on the list or lacking items
measurementImpact = 0.1 // The impact a single measurement has on a peer's final throughput
value.
)

// Head hash and total difficulty retriever for
type currentHeadRetrievalFn func() (common.Hash, *big.Int)

// Block header and body fetchers belonging to eth/62 and above
type relativeHeaderFetcherFn func(common.Hash, int, int, bool) error
type absoluteHeaderFetcherFn func(uint64, int, int, bool) error
type blockBodyFetcherFn func([]common.Hash) error
type receiptFetcherFn func([]common.Hash) error
type stateFetcherFn func([]common.Hash) error

var (
errAlreadyFetching   = errors.New("already fetching blocks from peer")

```
errAlreadyRegistered = errors.New("peer is already registered")
errNotRegistered     = errors.New("peer is not registered")
)

// peer represents an active peer from which hashes and blocks are retrieved.
type peer struct {
id string // Unique identifier of the peer

headerIdle  int32 // Current header activity state of the peer (idle = 0, active = 1)
blockIdle   int32 // Current block activity state of the peer (idle = 0, active = 1)
receiptIdle int32 // Current receipt activity state of the peer (idle = 0, active = 1)
stateIdle   int32 // Current node data activity state of the peer (idle = 0, active = 1)

headerThroughput  float64 // Number of headers measured to be retrievable per second
blockThroughput   float64 // Number of blocks (bodies) measured to be retrievable per second
receiptThroughput float64 // Number of receipts measured to be retrievable per second
stateThroughput   float64 // Number of node data pieces measured to be retrievable per second

rtt time.Duration // Request round trip time to track responsiveness (QoS)

headerStarted  time.Time // Time instance when the last header fetch was started
blockStarted   time.Time // Time instance when the last block (body) fetch was started
receiptStarted time.Time // Time instance when the last receipt fetch was started
stateStarted   time.Time // Time instance when the last node data fetch was started

lacking map[common.Hash]struct{} // Set of hashes not to request (didn't have previously)

currentHead currentHeadRetrievalFn // Method to fetch the currently known head of the peer

getRelHeaders  relativeHeaderFetcherFn // [eth/62] Method to retrieve a batch of headers from an
origin hash
getAbsHeaders  absoluteHeaderFetcherFn // [eth/62] Method to retrieve a batch of headers from
an absolute position
getBlockBodies blockBodyFetcherFn      // [eth/62] Method to retrieve a batch of block bodies

getReceipts receiptFetcherFn // [eth/63] Method to retrieve a batch of block transaction receipts
getNodeData stateFetcherFn   // [eth/63] Method to retrieve a batch of state trie data

version int        // Eth protocol version number to switch strategies
log     log.Logger // Contextual logger to add extra infos to peer logs
lock    sync.RWMutex
}
```

```
// newPeer create a new downloader peer, with specific hash and block retrieval
// mechanisms.
func newPeer(id string, version int, currentHead currentHeadRetrievalFn,
getRelHeaders relativeHeaderFetcherFn, getAbsHeaders absoluteHeaderFetcherFn,
getBlockBodies blockBodyFetcherFn,
getReceipts receiptFetcherFn, getNodeData stateFetcherFn, logger log.Logger) *peer {

return &peer{
id:      id,
lacking: make(map[common.Hash]struct{}),

currentHead:    currentHead,
getRelHeaders:  getRelHeaders,
getAbsHeaders:  getAbsHeaders,
getBlockBodies: getBlockBodies,

getReceipts: getReceipts,
getNodeData: getNodeData,

version: version,
log:     logger,
}
}

// Reset clears the internal state of a peer entity.
func (p *peer) Reset() {
p.lock.Lock()
defer p.lock.Unlock()

atomic.StoreInt32(&p.headerIdle, 0)
atomic.StoreInt32(&p.blockIdle, 0)
atomic.StoreInt32(&p.receiptIdle, 0)
atomic.StoreInt32(&p.stateIdle, 0)

p.headerThroughput = 0
p.blockThroughput = 0
p.receiptThroughput = 0
p.stateThroughput = 0

p.lacking = make(map[common.Hash]struct{})
}
```

```
// FetchHeaders sends a header retrieval request to the remote peer.
func (p *peer) FetchHeaders(from uint64, count int) error {
// Sanity check the protocol version
if p.version < 62 {
panic(fmt.Sprintf("header fetch [eth/62+] requested on eth/%d", p.version))
}
// Short circuit if the peer is already fetching
if !atomic.CompareAndSwapInt32(&p.headerIdle, 0, 1) {
return errAlreadyFetching
}
p.headerStarted = time.Now()

// Issue the header retrieval request (absolut upwards without gaps)
go p.getAbsHeaders(from, count, 0, false)

return nil
}

// FetchBodies sends a block body retrieval request to the remote peer.
func (p *peer) FetchBodies(request *fetchRequest) error {
// Sanity check the protocol version
if p.version < 62 {
panic(fmt.Sprintf("body fetch [eth/62+] requested on eth/%d", p.version))
}
// Short circuit if the peer is already fetching
if !atomic.CompareAndSwapInt32(&p.blockIdle, 0, 1) {
return errAlreadyFetching
}
p.blockStarted = time.Now()

// Convert the header set to a retrievable slice
hashes := make([]common.Hash, 0, len(request.Headers))
for _, header := range request.Headers {
hashes = append(hashes, header.Hash())
}
go p.getBlockBodies(hashes)

return nil
}

// FetchReceipts sends a receipt retrieval request to the remote peer.
```

```
func (p *peer) FetchReceipts(request *fetchRequest) error {
// Sanity check the protocol version
if p.version < 63 {
panic(fmt.Sprintf("body fetch [eth/63+] requested on eth/%d", p.version))
}
// Short circuit if the peer is already fetching
if !atomic.CompareAndSwapInt32(&p.receiptIdle, 0, 1) {
return errAlreadyFetching
}
p.receiptStarted = time.Now()

// Convert the header set to a retrievable slice
hashes := make([]common.Hash, 0, len(request.Headers))
for _, header := range request.Headers {
hashes = append(hashes, header.Hash())
}
go p.getReceipts(hashes)

return nil
}

// FetchNodeData sends a node state data retrieval request to the remote peer.
func (p *peer) FetchNodeData(hashes []common.Hash) error {
// Sanity check the protocol version
if p.version < 63 {
panic(fmt.Sprintf("node data fetch [eth/63+] requested on eth/%d", p.version))
}
// Short circuit if the peer is already fetching
if !atomic.CompareAndSwapInt32(&p.stateIdle, 0, 1) {
return errAlreadyFetching
}
p.stateStarted = time.Now()
go p.getNodeData(hashes)
return nil
}

// SetHeadersIdle sets the peer to idle, allowing it to execute new header retrieval
// requests. Its estimated header retrieval throughput is updated with that measured
// just now.
func (p *peer) SetHeadersIdle(delivered int) {
p.setIdle(p.headerStarted, delivered, &p.headerThroughput, &p.headerIdle)
}
```

```
// SetBlocksIdle sets the peer to idle, allowing it to execute new block retrieval
// requests. Its estimated block retrieval throughput is updated with that measured
// just now.
func (p *peer) SetBlocksIdle(delivered int) {
p.setIdle(p.blockStarted, delivered, &p.blockThroughput, &p.blockIdle)
}

// SetBodiesIdle sets the peer to idle, allowing it to execute block body retrieval
// requests. Its estimated body retrieval throughput is updated with that measured
// just now.
func (p *peer) SetBodiesIdle(delivered int) {
p.setIdle(p.blockStarted, delivered, &p.blockThroughput, &p.blockIdle)
}

// SetReceiptsIdle sets the peer to idle, allowing it to execute new receipt
// retrieval requests. Its estimated receipt retrieval throughput is updated
// with that measured just now.
func (p *peer) SetReceiptsIdle(delivered int) {
p.setIdle(p.receiptStarted, delivered, &p.receiptThroughput, &p.receiptIdle)
}

// SetNodeDataIdle sets the peer to idle, allowing it to execute new state trie
// data retrieval requests. Its estimated state retrieval throughput is updated
// with that measured just now.
func (p *peer) SetNodeDataIdle(delivered int) {
p.setIdle(p.stateStarted, delivered, &p.stateThroughput, &p.stateIdle)
}

// setIdle sets the peer to idle, allowing it to execute new retrieval requests.
// Its estimated retrieval throughput is updated with that measured just now.
func (p *peer) setIdle(started time.Time, delivered int, throughput *float64, idle *int32) {
// Irrelevant of the scaling, make sure the peer ends up idle
defer atomic.StoreInt32(idle, 0)

p.lock.Lock()
defer p.lock.Unlock()

// If nothing was delivered (hard timeout / unavailable data), reduce throughput to minimum
if delivered == 0 {
*throughput = 0
return
```

```
}
// Otherwise update the throughput with a new measurement
elapsed := time.Since(started) + 1 // +1 (ns) to ensure non-zero divisor
measured := float64(delivered) / (float64(elapsed) / float64(time.Second))

*throughput = (1-measurementImpact)*(*throughput) + measurementImpact*measured
p.rtt = time.Duration((1-measurementImpact)*float64(p.rtt) +
measurementImpact*float64(elapsed))

p.log.Trace("Peer throughput measurements updated",
"hps", p.headerThroughput, "bps", p.blockThroughput,
"rps", p.receiptThroughput, "sps", p.stateThroughput,
"miss", len(p.lacking), "rtt", p.rtt)
}

// HeaderCapacity retrieves the peers header download allowance based on its
// previously discovered throughput.
func (p *peer) HeaderCapacity(targetRTT time.Duration) int {
p.lock.RLock()
defer p.lock.RUnlock()

return int(math.Min(1+math.Max(1, p.headerThroughput*float64(targetRTT)/float64(time.Second)),
float64(MaxHeaderFetch)))
}

// BlockCapacity retrieves the peers block download allowance based on its
// previously discovered throughput.
func (p *peer) BlockCapacity(targetRTT time.Duration) int {
p.lock.RLock()
defer p.lock.RUnlock()

return int(math.Min(1+math.Max(1, p.blockThroughput*float64(targetRTT)/float64(time.Second)),
float64(MaxBlockFetch)))
}

// ReceiptCapacity retrieves the peers receipt download allowance based on its
// previously discovered throughput.
func (p *peer) ReceiptCapacity(targetRTT time.Duration) int {
p.lock.RLock()
defer p.lock.RUnlock()

return int(math.Min(1+math.Max(1, p.receiptThroughput*float64(targetRTT)/float64(time.Second)),
```

```
float64(MaxReceiptFetch)))
}

// NodeDataCapacity retrieves the peers state download allowance based on its
// previously discovered throughput.
func (p *peer) NodeDataCapacity(targetRTT time.Duration) int {
p.lock.RLock()
defer p.lock.RUnlock()

return int(math.Min(1+math.Max(1, p.stateThroughput*float64(targetRTT)/float64(time.Second)),
float64(MaxStateFetch)))
}

// MarkLacking appends a new entity to the set of items (blocks, receipts, states)
// that a peer is known not to have (i.e. have been requested before). If the
// set reaches its maximum allowed capacity, items are randomly dropped off.
func (p *peer) MarkLacking(hash common.Hash) {
p.lock.Lock()
defer p.lock.Unlock()

for len(p.lacking) >= maxLackingHashes {
for drop := range p.lacking {
delete(p.lacking, drop)
break
}
}
p.lacking[hash] = struct{}{}
}

// Lacks retrieves whether the hash of a blockchain item is on the peers lacking
// list (i.e. whether we know that the peer does not have it).
func (p *peer) Lacks(hash common.Hash) bool {
p.lock.RLock()
defer p.lock.RUnlock()

_, ok := p.lacking[hash]
return ok
}

// peerSet represents the collection of active peer participating in the chain
// download procedure.
type peerSet struct {
```

```
peers       map[string]*peer
newPeerFeed event.Feed
lock        sync.RWMutex
}

// newPeerSet creates a new peer set top track the active download sources.
func newPeerSet() *peerSet {
return &peerSet{
peers: make(map[string]*peer),
}
}

func (ps *peerSet) SubscribeNewPeers(ch chan<- *peer) event.Subscription {
return ps.newPeerFeed.Subscribe(ch)
}

// Reset iterates over the current peer set, and resets each of the known peers
// to prepare for a next batch of block retrieval.
func (ps *peerSet) Reset() {
ps.lock.RLock()
defer ps.lock.RUnlock()

for _, peer := range ps.peers {
peer.Reset()
}
}

// Register injects a new peer into the working set, or returns an error if the
// peer is already known.
//
// The method also sets the starting throughput values of the new peer to the
// average of all existing peers, to give it a realistic chance of being used
// for data retrievals.
func (ps *peerSet) Register(p *peer) error {
// Retrieve the current median RTT as a sane default
p.rtt = ps.medianRTT()

// Register the new peer with some meaningful defaults
ps.lock.Lock()
if _, ok := ps.peers[p.id]; ok {
ps.lock.Unlock()
return errAlreadyRegistered
```

```
}
if len(ps.peers) > 0 {
p.headerThroughput, p.blockThroughput, p.receiptThroughput, p.stateThroughput = 0, 0, 0, 0

for _, peer := range ps.peers {
peer.lock.RLock()
p.headerThroughput += peer.headerThroughput
p.blockThroughput += peer.blockThroughput
p.receiptThroughput += peer.receiptThroughput
p.stateThroughput += peer.stateThroughput
peer.lock.RUnlock()
}
p.headerThroughput /= float64(len(ps.peers))
p.blockThroughput /= float64(len(ps.peers))
p.receiptThroughput /= float64(len(ps.peers))
p.stateThroughput /= float64(len(ps.peers))
}
ps.peers[p.id] = p
ps.lock.Unlock()

ps.newPeerFeed.Send(p)
return nil
}

// Unregister removes a remote peer from the active set, disabling any further
// actions to/from that particular entity.
func (ps *peerSet) Unregister(id string) error {
ps.lock.Lock()
defer ps.lock.Unlock()

if _, ok := ps.peers[id]; !ok {
return errNotRegistered
}
delete(ps.peers, id)
return nil
}

// Peer retrieves the registered peer with the given id.
func (ps *peerSet) Peer(id string) *peer {
ps.lock.RLock()
defer ps.lock.RUnlock()
```

```
return ps.peers[id]
}

// Len returns if the current number of peers in the set.
func (ps *peerSet) Len() int {
ps.lock.RLock()
defer ps.lock.RUnlock()

return len(ps.peers)
}

// AllPeers retrieves a flat list of all the peers within the set.
func (ps *peerSet) AllPeers() []*peer {
ps.lock.RLock()
defer ps.lock.RUnlock()

list := make([]*peer, 0, len(ps.peers))
for _, p := range ps.peers {
list = append(list, p)
}
return list
}

// HeaderIdlePeers retrieves a flat list of all the currently header-idle peers
// within the active peer set, ordered by their reputation.
func (ps *peerSet) HeaderIdlePeers() ([]*peer, int) {
idle := func(p *peer) bool {
return atomic.LoadInt32(&p.headerIdle) == 0
}
throughput := func(p *peer) float64 {
p.lock.RLock()
defer p.lock.RUnlock()
return p.headerThroughput
}
return ps.idlePeers(62, 64, idle, throughput)
}

// BodyIdlePeers retrieves a flat list of all the currently body-idle peers within
// the active peer set, ordered by their reputation.
func (ps *peerSet) BodyIdlePeers() ([]*peer, int) {
idle := func(p *peer) bool {
return atomic.LoadInt32(&p.blockIdle) == 0
```

```
}
throughput := func(p *peer) float64 {
p.lock.RLock()
defer p.lock.RUnlock()
return p.blockThroughput
}
return ps.idlePeers(62, 64, idle, throughput)
}

// ReceiptIdlePeers retrieves a flat list of all the currently receipt-idle peers
// within the active peer set, ordered by their reputation.
func (ps *peerSet) ReceiptIdlePeers() ([]*peer, int) {
idle := func(p *peer) bool {
return atomic.LoadInt32(&p.receiptIdle) == 0
}
throughput := func(p *peer) float64 {
p.lock.RLock()
defer p.lock.RUnlock()
return p.receiptThroughput
}
return ps.idlePeers(63, 64, idle, throughput)
}

// NodeDataIdlePeers retrieves a flat list of all the currently node-data-idle
// peers within the active peer set, ordered by their reputation.
func (ps *peerSet) NodeDataIdlePeers() ([]*peer, int) {
idle := func(p *peer) bool {
return atomic.LoadInt32(&p.stateIdle) == 0
}
throughput := func(p *peer) float64 {
p.lock.RLock()
defer p.lock.RUnlock()
return p.stateThroughput
}
return ps.idlePeers(63, 64, idle, throughput)
}

// idlePeers retrieves a flat list of all currently idle peers satisfying the
// protocol version constraints, using the provided function to check idleness.
// The resulting set of peers are sorted by their measure throughput.
func (ps *peerSet) idlePeers(minProtocol, maxProtocol int, idleCheck func(*peer) bool, throughput
func(*peer) float64) ([]*peer, int) {
```

```
ps.lock.RLock()
defer ps.lock.RUnlock()

idle, total := make([]*peer, 0, len(ps.peers)), 0
for _, p := range ps.peers {
if p.version >= minProtocol && p.version <= maxProtocol {
if idleCheck(p) {
idle = append(idle, p)
}
total++
}
}
for i := 0; i < len(idle); i++ {
for j := i + 1; j < len(idle); j++ {
if throughput(idle[i]) < throughput(idle[j]) {
idle[i], idle[j] = idle[j], idle[i]
}
}
}
return idle, total
}

// medianRTT returns the median RTT of te peerset, considering only the tuning
// peers if there are more peers available.
func (ps *peerSet) medianRTT() time.Duration {
// Gather all the currnetly measured round trip times
ps.lock.RLock()
defer ps.lock.RUnlock()

rtts := make([]float64, 0, len(ps.peers))
for _, p := range ps.peers {
p.lock.RLock()
rtts = append(rtts, float64(p.rtt))
p.lock.RUnlock()
}
sort.Float64s(rtts)

median := rttMaxEstimate
if qosTuningPeers <= len(rtts) {
median = time.Duration(rtts[qosTuningPeers/2]) // Median of our tuning peers
} else if len(rtts) > 0 {
median = time.Duration(rtts[len(rtts)/2]) // Median of our connected peers (maintain even like this
```

```
some baseline qos)
}
// Restrict the RTT into some QoS defaults, irrelevant of true RTT
if median < rttMinEstimate {
median = rttMinEstimate
}
if median > rttMaxEstimate {
median = rttMaxEstimate
}
return median
}

5:F:\git\coin\ethereum\go-ethereum\eth\downloader\queue.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Contains the block download scheduler to collect download tasks and schedule
// them in an ordered, and throttled way.

package downloader

import (
"errors"
"fmt"
"sync"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/log"
"github.com/rcrowley/go-metrics"
"gopkg.in/karalabe/cookiejar.v2/collections/prque"
)

var blockCacheLimit = 8192 // Maximum number of blocks to cache before throttling the download

var (
errNoFetchesPending = errors.New("no fetches pending")
errStaleDelivery    = errors.New("stale delivery")
)

// fetchRequest is a currently running data retrieval operation.
type fetchRequest struct {
```

```
Peer    *peer              // Peer to which the request was sent
From    uint64             // [eth/62] Requested chain element index (used for skeleton fills only)
Hashes  map[common.Hash]int // [eth/61] Requested hashes with their insertion index (priority)
Headers []*types.Header    // [eth/62] Requested headers, sorted by request order
Time    time.Time          // Time when the request was made
}

// fetchResult is a struct collecting partial results from data fetchers until
// all outstanding pieces complete and the result as a whole can be processed.
type fetchResult struct {
Pending int // Number of data fetches still pending

Header       *types.Header
Uncles       []*types.Header
Transactions types.Transactions
Receipts     types.Receipts
}

// queue represents hashes that are either need fetching or are being fetched
type queue struct {
mode          SyncMode // Synchronisation mode to decide on the block parts to schedule for fetching
fastSyncPivot uint64   // Block number where the fast sync pivots into archive synchronisation mode

headerHead common.Hash // [eth/62] Hash of the last queued header to verify order

// Headers are "special", they download in batches, supported by a skeleton chain
headerTaskPool  map[uint64]*types.Header       // [eth/62] Pending header retrieval tasks, mapping starting indexes to skeleton headers
headerTaskQueue *prque.Prque                   // [eth/62] Priority queue of the skeleton indexes to fetch the filling headers for
headerPeerMiss  map[string]map[uint64]struct{} // [eth/62] Set of per-peer header batches known to be unavailable
headerPendPool  map[string]*fetchRequest       // [eth/62] Currently pending header retrieval operations
headerResults   []*types.Header                // [eth/62] Result cache accumulating the completed headers
headerProced    int                            // [eth/62] Number of headers already processed from the results
headerOffset    uint64                         // [eth/62] Number of the first header in the result cache
headerContCh    chan bool                      // [eth/62] Channel to notify when header download
```

```
finishes

// All data retrievals below are based on an already assembles header chain
blockTaskPool  map[common.Hash]*types.Header // [eth/62] Pending block (body) retrieval tasks, mapping hashes to headers
blockTaskQueue *prque.Prque                  // [eth/62] Priority queue of the headers to fetch the blocks (bodies) for
blockPendPool  map[string]*fetchRequest      // [eth/62] Currently pending block (body) retrieval operations
blockDonePool  map[common.Hash]struct{}      // [eth/62] Set of the completed block (body) fetches

receiptTaskPool  map[common.Hash]*types.Header // [eth/63] Pending receipt retrieval tasks, mapping hashes to headers
receiptTaskQueue *prque.Prque                  // [eth/63] Priority queue of the headers to fetch the receipts for
receiptPendPool  map[string]*fetchRequest      // [eth/63] Currently pending receipt retrieval operations
receiptDonePool  map[common.Hash]struct{}      // [eth/63] Set of the completed receipt fetches

resultCache  []*fetchResult // Downloaded but not yet delivered fetch results
resultOffset uint64         // Offset of the first cached fetch result in the block chain

lock   *sync.Mutex
active *sync.Cond
closed bool
}

// newQueue creates a new download queue for scheduling block retrieval.
func newQueue() *queue {
lock := new(sync.Mutex)
return &queue{
headerPendPool:   make(map[string]*fetchRequest),
headerContCh:     make(chan bool),
blockTaskPool:    make(map[common.Hash]*types.Header),
blockTaskQueue:   prque.New(),
blockPendPool:    make(map[string]*fetchRequest),
blockDonePool:    make(map[common.Hash]struct{}),
receiptTaskPool:  make(map[common.Hash]*types.Header),
receiptTaskQueue: prque.New(),
receiptPendPool:  make(map[string]*fetchRequest),
receiptDonePool:  make(map[common.Hash]struct{}),
```

```
resultCache:      make([]*fetchResult, blockCacheLimit),
active:           sync.NewCond(lock),
lock:             lock,
}
}

// Reset clears out the queue contents.
func (q *queue) Reset() {
q.lock.Lock()
defer q.lock.Unlock()

q.closed = false
q.mode = FullSync
q.fastSyncPivot = 0

q.headerHead = common.Hash{}

q.headerPendPool = make(map[string]*fetchRequest)

q.blockTaskPool = make(map[common.Hash]*types.Header)
q.blockTaskQueue.Reset()
q.blockPendPool = make(map[string]*fetchRequest)
q.blockDonePool = make(map[common.Hash]struct{})

q.receiptTaskPool = make(map[common.Hash]*types.Header)
q.receiptTaskQueue.Reset()
q.receiptPendPool = make(map[string]*fetchRequest)
q.receiptDonePool = make(map[common.Hash]struct{})

q.resultCache = make([]*fetchResult, blockCacheLimit)
q.resultOffset = 0
}

// Close marks the end of the sync, unblocking WaitResults.
// It may be called even if the queue is already closed.
func (q *queue) Close() {
q.lock.Lock()
q.closed = true
q.lock.Unlock()
q.active.Broadcast()
}
```

```
// PendingHeaders retrieves the number of header requests pending for retrieval.
func (q *queue) PendingHeaders() int {
q.lock.Lock()
defer q.lock.Unlock()

return q.headerTaskQueue.Size()
}

// PendingBlocks retrieves the number of block (body) requests pending for retrieval.
func (q *queue) PendingBlocks() int {
q.lock.Lock()
defer q.lock.Unlock()

return q.blockTaskQueue.Size()
}

// PendingReceipts retrieves the number of block receipts pending for retrieval.
func (q *queue) PendingReceipts() int {
q.lock.Lock()
defer q.lock.Unlock()

return q.receiptTaskQueue.Size()
}

// InFlightHeaders retrieves whether there are header fetch requests currently
// in flight.
func (q *queue) InFlightHeaders() bool {
q.lock.Lock()
defer q.lock.Unlock()

return len(q.headerPendPool) > 0
}

// InFlightBlocks retrieves whether there are block fetch requests currently in
// flight.
func (q *queue) InFlightBlocks() bool {
q.lock.Lock()
defer q.lock.Unlock()

return len(q.blockPendPool) > 0
}
```

```
// InFlightReceipts retrieves whether there are receipt fetch requests currently
// in flight.
func (q *queue) InFlightReceipts() bool {
q.lock.Lock()
defer q.lock.Unlock()

return len(q.receiptPendPool) > 0
}

// Idle returns if the queue is fully idle or has some data still inside.
func (q *queue) Idle() bool {
q.lock.Lock()
defer q.lock.Unlock()

queued := q.blockTaskQueue.Size() + q.receiptTaskQueue.Size()
pending := len(q.blockPendPool) + len(q.receiptPendPool)
cached := len(q.blockDonePool) + len(q.receiptDonePool)

return (queued + pending + cached) == 0
}

// FastSyncPivot retrieves the currently used fast sync pivot point.
func (q *queue) FastSyncPivot() uint64 {
q.lock.Lock()
defer q.lock.Unlock()

return q.fastSyncPivot
}

// ShouldThrottleBlocks checks if the download should be throttled (active block (body)
// fetches exceed block cache).
func (q *queue) ShouldThrottleBlocks() bool {
q.lock.Lock()
defer q.lock.Unlock()

// Calculate the currently in-flight block (body) requests
pending := 0
for _, request := range q.blockPendPool {
pending += len(request.Hashes) + len(request.Headers)
}
// Throttle if more blocks (bodies) are in-flight than free space in the cache
return pending >= len(q.resultCache)-len(q.blockDonePool)
```

```
}

// ShouldThrottleReceipts checks if the download should be throttled (active receipt
// fetches exceed block cache).
func (q *queue) ShouldThrottleReceipts() bool {
q.lock.Lock()
defer q.lock.Unlock()

// Calculate the currently in-flight receipt requests
pending := 0
for _, request := range q.receiptPendPool {
pending += len(request.Headers)
}
// Throttle if more receipts are in-flight than free space in the cache
return pending >= len(q.resultCache)-len(q.receiptDonePool)
}

// ScheduleSkeleton adds a batch of header retrieval tasks to the queue to fill
// up an already retrieved header skeleton.
func (q *queue) ScheduleSkeleton(from uint64, skeleton []*types.Header) {
q.lock.Lock()
defer q.lock.Unlock()

// No skeleton retrieval can be in progress, fail hard if so (huge implementation bug)
if q.headerResults != nil {
panic("skeleton assembly already in progress")
}
// Shedule all the header retrieval tasks for the skeleton assembly
q.headerTaskPool = make(map[uint64]*types.Header)
q.headerTaskQueue = prque.New()
q.headerPeerMiss = make(map[string]map[uint64]struct{}) // Reset availability to correct invalid
chains
q.headerResults = make([]*types.Header, len(skeleton)*MaxHeaderFetch)
q.headerProced = 0
q.headerOffset = from
q.headerContCh = make(chan bool, 1)

for i, header := range skeleton {
index := from + uint64(i*MaxHeaderFetch)

q.headerTaskPool[index] = header
q.headerTaskQueue.Push(index, -float32(index))
```

```
}
}

// RetrieveHeaders retrieves the header chain assemble based on the scheduled
// skeleton.
func (q *queue) RetrieveHeaders() ([]*types.Header, int) {
q.lock.Lock()
defer q.lock.Unlock()

headers, proced := q.headerResults, q.headerProced
q.headerResults, q.headerProced = nil, 0

return headers, proced
}

// Schedule adds a set of headers for the download queue for scheduling, returning
// the new headers encountered.
func (q *queue) Schedule(headers []*types.Header, from uint64) []*types.Header {
q.lock.Lock()
defer q.lock.Unlock()

// Insert all the headers prioritised by the contained block number
inserts := make([]*types.Header, 0, len(headers))
for _, header := range headers {
// Make sure chain order is honoured and preserved throughout
hash := header.Hash()
if header.Number == nil || header.Number.Uint64() != from {
log.Warn("Header broke chain ordering", "number", header.Number, "hash", hash, "expected",
from)
break
}
if q.headerHead != (common.Hash{}) && q.headerHead != header.ParentHash {
log.Warn("Header broke chain ancestry", "number", header.Number, "hash", hash)
break
}
// Make sure no duplicate requests are executed
if _, ok := q.blockTaskPool[hash]; ok {
log.Warn("Header  already scheduled for block fetch", "number", header.Number, "hash", hash)
continue
}
if _, ok := q.receiptTaskPool[hash]; ok {
log.Warn("Header already scheduled for receipt fetch", "number", header.Number, "hash", hash)
```

```
continue
}
// Queue the header for content retrieval
q.blockTaskPool[hash] = header
q.blockTaskQueue.Push(header, -float32(header.Number.Uint64()))

if q.mode == FastSync && header.Number.Uint64() <= q.fastSyncPivot {
// Fast phase of the fast sync, retrieve receipts too
q.receiptTaskPool[hash] = header
q.receiptTaskQueue.Push(header, -float32(header.Number.Uint64()))
}
inserts = append(inserts, header)
q.headerHead = hash
from++
}
return inserts
}

// WaitResults retrieves and permanently removes a batch of fetch
// results from the cache. the result slice will be empty if the queue
// has been closed.
func (q *queue) WaitResults() []*fetchResult {
q.lock.Lock()
defer q.lock.Unlock()

nproc := q.countProcessableItems()
for nproc == 0 && !q.closed {
q.active.Wait()
nproc = q.countProcessableItems()
}
results := make([]*fetchResult, nproc)
copy(results, q.resultCache[:nproc])
if len(results) > 0 {
// Mark results as done before dropping them from the cache.
for _, result := range results {
hash := result.Header.Hash()
delete(q.blockDonePool, hash)
delete(q.receiptDonePool, hash)
}
// Delete the results from the cache and clear the tail.
copy(q.resultCache, q.resultCache[nproc:])
for i := len(q.resultCache) - nproc; i < len(q.resultCache); i++ {
```

```
q.resultCache[i] = nil
}
// Advance the expected block number of the first cache entry.
q.resultOffset += uint64(nproc)
}
return results
}

// countProcessableItems counts the processable items.
func (q *queue) countProcessableItems() int {
for i, result := range q.resultCache {
// Don't process incomplete or unavailable items.
if result == nil || result.Pending > 0 {
return i
}
// Stop before processing the pivot block to ensure that
// resultCache has space for fsHeaderForceVerify items. Not
// doing this could leave us unable to download the required
// amount of headers.
if q.mode == FastSync && result.Header.Number.Uint64() == q.fastSyncPivot {
for j := 0; j < fsHeaderForceVerify; j++ {
if i+j+1 >= len(q.resultCache) || q.resultCache[i+j+1] == nil {
return i
}
}
}
}
return len(q.resultCache)
}

// ReserveHeaders reserves a set of headers for the given peer, skipping any
// previously failed batches.
func (q *queue) ReserveHeaders(p *peer, count int) *fetchRequest {
q.lock.Lock()
defer q.lock.Unlock()

// Short circuit if the peer's already downloading something (sanity check to
// not corrupt state)
if _, ok := q.headerPendPool[p.id]; ok {
return nil
}
// Retrieve a batch of hashes, skipping previously failed ones
```

```
send, skip := uint64(0), []uint64{}
for send == 0 && !q.headerTaskQueue.Empty() {
from, _ := q.headerTaskQueue.Pop()
if q.headerPeerMiss[p.id] != nil {
if _, ok := q.headerPeerMiss[p.id][from.(uint64)]; ok {
skip = append(skip, from.(uint64))
continue
}
}
send = from.(uint64)
}
// Merge all the skipped batches back
for _, from := range skip {
q.headerTaskQueue.Push(from, -float32(from))
}
// Assemble and return the block download request
if send == 0 {
return nil
}
request := &fetchRequest{
Peer: p,
From: send,
Time: time.Now(),
}
q.headerPendPool[p.id] = request
return request
}

// ReserveBodies reserves a set of body fetches for the given peer, skipping any
// previously failed downloads. Beside the next batch of needed fetches, it also
// returns a flag whether empty blocks were queued requiring processing.
func (q *queue) ReserveBodies(p *peer, count int) (*fetchRequest, bool, error) {
isNoop := func(header *types.Header) bool {
return header.TxHash == types.EmptyRootHash && header.UncleHash ==
types.EmptyUncleHash
}
q.lock.Lock()
defer q.lock.Unlock()

return q.reserveHeaders(p, count, q.blockTaskPool, q.blockTaskQueue, q.blockPendPool,
q.blockDonePool, isNoop)
}
```

```
// ReserveReceipts reserves a set of receipt fetches for the given peer, skipping
// any previously failed downloads. Beside the next batch of needed fetches, it
// also returns a flag whether empty receipts were queued requiring importing.
func (q *queue) ReserveReceipts(p *peer, count int) (*fetchRequest, bool, error) {
isNoop := func(header *types.Header) bool {
return header.ReceiptHash == types.EmptyRootHash
}
q.lock.Lock()
defer q.lock.Unlock()

return q.reserveHeaders(p, count, q.receiptTaskPool, q.receiptTaskQueue, q.receiptPendPool,
q.receiptDonePool, isNoop)
}

// reserveHeaders reserves a set of data download operations for a given peer,
// skipping any previously failed ones. This method is a generic version used
// by the individual special reservation functions.
//
// Note, this method expects the queue lock to be already held for writing. The
// reason the lock is not obtained in here is because the parameters already need
// to access the queue, so they already need a lock anyway.
func (q *queue) reserveHeaders(p *peer, count int, taskPool map[common.Hash]*types.Header,
taskQueue *prque.Prque,
pendPool map[string]*fetchRequest, donePool map[common.Hash]struct{}, isNoop
func(*types.Header) bool) (*fetchRequest, bool, error) {
// Short circuit if the pool has been depleted, or if the peer's already
// downloading something (sanity check not to corrupt state)
if taskQueue.Empty() {
return nil, false, nil
}
if _, ok := pendPool[p.id]; ok {
return nil, false, nil
}
// Calculate an upper limit on the items we might fetch (i.e. throttling)
space := len(q.resultCache) - len(donePool)
for _, request := range pendPool {
space -= len(request.Headers)
}
// Retrieve a batch of tasks, skipping previously failed ones
send := make([]*types.Header, 0, count)
skip := make([]*types.Header, 0)
```

```
progress := false
for proc := 0; proc < space && len(send) < count && !taskQueue.Empty(); proc++ {
header := taskQueue.PopItem().(*types.Header)

// If we're the first to request this task, initialise the result container
index := int(header.Number.Int64() - int64(q.resultOffset))
if index >= len(q.resultCache) || index < 0 {
common.Report("index allocation went beyond available resultCache space")
return nil, false, errInvalidChain
}
if q.resultCache[index] == nil {
components := 1
if q.mode == FastSync && header.Number.Uint64() <= q.fastSyncPivot {
components = 2
}
q.resultCache[index] = &fetchResult{
Pending: components,
Header:  header,
}
}
// If this fetch task is a noop, skip this fetch operation
if isNoop(header) {
donePool[header.Hash()] = struct{}{}
delete(taskPool, header.Hash())

space, proc = space-1, proc-1
q.resultCache[index].Pending--
progress = true
continue
}
// Otherwise unless the peer is known not to have the data, add to the retrieve list
if p.Lacks(header.Hash()) {
skip = append(skip, header)
} else {
send = append(send, header)
}
}
// Merge all the skipped headers back
for _, header := range skip {
taskQueue.Push(header, -float32(header.Number.Uint64()))
}
```

```
if progress {
// Wake WaitResults, resultCache was modified
q.active.Signal()
}
// Assemble and return the block download request
if len(send) == 0 {
return nil, progress, nil
}
request := &fetchRequest{
Peer:    p,
Headers: send,
Time:    time.Now(),
}
pendPool[p.id] = request

return request, progress, nil
}

// CancelHeaders aborts a fetch request, returning all pending skeleton indexes to the queue.
func (q *queue) CancelHeaders(request *fetchRequest) {
q.cancel(request, q.headerTaskQueue, q.headerPendPool)
}

// CancelBodies aborts a body fetch request, returning all pending headers to the
// task queue.
func (q *queue) CancelBodies(request *fetchRequest) {
q.cancel(request, q.blockTaskQueue, q.blockPendPool)
}

// CancelReceipts aborts a body fetch request, returning all pending headers to
// the task queue.
func (q *queue) CancelReceipts(request *fetchRequest) {
q.cancel(request, q.receiptTaskQueue, q.receiptPendPool)
}

// Cancel aborts a fetch request, returning all pending hashes to the task queue.
func (q *queue) cancel(request *fetchRequest, taskQueue *prque.Prque, pendPool
map[string]*fetchRequest) {
q.lock.Lock()
defer q.lock.Unlock()

if request.From > 0 {
```

```
taskQueue.Push(request.From, -float32(request.From))
}
for hash, index := range request.Hashes {
taskQueue.Push(hash, float32(index))
}
for _, header := range request.Headers {
taskQueue.Push(header, -float32(header.Number.Uint64()))
}
delete(pendPool, request.Peer.id)
}

// Revoke cancels all pending requests belonging to a given peer. This method is
// meant to be called during a peer drop to quickly reassign owned data fetches
// to remaining nodes.
func (q *queue) Revoke(peerId string) {
q.lock.Lock()
defer q.lock.Unlock()

if request, ok := q.blockPendPool[peerId]; ok {
for _, header := range request.Headers {
q.blockTaskQueue.Push(header, -float32(header.Number.Uint64()))
}
delete(q.blockPendPool, peerId)
}
if request, ok := q.receiptPendPool[peerId]; ok {
for _, header := range request.Headers {
q.receiptTaskQueue.Push(header, -float32(header.Number.Uint64()))
}
delete(q.receiptPendPool, peerId)
}
}

// ExpireHeaders checks for in flight requests that exceeded a timeout allowance,
// canceling them and returning the responsible peers for penalisation.
func (q *queue) ExpireHeaders(timeout time.Duration) map[string]int {
q.lock.Lock()
defer q.lock.Unlock()

return q.expire(timeout, q.headerPendPool, q.headerTaskQueue, headerTimeoutMeter)
}

// ExpireBodies checks for in flight block body requests that exceeded a timeout
```

```
// allowance, canceling them and returning the responsible peers for penalisation.
func (q *queue) ExpireBodies(timeout time.Duration) map[string]int {
q.lock.Lock()
defer q.lock.Unlock()

return q.expire(timeout, q.blockPendPool, q.blockTaskQueue, bodyTimeoutMeter)
}

// ExpireReceipts checks for in flight receipt requests that exceeded a timeout
// allowance, canceling them and returning the responsible peers for penalisation.
func (q *queue) ExpireReceipts(timeout time.Duration) map[string]int {
q.lock.Lock()
defer q.lock.Unlock()

return q.expire(timeout, q.receiptPendPool, q.receiptTaskQueue, receiptTimeoutMeter)
}

// expire is the generic check that move expired tasks from a pending pool back
// into a task pool, returning all entities caught with expired tasks.
//
// Note, this method expects the queue lock to be already held. The
// reason the lock is not obtained in here is because the parameters already need
// to access the queue, so they already need a lock anyway.
func (q *queue) expire(timeout time.Duration, pendPool map[string]*fetchRequest, taskQueue
*prque.Prque, timeoutMeter metrics.Meter) map[string]int {
// Iterate over the expired requests and return each to the queue
expiries := make(map[string]int)
for id, request := range pendPool {
if time.Since(request.Time) > timeout {
// Update the metrics with the timeout
timeoutMeter.Mark(1)

// Return any non satisfied requests to the pool
if request.From > 0 {
taskQueue.Push(request.From, -float32(request.From))
}
for hash, index := range request.Hashes {
taskQueue.Push(hash, float32(index))
}
for _, header := range request.Headers {
taskQueue.Push(header, -float32(header.Number.Uint64()))
}
```

```
// Add the peer to the expiry report along the the number of failed requests
expirations := len(request.Hashes)
if expirations < len(request.Headers) {
expirations = len(request.Headers)
}
expiries[id] = expirations
}
}
// Remove the expired requests from the pending pool
for id := range expiries {
delete(pendPool, id)
}
return expiries
}

// DeliverHeaders injects a header retrieval response into the header results
// cache. This method either accepts all headers it received, or none of them
// if they do not map correctly to the skeleton.
//
// If the headers are accepted, the method makes an attempt to deliver the set
// of ready headers to the processor to keep the pipeline full. However it will
// not block to prevent stalling other pending deliveries.
func (q *queue) DeliverHeaders(id string, headers []*types.Header, headerProcCh chan
[]*types.Header) (int, error) {
q.lock.Lock()
defer q.lock.Unlock()

// Short circuit if the data was never requested
request := q.headerPendPool[id]
if request == nil {
return 0, errNoFetchesPending
}
headerReqTimer.UpdateSince(request.Time)
delete(q.headerPendPool, id)

// Ensure headers can be mapped onto the skeleton chain
target := q.headerTaskPool[request.From].Hash()

accepted := len(headers) == MaxHeaderFetch
if accepted {
if headers[0].Number.Uint64() != request.From {
log.Trace("First header broke chain ordering", "peer", id, "number", headers[0].Number, "hash",
```

```
headers[0].Hash(), request.From)
accepted = false
} else if headers[len(headers)-1].Hash() != target {
log.Trace("Last header broke skeleton structure ", "peer", id, "number", headers[len(headers)-
1].Number, "hash", headers[len(headers)-1].Hash(), "expected", target)
accepted = false
}
}
if accepted {
for i, header := range headers[1:] {
hash := header.Hash()
if want := request.From + 1 + uint64(i); header.Number.Uint64() != want {
log.Warn("Header broke chain ordering", "peer", id, "number", header.Number, "hash", hash,
"expected", want)
accepted = false
break
}
if headers[i].Hash() != header.ParentHash {
log.Warn("Header broke chain ancestry", "peer", id, "number", header.Number, "hash", hash)
accepted = false
break
}
}
}
// If the batch of headers wasn't accepted, mark as unavailable
if !accepted {
log.Trace("Skeleton filling not accepted", "peer", id, "from", request.From)

miss := q.headerPeerMiss[id]
if miss == nil {
q.headerPeerMiss[id] = make(map[uint64]struct{})
miss = q.headerPeerMiss[id]
}
miss[request.From] = struct{}{}

q.headerTaskQueue.Push(request.From, -float32(request.From))
return 0, errors.New("delivery not accepted")
}
// Clean up a successful fetch and try to deliver any sub-results
copy(q.headerResults[request.From-q.headerOffset:], headers)
delete(q.headerTaskPool, request.From)
```

```
ready := 0
for q.headerProced+ready < len(q.headerResults) && q.headerResults[q.headerProced+ready] !=
nil {
ready += MaxHeaderFetch
}
if ready > 0 {
// Headers are ready for delivery, gather them and push forward (non blocking)
process := make([]*types.Header, ready)
copy(process, q.headerResults[q.headerProced:q.headerProced+ready])

select {
case headerProcCh <- process:
log.Trace("Pre-scheduled new headers", "peer", id, "count", len(process), "from",
process[0].Number)
q.headerProced += len(process)
default:
}
}
// Check for termination and return
if len(q.headerTaskPool) == 0 {
q.headerContCh <- false
}
return len(headers), nil
}

// DeliverBodies injects a block body retrieval response into the results queue.
// The method returns the number of blocks bodies accepted from the delivery and
// also wakes any threads waiting for data delivery.
func (q *queue) DeliverBodies(id string, txLists [][]*types.Transaction, uncleLists [][]*types.Header)
(int, error) {
q.lock.Lock()
defer q.lock.Unlock()

reconstruct := func(header *types.Header, index int, result *fetchResult) error {
if types.DeriveSha(types.Transactions(txLists[index])) != header.TxHash ||
types.CalcUncleHash(uncleLists[index]) != header.UncleHash {
return errInvalidBody
}
result.Transactions = txLists[index]
result.Uncles = uncleLists[index]
return nil
}
```

```
return q.deliver(id, q.blockTaskPool, q.blockTaskQueue, q.blockPendPool, q.blockDonePool,
bodyReqTimer, len(txLists), reconstruct)
}

// DeliverReceipts injects a receipt retrieval response into the results queue.
// The method returns the number of transaction receipts accepted from the delivery
// and also wakes any threads waiting for data delivery.
func (q *queue) DeliverReceipts(id string, receiptList [][]*types.Receipt) (int, error) {
q.lock.Lock()
defer q.lock.Unlock()

reconstruct := func(header *types.Header, index int, result *fetchResult) error {
if types.DeriveSha(types.Receipts(receiptList[index])) != header.ReceiptHash {
return errInvalidReceipt
}
result.Receipts = receiptList[index]
return nil
}
return q.deliver(id, q.receiptTaskPool, q.receiptTaskQueue, q.receiptPendPool,
q.receiptDonePool, receiptReqTimer, len(receiptList), reconstruct)
}

// deliver injects a data retrieval response into the results queue.
//
// Note, this method expects the queue lock to be already held for writing. The
// reason the lock is not obtained in here is because the parameters already need
// to access the queue, so they already need a lock anyway.
func (q *queue) deliver(id string, taskPool map[common.Hash]*types.Header, taskQueue
*prque.Prque,
pendPool map[string]*fetchRequest, donePool map[common.Hash]struct{}, reqTimer
metrics.Timer,
results int, reconstruct func(header *types.Header, index int, result *fetchResult) error) (int, error) {

// Short circuit if the data was never requested
request := pendPool[id]
if request == nil {
return 0, errNoFetchesPending
}
reqTimer.UpdateSince(request.Time)
delete(pendPool, id)

// If no data items were retrieved, mark them as unavailable for the origin peer
```

```
if results == 0 {
for _, header := range request.Headers {
request.Peer.MarkLacking(header.Hash())
}
}
// Assemble each of the results with their headers and retrieved data parts
var (
accepted int
failure  error
useful   bool
)
for i, header := range request.Headers {
// Short circuit assembly if no more fetch results are found
if i >= results {
break
}
// Reconstruct the next result if contents match up
index := int(header.Number.Int64() - int64(q.resultOffset))
if index >= len(q.resultCache) || index < 0 || q.resultCache[index] == nil {
failure = errInvalidChain
break
}
if err := reconstruct(header, i, q.resultCache[index]); err != nil {
failure = err
break
}
donePool[header.Hash()] = struct{}{}
q.resultCache[index].Pending--
useful = true
accepted++

// Clean up a successful fetch
request.Headers[i] = nil
delete(taskPool, header.Hash())
}
// Return all failed or missing fetches to the queue
for _, header := range request.Headers {
if header != nil {
taskQueue.Push(header, -float32(header.Number.Uint64()))
}
}
// Wake up WaitResults
```

```
if accepted > 0 {
q.active.Signal()
}
// If none of the data was good, it's a stale delivery
switch {
case failure == nil || failure == errInvalidChain:
return accepted, failure
case useful:
return accepted, fmt.Errorf("partial failure: %v", failure)
default:
return accepted, errStaleDelivery
}
}

// Prepare configures the result cache to allow accepting and caching inbound
// fetch results.
func (q *queue) Prepare(offset uint64, mode SyncMode, pivot uint64, head *types.Header) {
q.lock.Lock()
defer q.lock.Unlock()

// Prepare the queue for sync results
if q.resultOffset < offset {
q.resultOffset = offset
}
q.fastSyncPivot = pivot
q.mode = mode
}

6:F:\git\coin\ethereum\go-ethereum\eth\downloader\statesync.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package downloader

import (
"fmt"
"hash"
"sync"
"sync/atomic"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/state"
```

```
"github.com/ethereum/go-ethereum/crypto/sha3"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/trie"
)

// stateReq represents a batch of state fetch requests groupped together into
// a single data retrieval network packet.
type stateReq struct {
items    []common.Hash              // Hashes of the state items to download
tasks    map[common.Hash]*stateTask // Download tasks to track previous attempts
timeout  time.Duration              // Maximum round trip time for this to complete
timer    *time.Timer                // Timer to fire when the RTT timeout expires
peer     *peer                      // Peer that we're requesting from
response [][]byte                   // Response data of the peer (nil for timeouts)
}

// timedOut returns if this request timed out.
func (req *stateReq) timedOut() bool {
return req.response == nil
}

// stateSyncStats is a collection of progress stats to report during a state trie
// sync to RPC requests as well as to display in user logs.
type stateSyncStats struct {
processed  uint64 // Number of state entries processed
duplicate  uint64 // Number of state entries downloaded twice
unexpected uint64 // Number of non-requested state entries received
pending    uint64 // Number of still pending state entries
}

// syncState starts downloading state with the given root hash.
func (d *Downloader) syncState(root common.Hash) *stateSync {
s := newStateSync(d, root)
select {
case d.stateSyncStart <- s:
case <-d.quitCh:
s.err = errCancelStateFetch
close(s.done)
}
return s
}
```

```
// stateFetcher manages the active state sync and accepts requests
// on its behalf.
func (d *Downloader) stateFetcher() {
for {
select {
case s := <-d.stateSyncStart:
for next := s; next != nil; {
next = d.runStateSync(next)
}
case <-d.stateCh:
// Ignore state responses while no sync is running.
case <-d.quitCh:
return
}
}
}

// runStateSync runs a state synchronisation until it completes or another root
// hash is requested to be switched over to.
func (d *Downloader) runStateSync(s *stateSync) *stateSync {
var (
active   = make(map[string]*stateReq) // Currently in-flight requests
finished []*stateReq                  // Completed or failed requests
timeout  = make(chan *stateReq)       // Timed out active requests
)
defer func() {
// Cancel active request timers on exit. Also set peers to idle so they're
// available for the next sync.
for _, req := range active {
req.timer.Stop()
req.peer.SetNodeDataIdle(len(req.items))
}
}()
// Run the state sync.
go s.run()
defer s.Cancel()

for {
// Enable sending of the first buffered element if there is one.
var (
deliverReq   *stateReq
deliverReqCh chan *stateReq
```

```
)
if len(finished) > 0 {
deliverReq = finished[0]
deliverReqCh = s.deliver
}

select {
// The stateSync lifecycle:
case next := <-d.stateSyncStart:
return next

case <-s.done:
return nil

// Send the next finished request to the current sync:
case deliverReqCh <- deliverReq:
finished = append(finished[:0], finished[1:]...)

// Handle incoming state packs:
case pack := <-d.stateCh:
// Discard any data not requested (or previsouly timed out)
req := active[pack.PeerId()]
if req == nil {
log.Debug("Unrequested node data", "peer", pack.PeerId(), "len", pack.Items())
continue
}
// Finalize the request and queue up for processing
req.timer.Stop()
req.response = pack.(*statePack).states

finished = append(finished, req)
delete(active, pack.PeerId())

// Handle timed-out requests:
case req := <-timeout:
// If the peer is already requesting something else, ignore the stale timeout.
// This can happen when the timeout and the delivery happens simultaneously,
// causing both pathways to trigger.
if active[req.peer.id] != req {
continue
}
// Move the timed out data back into the download queue
```

```
finished = append(finished, req)
delete(active, req.peer.id)

// Track outgoing state requests:
case req := <-d.trackStateReq:
// If an active request already exists for this peer, we have a problem. In
// theory the trie node schedule must never assign two requests to the same
// peer. In practive however, a peer might receive a request, disconnect and
// immediately reconnect before the previous times out. In this case the first
// request is never honored, alas we must not silently overwrite it, as that
// causes valid requests to go missing and sync to get stuck.
if old := active[req.peer.id]; old != nil {
log.Warn("Busy peer assigned new state fetch", "peer", old.peer.id)

// Make sure the previous one doesn't get siletly lost
finished = append(finished, old)
}
// Start a timer to notify the sync loop if the peer stalled.
req.timer = time.AfterFunc(req.timeout, func() {
select {
case timeout <- req:
case <-s.done:
// Prevent leaking of timer goroutines in the unlikely case where a
// timer is fired just before exiting runStateSync.
}
})
active[req.peer.id] = req
}
}
}

// stateSync schedules requests for downloading a particular state trie defined
// by a given state root.
type stateSync struct {
d *Downloader // Downloader instance to access and manage current peerset

sched  *state.StateSync          // State trie sync scheduler defining the tasks
keccak hash.Hash                 // Keccak256 hasher to verify deliveries with
tasks  map[common.Hash]*stateTask // Set of tasks currently queued for retrieval

deliver    chan *stateReq // Delivery channel multiplexing peer responses
cancel     chan struct{}  // Channel to signal a termination request
```

```
cancelOnce sync.Once      // Ensures cancel only ever gets called once
done       chan struct{}  // Channel to signal termination completion
err        error          // Any error hit during sync (set before completion)
}

// stateTask represents a single trie node download taks, containing a set of
// peers already attempted retrieval from to detect stalled syncs and abort.
type stateTask struct {
attempts map[string]struct{}
}

// newStateSync creates a new state trie download scheduler. This method does not
// yet start the sync. The user needs to call run to initiate.
func newStateSync(d *Downloader, root common.Hash) *stateSync {
return &stateSync{
d:       d,
sched:   state.NewStateSync(root, d.stateDB),
keccak:  sha3.NewKeccak256(),
tasks:   make(map[common.Hash]*stateTask),
deliver: make(chan *stateReq),
cancel:  make(chan struct{}),
done:    make(chan struct{}),
}
}

// run starts the task assignment and response processing loop, blocking until
// it finishes, and finally notifying any goroutines waiting for the loop to
// finish.
func (s *stateSync) run() {
s.err = s.loop()
close(s.done)
}

// Wait blocks until the sync is done or canceled.
func (s *stateSync) Wait() error {
<-s.done
return s.err
}

// Cancel cancels the sync and waits until it has shut down.
func (s *stateSync) Cancel() error {
s.cancelOnce.Do(func() { close(s.cancel) })
```

```
return s.Wait()
}

// loop is the main event loop of a state trie sync. It it responsible for the
// assignment of new tasks to peers (including sending it to them) as well as
// for the processing of inbound data. Note, that the loop does not directly
// receive data from peers, rather those are buffered up in the downloader and
// pushed here async. The reason is to decouple processing from data receipt
// and timeouts.
func (s *stateSync) loop() error {
// Listen for new peer events to assign tasks to them
newPeer := make(chan *peer, 1024)
peerSub := s.d.peers.SubscribeNewPeers(newPeer)
defer peerSub.Unsubscribe()

// Keep assigning new tasks until the sync completes or aborts
for s.sched.Pending() > 0 {
if err := s.assignTasks(); err != nil {
return err
}
// Tasks assigned, wait for something to happen
select {
case <-newPeer:
// New peer arrived, try to assign it download tasks

case <-s.cancel:
return errCancelStateFetch

case req := <-s.deliver:
// Response or timeout triggered, drop the peer if stalling
log.Trace("Received node data response", "peer", req.peer.id, "count", len(req.response),
"timeout", req.timedOut())
if len(req.items) <= 2 && req.timedOut() {
// 2 items are the minimum requested, if even that times out, we've no use of
// this peer at the moment.
log.Warn("Stalling state sync, dropping peer", "peer", req.peer.id)
s.d.dropPeer(req.peer.id)
}
// Process all the received blobs and check for stale delivery
stale, err := s.process(req)
if err != nil {
log.Warn("Node data write error", "err", err)
```

```
return err
}
// The the delivery contains requested data, mark the node idle (otherwise it's a timed out delivery)
if !stale {
req.peer.SetNodeDataIdle(len(req.response))
}
}
}
return nil
}

// assignTasks attempts to assing new tasks to all idle peers, either from the
// batch currently being retried, or fetching new data from the trie sync itself.
func (s *stateSync) assignTasks() error {
// Iterate over all idle peers and try to assign them state fetches
peers, _ := s.d.peers.NodeDataIdlePeers()
for _, p := range peers {
// Assign a batch of fetches proportional to the estimated latency/bandwidth
cap := p.NodeDataCapacity(s.d.requestRTT())
req := &stateReq{peer: p, timeout: s.d.requestTTL()}
s.fillTasks(cap, req)

// If the peer was assigned tasks to fetch, send the network request
if len(req.items) > 0 {
req.peer.log.Trace("Requesting new batch of data", "type", "state", "count", len(req.items))

select {
case s.d.trackStateReq <- req:
req.peer.FetchNodeData(req.items)
case <-s.cancel:
}
}
}
return nil
}

// fillTasks fills the given request object with a maximum of n state download
// tasks to send to the remote peer.
func (s *stateSync) fillTasks(n int, req *stateReq) {
// Refill available tasks from the scheduler.
if len(s.tasks) < n {
new := s.sched.Missing(n - len(s.tasks))
```

```
for _, hash := range new {
s.tasks[hash] = &stateTask{make(map[string]struct{})}
}
}
// Find tasks that haven't been tried with the request's peer.
req.items = make([]common.Hash, 0, n)
req.tasks = make(map[common.Hash]*stateTask, n)
for hash, t := range s.tasks {
// Stop when we've gathered enough requests
if len(req.items) == n {
break
}
// Skip any requests we've already tried from this peer
if _, ok := t.attempts[req.peer.id]; ok {
continue
}
// Assign the request to this peer
t.attempts[req.peer.id] = struct{}{}
req.items = append(req.items, hash)
req.tasks[hash] = t
delete(s.tasks, hash)
}
}

// process iterates over a batch of delivered state data, injecting each item
// into a running state sync, re-queuing any items that were requested but not
// delivered.
func (s *stateSync) process(req *stateReq) (bool, error) {
// Collect processing stats and update progress if valid data was received
processed, written, duplicate, unexpected := 0, 0, 0, 0

defer func(start time.Time) {
if processed+written+duplicate+unexpected > 0 {
s.updateStats(processed, written, duplicate, unexpected, time.Since(start))
}
}(time.Now())

// Iterate over all the delivered data and inject one-by-one into the trie
progress, stale := false, len(req.response) > 0

for _, blob := range req.response {
prog, hash, err := s.processNodeData(blob)
```

```
switch err {
case nil:
processed++
case trie.ErrNotRequested:
unexpected++
case trie.ErrAlreadyProcessed:
duplicate++
default:
return stale, fmt.Errorf("invalid state node %s: %v", hash.TerminalString(), err)
}
if prog {
progress = true
}
// If the node delivered a requested item, mark the delivery non-stale
if _, ok := req.tasks[hash]; ok {
delete(req.tasks, hash)
stale = false
}
}
// If some data managed to hit the database, flush and reset failure counters
if progress {
// Flush any accumulated data out to disk
batch := s.d.stateDB.NewBatch()

count, err := s.sched.Commit(batch)
if err != nil {
return stale, err
}
if err := batch.Write(); err != nil {
return stale, err
}
written = count

// If we're inside the critical section, reset fail counter since we progressed
if atomic.LoadUint32(&s.d.fsPivotFails) > 1 {
log.Trace("Fast-sync progressed, resetting fail counter", "previous",
atomic.LoadUint32(&s.d.fsPivotFails))
atomic.StoreUint32(&s.d.fsPivotFails, 1) // Don't ever reset to 0, as that will unlock the pivot block
}
}
// Put unfulfilled tasks back into the retry queue
npeers := s.d.peers.Len()
```

```
for hash, task := range req.tasks {
// If the node did deliver something, missing items may be due to a protocol
// limit or a previous timeout + delayed delivery. Both cases should permit
// the node to retry the missing items (to avoid single-peer stalls).
if len(req.response) > 0 || req.timedOut() {
delete(task.attempts, req.peer.id)
}
// If we've requested the node too many times already, it may be a malicious
// sync where nobody has the right data. Abort.
if len(task.attempts) >= npeers {
return stale, fmt.Errorf("state node %s failed with all peers (%d tries, %d peers)",
hash.TerminalString(), len(task.attempts), npeers)
}
// Missing item, place into the retry queue.
s.tasks[hash] = task
}
return stale, nil
}

// processNodeData tries to inject a trie node data blob delivered from a remote
// peer into the state trie, returning whether anything useful was written or any
// error occurred.
func (s *stateSync) processNodeData(blob []byte) (bool, common.Hash, error) {
res := trie.SyncResult{Data: blob}

s.keccak.Reset()
s.keccak.Write(blob)
s.keccak.Sum(res.Hash[:0])

committed, _, err := s.sched.Process([]trie.SyncResult{res})
return committed, res.Hash, err
}

// updateStats bumps the various state sync progress counters and displays a log
// message for the user to see.
func (s *stateSync) updateStats(processed, written, duplicate, unexpected int, duration
time.Duration) {
s.d.syncStatsLock.Lock()
defer s.d.syncStatsLock.Unlock()

s.d.syncStatsState.pending = uint64(s.sched.Pending())
```

s.d.syncStatsState.processed += uint64(processed)
s.d.syncStatsState.duplicate += uint64(duplicate)
s.d.syncStatsState.unexpected += uint64(unexpected)

log.Info("Imported new state entries", "count", processed, "flushed", written, "elapsed",
common.PrettyDuration(duration), "processed", s.d.syncStatsState.processed, "pending",
s.d.syncStatsState.pending, "retry", len(s.tasks), "duplicate", s.d.syncStatsState.duplicate,
"unexpected", s.d.syncStatsState.unexpected)
}

7:F:\git\coin\ethereum\go-ethereum\eth\downloader\types.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package downloader

import (
"fmt"
"math/big"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
)

// headerCheckFn is a callback type for verifying a header's presence in the local chain.
type headerCheckFn func(common.Hash) bool

// blockAndStateCheckFn is a callback type for verifying block and associated states' presence in the local chain.
type blockAndStateCheckFn func(common.Hash) bool

// headerRetrievalFn is a callback type for retrieving a header from the local chain.
type headerRetrievalFn func(common.Hash) *types.Header

// blockRetrievalFn is a callback type for retrieving a block from the local chain.
type blockRetrievalFn func(common.Hash) *types.Block

// headHeaderRetrievalFn is a callback type for retrieving the head header from the local chain.
type headHeaderRetrievalFn func() *types.Header

// headBlockRetrievalFn is a callback type for retrieving the head block from the local chain.
type headBlockRetrievalFn func() *types.Block

```
// headFastBlockRetrievalFn is a callback type for retrieving the head fast block from the local chain.
type headFastBlockRetrievalFn func() *types.Block

// headBlockCommitterFn is a callback for directly committing the head block to a certain entity.
type headBlockCommitterFn func(common.Hash) error

// tdRetrievalFn is a callback type for retrieving the total difficulty of a local block.
type tdRetrievalFn func(common.Hash) *big.Int

// headerChainInsertFn is a callback type to insert a batch of headers into the local chain.
type headerChainInsertFn func([]*types.Header, int) (int, error)

// blockChainInsertFn is a callback type to insert a batch of blocks into the local chain.
type blockChainInsertFn func(types.Blocks) (int, error)

// receiptChainInsertFn is a callback type to insert a batch of receipts into the local chain.
type receiptChainInsertFn func(types.Blocks, []types.Receipts) (int, error)

// chainRollbackFn is a callback type to remove a few recently added elements from the local chain.
type chainRollbackFn func([]common.Hash)

// peerDropFn is a callback type for dropping a peer detected as malicious.
type peerDropFn func(id string)

// dataPack is a data message returned by a peer for some query.
type dataPack interface {
PeerId() string
Items() int
Stats() string
}

// headerPack is a batch of block headers returned by a peer.
type headerPack struct {
peerId  string
headers []*types.Header
}

func (p *headerPack) PeerId() string { return p.peerId }
func (p *headerPack) Items() int     { return len(p.headers) }
func (p *headerPack) Stats() string  { return fmt.Sprintf("%d", len(p.headers)) }
```

```
// bodyPack is a batch of block bodies returned by a peer.
type bodyPack struct {
peerId       string
transactions [][]*types.Transaction
uncles       [][]*types.Header
}

func (p *bodyPack) PeerId() string { return p.peerId }
func (p *bodyPack) Items() int {
if len(p.transactions) <= len(p.uncles) {
return len(p.transactions)
}
return len(p.uncles)
}
func (p *bodyPack) Stats() string { return fmt.Sprintf("%d:%d", len(p.transactions), len(p.uncles)) }

// receiptPack is a batch of receipts returned by a peer.
type receiptPack struct {
peerId   string
receipts [][]*types.Receipt
}

func (p *receiptPack) PeerId() string { return p.peerId }
func (p *receiptPack) Items() int     { return len(p.receipts) }
func (p *receiptPack) Stats() string  { return fmt.Sprintf("%d", len(p.receipts)) }

// statePack is a batch of states returned by a peer.
type statePack struct {
peerId string
states [][]byte
}

func (p *statePack) PeerId() string { return p.peerId }
func (p *statePack) Items() int     { return len(p.states) }
func (p *statePack) Stats() string  { return fmt.Sprintf("%d", len(p.states)) }

8:F:\git\coin\ethereum\go-ethereum\eth\fetcher\fetcher.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package fetcher contains the block announcement based synchronisation.
package fetcher
```

```
import (
"errors"
"math/rand"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/log"
"gopkg.in/karalabe/cookiejar.v2/collections/prque"
)

const (
arriveTimeout = 500 * time.Millisecond // Time allowance before an announced block is explicitly requested
gatherSlack   = 100 * time.Millisecond // Interval used to collate almost-expired announces with fetches
fetchTimeout  = 5 * time.Second        // Maximum allotted time to return an explicitly requested block
maxUncleDist  = 7                      // Maximum allowed backward distance from the chain head
maxQueueDist  = 32                     // Maximum allowed distance from the chain head to queue
hashLimit     = 256                    // Maximum number of unique blocks a peer may have announced
blockLimit    = 64                     // Maximum number of unique blocks a peer may have delivered
)

var (
errTerminated = errors.New("terminated")
)

// blockRetrievalFn is a callback type for retrieving a block from the local chain.
type blockRetrievalFn func(common.Hash) *types.Block

// headerRequesterFn is a callback type for sending a header retrieval request.
type headerRequesterFn func(common.Hash) error

// bodyRequesterFn is a callback type for sending a body retrieval request.
type bodyRequesterFn func([]common.Hash) error

// headerVerifierFn is a callback type to verify a block's header for fast propagation.
type headerVerifierFn func(header *types.Header) error
```

```
// blockBroadcasterFn is a callback type for broadcasting a block to connected peers.
type blockBroadcasterFn func(block *types.Block, propagate bool)

// chainHeightFn is a callback type to retrieve the current chain height.
type chainHeightFn func() uint64

// chainInsertFn is a callback type to insert a batch of blocks into the local chain.
type chainInsertFn func(types.Blocks) (int, error)

// peerDropFn is a callback type for dropping a peer detected as malicious.
type peerDropFn func(id string)

// announce is the hash notification of the availability of a new block in the
// network.
type announce struct {
hash   common.Hash   // Hash of the block being announced
number uint64        // Number of the block being announced (0 = unknown | old protocol)
header *types.Header // Header of the block partially reassembled (new protocol)
time   time.Time     // Timestamp of the announcement

origin string // Identifier of the peer originating the notification

fetchHeader headerRequesterFn // Fetcher function to retrieve the header of an announced block
fetchBodies bodyRequesterFn   // Fetcher function to retrieve the body of an announced block
}

// headerFilterTask represents a batch of headers needing fetcher filtering.
type headerFilterTask struct {
headers []*types.Header // Collection of headers to filter
time    time.Time       // Arrival time of the headers
}

// headerFilterTask represents a batch of block bodies (transactions and uncles)
// needing fetcher filtering.
type bodyFilterTask struct {
transactions [][]*types.Transaction // Collection of transactions per block bodies
uncles       [][]*types.Header      // Collection of uncles per block bodies
time         time.Time              // Arrival time of the blocks' contents
}

// inject represents a schedules import operation.
type inject struct {
```

```
origin string
block  *types.Block
}

// Fetcher is responsible for accumulating block announcements from various peers
// and scheduling them for retrieval.
type Fetcher struct {
// Various event channels
notify chan *announce
inject chan *inject

blockFilter  chan chan []*types.Block
headerFilter chan chan *headerFilterTask
bodyFilter   chan chan *bodyFilterTask

done chan common.Hash
quit chan struct{}

// Announce states
announces  map[string]int              // Per peer announce counts to prevent memory exhaustion
announced  map[common.Hash][]*announce // Announced blocks, scheduled for fetching
fetching   map[common.Hash]*announce   // Announced blocks, currently fetching
fetched    map[common.Hash][]*announce // Blocks with headers fetched, scheduled for body retrieval
completing map[common.Hash]*announce   // Blocks with headers, currently body-completing

// Block cache
queue  *prque.Prque            // Queue containing the import operations (block number sorted)
queues map[string]int          // Per peer block counts to prevent memory exhaustion
queued map[common.Hash]*inject // Set of already queued blocks (to dedup imports)

// Callbacks
getBlock       blockRetrievalFn   // Retrieves a block from the local chain
verifyHeader   headerVerifierFn   // Checks if a block's headers have a valid proof of work
broadcastBlock blockBroadcasterFn // Broadcasts a block to connected peers
chainHeight    chainHeightFn      // Retrieves the current chain's height
insertChain    chainInsertFn      // Injects a batch of blocks into the chain
dropPeer       peerDropFn         // Drops a peer for misbehaving

// Testing hooks
announceChangeHook func(common.Hash, bool) // Method to call upon adding or deleting a hash from the announce list
```

```
queueChangeHook    func(common.Hash, bool) // Method to call upon adding or deleting a block from the import queue
fetchingHook       func([]common.Hash)     // Method to call upon starting a block (eth/61) or header (eth/62) fetch
completingHook     func([]common.Hash)     // Method to call upon starting a block body fetch (eth/62)
importedHook       func(*types.Block)      // Method to call upon successful block import (both eth/61 and eth/62)
}

// New creates a block fetcher to retrieve blocks based on hash announcements.
func New(getBlock blockRetrievalFn, verifyHeader headerVerifierFn, broadcastBlock blockBroadcasterFn, chainHeight chainHeightFn, insertChain chainInsertFn, dropPeer peerDropFn) *Fetcher {
return &Fetcher{
notify:         make(chan *announce),
inject:         make(chan *inject),
blockFilter:    make(chan chan []*types.Block),
headerFilter:   make(chan chan *headerFilterTask),
bodyFilter:     make(chan chan *bodyFilterTask),
done:           make(chan common.Hash),
quit:           make(chan struct{}),
announces:      make(map[string]int),
announced:      make(map[common.Hash][]*announce),
fetching:       make(map[common.Hash]*announce),
fetched:        make(map[common.Hash][]*announce),
completing:     make(map[common.Hash]*announce),
queue:          prque.New(),
queues:         make(map[string]int),
queued:         make(map[common.Hash]*inject),
getBlock:       getBlock,
verifyHeader:   verifyHeader,
broadcastBlock: broadcastBlock,
chainHeight:    chainHeight,
insertChain:    insertChain,
dropPeer:       dropPeer,
}
}

// Start boots up the announcement based synchroniser, accepting and processing
// hash notifications and block fetches until termination requested.
func (f *Fetcher) Start() {
```

```
go f.loop()
}

// Stop terminates the announcement based synchroniser, canceling all pending
// operations.
func (f *Fetcher) Stop() {
close(f.quit)
}

// Notify announces the fetcher of the potential availability of a new block in
// the network.
func (f *Fetcher) Notify(peer string, hash common.Hash, number uint64, time time.Time,
headerFetcher headerRequesterFn, bodyFetcher bodyRequesterFn) error {
block := &announce{
hash:        hash,
number:      number,
time:        time,
origin:      peer,
fetchHeader: headerFetcher,
fetchBodies: bodyFetcher,
}
select {
case f.notify <- block:
return nil
case <-f.quit:
return errTerminated
}
}

// Enqueue tries to fill gaps the the fetcher's future import queue.
func (f *Fetcher) Enqueue(peer string, block *types.Block) error {
op := &inject{
origin: peer,
block:  block,
}
select {
case f.inject <- op:
return nil
case <-f.quit:
return errTerminated
}
}
```

```
// FilterHeaders extracts all the headers that were explicitly requested by the fetcher,
// returning those that should be handled differently.
func (f *Fetcher) FilterHeaders(headers []*types.Header, time time.Time) []*types.Header {
log.Trace("Filtering headers", "headers", len(headers))

// Send the filter channel to the fetcher
filter := make(chan *headerFilterTask)

select {
case f.headerFilter <- filter:
case <-f.quit:
return nil
}
// Request the filtering of the header list
select {
case filter <- &headerFilterTask{headers: headers, time: time}:
case <-f.quit:
return nil
}
// Retrieve the headers remaining after filtering
select {
case task := <-filter:
return task.headers
case <-f.quit:
return nil
}
}

// FilterBodies extracts all the block bodies that were explicitly requested by
// the fetcher, returning those that should be handled differently.
func (f *Fetcher) FilterBodies(transactions [][]*types.Transaction, uncles [][]*types.Header, time
time.Time) ([][]*types.Transaction, [][]*types.Header) {
log.Trace("Filtering bodies", "txs", len(transactions), "uncles", len(uncles))

// Send the filter channel to the fetcher
filter := make(chan *bodyFilterTask)

select {
case f.bodyFilter <- filter:
case <-f.quit:
return nil, nil
```

```
}
// Request the filtering of the body list
select {
case filter <- &bodyFilterTask{transactions: transactions, uncles: uncles, time: time}:
case <-f.quit:
return nil, nil
}
// Retrieve the bodies remaining after filtering
select {
case task := <-filter:
return task.transactions, task.uncles
case <-f.quit:
return nil, nil
}
}

// Loop is the main fetcher loop, checking and processing various notification
// events.
func (f *Fetcher) loop() {
// Iterate the block fetching until a quit is requested
fetchTimer := time.NewTimer(0)
completeTimer := time.NewTimer(0)

for {
// Clean up any expired block fetches
for hash, announce := range f.fetching {
if time.Since(announce.time) > fetchTimeout {
f.forgetHash(hash)
}
}
// Import any queued blocks that could potentially fit
height := f.chainHeight()
for !f.queue.Empty() {
op := f.queue.PopItem().(*inject)
if f.queueChangeHook != nil {
f.queueChangeHook(op.block.Hash(), false)
}
// If too high up the chain or phase, continue later
number := op.block.NumberU64()
if number > height+1 {
f.queue.Push(op, -float32(op.block.NumberU64()))
if f.queueChangeHook != nil {
```

```
f.queueChangeHook(op.block.Hash(), true)
}
break
}
// Otherwise if fresh and still unknown, try and import
hash := op.block.Hash()
if number+maxUncleDist < height || f.getBlock(hash) != nil {
f.forgetBlock(hash)
continue
}
f.insert(op.origin, op.block)
}
// Wait for an outside event to occur
select {
case <-f.quit:
// Fetcher terminating, abort all operations
return

case notification := <-f.notify:
// A block was announced, make sure the peer isn't DOSing us
propAnnounceInMeter.Mark(1)

count := f.announces[notification.origin] + 1
if count > hashLimit {
log.Debug("Peer exceeded outstanding announces", "peer", notification.origin, "limit", hashLimit)
propAnnounceDOSMeter.Mark(1)
break
}
// If we have a valid block number, check that it's potentially useful
if notification.number > 0 {
if dist := int64(notification.number) - int64(f.chainHeight()); dist < -maxUncleDist || dist >
maxQueueDist {
log.Debug("Peer discarded announcement", "peer", notification.origin, "number",
notification.number, "hash", notification.hash, "distance", dist)
propAnnounceDropMeter.Mark(1)
break
}
}
// All is well, schedule the announce if block's not yet downloading
if _, ok := f.fetching[notification.hash]; ok {
break
}
```

```
if _, ok := f.completing[notification.hash]; ok {
break
}
f.announces[notification.origin] = count
f.announced[notification.hash] = append(f.announced[notification.hash], notification)
if f.announceChangeHook != nil && len(f.announced[notification.hash]) == 1 {
f.announceChangeHook(notification.hash, true)
}
if len(f.announced) == 1 {
f.rescheduleFetch(fetchTimer)
}

case op := <-f.inject:
// A direct block insertion was requested, try and fill any pending gaps
propBroadcastInMeter.Mark(1)
f.enqueue(op.origin, op.block)

case hash := <-f.done:
// A pending import finished, remove all traces of the notification
f.forgetHash(hash)
f.forgetBlock(hash)

case <-fetchTimer.C:
// At least one block's timer ran out, check for needing retrieval
request := make(map[string][]common.Hash)

for hash, announces := range f.announced {
if time.Since(announces[0].time) > arriveTimeout-gatherSlack {
// Pick a random peer to retrieve from, reset all others
announce := announces[rand.Intn(len(announces))]
f.forgetHash(hash)

// If the block still didn't arrive, queue for fetching
if f.getBlock(hash) == nil {
request[announce.origin] = append(request[announce.origin], hash)
f.fetching[hash] = announce
}
}
}
// Send out all block header requests
for peer, hashes := range request {
log.Trace("Fetching scheduled headers", "peer", peer, "list", hashes)
```

```
// Create a closure of the fetch and schedule in on a new thread
fetchHeader, hashes := f.fetching[hashes[0]].fetchHeader, hashes
go func() {
if f.fetchingHook != nil {
f.fetchingHook(hashes)
}
for _, hash := range hashes {
headerFetchMeter.Mark(1)
fetchHeader(hash) // Suboptimal, but protocol doesn't allow batch header retrievals
}
}()
}
// Schedule the next fetch if blocks are still pending
f.rescheduleFetch(fetchTimer)

case <-completeTimer.C:
// At least one header's timer ran out, retrieve everything
request := make(map[string][]common.Hash)

for hash, announces := range f.fetched {
// Pick a random peer to retrieve from, reset all others
announce := announces[rand.Intn(len(announces))]
f.forgetHash(hash)

// If the block still didn't arrive, queue for completion
if f.getBlock(hash) == nil {
request[announce.origin] = append(request[announce.origin], hash)
f.completing[hash] = announce
}
}
// Send out all block body requests
for peer, hashes := range request {
log.Trace("Fetching scheduled bodies", "peer", peer, "list", hashes)

// Create a closure of the fetch and schedule in on a new thread
if f.completingHook != nil {
f.completingHook(hashes)
}
bodyFetchMeter.Mark(int64(len(hashes)))
go f.completing[hashes[0]].fetchBodies(hashes)
}
```

```
// Schedule the next fetch if blocks are still pending
f.rescheduleComplete(completeTimer)

case filter := <-f.headerFilter:
// Headers arrived from a remote peer. Extract those that were explicitly
// requested by the fetcher, and return everything else so it's delivered
// to other parts of the system.
var task *headerFilterTask
select {
case task = <-filter:
case <-f.quit:
return
}
headerFilterInMeter.Mark(int64(len(task.headers)))

// Split the batch of headers into unknown ones (to return to the caller),
// known incomplete ones (requiring body retrievals) and completed blocks.
unknown, incomplete, complete := []*types.Header{}, []*announce{}, []*types.Block{}
for _, header := range task.headers {
hash := header.Hash()

// Filter fetcher-requested headers from other synchronisation algorithms
if announce := f.fetching[hash]; announce != nil && f.fetched[hash] == nil && f.completing[hash] ==
nil && f.queued[hash] == nil {
// If the delivered header does not match the promised number, drop the announcer
if header.Number.Uint64() != announce.number {
log.Trace("Invalid block number fetched", "peer", announce.origin, "hash", header.Hash(),
"announced", announce.number, "provided", header.Number)
f.dropPeer(announce.origin)
f.forgetHash(hash)
continue
}
// Only keep if not imported by other means
if f.getBlock(hash) == nil {
announce.header = header
announce.time = task.time

// If the block is empty (header only), short circuit into the final import queue
if header.TxHash == types.DeriveSha(types.Transactions{}) && header.UncleHash ==
types.CalcUncleHash([]*types.Header{}) {
log.Trace("Block empty, skipping body retrieval", "peer", announce.origin, "number",
header.Number, "hash", header.Hash())
```

```
block := types.NewBlockWithHeader(header)
block.ReceivedAt = task.time

complete = append(complete, block)
f.completing[hash] = announce
continue
}
// Otherwise add to the list of blocks needing completion
incomplete = append(incomplete, announce)
} else {
log.Trace("Block already imported, discarding header", "peer", announce.origin, "number",
header.Number, "hash", header.Hash())
f.forgetHash(hash)
}
} else {
// Fetcher doesn't know about it, add to the return list
unknown = append(unknown, header)
}
}
headerFilterOutMeter.Mark(int64(len(unknown)))
select {
case filter <- &headerFilterTask{headers: unknown, time: task.time}:
case <-f.quit:
return
}
// Schedule the retrieved headers for body completion
for _, announce := range incomplete {
hash := announce.header.Hash()
if _, ok := f.completing[hash]; ok {
continue
}
f.fetched[hash] = append(f.fetched[hash], announce)
if len(f.fetched) == 1 {
f.rescheduleComplete(completeTimer)
}
}
// Schedule the header-only blocks for import
for _, block := range complete {
if announce := f.completing[block.Hash()]; announce != nil {
f.enqueue(announce.origin, block)
}
```

```
}

case filter := <-f.bodyFilter:
// Block bodies arrived, extract any explicitly requested blocks, return the rest
var task *bodyFilterTask
select {
case task = <-filter:
case <-f.quit:
return
}
bodyFilterInMeter.Mark(int64(len(task.transactions)))

blocks := []*types.Block{}
for i := 0; i < len(task.transactions) && i < len(task.uncles); i++ {
// Match up a body to any possible completion request
matched := false

for hash, announce := range f.completing {
if f.queued[hash] == nil {
txnHash := types.DeriveSha(types.Transactions(task.transactions[i]))
uncleHash := types.CalcUncleHash(task.uncles[i])

if txnHash == announce.header.TxHash && uncleHash == announce.header.UncleHash {
// Mark the body matched, reassemble if still unknown
matched = true

if f.getBlock(hash) == nil {
block := types.NewBlockWithHeader(announce.header).WithBody(task.transactions[i],
task.uncles[i])
block.ReceivedAt = task.time

blocks = append(blocks, block)
} else {
f.forgetHash(hash)
}
}
}
}
if matched {
task.transactions = append(task.transactions[:i], task.transactions[i+1:]...)
task.uncles = append(task.uncles[:i], task.uncles[i+1:]...)
i--
```

```
continue
}
}

bodyFilterOutMeter.Mark(int64(len(task.transactions)))
select {
case filter <- task:
case <-f.quit:
return
}
// Schedule the retrieved blocks for ordered import
for _, block := range blocks {
if announce := f.completing[block.Hash()]; announce != nil {
f.enqueue(announce.origin, block)
}
}
}
}
}

// rescheduleFetch resets the specified fetch timer to the next announce timeout.
func (f *Fetcher) rescheduleFetch(fetch *time.Timer) {
// Short circuit if no blocks are announced
if len(f.announced) == 0 {
return
}
// Otherwise find the earliest expiring announcement
earliest := time.Now()
for _, announces := range f.announced {
if earliest.After(announces[0].time) {
earliest = announces[0].time
}
}
fetch.Reset(arriveTimeout - time.Since(earliest))
}

// rescheduleComplete resets the specified completion timer to the next fetch timeout.
func (f *Fetcher) rescheduleComplete(complete *time.Timer) {
// Short circuit if no headers are fetched
if len(f.fetched) == 0 {
return
}
```

```
// Otherwise find the earliest expiring announcement
earliest := time.Now()
for _, announces := range f.fetched {
if earliest.After(announces[0].time) {
earliest = announces[0].time
}
}
complete.Reset(gatherSlack - time.Since(earliest))
}

// enqueue schedules a new future import operation, if the block to be imported
// has not yet been seen.
func (f *Fetcher) enqueue(peer string, block *types.Block) {
hash := block.Hash()

// Ensure the peer isn't DOSing us
count := f.queues[peer] + 1
if count > blockLimit {
log.Debug("Discarded propagated block, exceeded allowance", "peer", peer, "number",
block.Number(), "hash", hash, "limit", blockLimit)
propBroadcastDOSMeter.Mark(1)
f.forgetHash(hash)
return
}
// Discard any past or too distant blocks
if dist := int64(block.NumberU64()) - int64(f.chainHeight()); dist < -maxUncleDist || dist >
maxQueueDist {
log.Debug("Discarded propagated block, too far away", "peer", peer, "number", block.Number(),
"hash", hash, "distance", dist)
propBroadcastDropMeter.Mark(1)
f.forgetHash(hash)
return
}
// Schedule the block for future importing
if _, ok := f.queued[hash]; !ok {
op := &inject{
origin: peer,
block:  block,
}
f.queues[peer] = count
f.queued[hash] = op
f.queue.Push(op, -float32(block.NumberU64()))
```

```
if f.queueChangeHook != nil {
f.queueChangeHook(op.block.Hash(), true)
}
log.Debug("Queued propagated block", "peer", peer, "number", block.Number(), "hash", hash,
"queued", f.queue.Size())
}
}

// insert spawns a new goroutine to run a block insertion into the chain. If the
// block's number is at the same height as the current import phase, if updates
// the phase states accordingly.
func (f *Fetcher) insert(peer string, block *types.Block) {
hash := block.Hash()

// Run the import on a new thread
log.Debug("Importing propagated block", "peer", peer, "number", block.Number(), "hash", hash)
go func() {
defer func() { f.done <- hash }()

// If the parent's unknown, abort insertion
parent := f.getBlock(block.ParentHash())
if parent == nil {
log.Debug("Unknown parent of propagated block", "peer", peer, "number", block.Number(), "hash",
hash, "parent", block.ParentHash())
return
}
// Quickly validate the header and propagate the block if it passes
switch err := f.verifyHeader(block.Header()); err {
case nil:
// All ok, quickly propagate to our peers
propBroadcastOutTimer.UpdateSince(block.ReceivedAt)
go f.broadcastBlock(block, true)

case consensus.ErrFutureBlock:
// Weird future block, don't fail, but neither propagate

default:
// Something went very wrong, drop the peer
log.Debug("Propagated block verification failed", "peer", peer, "number", block.Number(), "hash",
hash, "err", err)
f.dropPeer(peer)
return
```

```
}
// Run the actual import and log any issues
if _, err := f.insertChain(types.Blocks{block}); err != nil {
log.Debug("Propagated block import failed", "peer", peer, "number", block.Number(), "hash", hash,
"err", err)
return
}
// If import succeeded, broadcast the block
propAnnounceOutTimer.UpdateSince(block.ReceivedAt)
go f.broadcastBlock(block, false)

// Invoke the testing hook if needed
if f.importedHook != nil {
f.importedHook(block)
}
}()
}

// forgetHash removes all traces of a block announcement from the fetcher's
// internal state.
func (f *Fetcher) forgetHash(hash common.Hash) {
// Remove all pending announces and decrement DOS counters
for _, announce := range f.announced[hash] {
f.announces[announce.origin]--
if f.announces[announce.origin] == 0 {
delete(f.announces, announce.origin)
}
}
delete(f.announced, hash)
if f.announceChangeHook != nil {
f.announceChangeHook(hash, false)
}
// Remove any pending fetches and decrement the DOS counters
if announce := f.fetching[hash]; announce != nil {
f.announces[announce.origin]--
if f.announces[announce.origin] == 0 {
delete(f.announces, announce.origin)
}
delete(f.fetching, hash)
}

// Remove any pending completion requests and decrement the DOS counters
```

```
for _, announce := range f.fetched[hash] {
f.announces[announce.origin]--
if f.announces[announce.origin] == 0 {
delete(f.announces, announce.origin)
}
}
delete(f.fetched, hash)

// Remove any pending completions and decrement the DOS counters
if announce := f.completing[hash]; announce != nil {
f.announces[announce.origin]--
if f.announces[announce.origin] == 0 {
delete(f.announces, announce.origin)
}
delete(f.completing, hash)
}
}

// forgetBlock removes all traces of a queued block from the fetcher's internal
// state.
func (f *Fetcher) forgetBlock(hash common.Hash) {
if insert := f.queued[hash]; insert != nil {
f.queues[insert.origin]--
if f.queues[insert.origin] == 0 {
delete(f.queues, insert.origin)
}
delete(f.queued, hash)
}
}

9:F:\git\coin\ethereum\go-ethereum\eth\fetcher\fetcher_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package fetcher

import (
"errors"
"math/big"
"sync"
"sync/atomic"
"testing"
"time"
```

```
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/params"
)

var (
testdb, _    = ethdb.NewMemDatabase()
testKey, _   =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
testAddress  = crypto.PubkeyToAddress(testKey.PublicKey)
genesis      = core.GenesisBlockForTesting(testdb, testAddress, big.NewInt(1000000000))
unknownBlock = types.NewBlock(&types.Header{GasLimit: params.GenesisGasLimit}, nil, nil, nil)
)

// makeChain creates a chain of n blocks starting at and including parent.
// the returned hash chain is ordered head->parent. In addition, every 3rd block
// contains a transaction and every 5th an uncle to allow testing correct block
// reassembly.
func makeChain(n int, seed byte, parent *types.Block) ([]common.Hash,
map[common.Hash]*types.Block) {
blocks, _ := core.GenerateChain(params.TestChainConfig, parent, testdb, n, func(i int, block
*core.BlockGen) {
block.SetCoinbase(common.Address{seed})

// If the block number is multiple of 3, send a bonus transaction to the miner
if parent == genesis && i%3 == 0 {
signer := types.MakeSigner(params.TestChainConfig, block.Number())
tx, err := types.SignTx(types.NewTransaction(block.TxNonce(testAddress),
common.Address{seed}, big.NewInt(1000), new(big.Int).SetUint64(params.TxGas), nil, nil), signer,
testKey)
if err != nil {
panic(err)
}
block.AddTx(tx)
}
// If the block number is a multiple of 5, add a bonus uncle to the block
if i%5 == 0 {
```

```
block.AddUncle(&types.Header{ParentHash: block.PrevBlock(i - 1).Hash(), Number:
big.NewInt(int64(i - 1))})
}
})
hashes := make([]common.Hash, n+1)
hashes[len(hashes)-1] = parent.Hash()
blockm := make(map[common.Hash]*types.Block, n+1)
blockm[parent.Hash()] = parent
for i, b := range blocks {
hashes[len(hashes)-i-2] = b.Hash()
blockm[b.Hash()] = b
}
return hashes, blockm
}

// fetcherTester is a test simulator for mocking out local block chain.
type fetcherTester struct {
fetcher *Fetcher

hashes []common.Hash                // Hash chain belonging to the tester
blocks map[common.Hash]*types.Block // Blocks belonging to the tester
drops  map[string]bool              // Map of peers dropped by the fetcher

lock sync.RWMutex
}

// newTester creates a new fetcher test mocker.
func newTester() *fetcherTester {
tester := &fetcherTester{
hashes: []common.Hash{genesis.Hash()},
blocks: map[common.Hash]*types.Block{genesis.Hash(): genesis},
drops:  make(map[string]bool),
}
tester.fetcher = New(tester.getBlock, tester.verifyHeader, tester.broadcastBlock,
tester.chainHeight, tester.insertChain, tester.dropPeer)
tester.fetcher.Start()

return tester
}

// getBlock retrieves a block from the tester's block chain.
func (f *fetcherTester) getBlock(hash common.Hash) *types.Block {
```

```
f.lock.RLock()
defer f.lock.RUnlock()

return f.blocks[hash]
}

// verifyHeader is a nop placeholder for the block header verification.
func (f *fetcherTester) verifyHeader(header *types.Header) error {
return nil
}

// broadcastBlock is a nop placeholder for the block broadcasting.
func (f *fetcherTester) broadcastBlock(block *types.Block, propagate bool) {
}

// chainHeight retrieves the current height (block number) of the chain.
func (f *fetcherTester) chainHeight() uint64 {
f.lock.RLock()
defer f.lock.RUnlock()

return f.blocks[f.hashes[len(f.hashes)-1]].NumberU64()
}

// insertChain injects a new blocks into the simulated chain.
func (f *fetcherTester) insertChain(blocks types.Blocks) (int, error) {
f.lock.Lock()
defer f.lock.Unlock()

for i, block := range blocks {
// Make sure the parent in known
if _, ok := f.blocks[block.ParentHash()]; !ok {
return i, errors.New("unknown parent")
}
// Discard any new blocks if the same height already exists
if block.NumberU64() <= f.blocks[f.hashes[len(f.hashes)-1]].NumberU64() {
return i, nil
}
// Otherwise build our current chain
f.hashes = append(f.hashes, block.Hash())
f.blocks[block.Hash()] = block
}
return 0, nil
```

```
}

// dropPeer is an emulator for the peer removal, simply accumulating the various
// peers dropped by the fetcher.
func (f *fetcherTester) dropPeer(peer string) {
f.lock.Lock()
defer f.lock.Unlock()

f.drops[peer] = true
}

// makeHeaderFetcher retrieves a block header fetcher associated with a simulated peer.
func (f *fetcherTester) makeHeaderFetcher(blocks map[common.Hash]*types.Block, drift
time.Duration) headerRequesterFn {
closure := make(map[common.Hash]*types.Block)
for hash, block := range blocks {
closure[hash] = block
}
// Create a function that return a header from the closure
return func(hash common.Hash) error {
// Gather the blocks to return
headers := make([]*types.Header, 0, 1)
if block, ok := closure[hash]; ok {
headers = append(headers, block.Header())
}
// Return on a new thread
go f.fetcher.FilterHeaders(headers, time.Now().Add(drift))

return nil
}
}

// makeBodyFetcher retrieves a block body fetcher associated with a simulated peer.
func (f *fetcherTester) makeBodyFetcher(blocks map[common.Hash]*types.Block, drift
time.Duration) bodyRequesterFn {
closure := make(map[common.Hash]*types.Block)
for hash, block := range blocks {
closure[hash] = block
}
// Create a function that returns blocks from the closure
return func(hashes []common.Hash) error {
// Gather the block bodies to return
```

```
transactions := make([][]*types.Transaction, 0, len(hashes))
uncles := make([][]*types.Header, 0, len(hashes))

for _, hash := range hashes {
if block, ok := closure[hash]; ok {
transactions = append(transactions, block.Transactions())
uncles = append(uncles, block.Uncles())
}
}
// Return on a new thread
go f.fetcher.FilterBodies(transactions, uncles, time.Now().Add(drift))

return nil
}
}

// verifyFetchingEvent verifies that one single event arrive on an fetching channel.
func verifyFetchingEvent(t *testing.T, fetching chan []common.Hash, arrive bool) {
if arrive {
select {
case <-fetching:
case <-time.After(time.Second):
t.Fatalf("fetching timeout")
}
} else {
select {
case <-fetching:
t.Fatalf("fetching invoked")
case <-time.After(10 * time.Millisecond):
}
}
}

// verifyCompletingEvent verifies that one single event arrive on an completing channel.
func verifyCompletingEvent(t *testing.T, completing chan []common.Hash, arrive bool) {
if arrive {
select {
case <-completing:
case <-time.After(time.Second):
t.Fatalf("completing timeout")
}
} else {
```

```
select {
case <-completing:
t.Fatalf("completing invoked")
case <-time.After(10 * time.Millisecond):
}
}
}

// verifyImportEvent verifies that one single event arrive on an import channel.
func verifyImportEvent(t *testing.T, imported chan *types.Block, arrive bool) {
if arrive {
select {
case <-imported:
case <-time.After(time.Second):
t.Fatalf("import timeout")
}
} else {
select {
case <-imported:
t.Fatalf("import invoked")
case <-time.After(10 * time.Millisecond):
}
}
}

// verifyImportCount verifies that exactly count number of events arrive on an
// import hook channel.
func verifyImportCount(t *testing.T, imported chan *types.Block, count int) {
for i := 0; i < count; i++ {
select {
case <-imported:
case <-time.After(time.Second):
t.Fatalf("block %d: import timeout", i+1)
}
}
verifyImportDone(t, imported)
}

// verifyImportDone verifies that no more events are arriving on an import channel.
func verifyImportDone(t *testing.T, imported chan *types.Block) {
select {
case <-imported:
```

```
t.Fatalf("extra block imported")
case <-time.After(50 * time.Millisecond):
}
}

// Tests that a fetcher accepts block announcements and initiates retrievals for
// them, successfully importing into the local chain.
func TestSequentialAnnouncements62(t *testing.T) { testSequentialAnnouncements(t, 62) }
func TestSequentialAnnouncements63(t *testing.T) { testSequentialAnnouncements(t, 63) }
func TestSequentialAnnouncements64(t *testing.T) { testSequentialAnnouncements(t, 64) }

func testSequentialAnnouncements(t *testing.T, protocol int) {
// Create a chain of blocks to import
targetBlocks := 4 * hashLimit
hashes, blocks := makeChain(targetBlocks, 0, genesis)

tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

// Iteratively announce blocks until all are imported
imported := make(chan *types.Block)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

for i := len(hashes) - 2; i >= 0; i-- {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerFetcher, bodyFetcher)
verifyImportEvent(t, imported, true)
}
verifyImportDone(t, imported)
}

// Tests that if blocks are announced by multiple peers (or even the same buggy
// peer), they will only get downloaded at most once.
func TestConcurrentAnnouncements62(t *testing.T) { testConcurrentAnnouncements(t, 62) }
func TestConcurrentAnnouncements63(t *testing.T) { testConcurrentAnnouncements(t, 63) }
func TestConcurrentAnnouncements64(t *testing.T) { testConcurrentAnnouncements(t, 64) }

func testConcurrentAnnouncements(t *testing.T, protocol int) {
// Create a chain of blocks to import
targetBlocks := 4 * hashLimit
hashes, blocks := makeChain(targetBlocks, 0, genesis)
```

```
// Assemble a tester with a built in counter for the requests
tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

counter := uint32(0)
headerWrapper := func(hash common.Hash) error {
atomic.AddUint32(&counter, 1)
return headerFetcher(hash)
}
// Iteratively announce blocks until all are imported
imported := make(chan *types.Block)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

for i := len(hashes) - 2; i >= 0; i-- {
tester.fetcher.Notify("first", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerWrapper, bodyFetcher)
tester.fetcher.Notify("second", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-
arriveTimeout+time.Millisecond), headerWrapper, bodyFetcher)
tester.fetcher.Notify("second", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout-
time.Millisecond), headerWrapper, bodyFetcher)
verifyImportEvent(t, imported, true)
}
verifyImportDone(t, imported)

// Make sure no blocks were retrieved twice
if int(counter) != targetBlocks {
t.Fatalf("retrieval count mismatch: have %v, want %v", counter, targetBlocks)
}
}

// Tests that announcements arriving while a previous is being fetched still
// results in a valid import.
func TestOverlappingAnnouncements62(t *testing.T) { testOverlappingAnnouncements(t, 62) }
func TestOverlappingAnnouncements63(t *testing.T) { testOverlappingAnnouncements(t, 63) }
func TestOverlappingAnnouncements64(t *testing.T) { testOverlappingAnnouncements(t, 64) }

func testOverlappingAnnouncements(t *testing.T, protocol int) {
// Create a chain of blocks to import
targetBlocks := 4 * hashLimit
hashes, blocks := makeChain(targetBlocks, 0, genesis)
```

```
tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

// Iteratively announce blocks, but overlap them continuously
overlap := 16
imported := make(chan *types.Block, len(hashes)-1)
for i := 0; i < overlap; i++ {
imported <- nil
}
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

for i := len(hashes) - 2; i >= 0; i-- {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerFetcher, bodyFetcher)
select {
case <-imported:
case <-time.After(time.Second):
t.Fatalf("block %d: import timeout", len(hashes)-i)
}
}
// Wait for all the imports to complete and check count
verifyImportCount(t, imported, overlap)
}

// Tests that announces already being retrieved will not be duplicated.
func TestPendingDeduplication62(t *testing.T) { testPendingDeduplication(t, 62) }
func TestPendingDeduplication63(t *testing.T) { testPendingDeduplication(t, 63) }
func TestPendingDeduplication64(t *testing.T) { testPendingDeduplication(t, 64) }

func testPendingDeduplication(t *testing.T, protocol int) {
// Create a hash and corresponding block
hashes, blocks := makeChain(1, 0, genesis)

// Assemble a tester with a built in counter and delayed fetcher
tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

delay := 50 * time.Millisecond
counter := uint32(0)
```

```
headerWrapper := func(hash common.Hash) error {
atomic.AddUint32(&counter, 1)

// Simulate a long running fetch
go func() {
time.Sleep(delay)
headerFetcher(hash)
}()
return nil
}
// Announce the same block many times until it's fetched (wait for any pending ops)
for tester.getBlock(hashes[0]) == nil {
tester.fetcher.Notify("repeater", hashes[0], 1, time.Now().Add(-arriveTimeout), headerWrapper,
bodyFetcher)
time.Sleep(time.Millisecond)
}
time.Sleep(delay)

// Check that all blocks were imported and none fetched twice
if imported := len(tester.blocks); imported != 2 {
t.Fatalf("synchronised block mismatch: have %v, want %v", imported, 2)
}
if int(counter) != 1 {
t.Fatalf("retrieval count mismatch: have %v, want %v", counter, 1)
}
}

// Tests that announcements retrieved in a random order are cached and eventually
// imported when all the gaps are filled in.
func TestRandomArrivalImport62(t *testing.T) { testRandomArrivalImport(t, 62) }
func TestRandomArrivalImport63(t *testing.T) { testRandomArrivalImport(t, 63) }
func TestRandomArrivalImport64(t *testing.T) { testRandomArrivalImport(t, 64) }

func testRandomArrivalImport(t *testing.T, protocol int) {
// Create a chain of blocks to import, and choose one to delay
targetBlocks := maxQueueDist
hashes, blocks := makeChain(targetBlocks, 0, genesis)
skip := targetBlocks / 2

tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)
```

```
// Iteratively announce blocks, skipping one entry
imported := make(chan *types.Block, len(hashes)-1)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

for i := len(hashes) - 1; i >= 0; i-- {
if i != skip {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerFetcher, bodyFetcher)
time.Sleep(time.Millisecond)
}
}
// Finally announce the skipped entry and check full import
tester.fetcher.Notify("valid", hashes[skip], uint64(len(hashes)-skip-1), time.Now().Add(-
arriveTimeout), headerFetcher, bodyFetcher)
verifyImportCount(t, imported, len(hashes)-1)
}

// Tests that direct block enqueues (due to block propagation vs. hash announce)
// are correctly schedule, filling and import queue gaps.
func TestQueueGapFill62(t *testing.T) { testQueueGapFill(t, 62) }
func TestQueueGapFill63(t *testing.T) { testQueueGapFill(t, 63) }
func TestQueueGapFill64(t *testing.T) { testQueueGapFill(t, 64) }

func testQueueGapFill(t *testing.T, protocol int) {
// Create a chain of blocks to import, and choose one to not announce at all
targetBlocks := maxQueueDist
hashes, blocks := makeChain(targetBlocks, 0, genesis)
skip := targetBlocks / 2

tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

// Iteratively announce blocks, skipping one entry
imported := make(chan *types.Block, len(hashes)-1)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

for i := len(hashes) - 1; i >= 0; i-- {
if i != skip {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerFetcher, bodyFetcher)
```

```
time.Sleep(time.Millisecond)
}
}
// Fill the missing block directly as if propagated
tester.fetcher.Enqueue("valid", blocks[hashes[skip]])
verifyImportCount(t, imported, len(hashes)-1)
}

// Tests that blocks arriving from various sources (multiple propagations, hash
// announces, etc) do not get scheduled for import multiple times.
func TestImportDeduplication62(t *testing.T) { testImportDeduplication(t, 62) }
func TestImportDeduplication63(t *testing.T) { testImportDeduplication(t, 63) }
func TestImportDeduplication64(t *testing.T) { testImportDeduplication(t, 64) }

func testImportDeduplication(t *testing.T, protocol int) {
// Create two blocks to import (one for duplication, the other for stalling)
hashes, blocks := makeChain(2, 0, genesis)

// Create the tester and wrap the importer with a counter
tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

counter := uint32(0)
tester.fetcher.insertChain = func(blocks types.Blocks) (int, error) {
atomic.AddUint32(&counter, uint32(len(blocks)))
return tester.insertChain(blocks)
}
// Instrument the fetching and imported events
fetching := make(chan []common.Hash)
imported := make(chan *types.Block, len(hashes)-1)
tester.fetcher.fetchingHook = func(hashes []common.Hash) { fetching <- hashes }
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

// Announce the duplicating block, wait for retrieval, and also propagate directly
tester.fetcher.Notify("valid", hashes[0], 1, time.Now().Add(-arriveTimeout), headerFetcher,
bodyFetcher)
<-fetching

tester.fetcher.Enqueue("valid", blocks[hashes[0]])
tester.fetcher.Enqueue("valid", blocks[hashes[0]])
tester.fetcher.Enqueue("valid", blocks[hashes[0]])
```

```
// Fill the missing block directly as if propagated, and check import uniqueness
tester.fetcher.Enqueue("valid", blocks[hashes[1]])
verifyImportCount(t, imported, 2)

if counter != 2 {
t.Fatalf("import invocation count mismatch: have %v, want %v", counter, 2)
}
}

// Tests that blocks with numbers much lower or higher than out current head get
// discarded to prevent wasting resources on useless blocks from faulty peers.
func TestDistantPropagationDiscarding(t *testing.T) {
// Create a long chain to import and define the discard boundaries
hashes, blocks := makeChain(3*maxQueueDist, 0, genesis)
head := hashes[len(hashes)/2]

low, high := len(hashes)/2+maxUncleDist+1, len(hashes)/2-maxQueueDist-1

// Create a tester and simulate a head block being the middle of the above chain
tester := newTester()

tester.lock.Lock()
tester.hashes = []common.Hash{head}
tester.blocks = map[common.Hash]*types.Block{head: blocks[head]}
tester.lock.Unlock()

// Ensure that a block with a lower number than the threshold is discarded
tester.fetcher.Enqueue("lower", blocks[hashes[low]])
time.Sleep(10 * time.Millisecond)
if !tester.fetcher.queue.Empty() {
t.Fatalf("fetcher queued stale block")
}
// Ensure that a block with a higher number than the threshold is discarded
tester.fetcher.Enqueue("higher", blocks[hashes[high]])
time.Sleep(10 * time.Millisecond)
if !tester.fetcher.queue.Empty() {
t.Fatalf("fetcher queued future block")
}
}

// Tests that announcements with numbers much lower or higher than out current
```

```
// head get discarded to prevent wasting resources on useless blocks from faulty
// peers.
func TestDistantAnnouncementDiscarding62(t *testing.T) { testDistantAnnouncementDiscarding(t,
62) }
func TestDistantAnnouncementDiscarding63(t *testing.T) { testDistantAnnouncementDiscarding(t,
63) }
func TestDistantAnnouncementDiscarding64(t *testing.T) { testDistantAnnouncementDiscarding(t,
64) }

func testDistantAnnouncementDiscarding(t *testing.T, protocol int) {
// Create a long chain to import and define the discard boundaries
hashes, blocks := makeChain(3*maxQueueDist, 0, genesis)
head := hashes[len(hashes)/2]

low, high := len(hashes)/2+maxUncleDist+1, len(hashes)/2-maxQueueDist-1

// Create a tester and simulate a head block being the middle of the above chain
tester := newTester()

tester.lock.Lock()
tester.hashes = []common.Hash{head}
tester.blocks = map[common.Hash]*types.Block{head: blocks[head]}
tester.lock.Unlock()

headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

fetching := make(chan struct{}, 2)
tester.fetcher.fetchingHook = func(hashes []common.Hash) { fetching <- struct{}{} }

// Ensure that a block with a lower number than the threshold is discarded
tester.fetcher.Notify("lower", hashes[low], blocks[hashes[low]].NumberU64(), time.Now().Add(-
arriveTimeout), headerFetcher, bodyFetcher)
select {
case <-time.After(50 * time.Millisecond):
case <-fetching:
t.Fatalf("fetcher requested stale header")
}
// Ensure that a block with a higher number than the threshold is discarded
tester.fetcher.Notify("higher", hashes[high], blocks[hashes[high]].NumberU64(), time.Now().Add(-
arriveTimeout), headerFetcher, bodyFetcher)
select {
```

```
case <-time.After(50 * time.Millisecond):
case <-fetching:
t.Fatalf("fetcher requested future header")
}
}

// Tests that peers announcing blocks with invalid numbers (i.e. not matching
// the headers provided afterwards) get dropped as malicious.
func TestInvalidNumberAnnouncement62(t *testing.T) { testInvalidNumberAnnouncement(t, 62) }
func TestInvalidNumberAnnouncement63(t *testing.T) { testInvalidNumberAnnouncement(t, 63) }
func TestInvalidNumberAnnouncement64(t *testing.T) { testInvalidNumberAnnouncement(t, 64) }

func testInvalidNumberAnnouncement(t *testing.T, protocol int) {
// Create a single block to import and check numbers against
hashes, blocks := makeChain(1, 0, genesis)

tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

imported := make(chan *types.Block)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

// Announce a block with a bad number, check for immediate drop
tester.fetcher.Notify("bad", hashes[0], 2, time.Now().Add(-arriveTimeout), headerFetcher,
bodyFetcher)
verifyImportEvent(t, imported, false)

tester.lock.RLock()
dropped := tester.drops["bad"]
tester.lock.RUnlock()

if !dropped {
t.Fatalf("peer with invalid numbered announcement not dropped")
}
// Make sure a good announcement passes without a drop
tester.fetcher.Notify("good", hashes[0], 1, time.Now().Add(-arriveTimeout), headerFetcher,
bodyFetcher)
verifyImportEvent(t, imported, true)

tester.lock.RLock()
dropped = tester.drops["good"]
```

```
tester.lock.RUnlock()

if dropped {
t.Fatalf("peer with valid numbered announcement dropped")
}
verifyImportDone(t, imported)
}

// Tests that if a block is empty (i.e. header only), no body request should be
// made, and instead the header should be assembled into a whole block in itself.
func TestEmptyBlockShortCircuit62(t *testing.T) { testEmptyBlockShortCircuit(t, 62) }
func TestEmptyBlockShortCircuit63(t *testing.T) { testEmptyBlockShortCircuit(t, 63) }
func TestEmptyBlockShortCircuit64(t *testing.T) { testEmptyBlockShortCircuit(t, 64) }

func testEmptyBlockShortCircuit(t *testing.T, protocol int) {
// Create a chain of blocks to import
hashes, blocks := makeChain(32, 0, genesis)

tester := newTester()
headerFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
bodyFetcher := tester.makeBodyFetcher(blocks, 0)

// Add a monitoring hook for all internal events
fetching := make(chan []common.Hash)
tester.fetcher.fetchingHook = func(hashes []common.Hash) { fetching <- hashes }

completing := make(chan []common.Hash)
tester.fetcher.completingHook = func(hashes []common.Hash) { completing <- hashes }

imported := make(chan *types.Block)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }

// Iteratively announce blocks until all are imported
for i := len(hashes) - 2; i >= 0; i-- {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
headerFetcher, bodyFetcher)

// All announces should fetch the header
verifyFetchingEvent(t, fetching, true)

// Only blocks with data contents should request bodies
verifyCompletingEvent(t, completing, len(blocks[hashes[i]].Transactions()) > 0 ||
```

```
len(blocks[hashes[i]].Uncles()) > 0)

// Irrelevant of the construct, import should succeed
verifyImportEvent(t, imported, true)
}
verifyImportDone(t, imported)
}

// Tests that a peer is unable to use unbounded memory with sending infinite
// block announcements to a node, but that even in the face of such an attack,
// the fetcher remains operational.
func TestHashMemoryExhaustionAttack62(t *testing.T) { testHashMemoryExhaustionAttack(t, 62)
}
func TestHashMemoryExhaustionAttack63(t *testing.T) { testHashMemoryExhaustionAttack(t, 63)
}
func TestHashMemoryExhaustionAttack64(t *testing.T) { testHashMemoryExhaustionAttack(t, 64)
}

func testHashMemoryExhaustionAttack(t *testing.T, protocol int) {
// Create a tester with instrumented import hooks
tester := newTester()

imported, announces := make(chan *types.Block), int32(0)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }
tester.fetcher.announceChangeHook = func(hash common.Hash, added bool) {
if added {
atomic.AddInt32(&announces, 1)
} else {
atomic.AddInt32(&announces, -1)
}
}
// Create a valid chain and an infinite junk chain
targetBlocks := hashLimit + 2*maxQueueDist
hashes, blocks := makeChain(targetBlocks, 0, genesis)
validHeaderFetcher := tester.makeHeaderFetcher(blocks, -gatherSlack)
validBodyFetcher := tester.makeBodyFetcher(blocks, 0)

attack, _ := makeChain(targetBlocks, 0, unknownBlock)
attackerHeaderFetcher := tester.makeHeaderFetcher(nil, -gatherSlack)
attackerBodyFetcher := tester.makeBodyFetcher(nil, 0)

// Feed the tester a huge hashset from the attacker, and a limited from the valid peer
```

```
for i := 0; i < len(attack); i++ {
if i < maxQueueDist {
tester.fetcher.Notify("valid", hashes[len(hashes)-2-i], uint64(i+1), time.Now(), validHeaderFetcher,
validBodyFetcher)
}
tester.fetcher.Notify("attacker", attack[i], 1 /* don't distance drop */, time.Now(),
attackerHeaderFetcher, attackerBodyFetcher)
}
if count := atomic.LoadInt32(&announces); count != hashLimit+maxQueueDist {
t.Fatalf("queued announce count mismatch: have %d, want %d", count,
hashLimit+maxQueueDist)
}
// Wait for fetches to complete
verifyImportCount(t, imported, maxQueueDist)

// Feed the remaining valid hashes to ensure DOS protection state remains clean
for i := len(hashes) - maxQueueDist - 2; i >= 0; i-- {
tester.fetcher.Notify("valid", hashes[i], uint64(len(hashes)-i-1), time.Now().Add(-arriveTimeout),
validHeaderFetcher, validBodyFetcher)
verifyImportEvent(t, imported, true)
}
verifyImportDone(t, imported)
}

// Tests that blocks sent to the fetcher (either through propagation or via hash
// announces and retrievals) don't pile up indefinitely, exhausting available
// system memory.
func TestBlockMemoryExhaustionAttack(t *testing.T) {
// Create a tester with instrumented import hooks
tester := newTester()

imported, enqueued := make(chan *types.Block), int32(0)
tester.fetcher.importedHook = func(block *types.Block) { imported <- block }
tester.fetcher.queueChangeHook = func(hash common.Hash, added bool) {
if added {
atomic.AddInt32(&enqueued, 1)
} else {
atomic.AddInt32(&enqueued, -1)
}
}
// Create a valid chain and a batch of dangling (but in range) blocks
targetBlocks := hashLimit + 2*maxQueueDist
```

```
hashes, blocks := makeChain(targetBlocks, 0, genesis)
attack := make(map[common.Hash]*types.Block)
for i := byte(0); len(attack) < blockLimit+2*maxQueueDist; i++ {
hashes, blocks := makeChain(maxQueueDist-1, i, unknownBlock)
for _, hash := range hashes[:maxQueueDist-2] {
attack[hash] = blocks[hash]
}
}
// Try to feed all the attacker blocks make sure only a limited batch is accepted
for _, block := range attack {
tester.fetcher.Enqueue("attacker", block)
}
time.Sleep(200 * time.Millisecond)
if queued := atomic.LoadInt32(&enqueued); queued != blockLimit {
t.Fatalf("queued block count mismatch: have %d, want %d", queued, blockLimit)
}
// Queue up a batch of valid blocks, and check that a new peer is allowed to do so
for i := 0; i < maxQueueDist-1; i++ {
tester.fetcher.Enqueue("valid", blocks[hashes[len(hashes)-3-i]])
}
time.Sleep(100 * time.Millisecond)
if queued := atomic.LoadInt32(&enqueued); queued != blockLimit+maxQueueDist-1 {
t.Fatalf("queued block count mismatch: have %d, want %d", queued, blockLimit+maxQueueDist-1)
}
// Insert the missing piece (and sanity check the import)
tester.fetcher.Enqueue("valid", blocks[hashes[len(hashes)-2]])
verifyImportCount(t, imported, maxQueueDist)

// Insert the remaining blocks in chunks to ensure clean DOS protection
for i := maxQueueDist; i < len(hashes)-1; i++ {
tester.fetcher.Enqueue("valid", blocks[hashes[len(hashes)-2-i]])
verifyImportEvent(t, imported, true)
}
verifyImportDone(t, imported)
}

10:F:\git\coin\ethereum\go-ethereum\eth\fetcher\metrics.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Contains the metrics collected by the fetcher.

package fetcher
```

```
import (
"github.com/ethereum/go-ethereum/metrics"
)

var (
propAnnounceInMeter   = metrics.NewMeter("eth/fetcher/prop/announces/in")
propAnnounceOutTimer  = metrics.NewTimer("eth/fetcher/prop/announces/out")
propAnnounceDropMeter = metrics.NewMeter("eth/fetcher/prop/announces/drop")
propAnnounceDOSMeter  = metrics.NewMeter("eth/fetcher/prop/announces/dos")

propBroadcastInMeter   = metrics.NewMeter("eth/fetcher/prop/broadcasts/in")
propBroadcastOutTimer  = metrics.NewTimer("eth/fetcher/prop/broadcasts/out")
propBroadcastDropMeter = metrics.NewMeter("eth/fetcher/prop/broadcasts/drop")
propBroadcastDOSMeter  = metrics.NewMeter("eth/fetcher/prop/broadcasts/dos")

headerFetchMeter = metrics.NewMeter("eth/fetcher/fetch/headers")
bodyFetchMeter   = metrics.NewMeter("eth/fetcher/fetch/bodies")

headerFilterInMeter  = metrics.NewMeter("eth/fetcher/filter/headers/in")
headerFilterOutMeter = metrics.NewMeter("eth/fetcher/filter/headers/out")
bodyFilterInMeter    = metrics.NewMeter("eth/fetcher/filter/bodies/in")
bodyFilterOutMeter   = metrics.NewMeter("eth/fetcher/filter/bodies/out")
)

11:F:\git\coin\ethereum\go-ethereum\eth\filters\api.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filters

import (
"context"
"encoding/json"
"errors"
"fmt"
"math/big"
"sync"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/hexutil"
"github.com/ethereum/go-ethereum/core/types"
```

```
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/rpc"
)

var (
deadline = 5 * time.Minute // consider a filter inactive if it has not been polled for within deadline
)

// filter is a helper struct that holds meta information over the filter type
// and associated subscription in the event system.
type filter struct {
typ      Type
deadline *time.Timer // filter is inactiv when deadline triggers
hashes   []common.Hash
crit     FilterCriteria
logs     []*types.Log
s        *Subscription // associated subscription in event system
}

// PublicFilterAPI offers support to create and manage filters. This will allow external clients to retrieve various
// information related to the Ethereum protocol such als blocks, transactions and logs.
type PublicFilterAPI struct {
backend   Backend
useMipMap bool
mux       *event.TypeMux
quit      chan struct{}
chainDb   ethdb.Database
events    *EventSystem
filtersMu sync.Mutex
filters   map[rpc.ID]*filter
}

// NewPublicFilterAPI returns a new PublicFilterAPI instance.
func NewPublicFilterAPI(backend Backend, lightMode bool) *PublicFilterAPI {
api := &PublicFilterAPI{
backend:   backend,
useMipMap: !lightMode,
mux:       backend.EventMux(),
chainDb:   backend.ChainDb(),
events:    NewEventSystem(backend.EventMux(), backend, lightMode),
```

```
filters:   make(map[rpc.ID]*filter),
}

go api.timeoutLoop()

return api
}

// timeoutLoop runs every 5 minutes and deletes filters that have not been recently used.
// Tt is started when the api is created.
func (api *PublicFilterAPI) timeoutLoop() {
ticker := time.NewTicker(5 * time.Minute)
for {
<-ticker.C
api.filtersMu.Lock()
for id, f := range api.filters {
select {
case <-f.deadline.C:
f.s.Unsubscribe()
delete(api.filters, id)
default:
continue
}
}
api.filtersMu.Unlock()
}
}

// NewPendingTransactionFilter creates a filter that fetches pending transaction hashes
// as transactions enter the pending state.
//
// It is part of the filter package because this filter can be used throug the
// `eth_getFilterChanges` polling method that is also used for log filters.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_newpendingtransactionfilter
func (api *PublicFilterAPI) NewPendingTransactionFilter() rpc.ID {
var (
pendingTxs   = make(chan common.Hash)
pendingTxSub = api.events.SubscribePendingTxEvents(pendingTxs)
)

api.filtersMu.Lock()
```

```
api.filters[pendingTxSub.ID] = &filter{typ: PendingTransactionsSubscription, deadline:
time.NewTimer(deadline), hashes: make([]common.Hash, 0), s: pendingTxSub}
api.filtersMu.Unlock()

go func() {
for {
select {
case ph := <-pendingTxs:
api.filtersMu.Lock()
if f, found := api.filters[pendingTxSub.ID]; found {
f.hashes = append(f.hashes, ph)
}
api.filtersMu.Unlock()
case <-pendingTxSub.Err():
api.filtersMu.Lock()
delete(api.filters, pendingTxSub.ID)
api.filtersMu.Unlock()
return
}
}
}()

return pendingTxSub.ID
}

// NewPendingTransactions creates a subscription that is triggered each time a transaction
// enters the transaction pool and was signed from one of the transactions this nodes manages.
func (api *PublicFilterAPI) NewPendingTransactions(ctx context.Context) (*rpc.Subscription, error)
{
notifier, supported := rpc.NotifierFromContext(ctx)
if !supported {
return &rpc.Subscription{}, rpc.ErrNotificationsUnsupported
}

rpcSub := notifier.CreateSubscription()

go func() {
txHashes := make(chan common.Hash)
pendingTxSub := api.events.SubscribePendingTxEvents(txHashes)

for {
select {
```

```
case h := <-txHashes:
notifier.Notify(rpcSub.ID, h)
case <-rpcSub.Err():
pendingTxSub.Unsubscribe()
return
case <-notifier.Closed():
pendingTxSub.Unsubscribe()
return
}
}
}()

return rpcSub, nil
}

// NewBlockFilter creates a filter that fetches blocks that are imported into the chain.
// It is part of the filter package since polling goes with eth_getFilterChanges.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_newblockfilter
func (api *PublicFilterAPI) NewBlockFilter() rpc.ID {
var (
headers   = make(chan *types.Header)
headerSub = api.events.SubscribeNewHeads(headers)
)

api.filtersMu.Lock()
api.filters[headerSub.ID] = &filter{typ: BlocksSubscription, deadline: time.NewTimer(deadline),
hashes: make([]common.Hash, 0), s: headerSub}
api.filtersMu.Unlock()

go func() {
for {
select {
case h := <-headers:
api.filtersMu.Lock()
if f, found := api.filters[headerSub.ID]; found {
f.hashes = append(f.hashes, h.Hash())
}
api.filtersMu.Unlock()
case <-headerSub.Err():
api.filtersMu.Lock()
delete(api.filters, headerSub.ID)
```

```
api.filtersMu.Unlock()
return
}
}
}()

return headerSub.ID
}

// NewHeads send a notification each time a new (header) block is appended to the chain.
func (api *PublicFilterAPI) NewHeads(ctx context.Context) (*rpc.Subscription, error) {
notifier, supported := rpc.NotifierFromContext(ctx)
if !supported {
return &rpc.Subscription{}, rpc.ErrNotificationsUnsupported
}

rpcSub := notifier.CreateSubscription()

go func() {
headers := make(chan *types.Header)
headersSub := api.events.SubscribeNewHeads(headers)

for {
select {
case h := <-headers:
notifier.Notify(rpcSub.ID, h)
case <-rpcSub.Err():
headersSub.Unsubscribe()
return
case <-notifier.Closed():
headersSub.Unsubscribe()
return
}
}
}()

return rpcSub, nil
}

// Logs creates a subscription that fires for all new log that match the given filter criteria.
func (api *PublicFilterAPI) Logs(ctx context.Context, crit FilterCriteria) (*rpc.Subscription, error) {
notifier, supported := rpc.NotifierFromContext(ctx)
```

```
if !supported {
return &rpc.Subscription{}, rpc.ErrNotificationsUnsupported
}

var (
rpcSub      = notifier.CreateSubscription()
matchedLogs = make(chan []*types.Log)
)

logsSub, err := api.events.SubscribeLogs(crit, matchedLogs)
if err != nil {
return nil, err
}

go func() {

for {
select {
case logs := <-matchedLogs:
for _, log := range logs {
notifier.Notify(rpcSub.ID, &log)
}
case <-rpcSub.Err(): // client send an unsubscribe request
logsSub.Unsubscribe()
return
case <-notifier.Closed(): // connection dropped
logsSub.Unsubscribe()
return
}
}
}()

return rpcSub, nil
}

// FilterCriteria represents a request to create a new filter.
type FilterCriteria struct {
FromBlock *big.Int
ToBlock   *big.Int
Addresses []common.Address
Topics    [][]common.Hash
}
```

```
// NewFilter creates a new filter and returns the filter id. It can be
// used to retrieve logs when the state changes. This method cannot be
// used to fetch logs that are already stored in the state.
//
// Default criteria for the from and to block are "latest".
// Using "latest" as block number will return logs for mined blocks.
// Using "pending" as block number returns logs for not yet mined (pending) blocks.
// In case logs are removed (chain reorg) previously returned logs are returned
// again but with the removed property set to true.
//
// In case "fromBlock" > "toBlock" an error is returned.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_newfilter
func (api *PublicFilterAPI) NewFilter(crit FilterCriteria) (rpc.ID, error) {
logs := make(chan []*types.Log)
logsSub, err := api.events.SubscribeLogs(crit, logs)
if err != nil {
return rpc.ID(""), err
}

api.filtersMu.Lock()
api.filters[logsSub.ID] = &filter{typ: LogsSubscription, crit: crit, deadline: time.NewTimer(deadline),
logs: make([]*types.Log, 0), s: logsSub}
api.filtersMu.Unlock()

go func() {
for {
select {
case l := <-logs:
api.filtersMu.Lock()
if f, found := api.filters[logsSub.ID]; found {
f.logs = append(f.logs, l...)
}
api.filtersMu.Unlock()
case <-logsSub.Err():
api.filtersMu.Lock()
delete(api.filters, logsSub.ID)
api.filtersMu.Unlock()
return
}
}
```

```
}()

return logsSub.ID, nil
}

// GetLogs returns logs matching the given argument that are stored within the state.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getlogs
func (api *PublicFilterAPI) GetLogs(ctx context.Context, crit FilterCriteria) ([]*types.Log, error) {
if crit.FromBlock == nil {
crit.FromBlock = big.NewInt(rpc.LatestBlockNumber.Int64())
}
if crit.ToBlock == nil {
crit.ToBlock = big.NewInt(rpc.LatestBlockNumber.Int64())
}

filter := New(api.backend, api.useMipMap)
filter.SetBeginBlock(crit.FromBlock.Int64())
filter.SetEndBlock(crit.ToBlock.Int64())
filter.SetAddresses(crit.Addresses)
filter.SetTopics(crit.Topics)

logs, err := filter.Find(ctx)
return returnLogs(logs), err
}

// UninstallFilter removes the filter with the given filter id.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_uninstallfilter
func (api *PublicFilterAPI) UninstallFilter(id rpc.ID) bool {
api.filtersMu.Lock()
f, found := api.filters[id]
if found {
delete(api.filters, id)
}
api.filtersMu.Unlock()
if found {
f.s.Unsubscribe()
}

return found
}
```

```
// GetFilterLogs returns the logs for the filter with the given id.
// If the filter could not be found an empty array of logs is returned.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getfilterlogs
func (api *PublicFilterAPI) GetFilterLogs(ctx context.Context, id rpc.ID) ([]*types.Log, error) {
api.filtersMu.Lock()
f, found := api.filters[id]
api.filtersMu.Unlock()

if !found || f.typ != LogsSubscription {
return nil, fmt.Errorf("filter not found")
}

filter := New(api.backend, api.useMipMap)
if f.crit.FromBlock != nil {
filter.SetBeginBlock(f.crit.FromBlock.Int64())
} else {
filter.SetBeginBlock(rpc.LatestBlockNumber.Int64())
}
if f.crit.ToBlock != nil {
filter.SetEndBlock(f.crit.ToBlock.Int64())
} else {
filter.SetEndBlock(rpc.LatestBlockNumber.Int64())
}
filter.SetAddresses(f.crit.Addresses)
filter.SetTopics(f.crit.Topics)

logs, err := filter.Find(ctx)
if err != nil {
return nil, err
}
return returnLogs(logs), nil
}

// GetFilterChanges returns the logs for the filter with the given id since
// last time is was called. This can be used for polling.
//
// For pending transaction and block filters the result is []common.Hash.
// (pending)Log filters return []Log.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_getfilterchanges
```

```
func (api *PublicFilterAPI) GetFilterChanges(id rpc.ID) (interface{}, error) {
api.filtersMu.Lock()
defer api.filtersMu.Unlock()

if f, found := api.filters[id]; found {
if !f.deadline.Stop() {
// timer expired but filter is not yet removed in timeout loop
// receive timer value and reset timer
<-f.deadline.C
}
f.deadline.Reset(deadline)

switch f.typ {
case PendingTransactionsSubscription, BlocksSubscription:
hashes := f.hashes
f.hashes = nil
return returnHashes(hashes), nil
case LogsSubscription:
logs := f.logs
f.logs = nil
return returnLogs(logs), nil
}
}

return []interface{}{}, fmt.Errorf("filter not found")
}

// returnHashes is a helper that will return an empty hash array case the given hash array is nil,
// otherwise the given hashes array is returned.
func returnHashes(hashes []common.Hash) []common.Hash {
if hashes == nil {
return []common.Hash{}
}
return hashes
}

// returnLogs is a helper that will return an empty log array in case the given logs array is nil,
// otherwise the given logs array is returned.
func returnLogs(logs []*types.Log) []*types.Log {
if logs == nil {
return []*types.Log{}
}
```

```
return logs
}

// UnmarshalJSON sets *args fields with given data.
func (args *FilterCriteria) UnmarshalJSON(data []byte) error {
type input struct {
From      *rpc.BlockNumber `json:"fromBlock"`
ToBlock   *rpc.BlockNumber `json:"toBlock"`
Addresses interface{}      `json:"address"`
Topics    []interface{}    `json:"topics"`
}

var raw input
if err := json.Unmarshal(data, &raw); err != nil {
return err
}

if raw.From != nil {
args.FromBlock = big.NewInt(raw.From.Int64())
}

if raw.ToBlock != nil {
args.ToBlock = big.NewInt(raw.ToBlock.Int64())
}

args.Addresses = []common.Address{}

if raw.Addresses != nil {
// raw.Address can contain a single address or an array of addresses
switch rawAddr := raw.Addresses.(type) {
case []interface{}:
for i, addr := range rawAddr {
if strAddr, ok := addr.(string); ok {
addr, err := decodeAddress(strAddr)
if err != nil {
return fmt.Errorf("invalid address at index %d: %v", i, err)
}
args.Addresses = append(args.Addresses, addr)
} else {
return fmt.Errorf("non-string address at index %d", i)
}
}
```

```
case string:
addr, err := decodeAddress(rawAddr)
if err != nil {
return fmt.Errorf("invalid address: %v", err)
}
args.Addresses = []common.Address{addr}
default:
return errors.New("invalid addresses in query")
}
}

// topics is an array consisting of strings and/or arrays of strings.
// JSON null values are converted to common.Hash{} and ignored by the filter manager.
if len(raw.Topics) > 0 {
args.Topics = make([][]common.Hash, len(raw.Topics))
for i, t := range raw.Topics {
switch topic := t.(type) {
case nil:
// ignore topic when matching logs
args.Topics[i] = []common.Hash{{}}

case string:
// match specific topic
top, err := decodeTopic(topic)
if err != nil {
return err
}
args.Topics[i] = []common.Hash{top}
case []interface{}:
// or case e.g. [null, "topic0", "topic1"]
for _, rawTopic := range topic {
if rawTopic == nil {
args.Topics[i] = append(args.Topics[i], common.Hash{})
} else if topic, ok := rawTopic.(string); ok {
parsed, err := decodeTopic(topic)
if err != nil {
return err
}
args.Topics[i] = append(args.Topics[i], parsed)
} else {
return fmt.Errorf("invalid topic(s)")
}
```

```
}
default:
return fmt.Errorf("invalid topic(s)")
}
}
}

return nil
}

func decodeAddress(s string) (common.Address, error) {
b, err := hexutil.Decode(s)
if err == nil && len(b) != common.AddressLength {
err = fmt.Errorf("hex has invalid length %d after decoding", len(b))
}
return common.BytesToAddress(b), err
}

func decodeTopic(s string) (common.Hash, error) {
b, err := hexutil.Decode(s)
if err == nil && len(b) != common.HashLength {
err = fmt.Errorf("hex has invalid length %d after decoding", len(b))
}
return common.BytesToHash(b), err
}

12:F:\git\coin\ethereum\go-ethereum\eth\filters\api_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filters

import (
"encoding/json"
"fmt"
"testing"

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

func TestUnmarshalJSONNewFilterArgs(t *testing.T) {
var (
```

```
fromBlock rpc.BlockNumber = 0x123435
toBlock   rpc.BlockNumber = 0xabcdef
address0                  =
common.StringToAddress("70c87d191324e6712a591f304b4eedef6ad9bb9d")
address1                  =
common.StringToAddress("9b2055d370f73ec7d8a03e965129118dc8f5bf83")
topic0                    =
common.HexToHash("3ac225168df54212a25c1c01fd35bebfea408fdac2e31ddd6f80a4bbf9a5f1ca
")
topic1                    =
common.HexToHash("9084a792d2f8b16a62b882fd56f7860c07bf5fa91dd8a2ae7e809e5180fef0b
3")
topic2                    =
common.HexToHash("6ccae1c4af4152f460ff510e573399795dfab5dcf1fa60d1f33ac8fdc1e480ce")
nullTopic                 = common.Hash{}
)

// default values
var test0 FilterCriteria
if err := json.Unmarshal([]byte("{}"), &test0); err != nil {
t.Fatal(err)
}
if test0.FromBlock != nil {
t.Fatalf("expected nil, got %d", test0.FromBlock)
}
if test0.ToBlock != nil {
t.Fatalf("expected nil, got %d", test0.ToBlock)
}
if len(test0.Addresses) != 0 {
t.Fatalf("expected 0 addresses, got %d", len(test0.Addresses))
}
if len(test0.Topics) != 0 {
t.Fatalf("expected 0 topics, got %d topics", len(test0.Topics))
}

// from, to block number
var test1 FilterCriteria
vector := fmt.Sprintf(`{"fromBlock":"0x%x","toBlock":"0x%x"}`, fromBlock, toBlock)
if err := json.Unmarshal([]byte(vector), &test1); err != nil {
t.Fatal(err)
}
if test1.FromBlock.Int64() != fromBlock.Int64() {
```

```
t.Fatalf("expected FromBlock %d, got %d", fromBlock, test1.FromBlock)
}
if test1.ToBlock.Int64() != toBlock.Int64() {
t.Fatalf("expected ToBlock %d, got %d", toBlock, test1.ToBlock)
}

// single address
var test2 FilterCriteria
vector = fmt.Sprintf(`{"address": "%s"}`, address0.Hex())
if err := json.Unmarshal([]byte(vector), &test2); err != nil {
t.Fatal(err)
}
if len(test2.Addresses) != 1 {
t.Fatalf("expected 1 address, got %d address(es)", len(test2.Addresses))
}
if test2.Addresses[0] != address0 {
t.Fatalf("expected address %x, got %x", address0, test2.Addresses[0])
}

// multiple address
var test3 FilterCriteria
vector = fmt.Sprintf(`{"address": ["%s", "%s"]}`, address0.Hex(), address1.Hex())
if err := json.Unmarshal([]byte(vector), &test3); err != nil {
t.Fatal(err)
}
if len(test3.Addresses) != 2 {
t.Fatalf("expected 2 addresses, got %d address(es)", len(test3.Addresses))
}
if test3.Addresses[0] != address0 {
t.Fatalf("expected address %x, got %x", address0, test3.Addresses[0])
}
if test3.Addresses[1] != address1 {
t.Fatalf("expected address %x, got %x", address1, test3.Addresses[1])
}

// single topic
var test4 FilterCriteria
vector = fmt.Sprintf(`{"topics": ["%s"]}`, topic0.Hex())
if err := json.Unmarshal([]byte(vector), &test4); err != nil {
t.Fatal(err)
}
if len(test4.Topics) != 1 {
```

```
t.Fatalf("expected 1 topic, got %d", len(test4.Topics))
}
if len(test4.Topics[0]) != 1 {
t.Fatalf("expected len(topics[0]) to be 1, got %d", len(test4.Topics[0]))
}
if test4.Topics[0][0] != topic0 {
t.Fatalf("got %x, expected %x", test4.Topics[0][0], topic0)
}

// test multiple "AND" topics
var test5 FilterCriteria
vector = fmt.Sprintf(`{"topics": ["%s", "%s"]}`, topic0.Hex(), topic1.Hex())
if err := json.Unmarshal([]byte(vector), &test5); err != nil {
t.Fatal(err)
}
if len(test5.Topics) != 2 {
t.Fatalf("expected 2 topics, got %d", len(test5.Topics))
}
if len(test5.Topics[0]) != 1 {
t.Fatalf("expected 1 topic, got %d", len(test5.Topics[0]))
}
if test5.Topics[0][0] != topic0 {
t.Fatalf("got %x, expected %x", test5.Topics[0][0], topic0)
}
if len(test5.Topics[1]) != 1 {
t.Fatalf("expected 1 topic, got %d", len(test5.Topics[1]))
}
if test5.Topics[1][0] != topic1 {
t.Fatalf("got %x, expected %x", test5.Topics[1][0], topic1)
}

// test optional topic
var test6 FilterCriteria
vector = fmt.Sprintf(`{"topics": ["%s", null, "%s"]}`, topic0.Hex(), topic2.Hex())
if err := json.Unmarshal([]byte(vector), &test6); err != nil {
t.Fatal(err)
}
if len(test6.Topics) != 3 {
t.Fatalf("expected 3 topics, got %d", len(test6.Topics))
}
if len(test6.Topics[0]) != 1 {
t.Fatalf("expected 1 topic, got %d", len(test6.Topics[0]))
```

```
}
if test6.Topics[0][0] != topic0 {
t.Fatalf("got %x, expected %x", test6.Topics[0][0], topic0)
}
if len(test6.Topics[1]) != 1 {
t.Fatalf("expected 1 topic, got %d", len(test6.Topics[1]))
}
if test6.Topics[1][0] != nullTopic {
t.Fatalf("got %x, expected empty hash", test6.Topics[1][0])
}
if len(test6.Topics[2]) != 1 {
t.Fatalf("expected 1 topic, got %d", len(test6.Topics[2]))
}
if test6.Topics[2][0] != topic2 {
t.Fatalf("got %x, expected %x", test6.Topics[2][0], topic2)
}

// test OR topics
var test7 FilterCriteria
vector = fmt.Sprintf(`{"topics": [["%s", "%s"], null, ["%s", null]]}`, topic0.Hex(), topic1.Hex(),
topic2.Hex())
if err := json.Unmarshal([]byte(vector), &test7); err != nil {
t.Fatal(err)
}
if len(test7.Topics) != 3 {
t.Fatalf("expected 3 topics, got %d topics", len(test7.Topics))
}
if len(test7.Topics[0]) != 2 {
t.Fatalf("expected 2 topics, got %d topics", len(test7.Topics[0]))
}
if test7.Topics[0][0] != topic0 || test7.Topics[0][1] != topic1 {
t.Fatalf("invalid topics expected [%x,%x], got [%x,%x]",
topic0, topic1, test7.Topics[0][0], test7.Topics[0][1],
)
}
if len(test7.Topics[1]) != 1 {
t.Fatalf("expected 1 topic, got %d topics", len(test7.Topics[1]))
}
if test7.Topics[1][0] != nullTopic {
t.Fatalf("expected empty hash, got %x", test7.Topics[1][0])
}
if len(test7.Topics[2]) != 2 {
```

```
t.Fatalf("expected 2 topics, got %d topics", len(test7.Topics[2]))
}
if test7.Topics[2][0] != topic2 || test7.Topics[2][1] != nullTopic {
t.Fatalf("invalid topics expected [%x,%x], got [%x,%x]",
topic2, nullTopic, test7.Topics[2][0], test7.Topics[2][1],
)
}
}
```

13:F:\git\coin\ethereum\go-ethereum\eth\filters\filter.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filters

import (
"context"
"math"
"math/big"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/rpc"
)

type Backend interface {
ChainDb() ethdb.Database
EventMux() *event.TypeMux
HeaderByNumber(ctx context.Context, blockNr rpc.BlockNumber) (*types.Header, error)
GetReceipts(ctx context.Context, blockHash common.Hash) (types.Receipts, error)
}

// Filter can be used to retrieve and filter logs.
type Filter struct {
backend   Backend
useMipMap bool

created time.Time
```

```
db         ethdb.Database
begin, end int64
addresses  []common.Address
topics     [][]common.Hash
}

// New creates a new filter which uses a bloom filter on blocks to figure out whether
// a particular block is interesting or not.
// MipMaps allow past blocks to be searched much more efficiently, but are not available
// to light clients.
func New(backend Backend, useMipMap bool) *Filter {
return &Filter{
backend:   backend,
useMipMap: useMipMap,
db:        backend.ChainDb(),
}
}

// SetBeginBlock sets the earliest block for filtering.
// -1 = latest block (i.e., the current block)
// hash = particular hash from-to
func (f *Filter) SetBeginBlock(begin int64) {
f.begin = begin
}

// SetEndBlock sets the latest block for filtering.
func (f *Filter) SetEndBlock(end int64) {
f.end = end
}

// SetAddresses matches only logs that are generated from addresses that are included
// in the given addresses.
func (f *Filter) SetAddresses(addr []common.Address) {
f.addresses = addr
}

// SetTopics matches only logs that have topics matching the given topics.
func (f *Filter) SetTopics(topics [][]common.Hash) {
f.topics = topics
}

// FindOnce searches the blockchain for matching log entries, returning
```

```
// all matching entries from the first block that contains matches,
// updating the start point of the filter accordingly. If no results are
// found, a nil slice is returned.
func (f *Filter) FindOnce(ctx context.Context) ([]*types.Log, error) {
head, _ := f.backend.HeaderByNumber(ctx, rpc.LatestBlockNumber)
if head == nil {
return nil, nil
}
headBlockNumber := head.Number.Uint64()

var beginBlockNo uint64 = uint64(f.begin)
if f.begin == -1 {
beginBlockNo = headBlockNumber
}
var endBlockNo uint64 = uint64(f.end)
if f.end == -1 {
endBlockNo = headBlockNumber
}

// if no addresses are present we can't make use of fast search which
// uses the mipmap bloom filters to check for fast inclusion and uses
// higher range probability in order to ensure at least a false positive
if !f.useMipMap || len(f.addresses) == 0 {
logs, blockNumber, err := f.getLogs(ctx, beginBlockNo, endBlockNo)
f.begin = int64(blockNumber + 1)
return logs, err
}

logs, blockNumber := f.mipFind(beginBlockNo, endBlockNo, 0)
f.begin = int64(blockNumber + 1)
return logs, nil
}

// Run filters logs with the current parameters set
func (f *Filter) Find(ctx context.Context) (logs []*types.Log, err error) {
for {
newLogs, err := f.FindOnce(ctx)
if len(newLogs) == 0 || err != nil {
return logs, err
}
logs = append(logs, newLogs...)
}
```

```
}

func (f *Filter) mipFind(start, end uint64, depth int) (logs []*types.Log, blockNumber uint64) {
level := core.MIPMapLevels[depth]
// normalise numerator so we can work in level specific batches and
// work with the proper range checks
for num := start / level * level; num <= end; num += level {
// find addresses in bloom filters
bloom := core.GetMipmapBloom(f.db, num, level)
// Don't bother checking the first time through the loop - we're probably picking
// up where a previous run left off.
first := true
for _, addr := range f.addresses {
if first || bloom.TestBytes(addr[:]) {
first = false
// range check normalised values and make sure that
// we're resolving the correct range instead of the
// normalised values.
start := uint64(math.Max(float64(num), float64(start)))
end := uint64(math.Min(float64(num+level-1), float64(end)))
if depth+1 == len(core.MIPMapLevels) {
l, blockNumber, _ := f.getLogs(context.Background(), start, end)
if len(l) > 0 {
return l, blockNumber
}
} else {
l, blockNumber := f.mipFind(start, end, depth+1)
if len(l) > 0 {
return l, blockNumber
}
}
}
}
}

return nil, end
}

func (f *Filter) getLogs(ctx context.Context, start, end uint64) (logs []*types.Log, blockNumber
uint64, err error) {
for i := start; i <= end; i++ {
blockNumber := rpc.BlockNumber(i)
```

```
header, err := f.backend.HeaderByNumber(ctx, blockNumber)
if header == nil || err != nil {
return logs, end, err
}

// Use bloom filtering to see if this block is interesting given the
// current parameters
if f.bloomFilter(header.Bloom) {
// Get the logs of the block
receipts, err := f.backend.GetReceipts(ctx, header.Hash())
if err != nil {
return nil, end, err
}
var unfiltered []*types.Log
for _, receipt := range receipts {
unfiltered = append(unfiltered, ([]*types.Log)(receipt.Logs)...)
}
logs = filterLogs(unfiltered, nil, nil, f.addresses, f.topics)
if len(logs) > 0 {
return logs, uint64(blockNumber), nil
}
}
}

return logs, end, nil
}

func includes(addresses []common.Address, a common.Address) bool {
for _, addr := range addresses {
if addr == a {
return true
}
}

return false
}

// filterLogs creates a slice of logs matching the given criteria.
func filterLogs(logs []*types.Log, fromBlock, toBlock *big.Int, addresses []common.Address, topics
[][]common.Hash) []*types.Log {
var ret []*types.Log
Logs:
```

```
for _, log := range logs {
if fromBlock != nil && fromBlock.Int64() >= 0 && fromBlock.Uint64() > log.BlockNumber {
continue
}
if toBlock != nil && toBlock.Int64() >= 0 && toBlock.Uint64() < log.BlockNumber {
continue
}

if len(addresses) > 0 && !includes(addresses, log.Address) {
continue
}

logTopics := make([]common.Hash, len(topics))
copy(logTopics, log.Topics)

// If the to filtered topics is greater than the amount of topics in logs, skip.
if len(topics) > len(log.Topics) {
continue Logs
}

for i, topics := range topics {
var match bool
for _, topic := range topics {
// common.Hash{} is a match all (wildcard)
if (topic == common.Hash{}) || log.Topics[i] == topic {
match = true
break
}
}

if !match {
continue Logs
}
}
ret = append(ret, log)
}

return ret
}

func (f *Filter) bloomFilter(bloom types.Bloom) bool {
return bloomFilter(bloom, f.addresses, f.topics)
```

```
}

func bloomFilter(bloom types.Bloom, addresses []common.Address, topics [][]common.Hash) bool
{
if len(addresses) > 0 {
var included bool
for _, addr := range addresses {
if types.BloomLookup(bloom, addr) {
included = true
break
}
}

if !included {
return false
}
}

for _, sub := range topics {
var included bool
for _, topic := range sub {
if (topic == common.Hash{}) || types.BloomLookup(bloom, topic) {
included = true
break
}
}
if !included {
return false
}
}

return true
}

14:F:\git\coin\ethereum\go-ethereum\eth\filters\filter_system.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package filters implements an ethereum filtering system for block,
// transactions and log events.
package filters

import (
```

```
"context"
"errors"
"fmt"
"sync"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/rpc"
)

// Type determines the kind of filter and is used to put the filter in to
// the correct bucket when added.
type Type byte

const (
// UnknownSubscription indicates an unknown subscription type
UnknownSubscription Type = iota
// LogsSubscription queries for new or removed (chain reorg) logs
LogsSubscription
// PendingLogsSubscription queries for logs in pending blocks
PendingLogsSubscription
// MinedAndPendingLogsSubscription queries for logs in mined and pending blocks.
MinedAndPendingLogsSubscription
// PendingTransactionsSubscription queries tx hashes for pending
// transactions entering the pending state
PendingTransactionsSubscription
// BlocksSubscription queries hashes for blocks that are imported
BlocksSubscription
// LastSubscription keeps track of the last index
LastIndexSubscription
)

var (
ErrInvalidSubscriptionID = errors.New("invalid id")
)

type subscription struct {
id        rpc.ID
typ       Type
```

```
created   time.Time
logsCrit  FilterCriteria
logs      chan []*types.Log
hashes    chan common.Hash
headers   chan *types.Header
installed chan struct{} // closed when the filter is installed
err       chan error    // closed when the filter is uninstalled
}

// EventSystem creates subscriptions, processes events and broadcasts them to the
// subscription which match the subscription criteria.
type EventSystem struct {
mux       *event.TypeMux
sub       *event.TypeMuxSubscription
backend   Backend
lightMode bool
lastHead  *types.Header
install   chan *subscription // install filter for event notification
uninstall chan *subscription // remove filter for event notification
}

// NewEventSystem creates a new manager that listens for event on the given mux,
// parses and filters them. It uses the all map to retrieve filter changes. The
// work loop holds its own index that is used to forward events to filters.
//
// The returned manager has a loop that needs to be stopped with the Stop function
// or by stopping the given mux.
func NewEventSystem(mux *event.TypeMux, backend Backend, lightMode bool) *EventSystem {
m := &EventSystem{
mux:       mux,
backend:   backend,
lightMode: lightMode,
install:   make(chan *subscription),
uninstall: make(chan *subscription),
}

go m.eventLoop()

return m
}

// Subscription is created when the client registers itself for a particular event.
```

```
type Subscription struct {
ID        rpc.ID
f         *subscription
es        *EventSystem
unsubOnce sync.Once
}

// Err returns a channel that is closed when unsubscribed.
func (sub *Subscription) Err() <-chan error {
return sub.f.err
}

// Unsubscribe uninstalls the subscription from the event broadcast loop.
func (sub *Subscription) Unsubscribe() {
sub.unsubOnce.Do(func() {
uninstallLoop:
for {
// write uninstall request and consume logs/hashes. This prevents
// the eventLoop broadcast method to deadlock when writing to the
// filter event channel while the subscription loop is waiting for
// this method to return (and thus not reading these events).
select {
case sub.es.uninstall <- sub.f:
break uninstallLoop
case <-sub.f.logs:
case <-sub.f.hashes:
case <-sub.f.headers:
}
}

// wait for filter to be uninstalled in work loop before returning
// this ensures that the manager won't use the event channel which
// will probably be closed by the client asap after this method returns.
<-sub.Err()
})
}

// subscribe installs the subscription in the event broadcast loop.
func (es *EventSystem) subscribe(sub *subscription) *Subscription {
es.install <- sub
<-sub.installed
return &Subscription{ID: sub.id, f: sub, es: es}
```

```
}

// SubscribeLogs creates a subscription that will write all logs matching the
// given criteria to the given logs channel. Default value for the from and to
// block is "latest". If the fromBlock > toBlock an error is returned.
func (es *EventSystem) SubscribeLogs(crit FilterCriteria, logs chan []*types.Log) (*Subscription,
error) {
var from, to rpc.BlockNumber
if crit.FromBlock == nil {
from = rpc.LatestBlockNumber
} else {
from = rpc.BlockNumber(crit.FromBlock.Int64())
}
if crit.ToBlock == nil {
to = rpc.LatestBlockNumber
} else {
to = rpc.BlockNumber(crit.ToBlock.Int64())
}

// only interested in pending logs
if from == rpc.PendingBlockNumber && to == rpc.PendingBlockNumber {
return es.subscribePendingLogs(crit, logs), nil
}
// only interested in new mined logs
if from == rpc.LatestBlockNumber && to == rpc.LatestBlockNumber {
return es.subscribeLogs(crit, logs), nil
}
// only interested in mined logs within a specific block range
if from >= 0 && to >= 0 && to >= from {
return es.subscribeLogs(crit, logs), nil
}
// interested in mined logs from a specific block number, new logs and pending logs
if from >= rpc.LatestBlockNumber && to == rpc.PendingBlockNumber {
return es.subscribeMinedPendingLogs(crit, logs), nil
}
// interested in logs from a specific block number to new mined blocks
if from >= 0 && to == rpc.LatestBlockNumber {
return es.subscribeLogs(crit, logs), nil
}
return nil, fmt.Errorf("invalid from and to block combination: from > to")
}
```

```
// subscribeMinedPendingLogs creates a subscription that returned mined and
// pending logs that match the given criteria.
func (es *EventSystem) subscribeMinedPendingLogs(crit FilterCriteria, logs chan []*types.Log)
*Subscription {
sub := &subscription{
id:        rpc.NewID(),
typ:       MinedAndPendingLogsSubscription,
logsCrit:  crit,
created:   time.Now(),
logs:      logs,
hashes:    make(chan common.Hash),
headers:   make(chan *types.Header),
installed: make(chan struct{}),
err:       make(chan error),
}

return es.subscribe(sub)
}

// subscribeLogs creates a subscription that will write all logs matching the
// given criteria to the given logs channel.
func (es *EventSystem) subscribeLogs(crit FilterCriteria, logs chan []*types.Log) *Subscription {
sub := &subscription{
id:        rpc.NewID(),
typ:       LogsSubscription,
logsCrit:  crit,
created:   time.Now(),
logs:      logs,
hashes:    make(chan common.Hash),
headers:   make(chan *types.Header),
installed: make(chan struct{}),
err:       make(chan error),
}

return es.subscribe(sub)
}

// subscribePendingLogs creates a subscription that writes transaction hashes for
// transactions that enter the transaction pool.
func (es *EventSystem) subscribePendingLogs(crit FilterCriteria, logs chan []*types.Log)
*Subscription {
sub := &subscription{
```

```
id:        rpc.NewID(),
typ:       PendingLogsSubscription,
logsCrit:  crit,
created:   time.Now(),
logs:      logs,
hashes:    make(chan common.Hash),
headers:   make(chan *types.Header),
installed: make(chan struct{}),
err:       make(chan error),
}

return es.subscribe(sub)
}

// SubscribeNewHeads creates a subscription that writes the header of a block that is
// imported in the chain.
func (es *EventSystem) SubscribeNewHeads(headers chan *types.Header) *Subscription {
sub := &subscription{
id:        rpc.NewID(),
typ:       BlocksSubscription,
created:   time.Now(),
logs:      make(chan []*types.Log),
hashes:    make(chan common.Hash),
headers:   headers,
installed: make(chan struct{}),
err:       make(chan error),
}

return es.subscribe(sub)
}

// SubscribePendingTxEvents creates a subscription that writes transaction hashes for
// transactions that enter the transaction pool.
func (es *EventSystem) SubscribePendingTxEvents(hashes chan common.Hash) *Subscription {
sub := &subscription{
id:        rpc.NewID(),
typ:       PendingTransactionsSubscription,
created:   time.Now(),
logs:      make(chan []*types.Log),
hashes:    hashes,
headers:   make(chan *types.Header),
installed: make(chan struct{}),
```

```
err:       make(chan error),
}

return es.subscribe(sub)
}

type filterIndex map[Type]map[rpc.ID]*subscription

// broadcast event to filters that match criteria.
func (es *EventSystem) broadcast(filters filterIndex, ev *event.TypeMuxEvent) {
if ev == nil {
return
}

switch e := ev.Data.(type) {
case []*types.Log:
if len(e) > 0 {
for _, f := range filters[LogsSubscription] {
if ev.Time.After(f.created) {
if matchedLogs := filterLogs(e, f.logsCrit.FromBlock, f.logsCrit.ToBlock, f.logsCrit.Addresses,
f.logsCrit.Topics); len(matchedLogs) > 0 {
f.logs <- matchedLogs
}
}
}
}
case core.RemovedLogsEvent:
for _, f := range filters[LogsSubscription] {
if ev.Time.After(f.created) {
if matchedLogs := filterLogs(e.Logs, f.logsCrit.FromBlock, f.logsCrit.ToBlock, f.logsCrit.Addresses,
f.logsCrit.Topics); len(matchedLogs) > 0 {
f.logs <- matchedLogs
}
}
}
case core.PendingLogsEvent:
for _, f := range filters[PendingLogsSubscription] {
if ev.Time.After(f.created) {
if matchedLogs := filterLogs(e.Logs, nil, f.logsCrit.ToBlock, f.logsCrit.Addresses, f.logsCrit.Topics);
len(matchedLogs) > 0 {
f.logs <- matchedLogs
}
```

```
}
}
case core.TxPreEvent:
for _, f := range filters[PendingTransactionsSubscription] {
if ev.Time.After(f.created) {
f.hashes <- e.Tx.Hash()
}
}
case core.ChainEvent:
for _, f := range filters[BlocksSubscription] {
if ev.Time.After(f.created) {
f.headers <- e.Block.Header()
}
}
if es.lightMode && len(filters[LogsSubscription]) > 0 {
es.lightFilterNewHead(e.Block.Header(), func(header *types.Header, remove bool) {
for _, f := range filters[LogsSubscription] {
if ev.Time.After(f.created) {
if matchedLogs := es.lightFilterLogs(header, f.logsCrit.Addresses, f.logsCrit.Topics, remove);
len(matchedLogs) > 0 {
f.logs <- matchedLogs
}
}
}
})
}
}
}

func (es *EventSystem) lightFilterNewHead(newHeader *types.Header, callBack
func(*types.Header, bool)) {
oldh := es.lastHead
es.lastHead = newHeader
if oldh == nil {
return
}
newh := newHeader
// find common ancestor, create list of rolled back and new block hashes
var oldHeaders, newHeaders []*types.Header
for oldh.Hash() != newh.Hash() {
if oldh.Number.Uint64() >= newh.Number.Uint64() {
oldHeaders = append(oldHeaders, oldh)
```

```
oldh = core.GetHeader(es.backend.ChainDb(), oldh.ParentHash, oldh.Number.Uint64()-1)
}
if oldh.Number.Uint64() < newh.Number.Uint64() {
newHeaders = append(newHeaders, newh)
newh = core.GetHeader(es.backend.ChainDb(), newh.ParentHash, newh.Number.Uint64()-1)
if newh == nil {
// happens when CHT syncing, nothing to do
newh = oldh
}
}
}
// roll back old blocks
for _, h := range oldHeaders {
callBack(h, true)
}
// check new blocks (array is in reverse order)
for i := len(newHeaders) - 1; i >= 0; i-- {
callBack(newHeaders[i], false)
}
}

// filter logs of a single header in light client mode
func (es *EventSystem) lightFilterLogs(header *types.Header, addresses []common.Address,
topics [][]common.Hash, remove bool) []*types.Log {
if bloomFilter(header.Bloom, addresses, topics) {
// Get the logs of the block
ctx, cancel := context.WithTimeout(context.Background(), time.Second*5)
defer cancel()
receipts, err := es.backend.GetReceipts(ctx, header.Hash())
if err != nil {
return nil
}
var unfiltered []*types.Log
for _, receipt := range receipts {
for _, log := range receipt.Logs {
logcopy := *log
logcopy.Removed = remove
unfiltered = append(unfiltered, &logcopy)
}
}
logs := filterLogs(unfiltered, nil, nil, addresses, topics)
return logs
```

```
}
return nil
}

// eventLoop (un)installs filters and processes mux events.
func (es *EventSystem) eventLoop() {
var (
index = make(filterIndex)
sub   = es.mux.Subscribe(core.PendingLogsEvent{}, core.RemovedLogsEvent{}, []*types.Log{},
core.TxPreEvent{}, core.ChainEvent{})
)

for i := UnknownSubscription; i < LastIndexSubscription; i++ {
index[i] = make(map[rpc.ID]*subscription)
}

for {
select {
case ev, active := <-sub.Chan():
if !active { // system stopped
return
}
es.broadcast(index, ev)
case f := <-es.install:
if f.typ == MinedAndPendingLogsSubscription {
// the type are logs and pending logs subscriptions
index[LogsSubscription][f.id] = f
index[PendingLogsSubscription][f.id] = f
} else {
index[f.typ][f.id] = f
}
close(f.installed)
case f := <-es.uninstall:
if f.typ == MinedAndPendingLogsSubscription {
// the type are logs and pending logs subscriptions
delete(index[LogsSubscription], f.id)
delete(index[PendingLogsSubscription], f.id)
} else {
delete(index[f.typ], f.id)
}
close(f.err)
}
```

```
}
}
```

15:F:\git\coin\ethereum\go-ethereum\eth\filters\filter_system_test.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filters

import (
"context"
"math/big"
"reflect"
"testing"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rpc"
)

type testBackend struct {
mux *event.TypeMux
db  ethdb.Database
}

func (b *testBackend) ChainDb() ethdb.Database {
return b.db
}

func (b *testBackend) EventMux() *event.TypeMux {
return b.mux
}

func (b *testBackend) HeaderByNumber(ctx context.Context, blockNr rpc.BlockNumber)
(*types.Header, error) {
var hash common.Hash
var num uint64
if blockNr == rpc.LatestBlockNumber {
```

```
hash = core.GetHeadBlockHash(b.db)
num = core.GetBlockNumber(b.db, hash)
} else {
num = uint64(blockNr)
hash = core.GetCanonicalHash(b.db, num)
}
return core.GetHeader(b.db, hash, num), nil
}

func (b *testBackend) GetReceipts(ctx context.Context, blockHash common.Hash)
(types.Receipts, error) {
num := core.GetBlockNumber(b.db, blockHash)
return core.GetBlockReceipts(b.db, blockHash, num), nil
}

// TestBlockSubscription tests if a block subscription returns block hashes for posted chain events.
// It creates multiple subscriptions:
// - one at the start and should receive all posted chain events and a second (blockHashes)
// - one that is created after a cutoff moment and uninstalled after a second cutoff moment
(blockHashes[cutoff1:cutoff2])
// - one that is created after the second cutoff moment (blockHashes[cutoff2:])
func TestBlockSubscription(t *testing.T) {
t.Parallel()

var (
mux         = new(event.TypeMux)
db, _       = ethdb.NewMemDatabase()
backend     = &testBackend{mux, db}
api         = NewPublicFilterAPI(backend, false)
genesis     = new(core.Genesis).MustCommit(db)
chain, _    = core.GenerateChain(params.TestChainConfig, genesis, db, 10, func(i int, gen
*core.BlockGen) {})
chainEvents = []core.ChainEvent{}
)

for _, blk := range chain {
chainEvents = append(chainEvents, core.ChainEvent{Hash: blk.Hash(), Block: blk})
}

chan0 := make(chan *types.Header)
sub0 := api.events.SubscribeNewHeads(chan0)
chan1 := make(chan *types.Header)
```

```
sub1 := api.events.SubscribeNewHeads(chan1)

go func() { // simulate client
i1, i2 := 0, 0
for i1 != len(chainEvents) || i2 != len(chainEvents) {
select {
case header := <-chan0:
if chainEvents[i1].Hash != header.Hash() {
t.Errorf("sub0 received invalid hash on index %d, want %x, got %x", i1, chainEvents[i1].Hash,
header.Hash())
}
i1++
case header := <-chan1:
if chainEvents[i2].Hash != header.Hash() {
t.Errorf("sub1 received invalid hash on index %d, want %x, got %x", i2, chainEvents[i2].Hash,
header.Hash())
}
i2++
}
}

sub0.Unsubscribe()
sub1.Unsubscribe()
}()

time.Sleep(1 * time.Second)
for _, e := range chainEvents {
mux.Post(e)
}

<-sub0.Err()
<-sub1.Err()
}

// TestPendingTxFilter tests whether pending tx filters retrieve all pending transactions that are
posted to the event mux.
func TestPendingTxFilter(t *testing.T) {
t.Parallel()

var (
mux     = new(event.TypeMux)
db, _   = ethdb.NewMemDatabase()
```

```
backend = &testBackend{mux, db}
api     = NewPublicFilterAPI(backend, false)

transactions = []*types.Transaction{
types.NewTransaction(0,
common.HexToAddress("0xb794f5ea0ba39494ce83a213fffba74279579268"), new(big.Int),
new(big.Int), new(big.Int), nil),
types.NewTransaction(1,
common.HexToAddress("0xb794f5ea0ba39494ce83a213fffba74279579268"), new(big.Int),
new(big.Int), new(big.Int), nil),
types.NewTransaction(2,
common.HexToAddress("0xb794f5ea0ba39494ce83a213fffba74279579268"), new(big.Int),
new(big.Int), new(big.Int), nil),
types.NewTransaction(3,
common.HexToAddress("0xb794f5ea0ba39494ce83a213fffba74279579268"), new(big.Int),
new(big.Int), new(big.Int), nil),
types.NewTransaction(4,
common.HexToAddress("0xb794f5ea0ba39494ce83a213fffba74279579268"), new(big.Int),
new(big.Int), new(big.Int), nil),
}

hashes []common.Hash
)

fid0 := api.NewPendingTransactionFilter()

time.Sleep(1 * time.Second)
for _, tx := range transactions {
ev := core.TxPreEvent{Tx: tx}
mux.Post(ev)
}

for {
results, err := api.GetFilterChanges(fid0)
if err != nil {
t.Fatalf("Unable to retrieve logs: %v", err)
}

h := results.([]common.Hash)
hashes = append(hashes, h...)
if len(hashes) >= len(transactions) {
break
```

```
}

time.Sleep(100 * time.Millisecond)
}

for i := range hashes {
if hashes[i] != transactions[i].Hash() {
t.Errorf("hashes[%d] invalid, want %x, got %x", i, transactions[i].Hash(), hashes[i])
}
}
}

// TestLogFilterCreation test whether a given filter criteria makes sense.
// If not it must return an error.
func TestLogFilterCreation(t *testing.T) {
var (
mux     = new(event.TypeMux)
db, _   = ethdb.NewMemDatabase()
backend = &testBackend{mux, db}
api     = NewPublicFilterAPI(backend, false)

testCases = []struct {
crit    FilterCriteria
success bool
}{
// defaults
{FilterCriteria{}, true},
// valid block number range
{FilterCriteria{FromBlock: big.NewInt(1), ToBlock: big.NewInt(2)}, true},
// "mined" block range to pending
{FilterCriteria{FromBlock: big.NewInt(1), ToBlock: big.NewInt(rpc.LatestBlockNumber.Int64())},
true},
// new mined and pending blocks
{FilterCriteria{FromBlock: big.NewInt(rpc.LatestBlockNumber.Int64()), ToBlock:
big.NewInt(rpc.PendingBlockNumber.Int64())}, true},
// from block "higher" than to block
{FilterCriteria{FromBlock: big.NewInt(2), ToBlock: big.NewInt(1)}, false},
// from block "higher" than to block
{FilterCriteria{FromBlock: big.NewInt(rpc.LatestBlockNumber.Int64()), ToBlock: big.NewInt(100)},
false},
// from block "higher" than to block
{FilterCriteria{FromBlock: big.NewInt(rpc.PendingBlockNumber.Int64()), ToBlock:
```

```
big.NewInt(100)}, false},
// from block "higher" than to block
{FilterCriteria{FromBlock: big.NewInt(rpc.PendingBlockNumber.Int64()), ToBlock:
big.NewInt(rpc.LatestBlockNumber.Int64())}, false},
}
)

for i, test := range testCases {
_, err := api.NewFilter(test.crit)
if test.success && err != nil {
t.Errorf("expected filter creation for case %d to success, got %v", i, err)
}
if !test.success && err == nil {
t.Errorf("expected testcase %d to fail with an error", i)
}
}
}

// TestInvalidLogFilterCreation tests whether invalid filter log criteria results in an error
// when the filter is created.
func TestInvalidLogFilterCreation(t *testing.T) {
t.Parallel()

var (
mux     = new(event.TypeMux)
db, _   = ethdb.NewMemDatabase()
backend = &testBackend{mux, db}
api     = NewPublicFilterAPI(backend, false)
)

// different situations where log filter creation should fail.
// Reason: fromBlock > toBlock
testCases := []FilterCriteria{
0: {FromBlock: big.NewInt(rpc.PendingBlockNumber.Int64()), ToBlock:
big.NewInt(rpc.LatestBlockNumber.Int64())},
1: {FromBlock: big.NewInt(rpc.PendingBlockNumber.Int64()), ToBlock: big.NewInt(100)},
2: {FromBlock: big.NewInt(rpc.LatestBlockNumber.Int64()), ToBlock: big.NewInt(100)},
}

for i, test := range testCases {
if _, err := api.NewFilter(test); err == nil {
t.Errorf("Expected NewFilter for case #%d to fail", i)
```

```
}
}
}

// TestLogFilter tests whether log filters match the correct logs that are posted to the event mux.
func TestLogFilter(t *testing.T) {
t.Parallel()

var (
mux     = new(event.TypeMux)
db, _   = ethdb.NewMemDatabase()
backend = &testBackend{mux, db}
api     = NewPublicFilterAPI(backend, false)

firstAddr      = common.HexToAddress("0x1111111111111111111111111111111111111111")
secondAddr     = common.HexToAddress("0x2222222222222222222222222222222222222222")
thirdAddress   = common.HexToAddress("0x3333333333333333333333333333333333333333")
notUsedAddress =
common.HexToAddress("0x9999999999999999999999999999999999999999")
firstTopic     =
common.HexToHash("0x1111111111111111111111111111111111111111111111111111111111111111")
secondTopic    =
common.HexToHash("0x2222222222222222222222222222222222222222222222222222222222222222")
notUsedTopic   =
common.HexToHash("0x9999999999999999999999999999999999999999999999999999999999999999")

// posted twice, once as vm.Logs and once as core.PendingLogsEvent
allLogs = []*types.Log{
{Address: firstAddr},
{Address: firstAddr, Topics: []common.Hash{firstTopic}, BlockNumber: 1},
{Address: secondAddr, Topics: []common.Hash{firstTopic}, BlockNumber: 1},
{Address: thirdAddress, Topics: []common.Hash{secondTopic}, BlockNumber: 2},
{Address: thirdAddress, Topics: []common.Hash{secondTopic}, BlockNumber: 3},
}

expectedCase7  = []*types.Log{allLogs[3], allLogs[4], allLogs[0], allLogs[1], allLogs[2], allLogs[3], allLogs[4]}
expectedCase11 = []*types.Log{allLogs[1], allLogs[2], allLogs[1], allLogs[2]}
```

```
testCases = []struct {
crit     FilterCriteria
expected []*types.Log
id       rpc.ID
}{
// match all
0: {FilterCriteria{}, allLogs, ""},
// match none due to no matching addresses
1: {FilterCriteria{Addresses: []common.Address{{}, notUsedAddress}, Topics:
[][]common.Hash{allLogs[0].Topics}}, []*types.Log{}, ""},
// match logs based on addresses, ignore topics
2: {FilterCriteria{Addresses: []common.Address{firstAddr}}, allLogs[:2], ""},
// match none due to no matching topics (match with address)
3: {FilterCriteria{Addresses: []common.Address{secondAddr}, Topics:
[][]common.Hash{{notUsedTopic}}}, []*types.Log{}, ""},
// match logs based on addresses and topics
4: {FilterCriteria{Addresses: []common.Address{thirdAddress}, Topics:
[][]common.Hash{{firstTopic, secondTopic}}}, allLogs[3:5], ""},
// match logs based on multiple addresses and "or" topics
5: {FilterCriteria{Addresses: []common.Address{secondAddr, thirdAddress}, Topics:
[][]common.Hash{{firstTopic, secondTopic}}}, allLogs[2:5], ""},
// logs in the pending block
6: {FilterCriteria{Addresses: []common.Address{firstAddr}, FromBlock:
big.NewInt(rpc.PendingBlockNumber.Int64()), ToBlock:
big.NewInt(rpc.PendingBlockNumber.Int64())}, allLogs[:2], ""},
// mined logs with block num >= 2 or pending logs
7: {FilterCriteria{FromBlock: big.NewInt(2), ToBlock:
big.NewInt(rpc.PendingBlockNumber.Int64())}, expectedCase7, ""},
// all "mined" logs with block num >= 2
8: {FilterCriteria{FromBlock: big.NewInt(2), ToBlock: big.NewInt(rpc.LatestBlockNumber.Int64())},
allLogs[3:], ""},
// all "mined" logs
9: {FilterCriteria{ToBlock: big.NewInt(rpc.LatestBlockNumber.Int64())}, allLogs, ""},
// all "mined" logs with 1>= block num <=2 and topic secondTopic
10: {FilterCriteria{FromBlock: big.NewInt(1), ToBlock: big.NewInt(2), Topics:
[][]common.Hash{{secondTopic}}}, allLogs[3:4], ""},
// all "mined" and pending logs with topic firstTopic
11: {FilterCriteria{FromBlock: big.NewInt(rpc.LatestBlockNumber.Int64()), ToBlock:
big.NewInt(rpc.PendingBlockNumber.Int64()), Topics: [][]common.Hash{{firstTopic}}},
expectedCase11, ""},
}
)
```

```
// create all filters
for i := range testCases {
testCases[i].id, _ = api.NewFilter(testCases[i].crit)
}

// raise events
time.Sleep(1 * time.Second)
if err := mux.Post(allLogs); err != nil {
t.Fatal(err)
}
if err := mux.Post(core.PendingLogsEvent{Logs: allLogs}); err != nil {
t.Fatal(err)
}

for i, tt := range testCases {
var fetched []*types.Log
for { // fetch all expected logs
results, err := api.GetFilterChanges(tt.id)
if err != nil {
t.Fatalf("Unable to fetch logs: %v", err)
}

fetched = append(fetched, results.([]*types.Log)...)
if len(fetched) >= len(tt.expected) {
break
}

time.Sleep(100 * time.Millisecond)
}

if len(fetched) != len(tt.expected) {
t.Errorf("invalid number of logs for case %d, want %d log(s), got %d", i, len(tt.expected),
len(fetched))
return
}

for l := range fetched {
if fetched[l].Removed {
t.Errorf("expected log not to be removed for log %d in case %d", l, i)
}
if !reflect.DeepEqual(fetched[l], tt.expected[l]) {
```

```
t.Errorf("invalid log on index %d for case %d", l, i)
}
}
}
}

// TestPendingLogsSubscription tests if a subscription receives the correct pending logs that are posted to the event mux.
func TestPendingLogsSubscription(t *testing.T) {
t.Parallel()

var (
mux     = new(event.TypeMux)
db, _   = ethdb.NewMemDatabase()
backend = &testBackend{mux, db}
api     = NewPublicFilterAPI(backend, false)

firstAddr      = common.HexToAddress("0x1111111111111111111111111111111111111111")
secondAddr     = common.HexToAddress("0x2222222222222222222222222222222222222222")
thirdAddress   = common.HexToAddress("0x3333333333333333333333333333333333333333")
notUsedAddress =
common.HexToAddress("0x9999999999999999999999999999999999999999")
firstTopic     =
common.HexToHash("0x1111111111111111111111111111111111111111111111111111111111111111")
secondTopic    =
common.HexToHash("0x2222222222222222222222222222222222222222222222222222222222222222")
thirdTopic     =
common.HexToHash("0x3333333333333333333333333333333333333333333333333333333333333333")
forthTopic     =
common.HexToHash("0x4444444444444444444444444444444444444444444444444444444444444444")
notUsedTopic   =
common.HexToHash("0x9999999999999999999999999999999999999999999999999999999999999999")

allLogs = []core.PendingLogsEvent{
{Logs: []*types.Log{{Address: firstAddr, Topics: []common.Hash{}, BlockNumber: 0}}},
{Logs: []*types.Log{{Address: firstAddr, Topics: []common.Hash{firstTopic}, BlockNumber: 1}}},
{Logs: []*types.Log{{Address: secondAddr, Topics: []common.Hash{firstTopic}, BlockNumber: 2}}},
```

```
{Logs: []*types.Log{{Address: thirdAddress, Topics: []common.Hash{secondTopic}, BlockNumber: 3}}},
{Logs: []*types.Log{{Address: thirdAddress, Topics: []common.Hash{secondTopic}, BlockNumber: 4}}},
{Logs: []*types.Log{
{Address: thirdAddress, Topics: []common.Hash{firstTopic}, BlockNumber: 5},
{Address: thirdAddress, Topics: []common.Hash{thirdTopic}, BlockNumber: 5},
{Address: thirdAddress, Topics: []common.Hash{forthTopic}, BlockNumber: 5},
{Address: firstAddr, Topics: []common.Hash{firstTopic}, BlockNumber: 5},
}},
}

convertLogs = func(pl []core.PendingLogsEvent) []*types.Log {
var logs []*types.Log
for _, l := range pl {
logs = append(logs, l.Logs...)
}
return logs
}

testCases = []struct {
crit     FilterCriteria
expected []*types.Log
c        chan []*types.Log
sub      *Subscription
}{
// match all
{FilterCriteria{}, convertLogs(allLogs), nil, nil},
// match none due to no matching addresses
{FilterCriteria{Addresses: []common.Address{{}, notUsedAddress}, Topics: [][]common.Hash{{}}},
[]*types.Log{}, nil, nil},
// match logs based on addresses, ignore topics
{FilterCriteria{Addresses: []common.Address{firstAddr}}, append(convertLogs(allLogs[:2]),
allLogs[5].Logs[3]), nil, nil},
// match none due to no matching topics (match with address)
{FilterCriteria{Addresses: []common.Address{secondAddr}, Topics:
[][]common.Hash{{notUsedTopic}}}, []*types.Log{}, nil, nil},
// match logs based on addresses and topics
{FilterCriteria{Addresses: []common.Address{thirdAddress}, Topics: [][]common.Hash{{firstTopic,
secondTopic}}}, append(convertLogs(allLogs[3:5]), allLogs[5].Logs[0]), nil, nil},
// match logs based on multiple addresses and "or" topics
{FilterCriteria{Addresses: []common.Address{secondAddr, thirdAddress}, Topics:
```

```
[][]common.Hash{{firstTopic, secondTopic}}}, append(convertLogs(allLogs[2:5]),
allLogs[5].Logs[0]), nil, nil},
// block numbers are ignored for filters created with New***Filter, these return all logs that match
the given criteria when the state changes
{FilterCriteria{Addresses: []common.Address{firstAddr}, FromBlock: big.NewInt(2), ToBlock:
big.NewInt(3)}, append(convertLogs(allLogs[:2]), allLogs[5].Logs[3]), nil, nil},
// multiple pending logs, should match only 2 topics from the logs in block 5
{FilterCriteria{Addresses: []common.Address{thirdAddress}, Topics: [][]common.Hash{{firstTopic,
forthTopic}}}, []*types.Log{allLogs[5].Logs[0], allLogs[5].Logs[2]}, nil, nil},
}
)

// create all subscriptions, this ensures all subscriptions are created before the events are posted.
// on slow machines this could otherwise lead to missing events when the subscription is created
after
// (some) events are posted.
for i := range testCases {
testCases[i].c = make(chan []*types.Log)
testCases[i].sub, _ = api.events.SubscribeLogs(testCases[i].crit, testCases[i].c)
}

for n, test := range testCases {
i := n
tt := test
go func() {
var fetched []*types.Log
fetchLoop:
for {
logs := <-tt.c
fetched = append(fetched, logs...)
if len(fetched) >= len(tt.expected) {
break fetchLoop
}
}

if len(fetched) != len(tt.expected) {
t.Fatalf("invalid number of logs for case %d, want %d log(s), got %d", i, len(tt.expected),
len(fetched))
}

for l := range fetched {
if fetched[l].Removed {
```

```
t.Errorf("expected log not to be removed for log %d in case %d", l, i)
}
if !reflect.DeepEqual(fetched[l], tt.expected[l]) {
t.Errorf("invalid log on index %d for case %d", l, i)
}
}
}()
}

// raise events
time.Sleep(1 * time.Second)
for _, l := range allLogs {
if err := mux.Post(l); err != nil {
t.Fatal(err)
}
}
}

16:F:\git\coin\ethereum\go-ethereum\eth\filters\filter_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filters

import (
"context"
"io/ioutil"
"math/big"
"os"
"testing"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/params"
)

func makeReceipt(addr common.Address) *types.Receipt {
receipt := types.NewReceipt(nil, new(big.Int))
receipt.Logs = []*types.Log{
```

```
{Address: addr},
}
receipt.Bloom = types.CreateBloom(types.Receipts{receipt})
return receipt
}

func BenchmarkMipmaps(b *testing.B) {
dir, err := ioutil.TempDir("", "mipmap")
if err != nil {
b.Fatal(err)
}
defer os.RemoveAll(dir)

var (
db, _   = ethdb.NewLDBDatabase(dir, 0, 0)
mux     = new(event.TypeMux)
backend = &testBackend{mux, db}
key1, _ =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
addr1   = crypto.PubkeyToAddress(key1.PublicKey)
addr2   = common.BytesToAddress([]byte("jeff"))
addr3   = common.BytesToAddress([]byte("ethereum"))
addr4   = common.BytesToAddress([]byte("random addresses please"))
)
defer db.Close()

genesis := core.GenesisBlockForTesting(db, addr1, big.NewInt(1000000))
chain, receipts := core.GenerateChain(params.TestChainConfig, genesis, db, 100010, func(i int,
gen *core.BlockGen) {
var receipts types.Receipts
switch i {
case 2403:
receipt := makeReceipt(addr1)
receipts = types.Receipts{receipt}
gen.AddUncheckedReceipt(receipt)
case 1034:
receipt := makeReceipt(addr2)
receipts = types.Receipts{receipt}
gen.AddUncheckedReceipt(receipt)
case 34:
receipt := makeReceipt(addr3)
```

```
receipts = types.Receipts{receipt}
gen.AddUncheckedReceipt(receipt)
case 99999:
receipt := makeReceipt(addr4)
receipts = types.Receipts{receipt}
gen.AddUncheckedReceipt(receipt)

}

// store the receipts
err := core.WriteReceipts(db, receipts)
if err != nil {
b.Fatal(err)
}
core.WriteMipmapBloom(db, uint64(i+1), receipts)
})
for i, block := range chain {
core.WriteBlock(db, block)
if err := core.WriteCanonicalHash(db, block.Hash(), block.NumberU64()); err != nil {
b.Fatalf("failed to insert block number: %v", err)
}
if err := core.WriteHeadBlockHash(db, block.Hash()); err != nil {
b.Fatalf("failed to insert block number: %v", err)
}
if err := core.WriteBlockReceipts(db, block.Hash(), block.NumberU64(), receipts[i]); err != nil {
b.Fatal("error writing block receipts:", err)
}
}
b.ResetTimer()

filter := New(backend, true)
filter.SetAddresses([]common.Address{addr1, addr2, addr3, addr4})
filter.SetBeginBlock(0)
filter.SetEndBlock(-1)

for i := 0; i < b.N; i++ {
logs, _ := filter.Find(context.Background())
if len(logs) != 4 {
b.Fatal("expected 4 logs, got", len(logs))
}
}
}
```

```
func TestFilters(t *testing.T) {
dir, err := ioutil.TempDir("", "mipmap")
if err != nil {
t.Fatal(err)
}
defer os.RemoveAll(dir)

var (
db, _   = ethdb.NewLDBDatabase(dir, 0, 0)
mux     = new(event.TypeMux)
backend = &testBackend{mux, db}
key1, _ =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
addr    = crypto.PubkeyToAddress(key1.PublicKey)

hash1 = common.BytesToHash([]byte("topic1"))
hash2 = common.BytesToHash([]byte("topic2"))
hash3 = common.BytesToHash([]byte("topic3"))
hash4 = common.BytesToHash([]byte("topic4"))
)
defer db.Close()

genesis := core.GenesisBlockForTesting(db, addr, big.NewInt(1000000))
chain, receipts := core.GenerateChain(params.TestChainConfig, genesis, db, 1000, func(i int, gen
*core.BlockGen) {
var receipts types.Receipts
switch i {
case 1:
receipt := types.NewReceipt(nil, new(big.Int))
receipt.Logs = []*types.Log{
{
Address: addr,
Topics:  []common.Hash{hash1},
},
}
gen.AddUncheckedReceipt(receipt)
receipts = types.Receipts{receipt}
case 2:
receipt := types.NewReceipt(nil, new(big.Int))
receipt.Logs = []*types.Log{
```

```
{
Address: addr,
Topics:  []common.Hash{hash2},
},
}
gen.AddUncheckedReceipt(receipt)
receipts = types.Receipts{receipt}
case 998:
receipt := types.NewReceipt(nil, new(big.Int))
receipt.Logs = []*types.Log{
{
Address: addr,
Topics:  []common.Hash{hash3},
},
}
gen.AddUncheckedReceipt(receipt)
receipts = types.Receipts{receipt}
case 999:
receipt := types.NewReceipt(nil, new(big.Int))
receipt.Logs = []*types.Log{
{
Address: addr,
Topics:  []common.Hash{hash4},
},
}
gen.AddUncheckedReceipt(receipt)
receipts = types.Receipts{receipt}
}

// store the receipts
err := core.WriteReceipts(db, receipts)
if err != nil {
t.Fatal(err)
}
// i is used as block number for the writes but since the i
// starts at 0 and block 0 (genesis) is already present increment
// by one
core.WriteMipmapBloom(db, uint64(i+1), receipts)
})
for i, block := range chain {
core.WriteBlock(db, block)
if err := core.WriteCanonicalHash(db, block.Hash(), block.NumberU64()); err != nil {
```

```
t.Fatalf("failed to insert block number: %v", err)
}
if err := core.WriteHeadBlockHash(db, block.Hash()); err != nil {
t.Fatalf("failed to insert block number: %v", err)
}
if err := core.WriteBlockReceipts(db, block.Hash(), block.NumberU64(), receipts[i]); err != nil {
t.Fatal("error writing block receipts:", err)
}
}

filter := New(backend, true)
filter.SetAddresses([]common.Address{addr})
filter.SetTopics([][]common.Hash{{hash1, hash2, hash3, hash4}})
filter.SetBeginBlock(0)
filter.SetEndBlock(-1)

logs, _ := filter.Find(context.Background())
if len(logs) != 4 {
t.Error("expected 4 log, got", len(logs))
}

filter = New(backend, true)
filter.SetAddresses([]common.Address{addr})
filter.SetTopics([][]common.Hash{{hash3}})
filter.SetBeginBlock(900)
filter.SetEndBlock(999)
logs, _ = filter.Find(context.Background())
if len(logs) != 1 {
t.Error("expected 1 log, got", len(logs))
}
if len(logs) > 0 && logs[0].Topics[0] != hash3 {
t.Errorf("expected log[0].Topics[0] to be %x, got %x", hash3, logs[0].Topics[0])
}

filter = New(backend, true)
filter.SetAddresses([]common.Address{addr})
filter.SetTopics([][]common.Hash{{hash3}})
filter.SetBeginBlock(990)
filter.SetEndBlock(-1)
logs, _ = filter.Find(context.Background())
if len(logs) != 1 {
t.Error("expected 1 log, got", len(logs))
```

```
}
if len(logs) > 0 && logs[0].Topics[0] != hash3 {
t.Errorf("expected log[0].Topics[0] to be %x, got %x", hash3, logs[0].Topics[0])
}

filter = New(backend, true)
filter.SetTopics([][]common.Hash{{hash1, hash2}})
filter.SetBeginBlock(1)
filter.SetEndBlock(10)

logs, _ = filter.Find(context.Background())
if len(logs) != 2 {
t.Error("expected 2 log, got", len(logs))
}

failHash := common.BytesToHash([]byte("fail"))
filter = New(backend, true)
filter.SetTopics([][]common.Hash{{failHash}})
filter.SetBeginBlock(0)
filter.SetEndBlock(-1)

logs, _ = filter.Find(context.Background())
if len(logs) != 0 {
t.Error("expected 0 log, got", len(logs))
}

failAddr := common.BytesToAddress([]byte("failmenow"))
filter = New(backend, true)
filter.SetAddresses([]common.Address{failAddr})
filter.SetBeginBlock(0)
filter.SetEndBlock(-1)

logs, _ = filter.Find(context.Background())
if len(logs) != 0 {
t.Error("expected 0 log, got", len(logs))
}

filter = New(backend, true)
filter.SetTopics([][]common.Hash{{failHash}, {hash1}})
filter.SetBeginBlock(0)
filter.SetEndBlock(-1)
```

```
logs, _ = filter.Find(context.Background())
if len(logs) != 0 {
t.Error("expected 0 log, got", len(logs))
}
}

17:F:\git\coin\ethereum\go-ethereum\eth\gasprice\gasprice.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package gasprice

import (
"context"
"math/big"
"sort"
"sync"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/internal/ethapi"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rpc"
)

var maxPrice = big.NewInt(500 * params.Shannon)

type Config struct {
Blocks     int
Percentile int
Default    *big.Int `toml:",omitempty"`
}

// Oracle recommends gas prices based on the content of recent
// blocks. Suitable for both light and full clients.
type Oracle struct {
backend   ethapi.Backend
lastHead  common.Hash
lastPrice *big.Int
cacheLock sync.RWMutex
fetchLock sync.Mutex

checkBlocks, maxEmpty, maxBlocks int
percentile                       int
```

```
}

// NewOracle returns a new oracle.
func NewOracle(backend ethapi.Backend, params Config) *Oracle {
blocks := params.Blocks
if blocks < 1 {
blocks = 1
}
percent := params.Percentile
if percent < 0 {
percent = 0
}
if percent > 100 {
percent = 100
}
return &Oracle{
backend:     backend,
lastPrice:   params.Default,
checkBlocks: blocks,
maxEmpty:    blocks / 2,
maxBlocks:   blocks * 5,
percentile:  percent,
}
}

// SuggestPrice returns the recommended gas price.
func (gpo *Oracle) SuggestPrice(ctx context.Context) (*big.Int, error) {
gpo.cacheLock.RLock()
lastHead := gpo.lastHead
lastPrice := gpo.lastPrice
gpo.cacheLock.RUnlock()

head, _ := gpo.backend.HeaderByNumber(ctx, rpc.LatestBlockNumber)
headHash := head.Hash()
if headHash == lastHead {
return lastPrice, nil
}

gpo.fetchLock.Lock()
defer gpo.fetchLock.Unlock()

// try checking the cache again, maybe the last fetch fetched what we need
```

```
gpo.cacheLock.RLock()
lastHead = gpo.lastHead
lastPrice = gpo.lastPrice
gpo.cacheLock.RUnlock()
if headHash == lastHead {
return lastPrice, nil
}

blockNum := head.Number.Uint64()
ch := make(chan getBlockPricesResult, gpo.checkBlocks)
sent := 0
exp := 0
var txPrices []*big.Int
for sent < gpo.checkBlocks && blockNum > 0 {
go gpo.getBlockPrices(ctx, blockNum, ch)
sent++
exp++
blockNum--
}
maxEmpty := gpo.maxEmpty
for exp > 0 {
res := <-ch
if res.err != nil {
return lastPrice, res.err
}
exp--
if len(res.prices) > 0 {
txPrices = append(txPrices, res.prices...)
continue
}
if maxEmpty > 0 {
maxEmpty--
continue
}
if blockNum > 0 && sent < gpo.maxBlocks {
go gpo.getBlockPrices(ctx, blockNum, ch)
sent++
exp++
blockNum--
}
}
price := lastPrice
```

```
if len(txPrices) > 0 {
sort.Sort(bigIntArray(txPrices))
price = txPrices[(len(txPrices)-1)*gpo.percentile/100]
}
if price.Cmp(maxPrice) > 0 {
price = new(big.Int).Set(maxPrice)
}

gpo.cacheLock.Lock()
gpo.lastHead = headHash
gpo.lastPrice = price
gpo.cacheLock.Unlock()
return price, nil
}

type getBlockPricesResult struct {
prices []*big.Int
err    error
}

// getLowestPrice calculates the lowest transaction gas price in a given block
// and sends it to the result channel. If the block is empty, price is nil.
func (gpo *Oracle) getBlockPrices(ctx context.Context, blockNum uint64, ch chan
getBlockPricesResult) {
block, err := gpo.backend.BlockByNumber(ctx, rpc.BlockNumber(blockNum))
if block == nil {
ch <- getBlockPricesResult{nil, err}
return
}
txs := block.Transactions()
prices := make([]*big.Int, len(txs))
for i, tx := range txs {
prices[i] = tx.GasPrice()
}
ch <- getBlockPricesResult{prices, nil}
}

type bigIntArray []*big.Int

func (s bigIntArray) Len() int           { return len(s) }
func (s bigIntArray) Less(i, j int) bool { return s[i].Cmp(s[j]) < 0 }
func (s bigIntArray) Swap(i, j int)      { s[i], s[j] = s[j], s[i] }
```

```
18:F:\git\coin\ethereum\go-ethereum\eth\gen_config.go
func (c Config) MarshalTOML() (interface{}, error) {
type Config struct {
Genesis                 *core.Genesis `toml:",omitempty"`
NetworkId               uint64
SyncMode                downloader.SyncMode
LightServ               int  `toml:",omitempty"`
LightPeers              int  `toml:",omitempty"`
MaxPeers                int  `toml:"-"`
SkipBcVersionCheck      bool `toml:"-"`
DatabaseHandles         int  `toml:"-"`
DatabaseCache           int
Etherbase               common.Address `toml:",omitempty"`
MinerThreads            int            `toml:",omitempty"`
ExtraData               hexutil.Bytes  `toml:",omitempty"`
GasPrice                *big.Int
EthashCacheDir          string
EthashCachesInMem       int
EthashCachesOnDisk      int
EthashDatasetDir        string
EthashDatasetsInMem     int
EthashDatasetsOnDisk    int
TxPool                  core.TxPoolConfig
GPO                     gasprice.Config
EnablePreimageRecording bool
DocRoot                 string `toml:"-"`
PowFake                 bool   `toml:"-"`
PowTest                 bool   `toml:"-"`
PowShared               bool   `toml:"-"`
}
var enc Config
enc.Genesis = c.Genesis
enc.NetworkId = c.NetworkId
enc.SyncMode = c.SyncMode
enc.LightServ = c.LightServ
enc.LightPeers = c.LightPeers
enc.MaxPeers = c.MaxPeers
enc.SkipBcVersionCheck = c.SkipBcVersionCheck
enc.DatabaseHandles = c.DatabaseHandles
enc.DatabaseCache = c.DatabaseCache
enc.Etherbase = c.Etherbase
```

```
enc.MinerThreads = c.MinerThreads
enc.ExtraData = c.ExtraData
enc.GasPrice = c.GasPrice
enc.EthashCacheDir = c.EthashCacheDir
enc.EthashCachesInMem = c.EthashCachesInMem
enc.EthashCachesOnDisk = c.EthashCachesOnDisk
enc.EthashDatasetDir = c.EthashDatasetDir
enc.EthashDatasetsInMem = c.EthashDatasetsInMem
enc.EthashDatasetsOnDisk = c.EthashDatasetsOnDisk
enc.TxPool = c.TxPool
enc.GPO = c.GPO
enc.EnablePreimageRecording = c.EnablePreimageRecording
enc.DocRoot = c.DocRoot
enc.PowFake = c.PowFake
enc.PowTest = c.PowTest
enc.PowShared = c.PowShared
return &enc, nil
}

func (c *Config) UnmarshalTOML(unmarshal func(interface{}) error) error {
type Config struct {
Genesis                 *core.Genesis `toml:",omitempty"`
NetworkId               *uint64
SyncMode                *downloader.SyncMode
LightServ               *int  `toml:",omitempty"`
LightPeers              *int  `toml:",omitempty"`
MaxPeers                *int  `toml:"-"`
SkipBcVersionCheck      *bool `toml:"-"`
DatabaseHandles         *int  `toml:"-"`
DatabaseCache           *int
Etherbase               *common.Address `toml:",omitempty"`
MinerThreads            *int            `toml:",omitempty"`
ExtraData               hexutil.Bytes   `toml:",omitempty"`
GasPrice                *big.Int
EthashCacheDir          *string
EthashCachesInMem       *int
EthashCachesOnDisk      *int
EthashDatasetDir        *string
EthashDatasetsInMem     *int
EthashDatasetsOnDisk    *int
TxPool                  *core.TxPoolConfig
GPO                     *gasprice.Config
```

```
EnablePreimageRecording *bool
DocRoot                 *string `toml:"-"`
PowFake                 *bool   `toml:"-"`
PowTest                 *bool   `toml:"-"`
PowShared               *bool   `toml:"-"`
}
var dec Config
if err := unmarshal(&dec); err != nil {
return err
}
if dec.Genesis != nil {
c.Genesis = dec.Genesis
}
if dec.NetworkId != nil {
c.NetworkId = *dec.NetworkId
}
if dec.SyncMode != nil {
c.SyncMode = *dec.SyncMode
}
if dec.LightServ != nil {
c.LightServ = *dec.LightServ
}
if dec.LightPeers != nil {
c.LightPeers = *dec.LightPeers
}
if dec.MaxPeers != nil {
c.MaxPeers = *dec.MaxPeers
}
if dec.SkipBcVersionCheck != nil {
c.SkipBcVersionCheck = *dec.SkipBcVersionCheck
}
if dec.DatabaseHandles != nil {
c.DatabaseHandles = *dec.DatabaseHandles
}
if dec.DatabaseCache != nil {
c.DatabaseCache = *dec.DatabaseCache
}
if dec.Etherbase != nil {
c.Etherbase = *dec.Etherbase
}
if dec.MinerThreads != nil {
c.MinerThreads = *dec.MinerThreads
```

```
}
if dec.ExtraData != nil {
c.ExtraData = dec.ExtraData
}
if dec.GasPrice != nil {
c.GasPrice = dec.GasPrice
}
if dec.EthashCacheDir != nil {
c.EthashCacheDir = *dec.EthashCacheDir
}
if dec.EthashCachesInMem != nil {
c.EthashCachesInMem = *dec.EthashCachesInMem
}
if dec.EthashCachesOnDisk != nil {
c.EthashCachesOnDisk = *dec.EthashCachesOnDisk
}
if dec.EthashDatasetDir != nil {
c.EthashDatasetDir = *dec.EthashDatasetDir
}
if dec.EthashDatasetsInMem != nil {
c.EthashDatasetsInMem = *dec.EthashDatasetsInMem
}
if dec.EthashDatasetsOnDisk != nil {
c.EthashDatasetsOnDisk = *dec.EthashDatasetsOnDisk
}
if dec.TxPool != nil {
c.TxPool = *dec.TxPool
}
if dec.GPO != nil {
c.GPO = *dec.GPO
}
if dec.EnablePreimageRecording != nil {
c.EnablePreimageRecording = *dec.EnablePreimageRecording
}
if dec.DocRoot != nil {
c.DocRoot = *dec.DocRoot
}
if dec.PowFake != nil {
c.PowFake = *dec.PowFake
}
if dec.PowTest != nil {
c.PowTest = *dec.PowTest
```

```
}
if dec.PowShared != nil {
c.PowShared = *dec.PowShared
}
return nil
}

19:F:\git\coin\ethereum\go-ethereum\eth\handler.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"encoding/json"
"errors"
"fmt"
"math"
"math/big"
"sync"
"sync/atomic"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/consensus/misc"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/eth/fetcher"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/p2p/discover"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rlp"
)

const (
softResponseLimit = 2 * 1024 * 1024 // Target maximum size of returned blocks, headers or node
data.
estHeaderRlpSize  = 500             // Approximate size of an RLP encoded block header
```

```
)

var (
daoChallengeTimeout = 15 * time.Second // Time allowance for a node to reply to the DAO
handshake challenge
)

// errIncompatibleConfig is returned if the requested protocols and configs are
// not compatible (low protocol version restrictions and high requirements).
var errIncompatibleConfig = errors.New("incompatible configuration")

func errResp(code errCode, format string, v ...interface{}) error {
return fmt.Errorf("%v - %v", code, fmt.Sprintf(format, v...))
}

type ProtocolManager struct {
networkId uint64

fastSync  uint32 // Flag whether fast sync is enabled (gets disabled if we already have blocks)
acceptTxs uint32 // Flag whether we're considered synchronised (enables transaction processing)

txpool      txPool
blockchain  *core.BlockChain
chaindb     ethdb.Database
chainconfig *params.ChainConfig
maxPeers    int

downloader *downloader.Downloader
fetcher    *fetcher.Fetcher
peers      *peerSet

SubProtocols []p2p.Protocol

eventMux      *event.TypeMux
txSub         *event.TypeMuxSubscription
minedBlockSub *event.TypeMuxSubscription

// channels for fetcher, syncer, txsyncLoop
newPeerCh   chan *peer
txsyncCh    chan *txsync
quitSync    chan struct{}
noMorePeers chan struct{}
```

```
// wait group is used for graceful shutdowns during downloading
// and processing
wg sync.WaitGroup
}

// NewProtocolManager returns a new ethereum sub protocol manager. The Ethereum sub
protocol manages peers capable
// with the ethereum network.
func NewProtocolManager(config *params.ChainConfig, mode downloader.SyncMode, networkId
uint64, maxPeers int, mux *event.TypeMux, txpool txPool, engine consensus.Engine, blockchain
*core.BlockChain, chaindb ethdb.Database) (*ProtocolManager, error) {
// Create the protocol manager with the base fields
manager := &ProtocolManager{
networkId:   networkId,
eventMux:    mux,
txpool:      txpool,
blockchain:  blockchain,
chaindb:     chaindb,
chainconfig: config,
maxPeers:    maxPeers,
peers:       newPeerSet(),
newPeerCh:   make(chan *peer),
noMorePeers: make(chan struct{}),
txsyncCh:    make(chan *txsync),
quitSync:    make(chan struct{}),
}
// Figure out whether to allow fast sync or not
if mode == downloader.FastSync && blockchain.CurrentBlock().NumberU64() > 0 {
log.Warn("Blockchain not empty, fast sync disabled")
mode = downloader.FullSync
}
if mode == downloader.FastSync {
manager.fastSync = uint32(1)
}
// Initiate a sub-protocol for every implemented version we can handle
manager.SubProtocols = make([]p2p.Protocol, 0, len(ProtocolVersions))
for i, version := range ProtocolVersions {
// Skip protocol version if incompatible with the mode of operation
if mode == downloader.FastSync && version < eth63 {
continue
}
```

```
// Compatible; initialise the sub-protocol
version := version // Closure for the run
manager.SubProtocols = append(manager.SubProtocols, p2p.Protocol{
Name:    ProtocolName,
Version: version,
Length:  ProtocolLengths[i],
Run: func(p *p2p.Peer, rw p2p.MsgReadWriter) error {
peer := manager.newPeer(int(version), p, rw)
select {
case manager.newPeerCh <- peer:
manager.wg.Add(1)
defer manager.wg.Done()
return manager.handle(peer)
case <-manager.quitSync:
return p2p.DiscQuitting
}
},
NodeInfo: func() interface{} {
return manager.NodeInfo()
},
PeerInfo: func(id discover.NodeID) interface{} {
if p := manager.peers.Peer(fmt.Sprintf("%x", id[:8])); p != nil {
return p.Info()
}
return nil
},
})
}
if len(manager.SubProtocols) == 0 {
return nil, errIncompatibleConfig
}
// Construct the different synchronisation mechanisms
manager.downloader = downloader.New(mode, chaindb, manager.eventMux,
blockchain.HasHeader, blockchain.HasBlockAndState, blockchain.GetHeaderByHash,
blockchain.GetBlockByHash, blockchain.CurrentHeader, blockchain.CurrentBlock,
blockchain.CurrentFastBlock, blockchain.FastSyncCommitHead,
blockchain.GetTdByHash, blockchain.InsertHeaderChain, manager.blockchain.InsertChain,
blockchain.InsertReceiptChain, blockchain.Rollback,
manager.removePeer)

validator := func(header *types.Header) error {
return engine.VerifyHeader(blockchain, header, true)
```

```
}
heighter := func() uint64 {
return blockchain.CurrentBlock().NumberU64()
}
inserter := func(blocks types.Blocks) (int, error) {
// If fast sync is running, deny importing weird blocks
if atomic.LoadUint32(&manager.fastSync) == 1 {
log.Warn("Discarded bad propagated block", "number", blocks[0].Number(), "hash",
blocks[0].Hash())
return 0, nil
}
atomic.StoreUint32(&manager.acceptTxs, 1) // Mark initial sync done on any fetcher import
return manager.blockchain.InsertChain(blocks)
}
manager.fetcher = fetcher.New(blockchain.GetBlockByHash, validator, manager.BroadcastBlock,
heighter, inserter, manager.removePeer)

return manager, nil
}

func (pm *ProtocolManager) removePeer(id string) {
// Short circuit if the peer was already removed
peer := pm.peers.Peer(id)
if peer == nil {
return
}
log.Debug("Removing Ethereum peer", "peer", id)

// Unregister the peer from the downloader and Ethereum peer set
pm.downloader.UnregisterPeer(id)
if err := pm.peers.Unregister(id); err != nil {
log.Error("Peer removal failed", "peer", id, "err", err)
}
// Hard disconnect at the networking layer
if peer != nil {
peer.Peer.Disconnect(p2p.DiscUselessPeer)
}
}

func (pm *ProtocolManager) Start() {
// broadcast transactions
pm.txSub = pm.eventMux.Subscribe(core.TxPreEvent{})
```

```
go pm.txBroadcastLoop()
// broadcast mined blocks
pm.minedBlockSub = pm.eventMux.Subscribe(core.NewMinedBlockEvent{})
go pm.minedBroadcastLoop()

// start sync handlers
go pm.syncer()
go pm.txsyncLoop()
}

func (pm *ProtocolManager) Stop() {
log.Info("Stopping Ethereum protocol")

pm.txSub.Unsubscribe()         // quits txBroadcastLoop
pm.minedBlockSub.Unsubscribe() // quits blockBroadcastLoop

// Quit the sync loop.
// After this send has completed, no new peers will be accepted.
pm.noMorePeers <- struct{}{}

// Quit fetcher, txsyncLoop.
close(pm.quitSync)

// Disconnect existing sessions.
// This also closes the gate for any new registrations on the peer set.
// sessions which are already established but not added to pm.peers yet
// will exit when they try to register.
pm.peers.Close()

// Wait for all peer handler goroutines and the loops to come down.
pm.wg.Wait()

log.Info("Ethereum protocol stopped")
}

func (pm *ProtocolManager) newPeer(pv int, p *p2p.Peer, rw p2p.MsgReadWriter) *peer {
return newPeer(pv, p, newMeteredMsgWriter(rw))
}

// handle is the callback invoked to manage the life cycle of an eth peer. When
// this function terminates, the peer is disconnected.
func (pm *ProtocolManager) handle(p *peer) error {
```

```
if pm.peers.Len() >= pm.maxPeers {
return p2p.DiscTooManyPeers
}
p.Log().Debug("Ethereum peer connected", "name", p.Name())

// Execute the Ethereum handshake
td, head, genesis := pm.blockchain.Status()
if err := p.Handshake(pm.networkId, td, head, genesis); err != nil {
p.Log().Debug("Ethereum handshake failed", "err", err)
return err
}
if rw, ok := p.rw.(*meteredMsgReadWriter); ok {
rw.Init(p.version)
}
// Register the peer locally
if err := pm.peers.Register(p); err != nil {
p.Log().Error("Ethereum peer registration failed", "err", err)
return err
}
defer pm.removePeer(p.id)

// Register the peer in the downloader. If the downloader considers it banned, we disconnect
if err := pm.downloader.RegisterPeer(p.id, p.version, p.Head, p.RequestHeadersByHash,
p.RequestHeadersByNumber, p.RequestBodies, p.RequestReceipts, p.RequestNodeData); err !=
nil {
return err
}
// Propagate existing transactions. new transactions appearing
// after this will be sent via broadcasts.
pm.syncTransactions(p)

// If we're DAO hard-fork aware, validate any remote peer with regard to the hard-fork
if daoBlock := pm.chainconfig.DAOForkBlock; daoBlock != nil {
// Request the peer's DAO fork header for extra-data validation
if err := p.RequestHeadersByNumber(daoBlock.Uint64(), 1, 0, false); err != nil {
return err
}
// Start a timer to disconnect if the peer doesn't reply in time
p.forkDrop = time.AfterFunc(daoChallengeTimeout, func() {
p.Log().Debug("Timed out DAO fork-check, dropping")
pm.removePeer(p.id)
})
```

```
// Make sure it's cleaned up if the peer dies off
defer func() {
if p.forkDrop != nil {
p.forkDrop.Stop()
p.forkDrop = nil
}
}()
}
// main loop. handle incoming messages.
for {
if err := pm.handleMsg(p); err != nil {
p.Log().Debug("Ethereum message handling failed", "err", err)
return err
}
}
}

// handleMsg is invoked whenever an inbound message is received from a remote
// peer. The remote connection is torn down upon returning any error.
func (pm *ProtocolManager) handleMsg(p *peer) error {
// Read the next message from the remote peer, and ensure it's fully consumed
msg, err := p.rw.ReadMsg()
if err != nil {
return err
}
if msg.Size > ProtocolMaxMsgSize {
return errResp(ErrMsgTooLarge, "%v > %v", msg.Size, ProtocolMaxMsgSize)
}
defer msg.Discard()

// Handle the message depending on its contents
switch {
case msg.Code == StatusMsg:
// Status messages should never arrive after the handshake
return errResp(ErrExtraStatusMsg, "uncontrolled status message")

// Block header query, collect the requested headers and reply
case msg.Code == GetBlockHeadersMsg:
// Decode the complex header query
var query getBlockHeadersData
if err := msg.Decode(&query); err != nil {
return errResp(ErrDecode, "%v: %v", msg, err)
```

```
}
hashMode := query.Origin.Hash != (common.Hash{})

// Gather headers until the fetch or network limits is reached
var (
bytes   common.StorageSize
headers []*types.Header
unknown bool
)
for !unknown && len(headers) < int(query.Amount) && bytes < softResponseLimit && len(headers)
< downloader.MaxHeaderFetch {
// Retrieve the next header satisfying the query
var origin *types.Header
if hashMode {
origin = pm.blockchain.GetHeaderByHash(query.Origin.Hash)
} else {
origin = pm.blockchain.GetHeaderByNumber(query.Origin.Number)
}
if origin == nil {
break
}
number := origin.Number.Uint64()
headers = append(headers, origin)
bytes += estHeaderRlpSize

// Advance to the next header of the query
switch {
case query.Origin.Hash != (common.Hash{}) && query.Reverse:
// Hash based traversal towards the genesis block
for i := 0; i < int(query.Skip)+1; i++ {
if header := pm.blockchain.GetHeader(query.Origin.Hash, number); header != nil {
query.Origin.Hash = header.ParentHash
number--
} else {
unknown = true
break
}
}
case query.Origin.Hash != (common.Hash{}) && !query.Reverse:
// Hash based traversal towards the leaf block
var (
current = origin.Number.Uint64()
```

```
next    = current + query.Skip + 1
)
if next <= current {
infos, _ := json.MarshalIndent(p.Peer.Info(), "", "  ")
p.Log().Warn("GetBlockHeaders skip overflow attack", "current", current, "skip", query.Skip, "next",
next, "attacker", infos)
unknown = true
} else {
if header := pm.blockchain.GetHeaderByNumber(next); header != nil {
if pm.blockchain.GetBlockHashesFromHash(header.Hash(), query.Skip+1)[query.Skip] ==
query.Origin.Hash {
query.Origin.Hash = header.Hash()
} else {
unknown = true
}
} else {
unknown = true
}
}
case query.Reverse:
// Number based traversal towards the genesis block
if query.Origin.Number >= query.Skip+1 {
query.Origin.Number -= (query.Skip + 1)
} else {
unknown = true
}

case !query.Reverse:
// Number based traversal towards the leaf block
query.Origin.Number += (query.Skip + 1)
}
}
return p.SendBlockHeaders(headers)

case msg.Code == BlockHeadersMsg:
// A batch of headers arrived to one of our previous requests
var headers []*types.Header
if err := msg.Decode(&headers); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// If no headers were received, but we're expending a DAO fork check, maybe it's that
if len(headers) == 0 && p.forkDrop != nil {
```

```
// Possibly an empty reply to the fork header checks, sanity check TDs
verifyDAO := true

// If we already have a DAO header, we can check the peer's TD against it. If
// the peer's ahead of this, it too must have a reply to the DAO check
if daoHeader := pm.blockchain.GetHeaderByNumber(pm.chainconfig.DAOForkBlock.Uint64());
daoHeader != nil {
if _, td := p.Head(); td.Cmp(pm.blockchain.GetTd(daoHeader.Hash(),
daoHeader.Number.Uint64())) >= 0 {
verifyDAO = false
}
}
// If we're seemingly on the same chain, disable the drop timer
if verifyDAO {
p.Log().Debug("Seems to be on the same side of the DAO fork")
p.forkDrop.Stop()
p.forkDrop = nil
return nil
}
}
// Filter out any explicitly requested headers, deliver the rest to the downloader
filter := len(headers) == 1
if filter {
// If it's a potential DAO fork check, validate against the rules
if p.forkDrop != nil && pm.chainconfig.DAOForkBlock.Cmp(headers[0].Number) == 0 {
// Disable the fork drop timer
p.forkDrop.Stop()
p.forkDrop = nil

// Validate the header and either drop the peer or continue
if err := misc.VerifyDAOHeaderExtraData(pm.chainconfig, headers[0]); err != nil {
p.Log().Debug("Verified to be on the other side of the DAO fork, dropping")
return err
}
p.Log().Debug("Verified to be on the same side of the DAO fork")
return nil
}
// Irrelevant of the fork checks, send the header to the fetcher just in case
headers = pm.fetcher.FilterHeaders(headers, time.Now())
}
if len(headers) > 0 || !filter {
err := pm.downloader.DeliverHeaders(p.id, headers)
```

```
if err != nil {
log.Debug("Failed to deliver headers", "err", err)
}
}

case msg.Code == GetBlockBodiesMsg:
// Decode the retrieval message
msgStream := rlp.NewStream(msg.Payload, uint64(msg.Size))
if _, err := msgStream.List(); err != nil {
return err
}
// Gather blocks until the fetch or network limits is reached
var (
hash   common.Hash
bytes  int
bodies []rlp.RawValue
)
for bytes < softResponseLimit && len(bodies) < downloader.MaxBlockFetch {
// Retrieve the hash of the next block
if err := msgStream.Decode(&hash); err == rlp.EOL {
break
} else if err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Retrieve the requested block body, stopping if enough was found
if data := pm.blockchain.GetBodyRLP(hash); len(data) != 0 {
bodies = append(bodies, data)
bytes += len(data)
}
}
return p.SendBlockBodiesRLP(bodies)

case msg.Code == BlockBodiesMsg:
// A batch of block bodies arrived to one of our previous requests
var request blockBodiesData
if err := msg.Decode(&request); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Deliver them all to the downloader for queuing
trasactions := make([][]*types.Transaction, len(request))
uncles := make([][]*types.Header, len(request))
```

```
for i, body := range request {
trasactions[i] = body.Transactions
uncles[i] = body.Uncles
}
// Filter out any explicitly requested bodies, deliver the rest to the downloader
filter := len(trasactions) > 0 || len(uncles) > 0
if filter {
trasactions, uncles = pm.fetcher.FilterBodies(trasactions, uncles, time.Now())
}
if len(trasactions) > 0 || len(uncles) > 0 || !filter {
err := pm.downloader.DeliverBodies(p.id, trasactions, uncles)
if err != nil {
log.Debug("Failed to deliver bodies", "err", err)
}
}

case p.version >= eth63 && msg.Code == GetNodeDataMsg:
// Decode the retrieval message
msgStream := rlp.NewStream(msg.Payload, uint64(msg.Size))
if _, err := msgStream.List(); err != nil {
return err
}
// Gather state data until the fetch or network limits is reached
var (
hash  common.Hash
bytes int
data  [][]byte
)
for bytes < softResponseLimit && len(data) < downloader.MaxStateFetch {
// Retrieve the hash of the next state entry
if err := msgStream.Decode(&hash); err == rlp.EOL {
break
} else if err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Retrieve the requested state entry, stopping if enough was found
if entry, err := pm.chaindb.Get(hash.Bytes()); err == nil {
data = append(data, entry)
bytes += len(entry)
}
}
return p.SendNodeData(data)
```

```
case p.version >= eth63 && msg.Code == NodeDataMsg:
// A batch of node state data arrived to one of our previous requests
var data [][]byte
if err := msg.Decode(&data); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Deliver all to the downloader
if err := pm.downloader.DeliverNodeData(p.id, data); err != nil {
log.Debug("Failed to deliver node state data", "err", err)
}

case p.version >= eth63 && msg.Code == GetReceiptsMsg:
// Decode the retrieval message
msgStream := rlp.NewStream(msg.Payload, uint64(msg.Size))
if _, err := msgStream.List(); err != nil {
return err
}
// Gather state data until the fetch or network limits is reached
var (
hash     common.Hash
bytes    int
receipts []rlp.RawValue
)
for bytes < softResponseLimit && len(receipts) < downloader.MaxReceiptFetch {
// Retrieve the hash of the next block
if err := msgStream.Decode(&hash); err == rlp.EOL {
break
} else if err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Retrieve the requested block's receipts, skipping if unknown to us
results := core.GetBlockReceipts(pm.chaindb, hash, core.GetBlockNumber(pm.chaindb, hash))
if results == nil {
if header := pm.blockchain.GetHeaderByHash(hash); header == nil || header.ReceiptHash !=
types.EmptyRootHash {
continue
}
}
// If known, encode and queue for response packet
if encoded, err := rlp.EncodeToBytes(results); err != nil {
log.Error("Failed to encode receipt", "err", err)
```

```
} else {
receipts = append(receipts, encoded)
bytes += len(encoded)
}
}
return p.SendReceiptsRLP(receipts)

case p.version >= eth63 && msg.Code == ReceiptsMsg:
// A batch of receipts arrived to one of our previous requests
var receipts [][]*types.Receipt
if err := msg.Decode(&receipts); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Deliver all to the downloader
if err := pm.downloader.DeliverReceipts(p.id, receipts); err != nil {
log.Debug("Failed to deliver receipts", "err", err)
}

case msg.Code == NewBlockHashesMsg:
var announces newBlockHashesData
if err := msg.Decode(&announces); err != nil {
return errResp(ErrDecode, "%v: %v", msg, err)
}
// Mark the hashes as present at the remote node
for _, block := range announces {
p.MarkBlock(block.Hash)
}
// Schedule all the unknown hashes for retrieval
unknown := make(newBlockHashesData, 0, len(announces))
for _, block := range announces {
if !pm.blockchain.HasBlock(block.Hash) {
unknown = append(unknown, block)
}
}
for _, block := range unknown {
pm.fetcher.Notify(p.id, block.Hash, block.Number, time.Now(), p.RequestOneHeader,
p.RequestBodies)
}

case msg.Code == NewBlockMsg:
// Retrieve and decode the propagated block
var request newBlockData
```

```
if err := msg.Decode(&request); err != nil {
return errResp(ErrDecode, "%v: %v", msg, err)
}
request.Block.ReceivedAt = msg.ReceivedAt
request.Block.ReceivedFrom = p

// Mark the peer as owning the block and schedule it for import
p.MarkBlock(request.Block.Hash())
pm.fetcher.Enqueue(p.id, request.Block)

// Assuming the block is importable by the peer, but possibly not yet done so,
// calculate the head hash and TD that the peer truly must have.
var (
trueHead = request.Block.ParentHash()
trueTD   = new(big.Int).Sub(request.TD, request.Block.Difficulty())
)
// Update the peers total difficulty if better than the previous
if _, td := p.Head(); trueTD.Cmp(td) > 0 {
p.SetHead(trueHead, trueTD)

// Schedule a sync if above ours. Note, this will not fire a sync for a gap of
// a singe block (as the true TD is below the propagated block), however this
// scenario should easily be covered by the fetcher.
currentBlock := pm.blockchain.CurrentBlock()
if trueTD.Cmp(pm.blockchain.GetTd(currentBlock.Hash(), currentBlock.NumberU64())) > 0 {
go pm.synchronise(p)
}
}

case msg.Code == TxMsg:
// Transactions arrived, make sure we have a valid and fresh chain to handle them
if atomic.LoadUint32(&pm.acceptTxs) == 0 {
break
}
// Transactions can be processed, parse all of them and deliver to the pool
var txs []*types.Transaction
if err := msg.Decode(&txs); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
for i, tx := range txs {
// Validate and mark the remote transaction
if tx == nil {
```

```
return errResp(ErrDecode, "transaction %d is nil", i)
}
p.MarkTransaction(tx.Hash())
}
pm.txpool.AddBatch(txs)

default:
return errResp(ErrInvalidMsgCode, "%v", msg.Code)
}
return nil
}

// BroadcastBlock will either propagate a block to a subset of it's peers, or
// will only announce it's availability (depending what's requested).
func (pm *ProtocolManager) BroadcastBlock(block *types.Block, propagate bool) {
hash := block.Hash()
peers := pm.peers.PeersWithoutBlock(hash)

// If propagation is requested, send to a subset of the peer
if propagate {
// Calculate the TD of the block (it's not imported yet, so block.Td is not valid)
var td *big.Int
if parent := pm.blockchain.GetBlock(block.ParentHash(), block.NumberU64()-1); parent != nil {
td = new(big.Int).Add(block.Difficulty(), pm.blockchain.GetTd(block.ParentHash(),
block.NumberU64()-1))
} else {
log.Error("Propagating dangling block", "number", block.Number(), "hash", hash)
return
}
// Send the block to a subset of our peers
transfer := peers[:int(math.Sqrt(float64(len(peers))))]
for _, peer := range transfer {
peer.SendNewBlock(block, td)
}
log.Trace("Propagated block", "hash", hash, "recipients", len(transfer), "duration",
common.PrettyDuration(time.Since(block.ReceivedAt)))
}
// Otherwise if the block is indeed in out own chain, announce it
if pm.blockchain.HasBlock(hash) {
for _, peer := range peers {
peer.SendNewBlockHashes([]common.Hash{hash}, []uint64{block.NumberU64()})
}
```

```
log.Trace("Announced block", "hash", hash, "recipients", len(peers), "duration",
common.PrettyDuration(time.Since(block.ReceivedAt)))
}
}

// BroadcastTx will propagate a transaction to all peers which are not known to
// already have the given transaction.
func (pm *ProtocolManager) BroadcastTx(hash common.Hash, tx *types.Transaction) {
// Broadcast transaction to a batch of peers not knowing about it
peers := pm.peers.PeersWithoutTx(hash)
//FIXME include this again: peers = peers[:int(math.Sqrt(float64(len(peers))))]
for _, peer := range peers {
peer.SendTransactions(types.Transactions{tx})
}
log.Trace("Broadcast transaction", "hash", hash, "recipients", len(peers))
}

// Mined broadcast loop
func (self *ProtocolManager) minedBroadcastLoop() {
// automatically stops if unsubscribe
for obj := range self.minedBlockSub.Chan() {
switch ev := obj.Data.(type) {
case core.NewMinedBlockEvent:
self.BroadcastBlock(ev.Block, true)  // First propagate block to peers
self.BroadcastBlock(ev.Block, false) // Only then announce to the rest
}
}
}

func (self *ProtocolManager) txBroadcastLoop() {
// automatically stops if unsubscribe
for obj := range self.txSub.Chan() {
event := obj.Data.(core.TxPreEvent)
self.BroadcastTx(event.Tx.Hash(), event.Tx)
}
}

// EthNodeInfo represents a short summary of the Ethereum sub-protocol metadata known
// about the host peer.
type EthNodeInfo struct {
Network    uint64      `json:"network"`     // Ethereum network ID (1=Frontier, 2=Morden, Ropsten=3)
Difficulty *big.Int    `json:"difficulty"` // Total difficulty of the host's blockchain
```

```
Genesis    common.Hash `json:"genesis"`    // SHA3 hash of the host's genesis block
Head       common.Hash `json:"head"`       // SHA3 hash of the host's best owned block
}

// NodeInfo retrieves some protocol metadata about the running host node.
func (self *ProtocolManager) NodeInfo() *EthNodeInfo {
currentBlock := self.blockchain.CurrentBlock()
return &EthNodeInfo{
Network:    self.networkId,
Difficulty: self.blockchain.GetTd(currentBlock.Hash(), currentBlock.NumberU64()),
Genesis:    self.blockchain.Genesis().Hash(),
Head:       currentBlock.Hash(),
}
}

20:F:\git\coin\ethereum\go-ethereum\eth\handler_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"math"
"math/big"
"math/rand"
"testing"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus/ethash"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/state"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/params"
)

var bigTxGas = new(big.Int).SetUint64(params.TxGas)
```

```
// Tests that protocol versions and modes of operations are matched up properly.
func TestProtocolCompatibility(t *testing.T) {
// Define the compatibility chart
tests := []struct {
version    uint
mode       downloader.SyncMode
compatible bool
}{
{61, downloader.FullSync, true}, {62, downloader.FullSync, true}, {63, downloader.FullSync, true},
{61, downloader.FastSync, false}, {62, downloader.FastSync, false}, {63, downloader.FastSync,
true},
}
// Make sure anything we screw up is restored
backup := ProtocolVersions
defer func() { ProtocolVersions = backup }()

// Try all available compatibility configs and check for errors
for i, tt := range tests {
ProtocolVersions = []uint{tt.version}

pm, err := newTestProtocolManager(tt.mode, 0, nil, nil)
if pm != nil {
defer pm.Stop()
}
if (err == nil && !tt.compatible) || (err != nil && tt.compatible) {
t.Errorf("test %d: compatibility mismatch: have error %v, want compatibility %v", i, err,
tt.compatible)
}
}
}

// Tests that block headers can be retrieved from a remote chain based on user queries.
func TestGetBlockHeaders62(t *testing.T) { testGetBlockHeaders(t, 62) }
func TestGetBlockHeaders63(t *testing.T) { testGetBlockHeaders(t, 63) }

func testGetBlockHeaders(t *testing.T, protocol int) {
pm := newTestProtocolManagerMust(t, downloader.FullSync, downloader.MaxHashFetch+15, nil,
nil)
peer, _ := newTestPeer("peer", protocol, pm, true)
defer peer.close()
```

```
// Create a "random" unknown hash for testing
var unknown common.Hash
for i := range unknown {
unknown[i] = byte(i)
}
// Create a batch of tests for various scenarios
limit := uint64(downloader.MaxHeaderFetch)
tests := []struct {
query  *getBlockHeadersData // The query to execute for header retrieval
expect []common.Hash        // The hashes of the block whose headers are expected
}{
// A single random block should be retrievable by hash and number too
{
&getBlockHeadersData{Origin: hashOrNumber{Hash: pm.blockchain.GetBlockByNumber(limit /
2).Hash()}, Amount: 1},
[]common.Hash{pm.blockchain.GetBlockByNumber(limit / 2).Hash()},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 1},
[]common.Hash{pm.blockchain.GetBlockByNumber(limit / 2).Hash()},
},
// Multiple headers should be retrievable in both directions
{
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 3},
[]common.Hash{
pm.blockchain.GetBlockByNumber(limit / 2).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 + 1).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 + 2).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 3, Reverse: true},
[]common.Hash{
pm.blockchain.GetBlockByNumber(limit / 2).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 - 1).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 - 2).Hash(),
},
},
// Multiple headers with skip lists should be retrievable
{
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Skip: 3, Amount: 3},
[]common.Hash{
pm.blockchain.GetBlockByNumber(limit / 2).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 + 4).Hash(),
```

```
pm.blockchain.GetBlockByNumber(limit/2 + 8).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Skip: 3, Amount: 3, Reverse:
true},
[]common.Hash{
pm.blockchain.GetBlockByNumber(limit / 2).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 - 4).Hash(),
pm.blockchain.GetBlockByNumber(limit/2 - 8).Hash(),
},
},
// The chain endpoints should be retrievable
{
&getBlockHeadersData{Origin: hashOrNumber{Number: 0}, Amount: 1},
[]common.Hash{pm.blockchain.GetBlockByNumber(0).Hash()},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number:
pm.blockchain.CurrentBlock().NumberU64()}, Amount: 1},
[]common.Hash{pm.blockchain.CurrentBlock().Hash()},
},
// Ensure protocol limits are honored
{
&getBlockHeadersData{Origin: hashOrNumber{Number:
pm.blockchain.CurrentBlock().NumberU64() - 1}, Amount: limit + 10, Reverse: true},
pm.blockchain.GetBlockHashesFromHash(pm.blockchain.CurrentBlock().Hash(), limit),
},
// Check that requesting more than available is handled gracefully
{
&getBlockHeadersData{Origin: hashOrNumber{Number:
pm.blockchain.CurrentBlock().NumberU64() - 4}, Skip: 3, Amount: 3},
[]common.Hash{
pm.blockchain.GetBlockByNumber(pm.blockchain.CurrentBlock().NumberU64() - 4).Hash(),
pm.blockchain.GetBlockByNumber(pm.blockchain.CurrentBlock().NumberU64()).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: 4}, Skip: 3, Amount: 3, Reverse: true},
[]common.Hash{
pm.blockchain.GetBlockByNumber(4).Hash(),
pm.blockchain.GetBlockByNumber(0).Hash(),
},
},
// Check that requesting more than available is handled gracefully, even if mid skip
```

```
{
&getBlockHeadersData{Origin: hashOrNumber{Number:
pm.blockchain.CurrentBlock().NumberU64() - 4}, Skip: 2, Amount: 3},
[]common.Hash{
pm.blockchain.GetBlockByNumber(pm.blockchain.CurrentBlock().NumberU64() - 4).Hash(),
pm.blockchain.GetBlockByNumber(pm.blockchain.CurrentBlock().NumberU64() - 1).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: 4}, Skip: 2, Amount: 3, Reverse: true},
[]common.Hash{
pm.blockchain.GetBlockByNumber(4).Hash(),
pm.blockchain.GetBlockByNumber(1).Hash(),
},
},
// Check a corner case where requesting more can iterate past the endpoints
{
&getBlockHeadersData{Origin: hashOrNumber{Number: 2}, Amount: 5, Reverse: true},
[]common.Hash{
pm.blockchain.GetBlockByNumber(2).Hash(),
pm.blockchain.GetBlockByNumber(1).Hash(),
pm.blockchain.GetBlockByNumber(0).Hash(),
},
},
// Check a corner case where skipping overflow loops back into the chain start
{
&getBlockHeadersData{Origin: hashOrNumber{Hash:
pm.blockchain.GetBlockByNumber(3).Hash()}, Amount: 2, Reverse: false, Skip: math.MaxUint64 -
1},
[]common.Hash{
pm.blockchain.GetBlockByNumber(3).Hash(),
},
},
// Check a corner case where skipping overflow loops back to the same header
{
&getBlockHeadersData{Origin: hashOrNumber{Hash:
pm.blockchain.GetBlockByNumber(1).Hash()}, Amount: 2, Reverse: false, Skip: math.MaxUint64},
[]common.Hash{
pm.blockchain.GetBlockByNumber(1).Hash(),
},
},
// Check that non existing headers aren't returned
{
```

```
&getBlockHeadersData{Origin: hashOrNumber{Hash: unknown}, Amount: 1},
[]common.Hash{},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number:
pm.blockchain.CurrentBlock().NumberU64() + 1}, Amount: 1},
[]common.Hash{},
},
}
// Run each of the tests and verify the results against the chain
for i, tt := range tests {
// Collect the headers to expect in the response
headers := []*types.Header{}
for _, hash := range tt.expect {
headers = append(headers, pm.blockchain.GetBlockByHash(hash).Header())
}
// Send the hash request and verify the response
p2p.Send(peer.app, 0x03, tt.query)
if err := p2p.ExpectMsg(peer.app, 0x04, headers); err != nil {
t.Errorf("test %d: headers mismatch: %v", i, err)
}
// If the test used number origins, repeat with hashes as the too
if tt.query.Origin.Hash == (common.Hash{}) {
if origin := pm.blockchain.GetBlockByNumber(tt.query.Origin.Number); origin != nil {
tt.query.Origin.Hash, tt.query.Origin.Number = origin.Hash(), 0

p2p.Send(peer.app, 0x03, tt.query)
if err := p2p.ExpectMsg(peer.app, 0x04, headers); err != nil {
t.Errorf("test %d: headers mismatch: %v", i, err)
}
}
}
}
}

// Tests that block contents can be retrieved from a remote chain based on their hashes.
func TestGetBlockBodies62(t *testing.T) { testGetBlockBodies(t, 62) }
func TestGetBlockBodies63(t *testing.T) { testGetBlockBodies(t, 63) }

func testGetBlockBodies(t *testing.T, protocol int) {
pm := newTestProtocolManagerMust(t, downloader.FullSync, downloader.MaxBlockFetch+15, nil,
nil)
peer, _ := newTestPeer("peer", protocol, pm, true)
```

```
defer peer.close()

// Create a batch of tests for various scenarios
limit := downloader.MaxBlockFetch
tests := []struct {
random    int          // Number of blocks to fetch randomly from the chain
explicit  []common.Hash // Explicitly requested blocks
available []bool        // Availability of explicitly requested blocks
expected  int           // Total number of existing blocks to expect
}{
{1, nil, nil, 1},                                    // A single random block should be retrievable
{10, nil, nil, 10},                                  // Multiple random blocks should be retrievable
{limit, nil, nil, limit},                            // The maximum possible blocks should be
retrievable
{limit + 1, nil, nil, limit},                        // No more than the possible block count should
be returned
{0, []common.Hash{pm.blockchain.Genesis().Hash()}, []bool{true}, 1},      // The genesis block
should be retrievable
{0, []common.Hash{pm.blockchain.CurrentBlock().Hash()}, []bool{true}, 1}, // The chains head
block should be retrievable
{0, []common.Hash{{}}, []bool{false}, 0},                    // A non existent block should not be
returned

// Existing and non-existing blocks interleaved should not cause problems
{0, []common.Hash{
{},
pm.blockchain.GetBlockByNumber(1).Hash(),
{},
pm.blockchain.GetBlockByNumber(10).Hash(),
{},
pm.blockchain.GetBlockByNumber(100).Hash(),
{},
}, []bool{false, true, false, true, false, true, false}, 3},
}
// Run each of the tests and verify the results against the chain
for i, tt := range tests {
// Collect the hashes to request, and the response to expect
hashes, seen := []common.Hash{}, make(map[int64]bool)
bodies := []*blockBody{}

for j := 0; j < tt.random; j++ {
for {
```

```
num := rand.Int63n(int64(pm.blockchain.CurrentBlock().NumberU64()))
if !seen[num] {
seen[num] = true

block := pm.blockchain.GetBlockByNumber(uint64(num))
hashes = append(hashes, block.Hash())
if len(bodies) < tt.expected {
bodies = append(bodies, &blockBody{Transactions: block.Transactions(), Uncles: block.Uncles()})
}
break
}
}
}
for j, hash := range tt.explicit {
hashes = append(hashes, hash)
if tt.available[j] && len(bodies) < tt.expected {
block := pm.blockchain.GetBlockByHash(hash)
bodies = append(bodies, &blockBody{Transactions: block.Transactions(), Uncles: block.Uncles()})
}
}
// Send the hash request and verify the response
p2p.Send(peer.app, 0x05, hashes)
if err := p2p.ExpectMsg(peer.app, 0x06, bodies); err != nil {
t.Errorf("test %d: bodies mismatch: %v", i, err)
}
}
}

// Tests that the node state database can be retrieved based on hashes.
func TestGetNodeData63(t *testing.T) { testGetNodeData(t, 63) }

func testGetNodeData(t *testing.T, protocol int) {
// Define three accounts to simulate transactions with
acc1Key, _ :=
crypto.HexToECDSA("8a1f9a8f95be41cd7ccb6168179afb4504aefe388d1e14474d32c45c72ce7b
7a")
acc2Key, _ :=
crypto.HexToECDSA("49a7b37aa6f6645917e7b807e9d1c00d4fa71f18343b0d4122a4d2df64dd6f
ee")
acc1Addr := crypto.PubkeyToAddress(acc1Key.PublicKey)
acc2Addr := crypto.PubkeyToAddress(acc2Key.PublicKey)
```

```
signer := types.HomesteadSigner{}
// Create a chain generator with some simple transactions (blatantly stolen from
@fjl/chain_markets_test)
generator := func(i int, block *core.BlockGen) {
switch i {
case 0:
// In block 1, the test bank sends account #1 some ether.
tx, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBank), acc1Addr,
big.NewInt(10000), bigTxGas, nil, nil), signer, testBankKey)
block.AddTx(tx)
case 1:
// In block 2, the test bank sends some more ether to account #1.
// acc1Addr passes it on to account #2.
tx1, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBank), acc1Addr,
big.NewInt(1000), bigTxGas, nil, nil), signer, testBankKey)
tx2, _ := types.SignTx(types.NewTransaction(block.TxNonce(acc1Addr), acc2Addr,
big.NewInt(1000), bigTxGas, nil, nil), signer, acc1Key)
block.AddTx(tx1)
block.AddTx(tx2)
case 2:
// Block 3 is empty but was mined by account #2.
block.SetCoinbase(acc2Addr)
block.SetExtra([]byte("yeehaw"))
case 3:
// Block 4 includes blocks 2 and 3 as uncle headers (with modified extra data).
b2 := block.PrevBlock(1).Header()
b2.Extra = []byte("foo")
block.AddUncle(b2)
b3 := block.PrevBlock(2).Header()
b3.Extra = []byte("foo")
block.AddUncle(b3)
}
}
// Assemble the test environment
pm := newTestProtocolManagerMust(t, downloader.FullSync, 4, generator, nil)
peer, _ := newTestPeer("peer", protocol, pm, true)
defer peer.close()

// Fetch for now the entire chain db
hashes := []common.Hash{}
for _, key := range pm.chaindb.(*ethdb.MemDatabase).Keys() {
if len(key) == len(common.Hash{}) {
```

```
hashes = append(hashes, common.BytesToHash(key))
}
}
p2p.Send(peer.app, 0x0d, hashes)
msg, err := peer.app.ReadMsg()
if err != nil {
t.Fatalf("failed to read node data response: %v", err)
}
if msg.Code != 0x0e {
t.Fatalf("response packet code mismatch: have %x, want %x", msg.Code, 0x0c)
}
var data [][]byte
if err := msg.Decode(&data); err != nil {
t.Fatalf("failed to decode response node data: %v", err)
}
// Verify that all hashes correspond to the requested data, and reconstruct a state tree
for i, want := range hashes {
if hash := crypto.Keccak256Hash(data[i]); hash != want {
t.Errorf("data hash mismatch: have %x, want %x", hash, want)
}
}
statedb, _ := ethdb.NewMemDatabase()
for i := 0; i < len(data); i++ {
statedb.Put(hashes[i].Bytes(), data[i])
}
accounts := []common.Address{testBank, acc1Addr, acc2Addr}
for i := uint64(0); i <= pm.blockchain.CurrentBlock().NumberU64(); i++ {
trie, _ := state.New(pm.blockchain.GetBlockByNumber(i).Root(), state.NewDatabase(statedb))

for j, acc := range accounts {
state, _ := pm.blockchain.State()
bw := state.GetBalance(acc)
bh := trie.GetBalance(acc)

if (bw != nil && bh == nil) || (bw == nil && bh != nil) {
t.Errorf("test %d, account %d: balance mismatch: have %v, want %v", i, j, bh, bw)
}
if bw != nil && bh != nil && bw.Cmp(bw) != 0 {
t.Errorf("test %d, account %d: balance mismatch: have %v, want %v", i, j, bh, bw)
}
}
}
```

```
}

// Tests that the transaction receipts can be retrieved based on hashes.
func TestGetReceipt63(t *testing.T) { testGetReceipt(t, 63) }

func testGetReceipt(t *testing.T, protocol int) {
// Define three accounts to simulate transactions with
acc1Key, _ :=
crypto.HexToECDSA("8a1f9a8f95be41cd7ccb6168179afb4504aefe388d1e14474d32c45c72ce7b
7a")
acc2Key, _ :=
crypto.HexToECDSA("49a7b37aa6f6645917e7b807e9d1c00d4fa71f18343b0d4122a4d2df64dd6f
ee")
acc1Addr := crypto.PubkeyToAddress(acc1Key.PublicKey)
acc2Addr := crypto.PubkeyToAddress(acc2Key.PublicKey)

signer := types.HomesteadSigner{}
// Create a chain generator with some simple transactions (blatantly stolen from
@fjl/chain_markets_test)
generator := func(i int, block *core.BlockGen) {
switch i {
case 0:
// In block 1, the test bank sends account #1 some ether.
tx, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBank), acc1Addr,
big.NewInt(10000), bigTxGas, nil, nil), signer, testBankKey)
block.AddTx(tx)
case 1:
// In block 2, the test bank sends some more ether to account #1.
// acc1Addr passes it on to account #2.
tx1, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBank), acc1Addr,
big.NewInt(1000), bigTxGas, nil, nil), signer, testBankKey)
tx2, _ := types.SignTx(types.NewTransaction(block.TxNonce(acc1Addr), acc2Addr,
big.NewInt(1000), bigTxGas, nil, nil), signer, acc1Key)
block.AddTx(tx1)
block.AddTx(tx2)
case 2:
// Block 3 is empty but was mined by account #2.
block.SetCoinbase(acc2Addr)
block.SetExtra([]byte("yeehaw"))
case 3:
// Block 4 includes blocks 2 and 3 as uncle headers (with modified extra data).
b2 := block.PrevBlock(1).Header()
```

```
b2.Extra = []byte("foo")
block.AddUncle(b2)
b3 := block.PrevBlock(2).Header()
b3.Extra = []byte("foo")
block.AddUncle(b3)
}
}
// Assemble the test environment
pm := newTestProtocolManagerMust(t, downloader.FullSync, 4, generator, nil)
peer, _ := newTestPeer("peer", protocol, pm, true)
defer peer.close()

// Collect the hashes to request, and the response to expect
hashes, receipts := []common.Hash{}, []types.Receipts{}
for i := uint64(0); i <= pm.blockchain.CurrentBlock().NumberU64(); i++ {
block := pm.blockchain.GetBlockByNumber(i)

hashes = append(hashes, block.Hash())
receipts = append(receipts, core.GetBlockReceipts(pm.chaindb, block.Hash(),
block.NumberU64()))
}
// Send the hash request and verify the response
p2p.Send(peer.app, 0x0f, hashes)
if err := p2p.ExpectMsg(peer.app, 0x10, receipts); err != nil {
t.Errorf("receipts mismatch: %v", err)
}
}

// Tests that post eth protocol handshake, DAO fork-enabled clients also execute
// a DAO "challenge" verifying each others' DAO fork headers to ensure they're on
// compatible chains.
func TestDAOChallengeNoVsNo(t *testing.T)       { testDAOChallenge(t, false, false, false) }
func TestDAOChallengeNoVsPro(t *testing.T)      { testDAOChallenge(t, false, true, false) }
func TestDAOChallengeProVsNo(t *testing.T)      { testDAOChallenge(t, true, false, false) }
func TestDAOChallengeProVsPro(t *testing.T)     { testDAOChallenge(t, true, true, false) }
func TestDAOChallengeNoVsTimeout(t *testing.T)  { testDAOChallenge(t, false, false, true) }
func TestDAOChallengeProVsTimeout(t *testing.T) { testDAOChallenge(t, true, true, true) }

func testDAOChallenge(t *testing.T, localForked, remoteForked bool, timeout bool) {
// Reduce the DAO handshake challenge timeout
if timeout {
defer func(old time.Duration) { daoChallengeTimeout = old }(daoChallengeTimeout)
```

```
daoChallengeTimeout = 500 * time.Millisecond
}
// Create a DAO aware protocol manager
var (
evmux         = new(event.TypeMux)
pow           = ethash.NewFaker()
db, _         = ethdb.NewMemDatabase()
config        = &params.ChainConfig{DAOForkBlock: big.NewInt(1), DAOForkSupport: localForked}
gspec         = &core.Genesis{Config: config}
genesis       = gspec.MustCommit(db)
blockchain, _ = core.NewBlockChain(db, config, pow, evmux, vm.Config{})
)
pm, err := NewProtocolManager(config, downloader.FullSync, DefaultConfig.NetworkId, 1000,
evmux, new(testTxPool), pow, blockchain, db)
if err != nil {
t.Fatalf("failed to start test protocol manager: %v", err)
}
pm.Start()
defer pm.Stop()

// Connect a new peer and check that we receive the DAO challenge
peer, _ := newTestPeer("peer", eth63, pm, true)
defer peer.close()

challenge := &getBlockHeadersData{
Origin:  hashOrNumber{Number: config.DAOForkBlock.Uint64()},
Amount:  1,
Skip:    0,
Reverse: false,
}
if err := p2p.ExpectMsg(peer.app, GetBlockHeadersMsg, challenge); err != nil {
t.Fatalf("challenge mismatch: %v", err)
}
// Create a block to reply to the challenge if no timeout is simulated
if !timeout {
blocks, _ := core.GenerateChain(&params.ChainConfig{}, genesis, db, 1, func(i int, block
*core.BlockGen) {
if remoteForked {
block.SetExtra(params.DAOForkBlockExtra)
}
})
if err := p2p.Send(peer.app, BlockHeadersMsg, []*types.Header{blocks[0].Header()}); err != nil {
```

```
t.Fatalf("failed to answer challenge: %v", err)
}
time.Sleep(100 * time.Millisecond) // Sleep to avoid the verification racing with the drops
} else {
// Otherwise wait until the test timeout passes
time.Sleep(daoChallengeTimeout + 500*time.Millisecond)
}
// Verify that depending on fork side, the remote peer is maintained or dropped
if localForked == remoteForked && !timeout {
if peers := pm.peers.Len(); peers != 1 {
t.Fatalf("peer count mismatch: have %d, want %d", peers, 1)
}
} else {
if peers := pm.peers.Len(); peers != 0 {
t.Fatalf("peer count mismatch: have %d, want %d", peers, 0)
}
}
}
```

21:F:\git\coin\ethereum\go-ethereum\eth\helper_test.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// This file contains some shares testing functionality, common to  multiple
// different files and modules being tested.

package eth

import (
"crypto/ecdsa"
"crypto/rand"
"math/big"
"sort"
"sync"
"testing"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus/ethash"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/eth/downloader"
```

```
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/p2p/discover"
"github.com/ethereum/go-ethereum/params"
)

var (
testBankKey, _ =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
testBank       = crypto.PubkeyToAddress(testBankKey.PublicKey)
)

// newTestProtocolManager creates a new protocol manager for testing purposes,
// with the given number of blocks already known, and potential notification
// channels for different events.
func newTestProtocolManager(mode downloader.SyncMode, blocks int, generator func(int,
*core.BlockGen), newtx chan<- []*types.Transaction) (*ProtocolManager, error) {
var (
evmux  = new(event.TypeMux)
engine = ethash.NewFaker()
db, _  = ethdb.NewMemDatabase()
gspec  = &core.Genesis{
Config: params.TestChainConfig,
Alloc:  core.GenesisAlloc{testBank: {Balance: big.NewInt(1000000)}},
}
genesis       = gspec.MustCommit(db)
blockchain, _ = core.NewBlockChain(db, gspec.Config, engine, evmux, vm.Config{})
)
chain, _ := core.GenerateChain(gspec.Config, genesis, db, blocks, generator)
if _, err := blockchain.InsertChain(chain); err != nil {
panic(err)
}

pm, err := NewProtocolManager(gspec.Config, mode, DefaultConfig.NetworkId, 1000, evmux,
&testTxPool{added: newtx}, engine, blockchain, db)
if err != nil {
return nil, err
}
pm.Start()
return pm, nil
```

```
}

// newTestProtocolManagerMust creates a new protocol manager for testing purposes,
// with the given number of blocks already known, and potential notification
// channels for different events. In case of an error, the constructor force-
// fails the test.
func newTestProtocolManagerMust(t *testing.T, mode downloader.SyncMode, blocks int,
generator func(int, *core.BlockGen), newtx chan<- []*types.Transaction) *ProtocolManager {
pm, err := newTestProtocolManager(mode, blocks, generator, newtx)
if err != nil {
t.Fatalf("Failed to create protocol manager: %v", err)
}
return pm
}

// testTxPool is a fake, helper transaction pool for testing purposes
type testTxPool struct {
pool  []*types.Transaction        // Collection of all transactions
added chan<- []*types.Transaction // Notification channel for new transactions

lock sync.RWMutex // Protects the transaction pool
}

// AddBatch appends a batch of transactions to the pool, and notifies any
// listeners if the addition channel is non nil
func (p *testTxPool) AddBatch(txs []*types.Transaction) error {
p.lock.Lock()
defer p.lock.Unlock()

p.pool = append(p.pool, txs...)
if p.added != nil {
p.added <- txs
}

return nil
}

// Pending returns all the transactions known to the pool
func (p *testTxPool) Pending() (map[common.Address]types.Transactions, error) {
p.lock.RLock()
defer p.lock.RUnlock()
```

```
batches := make(map[common.Address]types.Transactions)
for _, tx := range p.pool {
from, _ := types.Sender(types.HomesteadSigner{}, tx)
batches[from] = append(batches[from], tx)
}
for _, batch := range batches {
sort.Sort(types.TxByNonce(batch))
}
return batches, nil
}

// newTestTransaction create a new dummy transaction.
func newTestTransaction(from *ecdsa.PrivateKey, nonce uint64, datasize int) *types.Transaction {
tx := types.NewTransaction(nonce, common.Address{}, big.NewInt(0), big.NewInt(100000),
big.NewInt(0), make([]byte, datasize))
tx, _ = types.SignTx(tx, types.HomesteadSigner{}, from)
return tx
}

// testPeer is a simulated peer to allow testing direct network calls.
type testPeer struct {
net p2p.MsgReadWriter // Network layer reader/writer to simulate remote messaging
app *p2p.MsgPipeRW    // Application layer reader/writer to simulate the local side
*peer
}

// newTestPeer creates a new peer registered at the given protocol manager.
func newTestPeer(name string, version int, pm *ProtocolManager, shake bool) (*testPeer, <-chan
error) {
// Create a message pipe to communicate through
app, net := p2p.MsgPipe()

// Generate a random id and create the peer
var id discover.NodeID
rand.Read(id[:])

peer := pm.newPeer(version, p2p.NewPeer(id, name, nil), net)

// Start the peer on a new thread
errc := make(chan error, 1)
go func() {
select {
```

```
case pm.newPeerCh <- peer:
errc <- pm.handle(peer)
case <-pm.quitSync:
errc <- p2p.DiscQuitting
}
}()
tp := &testPeer{app: app, net: net, peer: peer}
// Execute any implicitly requested handshakes and return
if shake {
td, head, genesis := pm.blockchain.Status()
tp.handshake(nil, td, head, genesis)
}
return tp, errc
}

// handshake simulates a trivial handshake that expects the same state from the
// remote side as we are simulating locally.
func (p *testPeer) handshake(t *testing.T, td *big.Int, head common.Hash, genesis common.Hash)
{
msg := &statusData{
ProtocolVersion: uint32(p.version),
NetworkId:       DefaultConfig.NetworkId,
TD:              td,
CurrentBlock:    head,
GenesisBlock:    genesis,
}
if err := p2p.ExpectMsg(p.app, StatusMsg, msg); err != nil {
t.Fatalf("status recv: %v", err)
}
if err := p2p.Send(p.app, StatusMsg, msg); err != nil {
t.Fatalf("status send: %v", err)
}
}

// close terminates the local side of the peer, notifying the remote protocol
// manager of termination.
func (p *testPeer) close() {
p.app.Close()
}

22:F:\git\coin\ethereum\go-ethereum\eth\metrics.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package eth

import (
"github.com/ethereum/go-ethereum/metrics"
"github.com/ethereum/go-ethereum/p2p"
)

var (
propTxnInPacketsMeter     = metrics.NewMeter("eth/prop/txns/in/packets")
propTxnInTrafficMeter     = metrics.NewMeter("eth/prop/txns/in/traffic")
propTxnOutPacketsMeter    = metrics.NewMeter("eth/prop/txns/out/packets")
propTxnOutTrafficMeter    = metrics.NewMeter("eth/prop/txns/out/traffic")
propHashInPacketsMeter    = metrics.NewMeter("eth/prop/hashes/in/packets")
propHashInTrafficMeter    = metrics.NewMeter("eth/prop/hashes/in/traffic")
propHashOutPacketsMeter   = metrics.NewMeter("eth/prop/hashes/out/packets")
propHashOutTrafficMeter   = metrics.NewMeter("eth/prop/hashes/out/traffic")
propBlockInPacketsMeter   = metrics.NewMeter("eth/prop/blocks/in/packets")
propBlockInTrafficMeter   = metrics.NewMeter("eth/prop/blocks/in/traffic")
propBlockOutPacketsMeter  = metrics.NewMeter("eth/prop/blocks/out/packets")
propBlockOutTrafficMeter  = metrics.NewMeter("eth/prop/blocks/out/traffic")
reqHeaderInPacketsMeter   = metrics.NewMeter("eth/req/headers/in/packets")
reqHeaderInTrafficMeter   = metrics.NewMeter("eth/req/headers/in/traffic")
reqHeaderOutPacketsMeter  = metrics.NewMeter("eth/req/headers/out/packets")
reqHeaderOutTrafficMeter  = metrics.NewMeter("eth/req/headers/out/traffic")
reqBodyInPacketsMeter     = metrics.NewMeter("eth/req/bodies/in/packets")
reqBodyInTrafficMeter     = metrics.NewMeter("eth/req/bodies/in/traffic")
reqBodyOutPacketsMeter    = metrics.NewMeter("eth/req/bodies/out/packets")
reqBodyOutTrafficMeter    = metrics.NewMeter("eth/req/bodies/out/traffic")
reqStateInPacketsMeter    = metrics.NewMeter("eth/req/states/in/packets")
reqStateInTrafficMeter    = metrics.NewMeter("eth/req/states/in/traffic")
reqStateOutPacketsMeter   = metrics.NewMeter("eth/req/states/out/packets")
reqStateOutTrafficMeter   = metrics.NewMeter("eth/req/states/out/traffic")
reqReceiptInPacketsMeter  = metrics.NewMeter("eth/req/receipts/in/packets")
reqReceiptInTrafficMeter  = metrics.NewMeter("eth/req/receipts/in/traffic")
reqReceiptOutPacketsMeter = metrics.NewMeter("eth/req/receipts/out/packets")
reqReceiptOutTrafficMeter = metrics.NewMeter("eth/req/receipts/out/traffic")
miscInPacketsMeter        = metrics.NewMeter("eth/misc/in/packets")
miscInTrafficMeter        = metrics.NewMeter("eth/misc/in/traffic")
miscOutPacketsMeter       = metrics.NewMeter("eth/misc/out/packets")
miscOutTrafficMeter       = metrics.NewMeter("eth/misc/out/traffic")
)
```

```
// meteredMsgReadWriter is a wrapper around a p2p.MsgReadWriter, capable of
// accumulating the above defined metrics based on the data stream contents.
type meteredMsgReadWriter struct {
p2p.MsgReadWriter     // Wrapped message stream to meter
version           int // Protocol version to select correct meters
}

// newMeteredMsgWriter wraps a p2p MsgReadWriter with metering support. If the
// metrics system is disabled, this function returns the original object.
func newMeteredMsgWriter(rw p2p.MsgReadWriter) p2p.MsgReadWriter {
if !metrics.Enabled {
return rw
}
return &meteredMsgReadWriter{MsgReadWriter: rw}
}

// Init sets the protocol version used by the stream to know which meters to
// increment in case of overlapping message ids between protocol versions.
func (rw *meteredMsgReadWriter) Init(version int) {
rw.version = version
}

func (rw *meteredMsgReadWriter) ReadMsg() (p2p.Msg, error) {
// Read the message and short circuit in case of an error
msg, err := rw.MsgReadWriter.ReadMsg()
if err != nil {
return msg, err
}
// Account for the data traffic
packets, traffic := miscInPacketsMeter, miscInTrafficMeter
switch {
case msg.Code == BlockHeadersMsg:
packets, traffic = reqHeaderInPacketsMeter, reqHeaderInTrafficMeter
case msg.Code == BlockBodiesMsg:
packets, traffic = reqBodyInPacketsMeter, reqBodyInTrafficMeter

case rw.version >= eth63 && msg.Code == NodeDataMsg:
packets, traffic = reqStateInPacketsMeter, reqStateInTrafficMeter
case rw.version >= eth63 && msg.Code == ReceiptsMsg:
packets, traffic = reqReceiptInPacketsMeter, reqReceiptInTrafficMeter
```

```
case msg.Code == NewBlockHashesMsg:
packets, traffic = propHashInPacketsMeter, propHashInTrafficMeter
case msg.Code == NewBlockMsg:
packets, traffic = propBlockInPacketsMeter, propBlockInTrafficMeter
case msg.Code == TxMsg:
packets, traffic = propTxnInPacketsMeter, propTxnInTrafficMeter
}
packets.Mark(1)
traffic.Mark(int64(msg.Size))

return msg, err
}

func (rw *meteredMsgReadWriter) WriteMsg(msg p2p.Msg) error {
// Account for the data traffic
packets, traffic := miscOutPacketsMeter, miscOutTrafficMeter
switch {
case msg.Code == BlockHeadersMsg:
packets, traffic = reqHeaderOutPacketsMeter, reqHeaderOutTrafficMeter
case msg.Code == BlockBodiesMsg:
packets, traffic = reqBodyOutPacketsMeter, reqBodyOutTrafficMeter

case rw.version >= eth63 && msg.Code == NodeDataMsg:
packets, traffic = reqStateOutPacketsMeter, reqStateOutTrafficMeter
case rw.version >= eth63 && msg.Code == ReceiptsMsg:
packets, traffic = reqReceiptOutPacketsMeter, reqReceiptOutTrafficMeter

case msg.Code == NewBlockHashesMsg:
packets, traffic = propHashOutPacketsMeter, propHashOutTrafficMeter
case msg.Code == NewBlockMsg:
packets, traffic = propBlockOutPacketsMeter, propBlockOutTrafficMeter
case msg.Code == TxMsg:
packets, traffic = propTxnOutPacketsMeter, propTxnOutTrafficMeter
}
packets.Mark(1)
traffic.Mark(int64(msg.Size))

// Send the packet to the p2p layer
return rw.MsgReadWriter.WriteMsg(msg)
}
```

23:F:\git\coin\ethereum\go-ethereum\eth\peer.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"errors"
"fmt"
"math/big"
"sync"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/rlp"
"gopkg.in/fatih/set.v0"
)

var (
errClosed            = errors.New("peer set is closed")
errAlreadyRegistered = errors.New("peer is already registered")
errNotRegistered     = errors.New("peer is not registered")
)

const (
maxKnownTxs      = 32768 // Maximum transactions hashes to keep in the known list (prevent
DOS)
maxKnownBlocks   = 1024  // Maximum block hashes to keep in the known list (prevent DOS)
handshakeTimeout = 5 * time.Second
)

// PeerInfo represents a short summary of the Ethereum sub-protocol metadata known
// about a connected peer.
type PeerInfo struct {
Version    int      `json:"version"`    // Ethereum protocol version negotiated
Difficulty *big.Int `json:"difficulty"` // Total difficulty of the peer's blockchain
Head       string   `json:"head"`       // SHA3 hash of the peer's best owned block
}

type peer struct {
id string
```

```
*p2p.Peer
rw p2p.MsgReadWriter

version  int         // Protocol version negotiated
forkDrop *time.Timer // Timed connection dropper if forks aren't validated in time

head common.Hash
td   *big.Int
lock sync.RWMutex

knownTxs    *set.Set // Set of transaction hashes known to be known by this peer
knownBlocks *set.Set // Set of block hashes known to be known by this peer
}

func newPeer(version int, p *p2p.Peer, rw p2p.MsgReadWriter) *peer {
id := p.ID()

return &peer{
Peer:        p,
rw:          rw,
version:     version,
id:          fmt.Sprintf("%x", id[:8]),
knownTxs:    set.New(),
knownBlocks: set.New(),
}
}

// Info gathers and returns a collection of metadata known about a peer.
func (p *peer) Info() *PeerInfo {
hash, td := p.Head()

return &PeerInfo{
Version:    p.version,
Difficulty: td,
Head:       hash.Hex(),
}
}

// Head retrieves a copy of the current head hash and total difficulty of the
// peer.
func (p *peer) Head() (hash common.Hash, td *big.Int) {
p.lock.RLock()
```

```
defer p.lock.RUnlock()

copy(hash[:], p.head[:])
return hash, new(big.Int).Set(p.td)
}

// SetHead updates the head hash and total difficulty of the peer.
func (p *peer) SetHead(hash common.Hash, td *big.Int) {
p.lock.Lock()
defer p.lock.Unlock()

copy(p.head[:], hash[:])
p.td.Set(td)
}

// MarkBlock marks a block as known for the peer, ensuring that the block will
// never be propagated to this particular peer.
func (p *peer) MarkBlock(hash common.Hash) {
// If we reached the memory allowance, drop a previously known block hash
for p.knownBlocks.Size() >= maxKnownBlocks {
p.knownBlocks.Pop()
}
p.knownBlocks.Add(hash)
}

// MarkTransaction marks a transaction as known for the peer, ensuring that it
// will never be propagated to this particular peer.
func (p *peer) MarkTransaction(hash common.Hash) {
// If we reached the memory allowance, drop a previously known transaction hash
for p.knownTxs.Size() >= maxKnownTxs {
p.knownTxs.Pop()
}
p.knownTxs.Add(hash)
}

// SendTransactions sends transactions to the peer and includes the hashes
// in its transaction hash set for future reference.
func (p *peer) SendTransactions(txs types.Transactions) error {
for _, tx := range txs {
p.knownTxs.Add(tx.Hash())
}
return p2p.Send(p.rw, TxMsg, txs)
```

```
}

// SendNewBlockHashes announces the availability of a number of blocks through
// a hash notification.
func (p *peer) SendNewBlockHashes(hashes []common.Hash, numbers []uint64) error {
for _, hash := range hashes {
p.knownBlocks.Add(hash)
}
request := make(newBlockHashesData, len(hashes))
for i := 0; i < len(hashes); i++ {
request[i].Hash = hashes[i]
request[i].Number = numbers[i]
}
return p2p.Send(p.rw, NewBlockHashesMsg, request)
}

// SendNewBlock propagates an entire block to a remote peer.
func (p *peer) SendNewBlock(block *types.Block, td *big.Int) error {
p.knownBlocks.Add(block.Hash())
return p2p.Send(p.rw, NewBlockMsg, []interface{}{block, td})
}

// SendBlockHeaders sends a batch of block headers to the remote peer.
func (p *peer) SendBlockHeaders(headers []*types.Header) error {
return p2p.Send(p.rw, BlockHeadersMsg, headers)
}

// SendBlockBodies sends a batch of block contents to the remote peer.
func (p *peer) SendBlockBodies(bodies []*blockBody) error {
return p2p.Send(p.rw, BlockBodiesMsg, blockBodiesData(bodies))
}

// SendBlockBodiesRLP sends a batch of block contents to the remote peer from
// an already RLP encoded format.
func (p *peer) SendBlockBodiesRLP(bodies []rlp.RawValue) error {
return p2p.Send(p.rw, BlockBodiesMsg, bodies)
}

// SendNodeDataRLP sends a batch of arbitrary internal data, corresponding to the
// hashes requested.
func (p *peer) SendNodeData(data [][]byte) error {
return p2p.Send(p.rw, NodeDataMsg, data)
```

```
}

// SendReceiptsRLP sends a batch of transaction receipts, corresponding to the
// ones requested from an already RLP encoded format.
func (p *peer) SendReceiptsRLP(receipts []rlp.RawValue) error {
return p2p.Send(p.rw, ReceiptsMsg, receipts)
}

// RequestOneHeader is a wrapper around the header query functions to fetch a
// single header. It is used solely by the fetcher.
func (p *peer) RequestOneHeader(hash common.Hash) error {
p.Log().Debug("Fetching single header", "hash", hash)
return p2p.Send(p.rw, GetBlockHeadersMsg, &getBlockHeadersData{Origin:
hashOrNumber{Hash: hash}, Amount: uint64(1), Skip: uint64(0), Reverse: false})
}

// RequestHeadersByHash fetches a batch of blocks' headers corresponding to the
// specified header query, based on the hash of an origin block.
func (p *peer) RequestHeadersByHash(origin common.Hash, amount int, skip int, reverse bool)
error {
p.Log().Debug("Fetching batch of headers", "count", amount, "fromhash", origin, "skip", skip,
"reverse", reverse)
return p2p.Send(p.rw, GetBlockHeadersMsg, &getBlockHeadersData{Origin:
hashOrNumber{Hash: origin}, Amount: uint64(amount), Skip: uint64(skip), Reverse: reverse})
}

// RequestHeadersByNumber fetches a batch of blocks' headers corresponding to the
// specified header query, based on the number of an origin block.
func (p *peer) RequestHeadersByNumber(origin uint64, amount int, skip int, reverse bool) error {
p.Log().Debug("Fetching batch of headers", "count", amount, "fromnum", origin, "skip", skip,
"reverse", reverse)
return p2p.Send(p.rw, GetBlockHeadersMsg, &getBlockHeadersData{Origin:
hashOrNumber{Number: origin}, Amount: uint64(amount), Skip: uint64(skip), Reverse: reverse})
}

// RequestBodies fetches a batch of blocks' bodies corresponding to the hashes
// specified.
func (p *peer) RequestBodies(hashes []common.Hash) error {
p.Log().Debug("Fetching batch of block bodies", "count", len(hashes))
return p2p.Send(p.rw, GetBlockBodiesMsg, hashes)
}
```

```
// RequestNodeData fetches a batch of arbitrary data from a node's known state
// data, corresponding to the specified hashes.
func (p *peer) RequestNodeData(hashes []common.Hash) error {
p.Log().Debug("Fetching batch of state data", "count", len(hashes))
return p2p.Send(p.rw, GetNodeDataMsg, hashes)
}

// RequestReceipts fetches a batch of transaction receipts from a remote node.
func (p *peer) RequestReceipts(hashes []common.Hash) error {
p.Log().Debug("Fetching batch of receipts", "count", len(hashes))
return p2p.Send(p.rw, GetReceiptsMsg, hashes)
}

// Handshake executes the eth protocol handshake, negotiating version number,
// network IDs, difficulties, head and genesis blocks.
func (p *peer) Handshake(network uint64, td *big.Int, head common.Hash, genesis common.Hash)
error {
// Send out own handshake in a new thread
errc := make(chan error, 2)
var status statusData // safe to read after two values have been received from errc

go func() {
errc <- p2p.Send(p.rw, StatusMsg, &statusData{
ProtocolVersion: uint32(p.version),
NetworkId:       network,
TD:              td,
CurrentBlock:    head,
GenesisBlock:    genesis,
})
}()
go func() {
errc <- p.readStatus(network, &status, genesis)
}()
timeout := time.NewTimer(handshakeTimeout)
defer timeout.Stop()
for i := 0; i < 2; i++ {
select {
case err := <-errc:
if err != nil {
return err
}
case <-timeout.C:
```

```
return p2p.DiscReadTimeout
}
}
p.td, p.head = status.TD, status.CurrentBlock
return nil
}

func (p *peer) readStatus(network uint64, status *statusData, genesis common.Hash) (err error) {
msg, err := p.rw.ReadMsg()
if err != nil {
return err
}
if msg.Code != StatusMsg {
return errResp(ErrNoStatusMsg, "first msg has code %x (!= %x)", msg.Code, StatusMsg)
}
if msg.Size > ProtocolMaxMsgSize {
return errResp(ErrMsgTooLarge, "%v > %v", msg.Size, ProtocolMaxMsgSize)
}
// Decode the handshake and make sure everything matches
if err := msg.Decode(&status); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
if status.GenesisBlock != genesis {
return errResp(ErrGenesisBlockMismatch, "%x (!= %x)", status.GenesisBlock[:8], genesis[:8])
}
if status.NetworkId != network {
return errResp(ErrNetworkIdMismatch, "%d (!= %d)", status.NetworkId, network)
}
if int(status.ProtocolVersion) != p.version {
return errResp(ErrProtocolVersionMismatch, "%d (!= %d)", status.ProtocolVersion, p.version)
}
return nil
}

// String implements fmt.Stringer.
func (p *peer) String() string {
return fmt.Sprintf("Peer %s [%s]", p.id,
fmt.Sprintf("eth/%2d", p.version),
)
}

// peerSet represents the collection of active peers currently participating in
```

```
// the Ethereum sub-protocol.
type peerSet struct {
peers  map[string]*peer
lock   sync.RWMutex
closed bool
}

// newPeerSet creates a new peer set to track the active participants.
func newPeerSet() *peerSet {
return &peerSet{
peers: make(map[string]*peer),
}
}

// Register injects a new peer into the working set, or returns an error if the
// peer is already known.
func (ps *peerSet) Register(p *peer) error {
ps.lock.Lock()
defer ps.lock.Unlock()

if ps.closed {
return errClosed
}
if _, ok := ps.peers[p.id]; ok {
return errAlreadyRegistered
}
ps.peers[p.id] = p
return nil
}

// Unregister removes a remote peer from the active set, disabling any further
// actions to/from that particular entity.
func (ps *peerSet) Unregister(id string) error {
ps.lock.Lock()
defer ps.lock.Unlock()

if _, ok := ps.peers[id]; !ok {
return errNotRegistered
}
delete(ps.peers, id)
return nil
}
```

```
// Peer retrieves the registered peer with the given id.
func (ps *peerSet) Peer(id string) *peer {
ps.lock.RLock()
defer ps.lock.RUnlock()

return ps.peers[id]
}

// Len returns if the current number of peers in the set.
func (ps *peerSet) Len() int {
ps.lock.RLock()
defer ps.lock.RUnlock()

return len(ps.peers)
}

// PeersWithoutBlock retrieves a list of peers that do not have a given block in
// their set of known hashes.
func (ps *peerSet) PeersWithoutBlock(hash common.Hash) []*peer {
ps.lock.RLock()
defer ps.lock.RUnlock()

list := make([]*peer, 0, len(ps.peers))
for _, p := range ps.peers {
if !p.knownBlocks.Has(hash) {
list = append(list, p)
}
}
return list
}

// PeersWithoutTx retrieves a list of peers that do not have a given transaction
// in their set of known hashes.
func (ps *peerSet) PeersWithoutTx(hash common.Hash) []*peer {
ps.lock.RLock()
defer ps.lock.RUnlock()

list := make([]*peer, 0, len(ps.peers))
for _, p := range ps.peers {
if !p.knownTxs.Has(hash) {
list = append(list, p)
```

```
}
}
return list
}

// BestPeer retrieves the known peer with the currently highest total difficulty.
func (ps *peerSet) BestPeer() *peer {
ps.lock.RLock()
defer ps.lock.RUnlock()

var (
bestPeer *peer
bestTd   *big.Int
)
for _, p := range ps.peers {
if _, td := p.Head(); bestPeer == nil || td.Cmp(bestTd) > 0 {
bestPeer, bestTd = p, td
}
}
return bestPeer
}

// Close disconnects all peers.
// No new peers can be registered after Close has returned.
func (ps *peerSet) Close() {
ps.lock.Lock()
defer ps.lock.Unlock()

for _, p := range ps.peers {
p.Disconnect(p2p.DiscQuitting)
}
ps.closed = true
}

24:F:\git\coin\ethereum\go-ethereum\eth\protocol.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"fmt"
"io"
```

```
"math/big"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/rlp"
)

// Constants to match up protocol versions and messages
const (
eth62 = 62
eth63 = 63
)

// Official short name of the protocol used during capability negotiation.
var ProtocolName = "eth"

// Supported versions of the eth protocol (first is primary).
var ProtocolVersions = []uint{eth63, eth62}

// Number of implemented message corresponding to different protocol versions.
var ProtocolLengths = []uint64{17, 8}

const ProtocolMaxMsgSize = 10 * 1024 * 1024 // Maximum cap on the size of a protocol message

// eth protocol message codes
const (
// Protocol messages belonging to eth/62
StatusMsg          = 0x00
NewBlockHashesMsg  = 0x01
TxMsg              = 0x02
GetBlockHeadersMsg = 0x03
BlockHeadersMsg    = 0x04
GetBlockBodiesMsg  = 0x05
BlockBodiesMsg     = 0x06
NewBlockMsg        = 0x07

// Protocol messages belonging to eth/63
GetNodeDataMsg = 0x0d
NodeDataMsg    = 0x0e
GetReceiptsMsg = 0x0f
ReceiptsMsg    = 0x10
)
```

```
type errCode int

const (
ErrMsgTooLarge = iota
ErrDecode
ErrInvalidMsgCode
ErrProtocolVersionMismatch
ErrNetworkIdMismatch
ErrGenesisBlockMismatch
ErrNoStatusMsg
ErrExtraStatusMsg
ErrSuspendedPeer
)

func (e errCode) String() string {
return errorToString[int(e)]
}

// XXX change once legacy code is out
var errorToString = map[int]string{
ErrMsgTooLarge:             "Message too long",
ErrDecode:                  "Invalid message",
ErrInvalidMsgCode:          "Invalid message code",
ErrProtocolVersionMismatch: "Protocol version mismatch",
ErrNetworkIdMismatch:       "NetworkId mismatch",
ErrGenesisBlockMismatch:    "Genesis block mismatch",
ErrNoStatusMsg:             "No status message",
ErrExtraStatusMsg:          "Extra status message",
ErrSuspendedPeer:           "Suspended peer",
}

type txPool interface {
// AddBatch should add the given transactions to the pool.
AddBatch([]*types.Transaction) error

// Pending should return pending transactions.
// The slice should be modifiable by the caller.
Pending() (map[common.Address]types.Transactions, error)
}

// statusData is the network packet for the status message.
```

```
type statusData struct {
ProtocolVersion uint32
NetworkId       uint64
TD              *big.Int
CurrentBlock    common.Hash
GenesisBlock    common.Hash
}

// newBlockHashesData is the network packet for the block announcements.
type newBlockHashesData []struct {
Hash   common.Hash // Hash of one particular block being announced
Number uint64      // Number of one particular block being announced
}

// getBlockHeadersData represents a block header query.
type getBlockHeadersData struct {
Origin  hashOrNumber // Block from which to retrieve headers
Amount  uint64       // Maximum number of headers to retrieve
Skip    uint64       // Blocks to skip between consecutive headers
Reverse bool         // Query direction (false = rising towards latest, true = falling towards genesis)
}

// hashOrNumber is a combined field for specifying an origin block.
type hashOrNumber struct {
Hash   common.Hash // Block hash from which to retrieve headers (excludes Number)
Number uint64      // Block hash from which to retrieve headers (excludes Hash)
}

// EncodeRLP is a specialized encoder for hashOrNumber to encode only one of the
// two contained union fields.
func (hn *hashOrNumber) EncodeRLP(w io.Writer) error {
if hn.Hash == (common.Hash{}) {
return rlp.Encode(w, hn.Number)
}
if hn.Number != 0 {
return fmt.Errorf("both origin hash (%x) and number (%d) provided", hn.Hash, hn.Number)
}
return rlp.Encode(w, hn.Hash)
}

// DecodeRLP is a specialized decoder for hashOrNumber to decode the contents
// into either a block hash or a block number.
```

```
func (hn *hashOrNumber) DecodeRLP(s *rlp.Stream) error {
_, size, _ := s.Kind()
origin, err := s.Raw()
if err == nil {
switch {
case size == 32:
err = rlp.DecodeBytes(origin, &hn.Hash)
case size <= 8:
err = rlp.DecodeBytes(origin, &hn.Number)
default:
err = fmt.Errorf("invalid input size %d for origin", size)
}
}
return err
}

// newBlockData is the network packet for the block propagation message.
type newBlockData struct {
Block *types.Block
TD    *big.Int
}

// blockBody represents the data content of a single block.
type blockBody struct {
Transactions []*types.Transaction // Transactions contained within a block
Uncles       []*types.Header      // Uncles contained within a block
}

// blockBodiesData is the network packet for block content distribution.
type blockBodiesData []*blockBody

25:F:\git\coin\ethereum\go-ethereum\eth\protocol_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"fmt"
"sync"
"testing"
"time"
```

```
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/rlp"
)

func init() {
// log.Root().SetHandler(log.LvlFilterHandler(log.LvlTrace, log.StreamHandler(os.Stderr,
log.TerminalFormat(false))))
}

var testAccount, _ =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")

// Tests that handshake failures are detected and reported correctly.
func TestStatusMsgErrors62(t *testing.T) { testStatusMsgErrors(t, 62) }
func TestStatusMsgErrors63(t *testing.T) { testStatusMsgErrors(t, 63) }

func testStatusMsgErrors(t *testing.T, protocol int) {
pm := newTestProtocolManagerMust(t, downloader.FullSync, 0, nil, nil)
td, currentBlock, genesis := pm.blockchain.Status()
defer pm.Stop()

tests := []struct {
code      uint64
data      interface{}
wantError error
}{
{
code: TxMsg, data: []interface{}{},
wantError: errResp(ErrNoStatusMsg, "first msg has code 2 (!= 0)"),
},
{
code: StatusMsg, data: statusData{10, DefaultConfig.NetworkId, td, currentBlock, genesis},
wantError: errResp(ErrProtocolVersionMismatch, "10 (!= %d)", protocol),
},
{
code: StatusMsg, data: statusData{uint32(protocol), 999, td, currentBlock, genesis},
wantError: errResp(ErrNetworkIdMismatch, "999 (!= 1)"),
```

```
},
{
code: StatusMsg, data: statusData{uint32(protocol), DefaultConfig.NetworkId, td, currentBlock,
common.Hash{3}},
wantError: errResp(ErrGenesisBlockMismatch, "0300000000000000 (!= %x)", genesis[:8]),
},
}

for i, test := range tests {
p, errc := newTestPeer("peer", protocol, pm, false)
// The send call might hang until reset because
// the protocol might not read the payload.
go p2p.Send(p.app, test.code, test.data)

select {
case err := <-errc:
if err == nil {
t.Errorf("test %d: protocol returned nil error, want %q", i, test.wantError)
} else if err.Error() != test.wantError.Error() {
t.Errorf("test %d: wrong error: got %q, want %q", i, err, test.wantError)
}
case <-time.After(2 * time.Second):
t.Errorf("protocol did not shut down within 2 seconds")
}
p.close()
}
}

// This test checks that received transactions are added to the local pool.
func TestRecvTransactions62(t *testing.T) { testRecvTransactions(t, 62) }
func TestRecvTransactions63(t *testing.T) { testRecvTransactions(t, 63) }

func testRecvTransactions(t *testing.T, protocol int) {
txAdded := make(chan []*types.Transaction)
pm := newTestProtocolManagerMust(t, downloader.FullSync, 0, nil, txAdded)
pm.acceptTxs = 1 // mark synced to accept transactions
p, _ := newTestPeer("peer", protocol, pm, true)
defer pm.Stop()
defer p.close()

tx := newTestTransaction(testAccount, 0, 0)
if err := p2p.Send(p.app, TxMsg, []interface{}{tx}); err != nil {
```

```
t.Fatalf("send error: %v", err)
}
select {
case added := <-txAdded:
if len(added) != 1 {
t.Errorf("wrong number of added transactions: got %d, want 1", len(added))
} else if added[0].Hash() != tx.Hash() {
t.Errorf("added wrong tx hash: got %v, want %v", added[0].Hash(), tx.Hash())
}
case <-time.After(2 * time.Second):
t.Errorf("no TxPreEvent received within 2 seconds")
}
}

// This test checks that pending transactions are sent.
func TestSendTransactions62(t *testing.T) { testSendTransactions(t, 62) }
func TestSendTransactions63(t *testing.T) { testSendTransactions(t, 63) }

func testSendTransactions(t *testing.T, protocol int) {
pm := newTestProtocolManagerMust(t, downloader.FullSync, 0, nil, nil)
defer pm.Stop()

// Fill the pool with big transactions.
const txsize = txsyncPackSize / 10
alltxs := make([]*types.Transaction, 100)
for nonce := range alltxs {
alltxs[nonce] = newTestTransaction(testAccount, uint64(nonce), txsize)
}
pm.txpool.AddBatch(alltxs)

// Connect several peers. They should all receive the pending transactions.
var wg sync.WaitGroup
checktxs := func(p *testPeer) {
defer wg.Done()
defer p.close()
seen := make(map[common.Hash]bool)
for _, tx := range alltxs {
seen[tx.Hash()] = false
}
for n := 0; n < len(alltxs) && !t.Failed(); {
var txs []*types.Transaction
msg, err := p.app.ReadMsg()
```

```
if err != nil {
t.Errorf("%v: read error: %v", p.Peer, err)
} else if msg.Code != TxMsg {
t.Errorf("%v: got code %d, want TxMsg", p.Peer, msg.Code)
}
if err := msg.Decode(&txs); err != nil {
t.Errorf("%v: %v", p.Peer, err)
}
for _, tx := range txs {
hash := tx.Hash()
seentx, want := seen[hash]
if seentx {
t.Errorf("%v: got tx more than once: %x", p.Peer, hash)
}
if !want {
t.Errorf("%v: got unexpected tx: %x", p.Peer, hash)
}
seen[hash] = true
n++
}
}
}
for i := 0; i < 3; i++ {
p, _ := newTestPeer(fmt.Sprintf("peer #%d", i), protocol, pm, true)
wg.Add(1)
go checktxs(p)
}
wg.Wait()
}

// Tests that the custom union field encoder and decoder works correctly.
func TestGetBlockHeadersDataEncodeDecode(t *testing.T) {
// Create a "random" hash for testing
var hash common.Hash
for i := range hash {
hash[i] = byte(i)
}
// Assemble some table driven tests
tests := []struct {
packet *getBlockHeadersData
fail   bool
}{
```

```
// Providing the origin as either a hash or a number should both work
{fail: false, packet: &getBlockHeadersData{Origin: hashOrNumber{Number: 314}}},
{fail: false, packet: &getBlockHeadersData{Origin: hashOrNumber{Hash: hash}}},

// Providing arbitrary query field should also work
{fail: false, packet: &getBlockHeadersData{Origin: hashOrNumber{Number: 314}, Amount: 314,
Skip: 1, Reverse: true}},
{fail: false, packet: &getBlockHeadersData{Origin: hashOrNumber{Hash: hash}, Amount: 314,
Skip: 1, Reverse: true}},

// Providing both the origin hash and origin number must fail
{fail: true, packet: &getBlockHeadersData{Origin: hashOrNumber{Hash: hash, Number: 314}}},
}
// Iterate over each of the tests and try to encode and then decode
for i, tt := range tests {
bytes, err := rlp.EncodeToBytes(tt.packet)
if err != nil && !tt.fail {
t.Fatalf("test %d: failed to encode packet: %v", i, err)
} else if err == nil && tt.fail {
t.Fatalf("test %d: encode should have failed", i)
}
if !tt.fail {
packet := new(getBlockHeadersData)
if err := rlp.DecodeBytes(bytes, packet); err != nil {
t.Fatalf("test %d: failed to decode packet: %v", i, err)
}
if packet.Origin.Hash != tt.packet.Origin.Hash || packet.Origin.Number != tt.packet.Origin.Number
|| packet.Amount != tt.packet.Amount ||
packet.Skip != tt.packet.Skip || packet.Reverse != tt.packet.Reverse {
t.Fatalf("test %d: encode decode mismatch: have %+v, want %+v", i, packet, tt.packet)
}
}
}
}

26:F:\git\coin\ethereum\go-ethereum\eth\sync.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"math/rand"
```

```
"sync/atomic"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/p2p/discover"
)

const (
forceSyncCycle      = 10 * time.Second // Time interval to force syncs, even if few peers are
available
minDesiredPeerCount = 5                // Amount of peers desired to start syncing

// This is the target size for the packs of transactions sent by txsyncLoop.
// A pack can get larger than this if a single transactions exceeds this size.
txsyncPackSize = 100 * 1024
)

type txsync struct {
p   *peer
txs []*types.Transaction
}

// syncTransactions starts sending all currently pending transactions to the given peer.
func (pm *ProtocolManager) syncTransactions(p *peer) {
var txs types.Transactions
pending, _ := pm.txpool.Pending()
for _, batch := range pending {
txs = append(txs, batch...)
}
if len(txs) == 0 {
return
}
select {
case pm.txsyncCh <- &txsync{p, txs}:
case <-pm.quitSync:
}
}

// txsyncLoop takes care of the initial transaction sync for each new
```

```
// connection. When a new peer appears, we relay all currently pending
// transactions. In order to minimise egress bandwidth usage, we send
// the transactions in small packs to one peer at a time.
func (pm *ProtocolManager) txsyncLoop() {
var (
pending = make(map[discover.NodeID]*txsync)
sending = false               // whether a send is active
pack    = new(txsync)         // the pack that is being sent
done    = make(chan error, 1) // result of the send
)

// send starts a sending a pack of transactions from the sync.
send := func(s *txsync) {
// Fill pack with transactions up to the target size.
size := common.StorageSize(0)
pack.p = s.p
pack.txs = pack.txs[:0]
for i := 0; i < len(s.txs) && size < txsyncPackSize; i++ {
pack.txs = append(pack.txs, s.txs[i])
size += s.txs[i].Size()
}
// Remove the transactions that will be sent.
s.txs = s.txs[:copy(s.txs, s.txs[len(pack.txs):])]
if len(s.txs) == 0 {
delete(pending, s.p.ID())
}
// Send the pack in the background.
s.p.Log().Trace("Sending batch of transactions", "count", len(pack.txs), "bytes", size)
sending = true
go func() { done <- pack.p.SendTransactions(pack.txs) }()
}

// pick chooses the next pending sync.
pick := func() *txsync {
if len(pending) == 0 {
return nil
}
n := rand.Intn(len(pending)) + 1
for _, s := range pending {
if n--; n == 0 {
return s
}
```

```
}
return nil
}

for {
select {
case s := <-pm.txsyncCh:
pending[s.p.ID()] = s
if !sending {
send(s)
}
case err := <-done:
sending = false
// Stop tracking peers that cause send failures.
if err != nil {
pack.p.Log().Debug("Transaction send failed", "err", err)
delete(pending, pack.p.ID())
}
// Schedule the next send.
if s := pick(); s != nil {
send(s)
}
case <-pm.quitSync:
return
}
}
}

// syncer is responsible for periodically synchronising with the network, both
// downloading hashes and blocks as well as handling the announcement handler.
func (pm *ProtocolManager) syncer() {
// Start and ensure cleanup of sync mechanisms
pm.fetcher.Start()
defer pm.fetcher.Stop()
defer pm.downloader.Terminate()

// Wait for different events to fire synchronisation operations
forceSync := time.Tick(forceSyncCycle)
for {
select {
case <-pm.newPeerCh:
// Make sure we have peers to select from, then sync
```

```
if pm.peers.Len() < minDesiredPeerCount {
break
}
go pm.synchronise(pm.peers.BestPeer())

case <-forceSync:
// Force a sync even if not enough peers are present
go pm.synchronise(pm.peers.BestPeer())

case <-pm.noMorePeers:
return
}
}
}

// synchronise tries to sync up our local block chain with a remote peer.
func (pm *ProtocolManager) synchronise(peer *peer) {
// Short circuit if no peers are available
if peer == nil {
return
}
// Make sure the peer's TD is higher than our own
currentBlock := pm.blockchain.CurrentBlock()
td := pm.blockchain.GetTd(currentBlock.Hash(), currentBlock.NumberU64())

pHead, pTd := peer.Head()
if pTd.Cmp(td) <= 0 {
return
}
// Otherwise try to sync with the downloader
mode := downloader.FullSync
if atomic.LoadUint32(&pm.fastSync) == 1 {
// Fast sync was explicitly requested, and explicitly granted
mode = downloader.FastSync
} else if currentBlock.NumberU64() == 0 && pm.blockchain.CurrentFastBlock().NumberU64() > 0 {
// The database seems empty as the current block is the genesis. Yet the fast
// block is ahead, so fast sync was enabled for this node at a certain point.
// The only scenario where this can happen is if the user manually (or via a
// bad block) rolled back a fast sync node below the sync point. In this case
// however it's safe to reenable fast sync.
atomic.StoreUint32(&pm.fastSync, 1)
mode = downloader.FastSync
```

```
}
if err := pm.downloader.Synchronise(peer.id, pHead, pTd, mode); err != nil {
return
}
atomic.StoreUint32(&pm.acceptTxs, 1) // Mark initial sync done
if head := pm.blockchain.CurrentBlock(); head.NumberU64() > 0 {
// We've completed a sync cycle, notify all peers of new state. This path is
// essential in star-topology networks where a gateway node needs to notify
// all its out-of-date peers of the availability of a new block. This failure
// scenario will most often crop up in private and hackathon networks with
// degenerate connectivity, but it should be healthy for the mainnet too to
// more reliably update peers or the local TD state.
go pm.BroadcastBlock(head, false)
}
// If fast sync was enabled, and we synced up, disable it
if atomic.LoadUint32(&pm.fastSync) == 1 {
// Disable fast sync if we indeed have something in our chain
if pm.blockchain.CurrentBlock().NumberU64() > 0 {
log.Info("Fast sync complete, auto disabling")
atomic.StoreUint32(&pm.fastSync, 0)
}
}
}
```

27:F:\git\coin\ethereum\go-ethereum\eth\sync_test.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package eth

import (
"sync/atomic"
"testing"
"time"

"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/p2p/discover"
)

// Tests that fast sync gets disabled as soon as a real block is successfully
// imported into the blockchain.
func TestFastSyncDisabling(t *testing.T) {
```

```
// Create a pristine protocol manager, check that fast sync is left enabled
pmEmpty := newTestProtocolManagerMust(t, downloader.FastSync, 0, nil, nil)
if atomic.LoadUint32(&pmEmpty.fastSync) == 0 {
t.Fatalf("fast sync disabled on pristine blockchain")
}
// Create a full protocol manager, check that fast sync gets disabled
pmFull := newTestProtocolManagerMust(t, downloader.FastSync, 1024, nil, nil)
if atomic.LoadUint32(&pmFull.fastSync) == 1 {
t.Fatalf("fast sync not disabled on non-empty blockchain")
}
// Sync up the two peers
io1, io2 := p2p.MsgPipe()

go pmFull.handle(pmFull.newPeer(63, p2p.NewPeer(discover.NodeID{}, "empty", nil), io2))
go pmEmpty.handle(pmEmpty.newPeer(63, p2p.NewPeer(discover.NodeID{}, "full", nil), io1))

time.Sleep(250 * time.Millisecond)
pmEmpty.synchronise(pmEmpty.peers.BestPeer())

// Check that fast sync was disabled
if atomic.LoadUint32(&pmEmpty.fastSync) == 1 {
t.Fatalf("fast sync not disabled after successful synchronisation")
}
}

28:F:\git\coin\ethereum\go-ethereum\ethclient\ethclient.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package ethclient provides a client for the Ethereum RPC API.
package ethclient

import (
"context"
"encoding/json"
"fmt"
"math/big"

"github.com/ethereum/go-ethereum"
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/hexutil"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/rlp"
```

```
"github.com/ethereum/go-ethereum/rpc"
)

// Client defines 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) {
c, err := rpc.Dial(rawurl)
if err != nil {
return nil, err
}
return NewClient(c), nil
}

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

// Blockchain Access

// BlockByHash returns the given full block.
//
// 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) {
return ec.getBlock(ctx, "eth_getBlockByHash", hash, true)
}

// BlockByNumber returns a block from the current canonical chain. If number is nil, the
// latest known block is returned.
//
// 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) {
return ec.getBlock(ctx, "eth_getBlockByNumber", toBlockNumArg(number), true)
}

type rpcBlock struct {
```

```
Hash         common.Hash          `json:"hash"`
Transactions []*types.Transaction `json:"transactions"`
UncleHashes  []common.Hash        `json:"uncles"`
}

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 {
return nil, err
} else if len(raw) == 0 {
return nil, ethereum.NotFound
}
// Decode header and transactions.
var head *types.Header
var body rpcBlock
if err := json.Unmarshal(raw, &head); err != nil {
return nil, err
}
if err := json.Unmarshal(raw, &body); err != nil {
return nil, err
}
// Quick-verify transaction and uncle lists. This mostly helps with debugging the server.
if head.UncleHash == types.EmptyUncleHash && len(body.UncleHashes) > 0 {
return nil, fmt.Errorf("server returned non-empty uncle list but block header indicates no uncles")
}
if head.UncleHash != types.EmptyUncleHash && len(body.UncleHashes) == 0 {
return nil, fmt.Errorf("server returned empty uncle list but block header indicates uncles")
}
if head.TxHash == types.EmptyRootHash && len(body.Transactions) > 0 {
return nil, fmt.Errorf("server returned non-empty transaction list but block header indicates no
transactions")
}
if head.TxHash != types.EmptyRootHash && len(body.Transactions) == 0 {
return nil, fmt.Errorf("server returned empty transaction list but block header indicates
transactions")
}
// Load uncles because they are not included in the block response.
var uncles []*types.Header
if len(body.UncleHashes) > 0 {
uncles = make([]*types.Header, len(body.UncleHashes))
```

```
reqs := make([]rpc.BatchElem, len(body.UncleHashes))
for i := range reqs {
reqs[i] = rpc.BatchElem{
Method: "eth_getUncleByBlockHashAndIndex",
Args:   []interface{}{body.Hash, hexutil.EncodeUint64(uint64(i))},
Result: &uncles[i],
}
}
if err := ec.c.BatchCallContext(ctx, reqs); err != nil {
return nil, err
}
for i := range reqs {
if reqs[i].Error != nil {
return nil, reqs[i].Error
}
if uncles[i] == nil {
return nil, fmt.Errorf("got null header for uncle %d of block %x", i, body.Hash[:])
}
}
}
return types.NewBlockWithHeader(head).WithBody(body.Transactions, uncles), nil
}

// HeaderByHash returns the block header with the given hash.
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 {
err = ethereum.NotFound
}
return head, err
}

// 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) {
var head *types.Header
err := ec.c.CallContext(ctx, &head, "eth_getBlockByNumber", toBlockNumArg(number), false)
if err == nil && head == nil {
err = ethereum.NotFound
}
return head, err
```

```
}

// 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) {
var raw json.RawMessage
err = ec.c.CallContext(ctx, &raw, "eth_getTransactionByHash", hash)
if err != nil {
return nil, false, err
} else if len(raw) == 0 {
return nil, false, ethereum.NotFound
}
if err := json.Unmarshal(raw, &tx); err != nil {
return nil, false, err
} else if _, r, _ := tx.RawSignatureValues(); r == nil {
return nil, false, fmt.Errorf("server returned transaction without signature")
}
var block struct{ BlockNumber *string }
if err := json.Unmarshal(raw, &block); err != nil {
return nil, false, err
}
return tx, block.BlockNumber == nil, nil
}

// TransactionCount returns the total number of transactions in the given block.
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) {
var tx *types.Transaction
err := ec.c.CallContext(ctx, &tx, "eth_getTransactionByBlockHashAndIndex", blockHash,
hexutil.Uint64(index))
if err == nil {
if tx == nil {
return nil, ethereum.NotFound
} else if _, r, _ := tx.RawSignatureValues(); r == nil {
return nil, fmt.Errorf("server returned transaction without signature")
```

```
}
}
return tx, err
}

// 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) {
var r *types.Receipt
err := ec.c.CallContext(ctx, &r, "eth_getTransactionReceipt", txHash)
if err == nil {
if r == nil {
return nil, ethereum.NotFound
} else if len(r.PostState) == 0 {
return nil, fmt.Errorf("server returned receipt without post state")
}
}
return r, err
}

func toBlockNumArg(number *big.Int) string {
if number == nil {
return "latest"
}
return hexutil.EncodeBig(number)
}

type rpcProgress struct {
StartingBlock hexutil.Uint64
CurrentBlock  hexutil.Uint64
HighestBlock  hexutil.Uint64
PulledStates  hexutil.Uint64
KnownStates   hexutil.Uint64
}

// SyncProgress retrieves the current progress of the sync algorithm. If there's
// no sync currently running, it returns nil.
func (ec *Client) SyncProgress(ctx context.Context) (*ethereum.SyncProgress, error) {
var raw json.RawMessage
if err := ec.c.CallContext(ctx, &raw, "eth_syncing"); err != nil {
return nil, err
```

```
}
// Handle the possible response types
var syncing bool
if err := json.Unmarshal(raw, &syncing); err == nil {
return nil, nil // Not syncing (always false)
}
var progress *rpcProgress
if err := json.Unmarshal(raw, &progress); err != nil {
return nil, err
}
return &ethereum.SyncProgress{
StartingBlock: uint64(progress.StartingBlock),
CurrentBlock:  uint64(progress.CurrentBlock),
HighestBlock:  uint64(progress.HighestBlock),
PulledStates:  uint64(progress.PulledStates),
KnownStates:   uint64(progress.KnownStates),
}, nil
}

// SubscribeNewHead subscribes to notifications about the current blockchain head
// on the given channel.
func (ec *Client) SubscribeNewHead(ctx context.Context, ch chan<- *types.Header)
(ethereum.Subscription, error) {
return ec.c.EthSubscribe(ctx, ch, "newHeads", map[string]struct{}{})
}

// State Access

// 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) {
var result hexutil.Big
err := ec.c.CallContext(ctx, &result, "eth_getBalance", account, toBlockNumArg(blockNumber))
return (*big.Int)(&result), err
}

// 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) {
var result hexutil.Bytes
```

```
err := ec.c.CallContext(ctx, &result, "eth_getStorageAt", account, key,
toBlockNumArg(blockNumber))
return result, err
}

// 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) {
var result hexutil.Bytes
err := ec.c.CallContext(ctx, &result, "eth_getCode", account, toBlockNumArg(blockNumber))
return result, err
}

// 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) {
var result hexutil.Uint64
err := ec.c.CallContext(ctx, &result, "eth_getTransactionCount", account,
toBlockNumArg(blockNumber))
return uint64(result), err
}

// Filters

// FilterLogs executes a filter query.
func (ec *Client) FilterLogs(ctx context.Context, q ethereum.FilterQuery) ([]types.Log, error) {
var result []types.Log
err := ec.c.CallContext(ctx, &result, "eth_getLogs", toFilterArg(q))
return result, err
}

// SubscribeFilterLogs subscribes to the results of a streaming filter query.
func (ec *Client) SubscribeFilterLogs(ctx context.Context, q ethereum.FilterQuery, ch chan<-
types.Log) (ethereum.Subscription, error) {
return ec.c.EthSubscribe(ctx, ch, "logs", toFilterArg(q))
}

func toFilterArg(q ethereum.FilterQuery) interface{} {
arg := map[string]interface{}{
"fromBlock": toBlockNumArg(q.FromBlock),
```

```
"toBlock":   toBlockNumArg(q.ToBlock),
"address":   q.Addresses,
"topics":    q.Topics,
}
if q.FromBlock == nil {
arg["fromBlock"] = "0x0"
}
return arg
}

// Pending State

// PendingBalanceAt returns the wei balance of the given account in the pending state.
func (ec *Client) PendingBalanceAt(ctx context.Context, account common.Address) (*big.Int,
error) {
var result hexutil.Big
err := ec.c.CallContext(ctx, &result, "eth_getBalance", account, "pending")
return (*big.Int)(&result), err
}

// PendingStorageAt returns the value of key in the contract storage of the given account in the
pending state.
func (ec *Client) PendingStorageAt(ctx context.Context, account common.Address, key
common.Hash) ([]byte, error) {
var result hexutil.Bytes
err := ec.c.CallContext(ctx, &result, "eth_getStorageAt", account, key, "pending")
return result, err
}

// PendingCodeAt returns the contract code of the given account in the pending state.
func (ec *Client) PendingCodeAt(ctx context.Context, account common.Address) ([]byte, error) {
var result hexutil.Bytes
err := ec.c.CallContext(ctx, &result, "eth_getCode", account, "pending")
return result, err
}

// PendingNonceAt returns the account nonce of the given account in the pending state.
// This is the nonce that should be used for the next transaction.
func (ec *Client) PendingNonceAt(ctx context.Context, account common.Address) (uint64, error) {
var result hexutil.Uint64
err := ec.c.CallContext(ctx, &result, "eth_getTransactionCount", account, "pending")
return uint64(result), err
```

```
}

// PendingTransactionCount returns the total number of transactions in the pending state.
func (ec *Client) PendingTransactionCount(ctx context.Context) (uint, error) {
var num hexutil.Uint
err := ec.c.CallContext(ctx, &num, "eth_getBlockTransactionCountByNumber", "pending")
return uint(num), err
}

// TODO: SubscribePendingTransactions (needs server side)

// Contract Calling

// CallContract executes a message call transaction, which is directly executed in the VM
// of the node, but never mined into the blockchain.
//
// 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 ethereum.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 {
return nil, err
}
return hex, nil
}

// PendingCallContract executes a message call transaction using the EVM.
// The state seen by the contract call is the pending state.
func (ec *Client) PendingCallContract(ctx context.Context, msg ethereum.CallMsg) ([]byte, error) {
var hex hexutil.Bytes
err := ec.c.CallContext(ctx, &hex, "eth_call", toCallArg(msg), "pending")
if err != nil {
return nil, err
}
return hex, nil
}

// 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) {
var hex hexutil.Big
if err := ec.c.CallContext(ctx, &hex, "eth_gasPrice"); err != nil {
return nil, err
}
return (*big.Int)(&hex), nil
}

// EstimateGas tries to estimate the gas needed to execute a specific transaction based on
// 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 ethereum.CallMsg) (*big.Int, error) {
var hex hexutil.Big
err := ec.c.CallContext(ctx, &hex, "eth_estimateGas", toCallArg(msg))
if err != nil {
return nil, err
}
return (*big.Int)(&hex), nil
}

// SendTransaction injects a signed transaction into the pending pool for execution.
//
// 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 {
data, err := rlp.EncodeToBytes(tx)
if err != nil {
return err
}
return ec.c.CallContext(ctx, nil, "eth_sendRawTransaction", common.ToHex(data))
}

func toCallArg(msg ethereum.CallMsg) interface{} {
arg := map[string]interface{}{
"from": msg.From,
"to":   msg.To,
}
if len(msg.Data) > 0 {
arg["data"] = hexutil.Bytes(msg.Data)
}
if msg.Value != nil {
```

```
arg["value"] = (*hexutil.Big)(msg.Value)
}
if msg.Gas != nil {
arg["gas"] = (*hexutil.Big)(msg.Gas)
}
if msg.GasPrice != nil {
arg["gasPrice"] = (*hexutil.Big)(msg.GasPrice)
}
return arg
}

29:F:\git\coin\ethereum\go-ethereum\ethclient\ethclient_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethclient

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

// Verify that Client implements the ethereum interfaces.
var (
_ = ethereum.ChainReader(&Client{})
_ = ethereum.TransactionReader(&Client{})
_ = ethereum.ChainStateReader(&Client{})
_ = ethereum.ChainSyncReader(&Client{})
_ = ethereum.ContractCaller(&Client{})
_ = ethereum.GasEstimator(&Client{})
_ = ethereum.GasPricer(&Client{})
_ = ethereum.LogFilterer(&Client{})
_ = ethereum.PendingStateReader(&Client{})
// _ = ethereum.PendingStateEventer(&Client{})
_ = ethereum.PendingContractCaller(&Client{})
)

30:F:\git\coin\ethereum\go-ethereum\ethdb\database.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethdb

import (
"strconv"
"strings"
"sync"
```

```
"time"

"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/metrics"
"github.com/syndtr/goleveldb/leveldb"
"github.com/syndtr/goleveldb/leveldb/errors"
"github.com/syndtr/goleveldb/leveldb/filter"
"github.com/syndtr/goleveldb/leveldb/iterator"
"github.com/syndtr/goleveldb/leveldb/opt"

gometrics "github.com/rcrowley/go-metrics"
)

var OpenFileLimit = 64

type LDBDatabase struct {
fn string      // filename for reporting
db *leveldb.DB // LevelDB instance

getTimer       gometrics.Timer // Timer for measuring the database get request counts and
latencies
putTimer       gometrics.Timer // Timer for measuring the database put request counts and
latencies
delTimer       gometrics.Timer // Timer for measuring the database delete request counts and
latencies
missMeter      gometrics.Meter // Meter for measuring the missed database get requests
readMeter      gometrics.Meter // Meter for measuring the database get request data usage
writeMeter     gometrics.Meter // Meter for measuring the database put request data usage
compTimeMeter  gometrics.Meter // Meter for measuring the total time spent in database
compaction
compReadMeter  gometrics.Meter // Meter for measuring the data read during compaction
compWriteMeter gometrics.Meter // Meter for measuring the data written during compaction

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
}

// NewLDBDatabase returns a LevelDB wrapped object.
func NewLDBDatabase(file string, cache int, handles int) (*LDBDatabase, error) {
logger := log.New("database", file)
```

```
// Ensure we have some minimal caching and file guarantees
if cache < 16 {
cache = 16
}
if handles < 16 {
handles = 16
}
logger.Info("Allocated cache and file handles", "cache", cache, "handles", handles)

// Open the db and recover any potential corruptions
db, err := leveldb.OpenFile(file, &opt.Options{
OpenFilesCacheCapacity: handles,
BlockCacheCapacity:     cache / 2 * opt.MiB,
WriteBuffer:            cache / 4 * opt.MiB, // Two of these are used internally
Filter:                 filter.NewBloomFilter(10),
})
if _, corrupted := err.(*errors.ErrCorrupted); corrupted {
db, err = leveldb.RecoverFile(file, nil)
}
// (Re)check for errors and abort if opening of the db failed
if err != nil {
return nil, err
}
return &LDBDatabase{
fn:  file,
db:  db,
log: logger,
}, nil
}

// Path returns the path to the database directory.
func (db *LDBDatabase) Path() string {
return db.fn
}

// Put puts the given key / value to the queue
func (db *LDBDatabase) Put(key []byte, value []byte) error {
// Measure the database put latency, if requested
if db.putTimer != nil {
defer db.putTimer.UpdateSince(time.Now())
}
```

```
// Generate the data to write to disk, update the meter and write
//value = rle.Compress(value)

if db.writeMeter != nil {
db.writeMeter.Mark(int64(len(value)))
}
return db.db.Put(key, value, nil)
}

// Get returns the given key if it's present.
func (db *LDBDatabase) Get(key []byte) ([]byte, error) {
// Measure the database get latency, if requested
if db.getTimer != nil {
defer db.getTimer.UpdateSince(time.Now())
}
// Retrieve the key and increment the miss counter if not found
dat, err := db.db.Get(key, nil)
if err != nil {
if db.missMeter != nil {
db.missMeter.Mark(1)
}
return nil, err
}
// Otherwise update the actually retrieved amount of data
if db.readMeter != nil {
db.readMeter.Mark(int64(len(dat)))
}
return dat, nil
//return rle.Decompress(dat)
}

// Delete deletes the key from the queue and database
func (db *LDBDatabase) Delete(key []byte) error {
// Measure the database delete latency, if requested
if db.delTimer != nil {
defer db.delTimer.UpdateSince(time.Now())
}
// Execute the actual operation
return db.db.Delete(key, nil)
}

func (db *LDBDatabase) NewIterator() iterator.Iterator {
```

```
return db.db.NewIterator(nil, nil)
}

func (db *LDBDatabase) Close() {
// Stop the metrics collection to avoid internal database races
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)
}
}
err := db.db.Close()
if err == nil {
db.log.Info("Database closed")
} else {
db.log.Error("Failed to close database", "err", err)
}
}

func (db *LDBDatabase) LDB() *leveldb.DB {
return db.db
}

// Meter configures the database metrics collectors and
func (db *LDBDatabase) Meter(prefix string) {
// Short circuit metering if the metrics system is disabled
if !metrics.Enabled {
return
}
// Initialize all the metrics collector at the requested prefix
db.getTimer = metrics.NewTimer(prefix + "user/gets")
db.putTimer = metrics.NewTimer(prefix + "user/puts")
db.delTimer = metrics.NewTimer(prefix + "user/dels")
db.missMeter = metrics.NewMeter(prefix + "user/misses")
db.readMeter = metrics.NewMeter(prefix + "user/reads")
db.writeMeter = metrics.NewMeter(prefix + "user/writes")
db.compTimeMeter = metrics.NewMeter(prefix + "compact/time")
db.compReadMeter = metrics.NewMeter(prefix + "compact/input")
```

```
db.compWriteMeter = metrics.NewMeter(prefix + "compact/output")

// Create a quit channel for the periodic collector and run it
db.quitLock.Lock()
db.quitChan = make(chan chan error)
db.quitLock.Unlock()

go db.meter(3 * time.Second)
}

// meter periodically retrieves internal leveldb counters and reports them to
// the metrics subsystem.
//
// This is how a stats table look 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
func (db *LDBDatabase) meter(refresh time.Duration) {
// Create the counters to store current and previous values
counters := make([][]float64, 2)
for i := 0; i < 2; i++ {
counters[i] = make([]float64, 3)
}
// Iterate ad infinitum and collect the stats
for i := 1; ; 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)
return
}
// 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 table not found")
```

```
return
}
lines = lines[3:]

// Iterate over all the table rows, and accumulate the entries
for j := 0; j < len(counters[i%2]); j++ {
counters[i%2][j] = 0
}
for _, line := range lines {
parts := strings.Split(line, "|")
if len(parts) != 6 {
break
}
for idx, counter := range parts[3:] {
value, err := strconv.ParseFloat(strings.TrimSpace(counter), 64)
if err != nil {
db.log.Error("Compaction entry parsing failed", "err", err)
return
}
counters[i%2][idx] += value
}
}
// Update all the requested meters
if db.compTimeMeter != nil {
db.compTimeMeter.Mark(int64((counters[i%2][0] - counters[(i-1)%2][0]) * 1000 * 1000 * 1000))
}
if db.compReadMeter != nil {
db.compReadMeter.Mark(int64((counters[i%2][1] - counters[(i-1)%2][1]) * 1024 * 1024))
}
if db.compWriteMeter != nil {
db.compWriteMeter.Mark(int64((counters[i%2][2] - counters[(i-1)%2][2]) * 1024 * 1024))
}
// Sleep a bit, then repeat the stats collection
select {
case errc := <-db.quitChan:
// Quit requesting, stop hammering the database
errc <- nil
return

case <-time.After(refresh):
// Timeout, gather a new set of stats
}
```

```
}
}

// TODO: remove this stuff and expose leveldb directly

func (db *LDBDatabase) NewBatch() Batch {
return &ldbBatch{db: db.db, b: new(leveldb.Batch)}
}

type ldbBatch struct {
db *leveldb.DB
b  *leveldb.Batch
}

func (b *ldbBatch) Put(key, value []byte) error {
b.b.Put(key, value)
return nil
}

func (b *ldbBatch) Write() error {
return b.db.Write(b.b, nil)
}

type table struct {
db     Database
prefix string
}

// NewTable returns a Database object that prefixes all keys with a given
// string.
func NewTable(db Database, prefix string) Database {
return &table{
db:     db,
prefix: prefix,
}
}

func (dt *table) Put(key []byte, value []byte) error {
return dt.db.Put(append([]byte(dt.prefix), key...), value)
}

func (dt *table) Get(key []byte) ([]byte, error) {
```

```
return dt.db.Get(append([]byte(dt.prefix), key...))
}

func (dt *table) Delete(key []byte) error {
return dt.db.Delete(append([]byte(dt.prefix), key...))
}

func (dt *table) Close() {
// Do nothing; don't close the underlying DB.
}

type tableBatch struct {
batch  Batch
prefix string
}

// NewTableBatch returns a Batch object which prefixes all keys with a given string.
func NewTableBatch(db Database, prefix string) Batch {
return &tableBatch{db.NewBatch(), prefix}
}

func (dt *table) NewBatch() Batch {
return &tableBatch{dt.db.NewBatch(), dt.prefix}
}

func (tb *tableBatch) Put(key, value []byte) error {
return tb.batch.Put(append([]byte(tb.prefix), key...), value)
}

func (tb *tableBatch) Write() error {
return tb.batch.Write()
}

31:F:\git\coin\ethereum\go-ethereum\ethdb\database_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethdb

import (
"os"
"path/filepath"
```

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

func newDb() *LDBDatabase {
file := filepath.Join("/", "tmp", "ldbtesttmpfile")
if common.FileExist(file) {
os.RemoveAll(file)
}
db, _ := NewLDBDatabase(file, 0, 0)

return db
}
```

32:F:\git\coin\ethereum\go-ethereum\ethdb\interface.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethdb

type Database interface {
Put(key []byte, value []byte) error
Get(key []byte) ([]byte, error)
Delete(key []byte) error
Close()
NewBatch() Batch
}

type Batch interface {
Put(key, value []byte) error
Write() error
}
```

33:F:\git\coin\ethereum\go-ethereum\ethdb\memory_database.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethdb

import (
"errors"
"sync"

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

```
/*
 * This is a test memory database. Do not use for any production it does not get persisted
 */
type MemDatabase struct {
db   map[string][]byte
lock sync.RWMutex
}

func NewMemDatabase() (*MemDatabase, error) {
return &MemDatabase{
db: make(map[string][]byte),
}, nil
}

func (db *MemDatabase) Put(key []byte, value []byte) error {
db.lock.Lock()
defer db.lock.Unlock()

db.db[string(key)] = common.CopyBytes(value)
return nil
}

func (db *MemDatabase) Get(key []byte) ([]byte, error) {
db.lock.RLock()
defer db.lock.RUnlock()

if entry, ok := db.db[string(key)]; ok {
return entry, nil
}
return nil, errors.New("not found")
}

func (db *MemDatabase) Keys() [][]byte {
db.lock.RLock()
defer db.lock.RUnlock()

keys := [][]byte{}
for key := range db.db {
keys = append(keys, []byte(key))
}
return keys
```

```
}

/*
func (db *MemDatabase) GetKeys() []*common.Key {
data, _ := db.Get([]byte("KeyRing"))

return []*common.Key{common.NewKeyFromBytes(data)}
}
*/

func (db *MemDatabase) Delete(key []byte) error {
db.lock.Lock()
defer db.lock.Unlock()

delete(db.db, string(key))
return nil
}

func (db *MemDatabase) Close() {}

func (db *MemDatabase) NewBatch() Batch {
return &memBatch{db: db}
}

type kv struct{ k, v []byte }

type memBatch struct {
db     *MemDatabase
writes []kv
lock   sync.RWMutex
}

func (b *memBatch) Put(key, value []byte) error {
b.lock.Lock()
defer b.lock.Unlock()

b.writes = append(b.writes, kv{common.CopyBytes(key), common.CopyBytes(value)})
return nil
}

func (b *memBatch) Write() error {
b.lock.RLock()
```

```
defer b.lock.RUnlock()

b.db.lock.Lock()
defer b.db.lock.Unlock()

for _, kv := range b.writes {
b.db.db[string(kv.k)] = kv.v
}
return nil
}

34:F:\git\coin\ethereum\go-ethereum\ethstats\ethstats.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package ethstats implements the network stats reporting service.
package ethstats

import (
"context"
"encoding/json"
"errors"
"fmt"
"math/big"
"net"
"regexp"
"runtime"
"strconv"
"strings"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/mclock"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/eth"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/les"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/node"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/rpc"
```

```
"golang.org/x/net/websocket"
)

// historyUpdateRange is the number of blocks a node should report upon login or
// history request.
const historyUpdateRange = 50

// Service implements an Ethereum netstats reporting daemon that pushes local
// chain statistics up to a monitoring server.
type Service struct {
stack *node.Node // Temporary workaround, remove when API finalized

server *p2p.Server        // Peer-to-peer server to retrieve networking infos
eth    *eth.Ethereum      // Full Ethereum service if monitoring a full node
les    *les.LightEthereum // Light Ethereum service if monitoring a light node
engine consensus.Engine   // Consensus engine to retrieve variadic block fields

node string // Name of the node to display on the monitoring page
pass string // Password to authorize access to the monitoring page
host string // Remote address of the monitoring service

pongCh chan struct{} // Pong notifications are fed into this channel
histCh chan []uint64 // History request block numbers are fed into this channel
}

// New returns a monitoring service ready for stats reporting.
func New(url string, ethServ *eth.Ethereum, lesServ *les.LightEthereum) (*Service, error) {
// Parse the netstats connection url
re := regexp.MustCompile("([^:@]*)(:([^@]*))?@(.+)")
parts := re.FindStringSubmatch(url)
if len(parts) != 5 {
return nil, fmt.Errorf("invalid netstats url: \"%s\", should be nodename:secret@host:port", url)
}
// Assemble and return the stats service
var engine consensus.Engine
if ethServ != nil {
engine = ethServ.Engine()
} else {
engine = lesServ.Engine()
}
return &Service{
eth:    ethServ,
```

```
les:    lesServ,
engine: engine,
node:   parts[1],
pass:   parts[3],
host:   parts[4],
pongCh: make(chan struct{}),
histCh: make(chan []uint64, 1),
}, nil
}

// Protocols implements node.Service, returning the P2P network protocols used
// by the stats service (nil as it doesn't use the devp2p overlay network).
func (s *Service) Protocols() []p2p.Protocol { return nil }

// APIs implements node.Service, returning the RPC API endpoints provided by the
// stats service (nil as it doesn't provide any user callable APIs).
func (s *Service) APIs() []rpc.API { return nil }

// Start implements node.Service, starting up the monitoring and reporting daemon.
func (s *Service) Start(server *p2p.Server) error {
s.server = server
go s.loop()

log.Info("Stats daemon started")
return nil
}

// Stop implements node.Service, terminating the monitoring and reporting daemon.
func (s *Service) Stop() error {
log.Info("Stats daemon stopped")
return nil
}

// loop keeps trying to connect to the netstats server, reporting chain events
// until termination.
func (s *Service) loop() {
// Subscribe to chain events to execute updates on
var emux *event.TypeMux
if s.eth != nil {
emux = s.eth.EventMux()
} else {
emux = s.les.EventMux()
```

```
}
headSub := emux.Subscribe(core.ChainHeadEvent{})
defer headSub.Unsubscribe()

txSub := emux.Subscribe(core.TxPreEvent{})
defer txSub.Unsubscribe()

// Start a goroutine that exhausts the subsciptions to avoid events piling up
var (
quitCh = make(chan struct{})
headCh = make(chan *types.Block, 1)
txCh   = make(chan struct{}, 1)
)
go func() {
var lastTx mclock.AbsTime

for {
select {
// Notify of chain head events, but drop if too frequent
case head, ok := <-headSub.Chan():
if !ok { // node stopped
close(quitCh)
return
}
select {
case headCh <- head.Data.(core.ChainHeadEvent).Block:
default:
}

// Notify of new transaction events, but drop if too frequent
case _, ok := <-txSub.Chan():
if !ok { // node stopped
close(quitCh)
return
}
if time.Duration(mclock.Now()-lastTx) < time.Second {
continue
}
lastTx = mclock.Now()

select {
case txCh <- struct{}{}:
```

```
default:
}
}
}
}()
// Loop reporting until termination
for {
// Resolve the URL, defaulting to TLS, but falling back to none too
path := fmt.Sprintf("%s/api", s.host)
urls := []string{path}

if !strings.Contains(path, "://") { // url.Parse and url.IsAbs is unsuitable
(https://github.com/golang/go/issues/19779)
urls = []string{"wss://" + path, "ws://" + path}
}
// Establish a websocket connection to the server on any supported URL
var (
conf *websocket.Config
conn *websocket.Conn
err  error
)
for _, url := range urls {
if conf, err = websocket.NewConfig(url, "http://localhost/"); err != nil {
continue
}
conf.Dialer = &net.Dialer{Timeout: 5 * time.Second}
if conn, err = websocket.DialConfig(conf); err == nil {
break
}
}
if err != nil {
log.Warn("Stats server unreachable", "err", err)
time.Sleep(10 * time.Second)
continue
}
// Authenticate the client with the server
if err = s.login(conn); err != nil {
log.Warn("Stats login failed", "err", err)
conn.Close()
time.Sleep(10 * time.Second)
continue
}
```

```
go s.readLoop(conn)

// Send the initial stats so our node looks decent from the get go
if err = s.report(conn); err != nil {
log.Warn("Initial stats report failed", "err", err)
conn.Close()
continue
}
// Keep sending status updates until the connection breaks
fullReport := time.NewTicker(15 * time.Second)

for err == nil {
select {
case <-quitCh:
conn.Close()
return

case <-fullReport.C:
if err = s.report(conn); err != nil {
log.Warn("Full stats report failed", "err", err)
}
case list := <-s.histCh:
if err = s.reportHistory(conn, list); err != nil {
log.Warn("Requested history report failed", "err", err)
}
case head := <-headCh:
if err = s.reportBlock(conn, head); err != nil {
log.Warn("Block stats report failed", "err", err)
}
if err = s.reportPending(conn); err != nil {
log.Warn("Post-block transaction stats report failed", "err", err)
}
case <-txCh:
if err = s.reportPending(conn); err != nil {
log.Warn("Transaction stats report failed", "err", err)
}
}
}
// Make sure the connection is closed
conn.Close()
}
}
```

```
// readLoop loops as long as the connection is alive and retrieves data packets
// from the network socket. If any of them match an active request, it forwards
// it, if they themselves are requests it initiates a reply, and lastly it drops
// unknown packets.
func (s *Service) readLoop(conn *websocket.Conn) {
// If the read loop exists, close the connection
defer conn.Close()

for {
// Retrieve the next generic network packet and bail out on error
var msg map[string][]interface{}
if err := websocket.JSON.Receive(conn, &msg); err != nil {
log.Warn("Failed to decode stats server message", "err", err)
return
}
log.Trace("Received message from stats server", "msg", msg)
if len(msg["emit"]) == 0 {
log.Warn("Stats server sent non-broadcast", "msg", msg)
return
}
command, ok := msg["emit"][0].(string)
if !ok {
log.Warn("Invalid stats server message type", "type", msg["emit"][0])
return
}
// If the message is a ping reply, deliver (someone must be listening!)
if len(msg["emit"]) == 2 && command == "node-pong" {
select {
case s.pongCh <- struct{}{}:
// Pong delivered, continue listening
continue
default:
// Ping routine dead, abort
log.Warn("Stats server pinger seems to have died")
return
}
}
// If the message is a history request, forward to the event processor
if len(msg["emit"]) == 2 && command == "history" {
// Make sure the request is valid and doesn't crash us
request, ok := msg["emit"][1].(map[string]interface{})
```

```
if !ok {
log.Warn("Invalid stats history request", "msg", msg["emit"][1])
s.histCh <- nil
continue // Ethstats sometime sends invalid history requests, ignore those
}
list, ok := request["list"].([]interface{})
if !ok {
log.Warn("Invalid stats history block list", "list", request["list"])
return
}
// Convert the block number list to an integer list
numbers := make([]uint64, len(list))
for i, num := range list {
n, ok := num.(float64)
if !ok {
log.Warn("Invalid stats history block number", "number", num)
return
}
numbers[i] = uint64(n)
}
select {
case s.histCh <- numbers:
continue
default:
}
}
// Report anything else and continue
log.Info("Unknown stats message", "msg", msg)
}
}

// nodeInfo is the collection of metainformation about a node that is displayed
// on the monitoring page.
type nodeInfo struct {
Name     string `json:"name"`
Node     string `json:"node"`
Port     int    `json:"port"`
Network  string `json:"net"`
Protocol string `json:"protocol"`
API      string `json:"api"`
Os       string `json:"os"`
OsVer    string `json:"os_v"`
```

```
Client   string `json:"client"`
History  bool   `json:"canUpdateHistory"`
}

// authMsg is the authentication infos needed to login to a monitoring server.
type authMsg struct {
Id     string   `json:"id"`
Info   nodeInfo `json:"info"`
Secret string   `json:"secret"`
}

// login tries to authorize the client at the remote server.
func (s *Service) login(conn *websocket.Conn) error {
// Construct and send the login authentication
infos := s.server.NodeInfo()

var network, protocol string
if info := infos.Protocols["eth"]; info != nil {
network = fmt.Sprintf("%d", info.(*eth.EthNodeInfo).Network)
protocol = fmt.Sprintf("eth/%d", eth.ProtocolVersions[0])
} else {
network = fmt.Sprintf("%d", infos.Protocols["les"].(*eth.EthNodeInfo).Network)
protocol = fmt.Sprintf("les/%d", les.ProtocolVersions[0])
}
auth := &authMsg{
Id: s.node,
Info: nodeInfo{
Name:     s.node,
Node:     infos.Name,
Port:     infos.Ports.Listener,
Network:  network,
Protocol: protocol,
API:      "No",
Os:       runtime.GOOS,
OsVer:    runtime.GOARCH,
Client:   "0.1.1",
History:  true,
},
Secret: s.pass,
}
login := map[string][]interface{}{
"emit": {"hello", auth},
```

```
}
if err := websocket.JSON.Send(conn, login); err != nil {
return err
}
// Retrieve the remote ack or connection termination
var ack map[string][]string
if err := websocket.JSON.Receive(conn, &ack); err != nil || len(ack["emit"]) != 1 || ack["emit"][0] !=
"ready" {
return errors.New("unauthorized")
}
return nil
}

// report collects all possible data to report and send it to the stats server.
// This should only be used on reconnects or rarely to avoid overloading the
// server. Use the individual methods for reporting subscribed events.
func (s *Service) report(conn *websocket.Conn) error {
if err := s.reportLatency(conn); err != nil {
return err
}
if err := s.reportBlock(conn, nil); err != nil {
return err
}
if err := s.reportPending(conn); err != nil {
return err
}
if err := s.reportStats(conn); err != nil {
return err
}
return nil
}

// reportLatency sends a ping request to the server, measures the RTT time and
// finally sends a latency update.
func (s *Service) reportLatency(conn *websocket.Conn) error {
// Send the current time to the ethstats server
start := time.Now()

ping := map[string][]interface{}{
"emit": {"node-ping", map[string]string{
"id":         s.node,
"clientTime": start.String(),
```

```
}},
}
if err := websocket.JSON.Send(conn, ping); err != nil {
return err
}
// Wait for the pong request to arrive back
select {
case <-s.pongCh:
// Pong delivered, report the latency
case <-time.After(5 * time.Second):
// Ping timeout, abort
return errors.New("ping timed out")
}
latency := strconv.Itoa(int((time.Since(start) / time.Duration(2)).Nanoseconds() / 1000000))

// Send back the measured latency
log.Trace("Sending measured latency to ethstats", "latency", latency)

stats := map[string][]interface{}{
"emit": {"latency", map[string]string{
"id":      s.node,
"latency": latency,
}},
}
return websocket.JSON.Send(conn, stats)
}

// blockStats is the information to report about individual blocks.
type blockStats struct {
Number     *big.Int       `json:"number"`
Hash       common.Hash    `json:"hash"`
ParentHash common.Hash    `json:"parentHash"`
Timestamp  *big.Int       `json:"timestamp"`
Miner      common.Address `json:"miner"`
GasUsed    *big.Int       `json:"gasUsed"`
GasLimit   *big.Int       `json:"gasLimit"`
Diff       string         `json:"difficulty"`
TotalDiff  string         `json:"totalDifficulty"`
Txs        []txStats      `json:"transactions"`
TxHash     common.Hash    `json:"transactionsRoot"`
Root       common.Hash    `json:"stateRoot"`
Uncles     uncleStats     `json:"uncles"`
```

```
}

// txStats is the information to report about individual transactions.
type txStats struct {
Hash common.Hash `json:"hash"`
}

// uncleStats is a custom wrapper around an uncle array to force serializing
// empty arrays instead of returning null for them.
type uncleStats []*types.Header

func (s uncleStats) MarshalJSON() ([]byte, error) {
if uncles := ([]*types.Header)(s); len(uncles) > 0 {
return json.Marshal(uncles)
}
return []byte("[]"), nil
}

// reportBlock retrieves the current chain head and repors it to the stats server.
func (s *Service) reportBlock(conn *websocket.Conn, block *types.Block) error {
// Gather the block details from the header or block chain
details := s.assembleBlockStats(block)

// Assemble the block report and send it to the server
log.Trace("Sending new block to ethstats", "number", details.Number, "hash", details.Hash)

stats := map[string]interface{}{
"id":    s.node,
"block": details,
}
report := map[string][]interface{}{
"emit": {"block", stats},
}
return websocket.JSON.Send(conn, report)
}

// assembleBlockStats retrieves any required metadata to report a single block
// and assembles the block stats. If block is nil, the current head is processed.
func (s *Service) assembleBlockStats(block *types.Block) *blockStats {
// Gather the block infos from the local blockchain
var (
header *types.Header
```

```
td     *big.Int
txs    []txStats
uncles []*types.Header
)
if s.eth != nil {
// Full nodes have all needed information available
if block == nil {
block = s.eth.BlockChain().CurrentBlock()
}
header = block.Header()
td = s.eth.BlockChain().GetTd(header.Hash(), header.Number.Uint64())

txs = make([]txStats, len(block.Transactions()))
for i, tx := range block.Transactions() {
txs[i].Hash = tx.Hash()
}
uncles = block.Uncles()
} else {
// Light nodes would need on-demand lookups for transactions/uncles, skip
if block != nil {
header = block.Header()
} else {
header = s.les.BlockChain().CurrentHeader()
}
td = s.les.BlockChain().GetTd(header.Hash(), header.Number.Uint64())
txs = []txStats{}
}
// Assemble and return the block stats
author, _ := s.engine.Author(header)

return &blockStats{
Number:     header.Number,
Hash:       header.Hash(),
ParentHash: header.ParentHash,
Timestamp:  header.Time,
Miner:      author,
GasUsed:    new(big.Int).Set(header.GasUsed),
GasLimit:   new(big.Int).Set(header.GasLimit),
Diff:       header.Difficulty.String(),
TotalDiff:  td.String(),
Txs:        txs,
TxHash:     header.TxHash,
```

```
Root:       header.Root,
Uncles:     uncles,
}
}

// reportHistory retrieves the most recent batch of blocks and reports it to the
// stats server.
func (s *Service) reportHistory(conn *websocket.Conn, list []uint64) error {
// Figure out the indexes that need reporting
indexes := make([]uint64, 0, historyUpdateRange)
if len(list) > 0 {
// Specific indexes requested, send them back in particular
indexes = append(indexes, list...)
} else {
// No indexes requested, send back the top ones
var head int64
if s.eth != nil {
head = s.eth.BlockChain().CurrentHeader().Number.Int64()
} else {
head = s.les.BlockChain().CurrentHeader().Number.Int64()
}
start := head - historyUpdateRange + 1
if start < 0 {
start = 0
}
for i := uint64(start); i <= uint64(head); i++ {
indexes = append(indexes, i)
}
}
// Gather the batch of blocks to report
history := make([]*blockStats, len(indexes))
for i, number := range indexes {
// Retrieve the next block if it's known to us
var block *types.Block
if s.eth != nil {
block = s.eth.BlockChain().GetBlockByNumber(number)
} else {
if header := s.les.BlockChain().GetHeaderByNumber(number); header != nil {
block = types.NewBlockWithHeader(header)
}
}
// If we do have the block, add to the history and continue
```

```
if block != nil {
history[len(history)-1-i] = s.assembleBlockStats(block)
continue
}
// Ran out of blocks, cut the report short and send
history = history[len(history)-i:]
}
// Assemble the history report and send it to the server
if len(history) > 0 {
log.Trace("Sending historical blocks to ethstats", "first", history[0].Number, "last",
history[len(history)-1].Number)
} else {
log.Trace("No history to send to stats server")
}
stats := map[string]interface{}{
"id":      s.node,
"history": history,
}
report := map[string][]interface{}{
"emit": {"history", stats},
}
return websocket.JSON.Send(conn, report)
}

// pendStats is the information to report about pending transactions.
type pendStats struct {
Pending int `json:"pending"`
}

// reportPending retrieves the current number of pending transactions and reports
// it to the stats server.
func (s *Service) reportPending(conn *websocket.Conn) error {
// Retrieve the pending count from the local blockchain
var pending int
if s.eth != nil {
pending, _ = s.eth.TxPool().Stats()
} else {
pending = s.les.TxPool().Stats()
}
// Assemble the transaction stats and send it to the server
log.Trace("Sending pending transactions to ethstats", "count", pending)
```

```
stats := map[string]interface{}{
"id": s.node,
"stats": &pendStats{
Pending: pending,
},
}
report := map[string][]interface{}{
"emit": {"pending", stats},
}
return websocket.JSON.Send(conn, report)
}

// nodeStats is the information to report about the local node.
type nodeStats struct {
Active   bool `json:"active"`
Syncing  bool `json:"syncing"`
Mining   bool `json:"mining"`
Hashrate int  `json:"hashrate"`
Peers    int  `json:"peers"`
GasPrice int  `json:"gasPrice"`
Uptime   int  `json:"uptime"`
}

// reportPending retrieves various stats about the node at the networking and
// mining layer and reports it to the stats server.
func (s *Service) reportStats(conn *websocket.Conn) error {
// Gather the syncing and mining infos from the local miner instance
var (
mining   bool
hashrate int
syncing  bool
gasprice int
)
if s.eth != nil {
mining = s.eth.Miner().Mining()
hashrate = int(s.eth.Miner().HashRate())

sync := s.eth.Downloader().Progress()
syncing = s.eth.BlockChain().CurrentHeader().Number.Uint64() >= sync.HighestBlock

price, _ := s.eth.ApiBackend.SuggestPrice(context.Background())
gasprice = int(price.Uint64())
```

```
} else {
sync := s.les.Downloader().Progress()
syncing = s.les.BlockChain().CurrentHeader().Number.Uint64() >= sync.HighestBlock
}
// Assemble the node stats and send it to the server
log.Trace("Sending node details to ethstats")

stats := map[string]interface{}{
"id": s.node,
"stats": &nodeStats{
Active:   true,
Mining:   mining,
Hashrate: hashrate,
Peers:    s.server.PeerCount(),
GasPrice: gasprice,
Syncing:  syncing,
Uptime:   100,
},
}
report := map[string][]interface{}{
"emit": {"stats", stats},
}
return websocket.JSON.Send(conn, report)
}
```

35:F:\git\coin\ethereum\go-ethereum\event\event.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package event deals with subscriptions to real-time events.
package event

import (
"errors"
"fmt"
"reflect"
"sync"
"time"
)

// TypeMuxEvent is a time-tagged notification pushed to subscribers.
type TypeMuxEvent struct {
Time time.Time
```

```
Data interface{}
}

// A TypeMux dispatches events to registered receivers. Receivers can be
// registered to handle events of certain type. Any operation
// called after mux is stopped will return ErrMuxClosed.
//
// The zero value is ready to use.
//
// Deprecated: use Feed
type TypeMux struct {
mutex   sync.RWMutex
subm    map[reflect.Type][]*TypeMuxSubscription
stopped bool
}

// ErrMuxClosed is returned when Posting on a closed TypeMux.
var ErrMuxClosed = errors.New("event: mux closed")

// Subscribe creates a subscription for events of the given types. The
// subscription's channel is closed when it is unsubscribed
// or the mux is closed.
func (mux *TypeMux) Subscribe(types ...interface{}) *TypeMuxSubscription {
sub := newsub(mux)
mux.mutex.Lock()
defer mux.mutex.Unlock()
if mux.stopped {
// set the status to closed so that calling Unsubscribe after this
// call will short curuit
sub.closed = true
close(sub.postC)
} else {
if mux.subm == nil {
mux.subm = make(map[reflect.Type][]*TypeMuxSubscription)
}
for _, t := range types {
rtyp := reflect.TypeOf(t)
oldsubs := mux.subm[rtyp]
if find(oldsubs, sub) != -1 {
panic(fmt.Sprintf("event: duplicate type %s in Subscribe", rtyp))
}
subs := make([]*TypeMuxSubscription, len(oldsubs)+1)
```

```
copy(subs, oldsubs)
subs[len(oldsubs)] = sub
mux.subm[rtyp] = subs
}
}
return sub
}

// Post sends an event to all receivers registered for the given type.
// It returns ErrMuxClosed if the mux has been stopped.
func (mux *TypeMux) Post(ev interface{}) error {
event := &TypeMuxEvent{
Time: time.Now(),
Data: ev,
}
rtyp := reflect.TypeOf(ev)
mux.mutex.RLock()
if mux.stopped {
mux.mutex.RUnlock()
return ErrMuxClosed
}
subs := mux.subm[rtyp]
mux.mutex.RUnlock()
for _, sub := range subs {
sub.deliver(event)
}
return nil
}

// Stop closes a mux. The mux can no longer be used.
// Future Post calls will fail with ErrMuxClosed.
// Stop blocks until all current deliveries have finished.
func (mux *TypeMux) Stop() {
mux.mutex.Lock()
for _, subs := range mux.subm {
for _, sub := range subs {
sub.closewait()
}
}
mux.subm = nil
mux.stopped = true
mux.mutex.Unlock()
```

```
}

func (mux *TypeMux) del(s *TypeMuxSubscription) {
mux.mutex.Lock()
for typ, subs := range mux.subm {
if pos := find(subs, s); pos >= 0 {
if len(subs) == 1 {
delete(mux.subm, typ)
} else {
mux.subm[typ] = posdelete(subs, pos)
}
}
}
s.mux.mutex.Unlock()
}

func find(slice []*TypeMuxSubscription, item *TypeMuxSubscription) int {
for i, v := range slice {
if v == item {
return i
}
}
return -1
}

func posdelete(slice []*TypeMuxSubscription, pos int) []*TypeMuxSubscription {
news := make([]*TypeMuxSubscription, len(slice)-1)
copy(news[:pos], slice[:pos])
copy(news[pos:], slice[pos+1:])
return news
}

// TypeMuxSubscription is a subscription established through TypeMux.
type TypeMuxSubscription struct {
mux     *TypeMux
created time.Time
closeMu sync.Mutex
closing chan struct{}
closed  bool

// these two are the same channel. they are stored separately so
// postC can be set to nil without affecting the return value of
```

```
// Chan.
postMu sync.RWMutex
readC  <-chan *TypeMuxEvent
postC  chan<- *TypeMuxEvent
}

func newsub(mux *TypeMux) *TypeMuxSubscription {
c := make(chan *TypeMuxEvent)
return &TypeMuxSubscription{
mux:     mux,
created: time.Now(),
readC:   c,
postC:   c,
closing: make(chan struct{}),
}
}

func (s *TypeMuxSubscription) Chan() <-chan *TypeMuxEvent {
return s.readC
}

func (s *TypeMuxSubscription) Unsubscribe() {
s.mux.del(s)
s.closewait()
}

func (s *TypeMuxSubscription) closewait() {
s.closeMu.Lock()
defer s.closeMu.Unlock()
if s.closed {
return
}
close(s.closing)
s.closed = true

s.postMu.Lock()
close(s.postC)
s.postC = nil
s.postMu.Unlock()
}

func (s *TypeMuxSubscription) deliver(event *TypeMuxEvent) {
```

```
// Short circuit delivery if stale event
if s.created.After(event.Time) {
return
}
// Otherwise deliver the event
s.postMu.RLock()
defer s.postMu.RUnlock()

select {
case s.postC <- event:
case <-s.closing:
}
}

36:F:\git\coin\ethereum\go-ethereum\event\event_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import (
"math/rand"
"sync"
"testing"
"time"
)

type testEvent int

func TestSubCloseUnsub(t *testing.T) {
// the point of this test is **not** to panic
var mux TypeMux
mux.Stop()
sub := mux.Subscribe(int(0))
sub.Unsubscribe()
}

func TestSub(t *testing.T) {
mux := new(TypeMux)
defer mux.Stop()

sub := mux.Subscribe(testEvent(0))
go func() {
```

```
if err := mux.Post(testEvent(5)); err != nil {
t.Errorf("Post returned unexpected error: %v", err)
}
}()
ev := <-sub.Chan()

if ev.Data.(testEvent) != testEvent(5) {
t.Errorf("Got %v (%T), expected event %v (%T)",
ev, ev, testEvent(5), testEvent(5))
}
}

func TestMuxErrorAfterStop(t *testing.T) {
mux := new(TypeMux)
mux.Stop()

sub := mux.Subscribe(testEvent(0))
if _, isopen := <-sub.Chan(); isopen {
t.Errorf("subscription channel was not closed")
}
if err := mux.Post(testEvent(0)); err != ErrMuxClosed {
t.Errorf("Post error mismatch, got: %s, expected: %s", err, ErrMuxClosed)
}
}

func TestUnsubscribeUnblockPost(t *testing.T) {
mux := new(TypeMux)
defer mux.Stop()

sub := mux.Subscribe(testEvent(0))
unblocked := make(chan bool)
go func() {
mux.Post(testEvent(5))
unblocked <- true
}()

select {
case <-unblocked:
t.Errorf("Post returned before Unsubscribe")
default:
sub.Unsubscribe()
<-unblocked
```

```
}
}

func TestSubscribeDuplicateType(t *testing.T) {
mux := new(TypeMux)
expected := "event: duplicate type event.testEvent in Subscribe"

defer func() {
err := recover()
if err == nil {
t.Errorf("Subscribe didn't panic for duplicate type")
} else if err != expected {
t.Errorf("panic mismatch: got %#v, expected %#v", err, expected)
}
}()
mux.Subscribe(testEvent(1), testEvent(2))
}

func TestMuxConcurrent(t *testing.T) {
rand.Seed(time.Now().Unix())
mux := new(TypeMux)
defer mux.Stop()

recv := make(chan int)
poster := func() {
for {
err := mux.Post(testEvent(0))
if err != nil {
return
}
}
}
sub := func(i int) {
time.Sleep(time.Duration(rand.Intn(99)) * time.Millisecond)
sub := mux.Subscribe(testEvent(0))
<-sub.Chan()
sub.Unsubscribe()
recv <- i
}

go poster()
go poster()
```

```
go poster()
nsubs := 1000
for i := 0; i < nsubs; i++ {
go sub(i)
}

// wait until everyone has been served
counts := make(map[int]int, nsubs)
for i := 0; i < nsubs; i++ {
counts[<-recv]++
}
for i, count := range counts {
if count != 1 {
t.Errorf("receiver %d called %d times, expected only 1 call", i, count)
}
}
}

func emptySubscriber(mux *TypeMux, types ...interface{}) {
s := mux.Subscribe(testEvent(0))
go func() {
for range s.Chan() {
}
}()
}

func BenchmarkPost1000(b *testing.B) {
var (
mux              = new(TypeMux)
subscribed, done sync.WaitGroup
nsubs            = 1000
)
subscribed.Add(nsubs)
done.Add(nsubs)
for i := 0; i < nsubs; i++ {
go func() {
s := mux.Subscribe(testEvent(0))
subscribed.Done()
for range s.Chan() {
}
done.Done()
}()
```

```
}
subscribed.Wait()

// The actual benchmark.
b.ResetTimer()
for i := 0; i < b.N; i++ {
mux.Post(testEvent(0))
}

b.StopTimer()
mux.Stop()
done.Wait()
}

func BenchmarkPostConcurrent(b *testing.B) {
var mux = new(TypeMux)
defer mux.Stop()
emptySubscriber(mux, testEvent(0))
emptySubscriber(mux, testEvent(0))
emptySubscriber(mux, testEvent(0))

var wg sync.WaitGroup
poster := func() {
for i := 0; i < b.N; i++ {
mux.Post(testEvent(0))
}
wg.Done()
}
wg.Add(5)
for i := 0; i < 5; i++ {
go poster()
}
wg.Wait()
}

// for comparison
func BenchmarkChanSend(b *testing.B) {
c := make(chan interface{})
closed := make(chan struct{})
go func() {
for range c {
}
```

```
}()

for i := 0; i < b.N; i++ {
select {
case c <- i:
case <-closed:
}
}
}

37:F:\git\coin\ethereum\go-ethereum\event\example_feed_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event_test

import (
"fmt"

"github.com/ethereum/go-ethereum/event"
)

func ExampleFeed_acknowledgedEvents() {
// This example shows how the return value of Send can be used for request/reply
// interaction between event consumers and producers.
var feed event.Feed
type ackedEvent struct {
i   int
ack chan<- struct{}
}

// Consumers wait for events on the feed and acknowledge processing.
done := make(chan struct{})
defer close(done)
for i := 0; i < 3; i++ {
ch := make(chan ackedEvent, 100)
sub := feed.Subscribe(ch)
go func() {
defer sub.Unsubscribe()
for {
select {
case ev := <-ch:
fmt.Println(ev.i) // "process" the event
```

```
ev.ack <- struct{}{}
case <-done:
return
}
}
}()
}

// The producer sends values of type ackedEvent with increasing values of i.
// It waits for all consumers to acknowledge before sending the next event.
for i := 0; i < 3; i++ {
acksignal := make(chan struct{})
n := feed.Send(ackedEvent{i, acksignal})
for ack := 0; ack < n; ack++ {
<-acksignal
}
}
// Output:
// 0
// 0
// 0
// 1
// 1
// 1
// 2
// 2
// 2
}

38:F:\git\coin\ethereum\go-ethereum\event\example_scope_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event_test

import (
"fmt"
"sync"

"github.com/ethereum/go-ethereum/event"
)

// This example demonstrates how SubscriptionScope can be used to control the lifetime of
```

```
// subscriptions.
//
// Our example program consists of two servers, each of which performs a calculation when
// requested. The servers also allow subscribing to results of all computations.
type divServer struct{ results event.Feed }
type mulServer struct{ results event.Feed }

func (s *divServer) do(a, b int) int {
r := a / b
s.results.Send(r)
return r
}

func (s *mulServer) do(a, b int) int {
r := a * b
s.results.Send(r)
return r
}

// The servers are contained in an App. The app controls the servers and exposes them
// through its API.
type App struct {
divServer
mulServer
scope event.SubscriptionScope
}

func (s *App) Calc(op byte, a, b int) int {
switch op {
case '/':
return s.divServer.do(a, b)
case '*':
return s.mulServer.do(a, b)
default:
panic("invalid op")
}
}

// The app's SubscribeResults method starts sending calculation results to the given
// channel. Subscriptions created through this method are tied to the lifetime of the App
// because they are registered in the scope.
func (s *App) SubscribeResults(op byte, ch chan<- int) event.Subscription {
```

```
switch op {
case '/':
return s.scope.Track(s.divServer.results.Subscribe(ch))
case '*':
return s.scope.Track(s.mulServer.results.Subscribe(ch))
default:
panic("invalid op")
}
}

// Stop stops the App, closing all subscriptions created through SubscribeResults.
func (s *App) Stop() {
s.scope.Close()
}

func ExampleSubscriptionScope() {
// Create the app.
var (
app  App
wg   sync.WaitGroup
divs = make(chan int)
muls = make(chan int)
)

// Run a subscriber in the background.
divsub := app.SubscribeResults('/', divs)
mulsub := app.SubscribeResults('*', muls)
wg.Add(1)
go func() {
defer wg.Done()
defer fmt.Println("subscriber exited")
defer divsub.Unsubscribe()
defer mulsub.Unsubscribe()
for {
select {
case result := <-divs:
fmt.Println("division happened:", result)
case result := <-muls:
fmt.Println("multiplication happened:", result)
case <-divsub.Err():
return
case <-mulsub.Err():
```

```
return
}
}
}()

// Interact with the app.
app.Calc('/', 22, 11)
app.Calc('*', 3, 4)

// Stop the app. This shuts down the subscriptions, causing the subscriber to exit.
app.Stop()
wg.Wait()

// Output:
// division happened: 2
// multiplication happened: 12
// subscriber exited
}

39:F:\git\coin\ethereum\go-ethereum\event\example_subscription_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event_test

import (
"fmt"

"github.com/ethereum/go-ethereum/event"
)

func ExampleNewSubscription() {
// Create a subscription that sends 10 integers on ch.
ch := make(chan int)
sub := event.NewSubscription(func(quit <-chan struct{}) error {
for i := 0; i < 10; i++ {
select {
case ch <- i:
case <-quit:
fmt.Println("unsubscribed")
return nil
}
}
```

```
return nil
})

// This is the consumer. It reads 5 integers, then aborts the subscription.
// Note that Unsubscribe waits until the producer has shut down.
for i := range ch {
fmt.Println(i)
if i == 4 {
sub.Unsubscribe()
break
}
}
// Output:
// 0
// 1
// 2
// 3
// 4
// unsubscribed
}

40:F:\git\coin\ethereum\go-ethereum\event\example_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import "fmt"

func ExampleTypeMux() {
type someEvent struct{ I int }
type otherEvent struct{ S string }
type yetAnotherEvent struct{ X, Y int }

var mux TypeMux

// Start a subscriber.
done := make(chan struct{})
sub := mux.Subscribe(someEvent{}, otherEvent{})
go func() {
for event := range sub.Chan() {
fmt.Printf("Received: %#v\n", event.Data)
}
```

```
fmt.Println("done")
close(done)
}()

// Post some events.
mux.Post(someEvent{5})
mux.Post(yetAnotherEvent{X: 3, Y: 4})
mux.Post(someEvent{6})
mux.Post(otherEvent{"whoa"})

// Stop closes all subscription channels.
// The subscriber goroutine will print "done"
// and exit.
mux.Stop()

// Wait for subscriber to return.
<-done

// Output:
// Received: event.someEvent{I:5}
// Received: event.someEvent{I:6}
// Received: event.otherEvent{S:"whoa"}
// done
}

41:F:\git\coin\ethereum\go-ethereum\event\feed.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import (
"errors"
"reflect"
"sync"
)

var errBadChannel = errors.New("event: Subscribe argument does not have sendable channel type")

// Feed implements one-to-many subscriptions where the carrier of events is a channel.
// Values sent to a Feed are delivered to all subscribed channels simultaneously.
//
```

```
// Feeds can only be used with a single type. The type is determined by the first Send or
// Subscribe operation. Subsequent calls to these methods panic if the type does not
// match.
//
// The zero value is ready to use.
type Feed struct {
once      sync.Once        // ensures that init only runs once
sendLock  chan struct{}    // sendLock has a one-element buffer and is empty when held.It protects
sendCases.
removeSub chan interface{} // interrupts Send
sendCases caseList         // the active set of select cases used by Send

// The inbox holds newly subscribed channels until they are added to sendCases.
mu     sync.Mutex
inbox  caseList
etype  reflect.Type
closed bool
}

// This is the index of the first actual subscription channel in sendCases.
// sendCases[0] is a SelectRecv case for the removeSub channel.
const firstSubSendCase = 1

type feedTypeError struct {
got, want reflect.Type
op        string
}

func (e feedTypeError) Error() string {
return "event: wrong type in " + e.op + " got " + e.got.String() + ", want " + e.want.String()
}

func (f *Feed) init() {
f.removeSub = make(chan interface{})
f.sendLock = make(chan struct{}, 1)
f.sendLock <- struct{}{}
f.sendCases = caseList{{Chan: reflect.ValueOf(f.removeSub), Dir: reflect.SelectRecv}}
}

// Subscribe adds a channel to the feed. Future sends will be delivered on the channel
// until the subscription is canceled. All channels added must have the same element type.
//
```

```
// The channel should have ample buffer space to avoid blocking other subscribers.
// Slow subscribers are not dropped.
func (f *Feed) Subscribe(channel interface{}) Subscription {
f.once.Do(f.init)

chanval := reflect.ValueOf(channel)
chantyp := chanval.Type()
if chantyp.Kind() != reflect.Chan || chantyp.ChanDir()&reflect.SendDir == 0 {
panic(errBadChannel)
}
sub := &feedSub{feed: f, channel: chanval, err: make(chan error, 1)}

f.mu.Lock()
defer f.mu.Unlock()
if !f.typecheck(chantyp.Elem()) {
panic(feedTypeError{op: "Subscribe", got: chantyp, want: reflect.ChanOf(reflect.SendDir, f.etype)})
}
// Add the select case to the inbox.
// The next Send will add it to f.sendCases.
cas := reflect.SelectCase{Dir: reflect.SelectSend, Chan: chanval}
f.inbox = append(f.inbox, cas)
return sub
}

// note: callers must hold f.mu
func (f *Feed) typecheck(typ reflect.Type) bool {
if f.etype == nil {
f.etype = typ
return true
}
return f.etype == typ
}

func (f *Feed) remove(sub *feedSub) {
// Delete from inbox first, which covers channels
// that have not been added to f.sendCases yet.
ch := sub.channel.Interface()
f.mu.Lock()
index := f.inbox.find(ch)
if index != -1 {
f.inbox = f.inbox.delete(index)
f.mu.Unlock()
```

```
return
}
f.mu.Unlock()

select {
case f.removeSub <- ch:
// Send will remove the channel from f.sendCases.
case <-f.sendLock:
// No Send is in progress, delete the channel now that we have the send lock.
f.sendCases = f.sendCases.delete(f.sendCases.find(ch))
f.sendLock <- struct{}{}
}
}

// Send delivers to all subscribed channels simultaneously.
// It returns the number of subscribers that the value was sent to.
func (f *Feed) Send(value interface{}) (nsent int) {
f.once.Do(f.init)
<-f.sendLock

// Add new cases from the inbox after taking the send lock.
f.mu.Lock()
f.sendCases = append(f.sendCases, f.inbox...)
f.inbox = nil
f.mu.Unlock()

// Set the sent value on all channels.
rvalue := reflect.ValueOf(value)
if !f.typecheck(rvalue.Type()) {
f.sendLock <- struct{}{}
panic(feedTypeError{op: "Send", got: rvalue.Type(), want: f.etype})
}
for i := firstSubSendCase; i < len(f.sendCases); i++ {
f.sendCases[i].Send = rvalue
}

// Send until all channels except removeSub have been chosen.
cases := f.sendCases
for {
// Fast path: try sending without blocking before adding to the select set.
// This should usually succeed if subscribers are fast enough and have free
// buffer space.
```

```
for i := firstSubSendCase; i < len(cases); i++ {
if cases[i].Chan.TrySend(rvalue) {
nsent++
cases = cases.deactivate(i)
i--
}
}
if len(cases) == firstSubSendCase {
break
}
// Select on all the receivers, waiting for them to unblock.
chosen, recv, _ := reflect.Select(cases)
if chosen == 0 /* <-f.removeSub */ {
index := f.sendCases.find(recv.Interface())
f.sendCases = f.sendCases.delete(index)
if index >= 0 && index < len(cases) {
cases = f.sendCases[:len(cases)-1]
}
} else {
cases = cases.deactivate(chosen)
nsent++
}
}

// Forget about the sent value and hand off the send lock.
for i := firstSubSendCase; i < len(f.sendCases); i++ {
f.sendCases[i].Send = reflect.Value{}
}
f.sendLock <- struct{}{}
return nsent
}

type feedSub struct {
feed    *Feed
channel reflect.Value
errOnce sync.Once
err     chan error
}

func (sub *feedSub) Unsubscribe() {
sub.errOnce.Do(func() {
sub.feed.remove(sub)
```

```
close(sub.err)
})
}

func (sub *feedSub) Err() <-chan error {
return sub.err
}

type caseList []reflect.SelectCase

// find returns the index of a case containing the given channel.
func (cs caseList) find(channel interface{}) int {
for i, cas := range cs {
if cas.Chan.Interface() == channel {
return i
}
}
return -1
}

// delete removes the given case from cs.
func (cs caseList) delete(index int) caseList {
return append(cs[:index], cs[index+1:]...)
}

// deactivate moves the case at index into the non-accessible portion of the cs slice.
func (cs caseList) deactivate(index int) caseList {
last := len(cs) - 1
cs[index], cs[last] = cs[last], cs[index]
return cs[:last]
}

// func (cs caseList) String() string {
//     s := "["
//     for i, cas := range cs {
//             if i != 0 {
//                     s += ", "
//             }
//             switch cas.Dir {
//             case reflect.SelectSend:
//                     s += fmt.Sprintf("%v<-", cas.Chan.Interface())
//             case reflect.SelectRecv:
```

```
//                  s += fmt.Sprintf("<-%v", cas.Chan.Interface())
//              }
//      }
//      return s + "]"
// }

42:F:\git\coin\ethereum\go-ethereum\event\feed_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import (
"fmt"
"reflect"
"sync"
"testing"
"time"
)

func TestFeedPanics(t *testing.T) {
{
var f Feed
f.Send(int(2))
want := feedTypeError{op: "Send", got: reflect.TypeOf(uint64(0)), want: reflect.TypeOf(int(0))}
if err := checkPanic(want, func() { f.Send(uint64(2)) }); err != nil {
t.Error(err)
}
}
{
var f Feed
ch := make(chan int)
f.Subscribe(ch)
want := feedTypeError{op: "Send", got: reflect.TypeOf(uint64(0)), want: reflect.TypeOf(int(0))}
if err := checkPanic(want, func() { f.Send(uint64(2)) }); err != nil {
t.Error(err)
}
}
{
var f Feed
f.Send(int(2))
want := feedTypeError{op: "Subscribe", got: reflect.TypeOf(make(chan uint64)), want:
reflect.TypeOf(make(chan<- int))}
```

```
if err := checkPanic(want, func() { f.Subscribe(make(chan uint64)) }); err != nil {
t.Error(err)
}
}
{
var f Feed
if err := checkPanic(errBadChannel, func() { f.Subscribe(make(<-chan int)) }); err != nil {
t.Error(err)
}
}
{
var f Feed
if err := checkPanic(errBadChannel, func() { f.Subscribe(int(0)) }); err != nil {
t.Error(err)
}
}
}

func checkPanic(want error, fn func()) (err error) {
defer func() {
panic := recover()
if panic == nil {
err = fmt.Errorf("didn't panic")
} else if !reflect.DeepEqual(panic, want) {
err = fmt.Errorf("panicked with wrong error: got %q, want %q", panic, want)
}
}()
fn()
return nil
}

func TestFeed(t *testing.T) {
var feed Feed
var done, subscribed sync.WaitGroup
subscriber := func(i int) {
defer done.Done()

subchan := make(chan int)
sub := feed.Subscribe(subchan)
timeout := time.NewTimer(2 * time.Second)
subscribed.Done()
```

```
select {
case v := <-subchan:
if v != 1 {
t.Errorf("%d: received value %d, want 1", i, v)
}
case <-timeout.C:
t.Errorf("%d: receive timeout", i)
}

sub.Unsubscribe()
select {
case _, ok := <-sub.Err():
if ok {
t.Errorf("%d: error channel not closed after unsubscribe", i)
}
case <-timeout.C:
t.Errorf("%d: unsubscribe timeout", i)
}
}

const n = 1000
done.Add(n)
subscribed.Add(n)
for i := 0; i < n; i++ {
go subscriber(i)
}
subscribed.Wait()
if nsent := feed.Send(1); nsent != n {
t.Errorf("first send delivered %d times, want %d", nsent, n)
}
if nsent := feed.Send(2); nsent != 0 {
t.Errorf("second send delivered %d times, want 0", nsent)
}
done.Wait()
}

func TestFeedSubscribeSameChannel(t *testing.T) {
var (
feed Feed
done sync.WaitGroup
ch   = make(chan int)
sub1 = feed.Subscribe(ch)
```

```
sub2 = feed.Subscribe(ch)
_    = feed.Subscribe(ch)
)
expectSends := func(value, n int) {
if nsent := feed.Send(value); nsent != n {
t.Errorf("send delivered %d times, want %d", nsent, n)
}
done.Done()
}
expectRecv := func(wantValue, n int) {
for i := 0; i < n; i++ {
if v := <-ch; v != wantValue {
t.Errorf("received %d, want %d", v, wantValue)
}
}
}

done.Add(1)
go expectSends(1, 3)
expectRecv(1, 3)
done.Wait()

sub1.Unsubscribe()

done.Add(1)
go expectSends(2, 2)
expectRecv(2, 2)
done.Wait()

sub2.Unsubscribe()

done.Add(1)
go expectSends(3, 1)
expectRecv(3, 1)
done.Wait()
}

func TestFeedSubscribeBlockedPost(t *testing.T) {
var (
feed   Feed
nsends = 2000
ch1    = make(chan int)
```

```
ch2    = make(chan int)
wg     sync.WaitGroup
)
defer wg.Wait()

feed.Subscribe(ch1)
wg.Add(nsends)
for i := 0; i < nsends; i++ {
go func() {
feed.Send(99)
wg.Done()
}()
}

sub2 := feed.Subscribe(ch2)
defer sub2.Unsubscribe()

// We're done when ch1 has received N times.
// The number of receives on ch2 depends on scheduling.
for i := 0; i < nsends; {
select {
case <-ch1:
i++
case <-ch2:
}
}
}

func TestFeedUnsubscribeBlockedPost(t *testing.T) {
var (
feed   Feed
nsends = 200
chans  = make([]chan int, 2000)
subs   = make([]Subscription, len(chans))
bchan  = make(chan int)
bsub   = feed.Subscribe(bchan)
wg     sync.WaitGroup
)
for i := range chans {
chans[i] = make(chan int, nsends)
}
```

```
// Queue up some Sends. None of these can make progress while bchan isn't read.
wg.Add(nsends)
for i := 0; i < nsends; i++ {
go func() {
feed.Send(99)
wg.Done()
}()
}
// Subscribe the other channels.
for i, ch := range chans {
subs[i] = feed.Subscribe(ch)
}
// Unsubscribe them again.
for _, sub := range subs {
sub.Unsubscribe()
}
// Unblock the Sends.
bsub.Unsubscribe()
wg.Wait()
}

func TestFeedUnsubscribeFromInbox(t *testing.T) {
var (
feed Feed
ch1  = make(chan int)
ch2  = make(chan int)
sub1 = feed.Subscribe(ch1)
sub2 = feed.Subscribe(ch1)
sub3 = feed.Subscribe(ch2)
)
if len(feed.inbox) != 3 {
t.Errorf("inbox length != 3 after subscribe")
}
if len(feed.sendCases) != 1 {
t.Errorf("sendCases is non-empty after unsubscribe")
}

sub1.Unsubscribe()
sub2.Unsubscribe()
sub3.Unsubscribe()
if len(feed.inbox) != 0 {
t.Errorf("inbox is non-empty after unsubscribe")
```

```
}
if len(feed.sendCases) != 1 {
t.Errorf("sendCases is non-empty after unsubscribe")
}
}

func BenchmarkFeedSend1000(b *testing.B) {
var (
done  sync.WaitGroup
feed  Feed
nsubs = 1000
)
subscriber := func(ch <-chan int) {
for i := 0; i < b.N; i++ {
<-ch
}
done.Done()
}
done.Add(nsubs)
for i := 0; i < nsubs; i++ {
ch := make(chan int, 200)
feed.Subscribe(ch)
go subscriber(ch)
}

// The actual benchmark.
b.ResetTimer()
for i := 0; i < b.N; i++ {
if feed.Send(i) != nsubs {
panic("wrong number of sends")
}
}

b.StopTimer()
done.Wait()
}

43:F:\git\coin\ethereum\go-ethereum\event\filter\filter.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package filter implements event filters.
package filter
```

```
import "reflect"

type Filter interface {
Compare(Filter) bool
Trigger(data interface{})
}

type FilterEvent struct {
filter Filter
data   interface{}
}

type Filters struct {
id       int
watchers map[int]Filter
ch       chan FilterEvent

quit chan struct{}
}

func New() *Filters {
return &Filters{
ch:       make(chan FilterEvent),
watchers: make(map[int]Filter),
quit:     make(chan struct{}),
}
}

func (self *Filters) Start() {
go self.loop()
}

func (self *Filters) Stop() {
close(self.quit)
}

func (self *Filters) Notify(filter Filter, data interface{}) {
self.ch <- FilterEvent{filter, data}
}

func (self *Filters) Install(watcher Filter) int {
```

```
self.watchers[self.id] = watcher
self.id++

return self.id - 1
}

func (self *Filters) Uninstall(id int) {
delete(self.watchers, id)
}

func (self *Filters) loop() {
out:
for {
select {
case <-self.quit:
break out
case event := <-self.ch:
for _, watcher := range self.watchers {
if reflect.TypeOf(watcher) == reflect.TypeOf(event.filter) {
if watcher.Compare(event.filter) {
watcher.Trigger(event.data)
}
}
}
}
}
}

func (self *Filters) Match(a, b Filter) bool {
return reflect.TypeOf(a) == reflect.TypeOf(b) && a.Compare(b)
}

func (self *Filters) Get(i int) Filter {
return self.watchers[i]
}

44:F:\git\coin\ethereum\go-ethereum\event\filter\filter_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filter

import (
```

```
"testing"
"time"
)

// Simple test to check if baseline matching/mismatching filtering works.
func TestFilters(t *testing.T) {
fm := New()
fm.Start()

// Register two filters to catch posted data
first := make(chan struct{})
fm.Install(Generic{
Str1: "hello",
Fn: func(data interface{}) {
first <- struct{}{}
},
})
second := make(chan struct{})
fm.Install(Generic{
Str1: "hello1",
Str2: "hello",
Fn: func(data interface{}) {
second <- struct{}{}
},
})
// Post an event that should only match the first filter
fm.Notify(Generic{Str1: "hello"}, true)
fm.Stop()

// Ensure only the mathcing filters fire
select {
case <-first:
case <-time.After(100 * time.Millisecond):
t.Error("matching filter timed out")
}
select {
case <-second:
t.Error("mismatching filter fired")
case <-time.After(100 * time.Millisecond):
}
}
```

45:F:\git\coin\ethereum\go-ethereum\event\filter\generic_filter.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package filter

type Generic struct {
Str1, Str2, Str3 string
Data map[string]struct{}

Fn func(data interface{})
}

// self = registered, f = incoming
func (self Generic) Compare(f Filter) bool {
var strMatch, dataMatch = true, true

filter := f.(Generic)
if (len(self.Str1) > 0 && filter.Str1 != self.Str1) ||
(len(self.Str2) > 0 && filter.Str2 != self.Str2) ||
(len(self.Str3) > 0 && filter.Str3 != self.Str3) {
strMatch = false
}

for k := range self.Data {
if _, ok := filter.Data[k]; !ok {
return false
}
}

return strMatch && dataMatch
}

func (self Generic) Trigger(data interface{}) {
self.Fn(data)
}

46:F:\git\coin\ethereum\go-ethereum\event\subscription.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import (

```
"context"
"sync"
"time"

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

// Subscription represents a stream of events. The carrier of the events is typically a
// channel, but isn't part of the interface.
//
// Subscriptions can fail while established. Failures are reported through an error
// channel. It receives a value if there is an issue with the subscription (e.g. the
// network connection delivering the events has been closed). Only one value will ever be
// sent.
//
// The error channel is closed when the subscription ends successfully (i.e. when the
// source of events is closed). It is also closed when Unsubscribe is called.
//
// The Unsubscribe method cancels the sending of events. You must call Unsubscribe in all
// cases to ensure that resources related to the subscription are released. It can be
// called any number of times.
type Subscription interface {
Err() <-chan error // returns the error channel
Unsubscribe()      // cancels sending of events, closing the error channel
}

// NewSubscription runs a producer function as a subscription in a new goroutine. The
// channel given to the producer is closed when Unsubscribe is called. If fn returns an
// error, it is sent on the subscription's error channel.
func NewSubscription(producer func(<-chan struct{}) error) Subscription {
s := &funcSub{unsub: make(chan struct{}), err: make(chan error, 1)}
go func() {
defer close(s.err)
err := producer(s.unsub)
s.mu.Lock()
defer s.mu.Unlock()
if !s.unsubscribed {
if err != nil {
s.err <- err
}
s.unsubscribed = true
}
```

```
}()
return s
}

type funcSub struct {
unsub        chan struct{}
err          chan error
mu           sync.Mutex
unsubscribed bool
}

func (s *funcSub) Unsubscribe() {
s.mu.Lock()
if s.unsubscribed {
s.mu.Unlock()
return
}
s.unsubscribed = true
close(s.unsub)
s.mu.Unlock()
// Wait for producer shutdown.
<-s.err
}

func (s *funcSub) Err() <-chan error {
return s.err
}

// Resubscribe calls fn repeatedly to keep a subscription established. When the
// subscription is established, Resubscribe waits for it to fail and calls fn again. This
// process repeats until Unsubscribe is called or the active subscription ends
// successfully.
//
// Resubscribe applies backoff between calls to fn. The time between calls is adapted
// based on the error rate, but will never exceed backoffMax.
func Resubscribe(backoffMax time.Duration, fn ResubscribeFunc) Subscription {
s := &resubscribeSub{
waitTime:   backoffMax / 10,
backoffMax: backoffMax,
fn:         fn,
err:        make(chan error),
unsub:      make(chan struct{}),
```

```
}
go s.loop()
return s
}

// A ResubscribeFunc attempts to establish a subscription.
type ResubscribeFunc func(context.Context) (Subscription, error)

type resubscribeSub struct {
fn                  ResubscribeFunc
err                 chan error
unsub               chan struct{}
unsubOnce           sync.Once
lastTry             mclock.AbsTime
waitTime, backoffMax time.Duration
}

func (s *resubscribeSub) Unsubscribe() {
s.unsubOnce.Do(func() {
s.unsub <- struct{}{}
<-s.err
})
}

func (s *resubscribeSub) Err() <-chan error {
return s.err
}

func (s *resubscribeSub) loop() {
defer close(s.err)
var done bool
for !done {
sub := s.subscribe()
if sub == nil {
break
}
done = s.waitForError(sub)
sub.Unsubscribe()
}
}

func (s *resubscribeSub) subscribe() Subscription {
```

```
subscribed := make(chan error)
var sub Subscription
retry:
for {
s.lastTry = mclock.Now()
ctx, cancel := context.WithCancel(context.Background())
go func() {
rsub, err := s.fn(ctx)
sub = rsub
subscribed <- err
}()
select {
case err := <-subscribed:
cancel()
if err != nil {
// Subscribing failed, wait before launching the next try.
if s.backoffWait() {
return nil
}
continue retry
}
if sub == nil {
panic("event: ResubscribeFunc returned nil subscription and no error")
}
return sub
case <-s.unsub:
cancel()
return nil
}
}
}

func (s *resubscribeSub) waitForError(sub Subscription) bool {
defer sub.Unsubscribe()
select {
case err := <-sub.Err():
return err == nil
case <-s.unsub:
return true
}
}
```

```
func (s *resubscribeSub) backoffWait() bool {
if time.Duration(mclock.Now()-s.lastTry) > s.backoffMax {
s.waitTime = s.backoffMax / 10
} else {
s.waitTime *= 2
if s.waitTime > s.backoffMax {
s.waitTime = s.backoffMax
}
}

t := time.NewTimer(s.waitTime)
defer t.Stop()
select {
case <-t.C:
return false
case <-s.unsub:
return true
}
}

// SubscriptionScope provides a facility to unsubscribe multiple subscriptions at once.
//
// For code that handle more than one subscription, a scope can be used to conveniently
// unsubscribe all of them with a single call. The example demonstrates a typical use in a
// larger program.
//
// The zero value is ready to use.
type SubscriptionScope struct {
mu     sync.Mutex
subs   map[*scopeSub]struct{}
closed bool
}

type scopeSub struct {
sc *SubscriptionScope
s  Subscription
}

// Track starts tracking a subscription. If the scope is closed, Track returns nil. The
// returned subscription is a wrapper. Unsubscribing the wrapper removes it from the
// scope.
func (sc *SubscriptionScope) Track(s Subscription) Subscription {
```

```
sc.mu.Lock()
defer sc.mu.Unlock()
if sc.closed {
return nil
}
if sc.subs == nil {
sc.subs = make(map[*scopeSub]struct{})
}
ss := &scopeSub{sc, s}
sc.subs[ss] = struct{}{}
return ss
}

// Close calls Unsubscribe on all tracked subscriptions and prevents further additions to
// the tracked set. Calls to Track after Close return nil.
func (sc *SubscriptionScope) Close() {
sc.mu.Lock()
defer sc.mu.Unlock()
if sc.closed {
return
}
sc.closed = true
for s := range sc.subs {
s.s.Unsubscribe()
}
sc.subs = nil
}

// Count returns the number of tracked subscriptions.
// It is meant to be used for debugging.
func (sc *SubscriptionScope) Count() int {
sc.mu.Lock()
defer sc.mu.Unlock()
return len(sc.subs)
}

func (s *scopeSub) Unsubscribe() {
s.s.Unsubscribe()
s.sc.mu.Lock()
defer s.sc.mu.Unlock()
delete(s.sc.subs, s)
}
```

```
func (s *scopeSub) Err() <-chan error {
return s.s.Err()
}

47:F:\git\coin\ethereum\go-ethereum\event\subscription_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package event

import (
"context"
"errors"
"testing"
"time"
)

var errInts = errors.New("error in subscribeInts")

func subscribeInts(max, fail int, c chan<- int) Subscription {
return NewSubscription(func(quit <-chan struct{}) error {
for i := 0; i < max; i++ {
if i >= fail {
return errInts
}
select {
case c <- i:
case <-quit:
return nil
}
}
return nil
})
}

func TestNewSubscriptionError(t *testing.T) {
t.Parallel()

channel := make(chan int)
sub := subscribeInts(10, 2, channel)
loop:
for want := 0; want < 10; want++ {
```

```
select {
case got := <-channel:
if got != want {
t.Fatalf("wrong int %d, want %d", got, want)
}
case err := <-sub.Err():
if err != errInts {
t.Fatalf("wrong error: got %q, want %q", err, errInts)
}
if want != 2 {
t.Fatalf("got errInts at int %d, should be received at 2", want)
}
break loop
}
}
sub.Unsubscribe()

err, ok := <-sub.Err()
if err != nil {
t.Fatal("got non-nil error after Unsubscribe")
}
if ok {
t.Fatal("channel still open after Unsubscribe")
}
}

func TestResubscribe(t *testing.T) {
t.Parallel()

var i int
nfails := 6
sub := Resubscribe(100*time.Millisecond, func(ctx context.Context) (Subscription, error) {
// fmt.Printf("call #%d @ %v\n", i, time.Now())
i++
if i == 2 {
// Delay the second failure a bit to reset the resubscribe interval.
time.Sleep(200 * time.Millisecond)
}
if i < nfails {
return nil, errors.New("oops")
}
sub := NewSubscription(func(unsubscribed <-chan struct{}) error { return nil })
```

```
return sub, nil
})

<-sub.Err()
if i != nfails {
t.Fatalf("resubscribe function called %d times, want %d times", i, nfails)
}
}

func TestResubscribeAbort(t *testing.T) {
t.Parallel()

done := make(chan error)
sub := Resubscribe(0, func(ctx context.Context) (Subscription, error) {
select {
case <-ctx.Done():
done <- nil
case <-time.After(2 * time.Second):
done <- errors.New("context given to resubscribe function not canceled within 2s")
}
return nil, nil
})

sub.Unsubscribe()
if err := <-done; err != nil {
t.Fatal(err)
}
}

48:F:\git\coin\ethereum\go-ethereum\interfaces.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package ethereum defines interfaces for interacting with Ethereum.
package ethereum

import (
"context"
"errors"
"math/big"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/types"
```

```
)

// NotFound is returned by API methods if the requested item does not exist.
var NotFound = errors.New("not found")

// TODO: move subscription to package event

// Subscription represents an event subscription where events are
// delivered on a data channel.
type Subscription interface {
// Unsubscribe cancels the sending of events to the data channel
// and closes the error channel.
Unsubscribe()
// Err returns the subscription error channel. The error channel receives
// a value if there is an issue with the subscription (e.g. the network connection
// delivering the events has been closed). Only one value will ever be sent.
// The error channel is closed by Unsubscribe.
Err() <-chan error
}

// ChainReader provides access to the blockchain. The methods in this interface access raw
// data from either the canonical chain (when requesting by block number) or any
// blockchain fork that was previously downloaded and processed by the node. The block
// number argument can be nil to select the latest canonical block. Reading block headers
// should be preferred over full blocks whenever possible.
//
// The returned error is NotFound if the requested item does not exist.
type ChainReader interface {
BlockByHash(ctx context.Context, hash common.Hash) (*types.Block, error)
BlockByNumber(ctx context.Context, number *big.Int) (*types.Block, error)
HeaderByHash(ctx context.Context, hash common.Hash) (*types.Header, error)
HeaderByNumber(ctx context.Context, number *big.Int) (*types.Header, error)
TransactionCount(ctx context.Context, blockHash common.Hash) (uint, error)
TransactionInBlock(ctx context.Context, blockHash common.Hash, index uint)
(*types.Transaction, error)

// This method subscribes to notifications about changes of the head block of
// the canonical chain.
SubscribeNewHead(ctx context.Context, ch chan<- *types.Header) (Subscription, error)
}

// TransactionReader provides access to past transactions and their receipts.
```

```
// Implementations may impose arbitrary restrictions on the transactions and receipts that
// can be retrieved. Historic transactions may not be available.
//
// Avoid relying on this interface if possible. Contract logs (through the LogFilterer
// interface) are more reliable and usually safer in the presence of chain
// reorganisations.
//
// The returned error is NotFound if the requested item does not exist.
type TransactionReader interface {
// TransactionByHash checks the pool of pending transactions in addition to the
// blockchain. The isPending return value indicates whether the transaction has been
// mined yet. Note that the transaction may not be part of the canonical chain even if
// it's not pending.
TransactionByHash(ctx context.Context, txHash common.Hash) (tx *types.Transaction, isPending
bool, err error)
// TransactionReceipt returns the receipt of a mined transaction. Note that the
// transaction may not be included in the current canonical chain even if a receipt
// exists.
TransactionReceipt(ctx context.Context, txHash common.Hash) (*types.Receipt, error)
}

// ChainStateReader wraps access to the state trie of the canonical blockchain. Note that
// implementations of the interface may be unable to return state values for old blocks.
// In many cases, using CallContract can be preferable to reading raw contract storage.
type ChainStateReader interface {
BalanceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (*big.Int, error)
StorageAt(ctx context.Context, account common.Address, key common.Hash, blockNumber
*big.Int) ([]byte, error)
CodeAt(ctx context.Context, account common.Address, blockNumber *big.Int) ([]byte, error)
NonceAt(ctx context.Context, account common.Address, blockNumber *big.Int) (uint64, error)
}

// SyncProgress gives progress indications when the node is synchronising with
// the Ethereum network.
type SyncProgress struct {
StartingBlock uint64 // Block number where sync began
CurrentBlock  uint64 // Current block number where sync is at
HighestBlock  uint64 // Highest alleged block number in the chain
PulledStates  uint64 // Number of state trie entries already downloaded
KnownStates   uint64 // Total number os state trie entries known about
}
```

```
// ChainSyncReader wraps access to the node's current sync status. If there's no
// sync currently running, it returns nil.
type ChainSyncReader interface {
SyncProgress(ctx context.Context) (*SyncProgress, error)
}

// CallMsg contains parameters for contract calls.
type CallMsg struct {
From     common.Address  // the sender of the 'transaction'
To       *common.Address // the destination contract (nil for contract creation)
Gas      *big.Int        // if nil, the call executes with near-infinite gas
GasPrice *big.Int        // wei <-> gas exchange ratio
Value    *big.Int        // amount of wei sent along with the call
Data     []byte          // input data, usually an ABI-encoded contract method invocation
}

// A ContractCaller provides contract calls, essentially transactions that are executed by
// the EVM but not mined into the blockchain. ContractCall is a low-level method to
// execute such calls. For applications which are structured around specific contracts,
// the abigen tool provides a nicer, properly typed way to perform calls.
type ContractCaller interface {
CallContract(ctx context.Context, call CallMsg, blockNumber *big.Int) ([]byte, error)
}

// FilterQuery contains options for contact log filtering.
type FilterQuery struct {
FromBlock *big.Int         // beginning of the queried range, nil means genesis block
ToBlock   *big.Int         // end of the range, nil means latest block
Addresses []common.Address // restricts matches to events created by specific contracts

// The Topic list restricts matches to particular event topics. Each event has a list
// of topics. Topics matches a prefix of that list. An empty element slice matches any
// topic. Non-empty elements represent an alternative that matches any of the
// contained topics.
//
// Examples:
// {} or nil          matches any topic list
// {{A}}              matches topic A in first position
// {{}, {B}}          matches any topic in first position, B in second position
// {{A}}, {B}}        matches topic A in first position, B in second position
// {{A, B}}, {C, D}}  matches topic (A OR B) in first position, (C OR D) in second position
Topics [][]common.Hash
```

```
}

// LogFilterer provides access to contract log events using a one-off query or continuous
// event subscription.
//
// Logs received through a streaming query subscription may have Removed set to true,
// indicating that the log was reverted due to a chain reorganisation.
type LogFilterer interface {
FilterLogs(ctx context.Context, q FilterQuery) ([]types.Log, error)
SubscribeFilterLogs(ctx context.Context, q FilterQuery, ch chan<- types.Log) (Subscription, error)
}

// TransactionSender wraps transaction sending. The SendTransaction method injects a
// signed transaction into the pending transaction pool for execution. If the transaction
// was a contract creation, the TransactionReceipt method can be used to retrieve the
// contract address after the transaction has been mined.
//
// The transaction must be signed and have a valid nonce to be included. Consumers of the
// API can use package accounts to maintain local private keys and need can retrieve the
// next available nonce using PendingNonceAt.
type TransactionSender interface {
SendTransaction(ctx context.Context, tx *types.Transaction) error
}

// GasPricer wraps the gas price oracle, which monitors the blockchain to determine the
// optimal gas price given current fee market conditions.
type GasPricer interface {
SuggestGasPrice(ctx context.Context) (*big.Int, error)
}

// A PendingStateReader provides access to the pending state, which is the result of all
// known executable transactions which have not yet been included in the blockchain. It is
// commonly used to display the result of ’unconfirmed’ actions (e.g. wallet value
// transfers) initiated by the user. The PendingNonceAt operation is a good way to
// retrieve the next available transaction nonce for a specific account.
type PendingStateReader interface {
PendingBalanceAt(ctx context.Context, account common.Address) (*big.Int, error)
PendingStorageAt(ctx context.Context, account common.Address, key common.Hash) ([]byte,
error)
PendingCodeAt(ctx context.Context, account common.Address) ([]byte, error)
PendingNonceAt(ctx context.Context, account common.Address) (uint64, error)
PendingTransactionCount(ctx context.Context) (uint, error)
```

```
}

// PendingContractCaller can be used to perform calls against the pending state.
type PendingContractCaller interface {
PendingCallContract(ctx context.Context, call CallMsg) ([]byte, error)
}

// GasEstimator wraps EstimateGas, which tries to estimate the gas needed to execute a
// specific transaction based on the pending state. 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.
type GasEstimator interface {
EstimateGas(ctx context.Context, call CallMsg) (usedGas *big.Int, err error)
}

// A PendingStateEventer provides access to real time notifications about changes to the
// pending state.
type PendingStateEventer interface {
SubscribePendingTransactions(ctx context.Context, ch chan<- *types.Transaction) (Subscription,
error)
}

49:F:\git\coin\ethereum\go-ethereum\internal\build\archive.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package build

import (
"archive/tar"
"archive/zip"
"compress/gzip"
"fmt"
"io"
"os"
"path/filepath"
"strings"
)

type Archive interface {
// Directory adds a new directory entry to the archive and sets the
// directory for subsequent calls to Header.
Directory(name string) error
```

```
// Header adds a new file to the archive. The file is added to the directory
// set by Directory. The content of the file must be written to the returned
// writer.
Header(os.FileInfo) (io.Writer, error)

// Close flushes the archive and closes the underlying file.
Close() error
}

func NewArchive(file *os.File) (Archive, string) {
switch {
case strings.HasSuffix(file.Name(), ".zip"):
return NewZipArchive(file), strings.TrimSuffix(file.Name(), ".zip")
case strings.HasSuffix(file.Name(), ".tar.gz"):
return NewTarballArchive(file), strings.TrimSuffix(file.Name(), ".tar.gz")
default:
return nil, ""
}
}

// AddFile appends an existing file to an archive.
func AddFile(a Archive, file string) error {
fd, err := os.Open(file)
if err != nil {
return err
}
defer fd.Close()
fi, err := fd.Stat()
if err != nil {
return err
}
w, err := a.Header(fi)
if err != nil {
return err
}
if _, err := io.Copy(w, fd); err != nil {
return err
}
return nil
}
```

```
// WriteArchive creates an archive containing the given files.
func WriteArchive(name string, files []string) (err error) {
archfd, err := os.Create(name)
if err != nil {
return err
}

defer func() {
archfd.Close()
// Remove the half-written archive on failure.
if err != nil {
os.Remove(name)
}
}()
archive, basename := NewArchive(archfd)
if archive == nil {
return fmt.Errorf("unknown archive extension")
}
fmt.Println(name)
if err := archive.Directory(basename); err != nil {
return err
}
for _, file := range files {
fmt.Println("   +", filepath.Base(file))
if err := AddFile(archive, file); err != nil {
return err
}
}
return archive.Close()
}

type ZipArchive struct {
dir  string
zipw *zip.Writer
file io.Closer
}

func NewZipArchive(w io.WriteCloser) Archive {
return &ZipArchive{"", zip.NewWriter(w), w}
}

func (a *ZipArchive) Directory(name string) error {
```

```
a.dir = name + "/"
return nil
}

func (a *ZipArchive) Header(fi os.FileInfo) (io.Writer, error) {
head, err := zip.FileInfoHeader(fi)
if err != nil {
return nil, fmt.Errorf("can't make zip header: %v", err)
}
head.Name = a.dir + head.Name
head.Method = zip.Deflate
w, err := a.zipw.CreateHeader(head)
if err != nil {
return nil, fmt.Errorf("can't add zip header: %v", err)
}
return w, nil
}

func (a *ZipArchive) Close() error {
if err := a.zipw.Close(); err != nil {
return err
}
return a.file.Close()
}

type TarballArchive struct {
dir  string
tarw *tar.Writer
gzw  *gzip.Writer
file io.Closer
}

func NewTarballArchive(w io.WriteCloser) Archive {
gzw := gzip.NewWriter(w)
tarw := tar.NewWriter(gzw)
return &TarballArchive{"", tarw, gzw, w}
}

func (a *TarballArchive) Directory(name string) error {
a.dir = name + "/"
return a.tarw.WriteHeader(&tar.Header{
Name:     a.dir,
```

```
Mode:     0755,
Typeflag: tar.TypeDir,
})
}

func (a *TarballArchive) Header(fi os.FileInfo) (io.Writer, error) {
head, err := tar.FileInfoHeader(fi, "")
if err != nil {
return nil, fmt.Errorf("can't make tar header: %v", err)
}
head.Name = a.dir + head.Name
if err := a.tarw.WriteHeader(head); err != nil {
return nil, fmt.Errorf("can't add tar header: %v", err)
}
return a.tarw, nil
}

func (a *TarballArchive) Close() error {
if err := a.tarw.Close(); err != nil {
return err
}
if err := a.gzw.Close(); err != nil {
return err
}
return a.file.Close()
}

50:F:\git\coin\ethereum\go-ethereum\internal\build\azure.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package build

import (
"fmt"
"os"

storage "github.com/Azure/azure-storage-go"
)

// AzureBlobstoreConfig is an authentication and configuration struct containing
// the data needed by the Azure SDK to interact with a speicifc container in the
// blobstore.
```

```
type AzureBlobstoreConfig struct {
Account   string // Account name to authorize API requests with
Token     string // Access token for the above account
Container string // Blob container to upload files into
}

// AzureBlobstoreUpload uploads a local file to the Azure Blob Storage. Note, this
// method assumes a max file size of 64MB (Azure limitation). Larger files will
// need a multi API call approach implemented.
//
// See: https://msdn.microsoft.com/en-us/library/azure/dd179451.aspx#Anchor_3
func AzureBlobstoreUpload(path string, name string, config AzureBlobstoreConfig) error {
if *DryRunFlag {
fmt.Printf("would upload %q to %s/%s/%s\n", path, config.Account, config.Container, name)
return nil
}
// Create an authenticated client against the Azure cloud
rawClient, err := storage.NewBasicClient(config.Account, config.Token)
if err != nil {
return err
}
client := rawClient.GetBlobService()

// Stream the file to upload into the designated blobstore container
in, err := os.Open(path)
if err != nil {
return err
}
defer in.Close()

info, err := in.Stat()
if err != nil {
return err
}
return client.CreateBlockBlobFromReader(config.Container, name, uint64(info.Size()), in, nil)
}

// AzureBlobstoreList lists all the files contained within an azure blobstore.
func AzureBlobstoreList(config AzureBlobstoreConfig) ([]storage.Blob, error) {
// Create an authenticated client against the Azure cloud
rawClient, err := storage.NewBasicClient(config.Account, config.Token)
if err != nil {
```

```
return nil, err
}
client := rawClient.GetBlobService()

// List all the blobs from the container and return them
container := client.GetContainerReference(config.Container)

blobs, err := container.ListBlobs(storage.ListBlobsParameters{
MaxResults: 1024 * 1024 * 1024, // Yes, fetch all of them
Timeout:    3600,               // Yes, wait for all of them
})
if err != nil {
return nil, err
}
return blobs.Blobs, nil
}

// AzureBlobstoreDelete iterates over a list of files to delete and removes them
// from the blobstore.
func AzureBlobstoreDelete(config AzureBlobstoreConfig, blobs []storage.Blob) error {
if *DryRunFlag {
for _, blob := range blobs {
fmt.Printf("would delete %s (%s) from %s/%s\n", blob.Name, blob.Properties.LastModified,
config.Account, config.Container)
}
return nil
}
// Create an authenticated client against the Azure cloud
rawClient, err := storage.NewBasicClient(config.Account, config.Token)
if err != nil {
return err
}
client := rawClient.GetBlobService()

// Iterate over the blobs and delete them
for _, blob := range blobs {
if err := client.DeleteBlob(config.Container, blob.Name, nil); err != nil {
return err
}
}
return nil
}
```

51:F:\git\coin\ethereum\go-ethereum\internal\build\env.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package build

import (
"flag"
"fmt"
"os"
"strings"
)

var (
// These flags override values in build env.
GitCommitFlag   = flag.String("git-commit", "", `Overrides git commit hash embedded into executables`)
GitBranchFlag   = flag.String("git-branch", "", `Overrides git branch being built`)
GitTagFlag   = flag.String("git-tag", "", `Overrides git tag being built`)
BuildnumFlag   = flag.String("buildnum", "", `Overrides CI build number`)
PullRequestFlag = flag.Bool("pull-request", false, `Overrides pull request status of the build`)
CronJobFlag   = flag.Bool("cron-job", false, `Overrides cron job status of the build`)
)

// Environment contains metadata provided by the build environment.
type Environment struct {
Name   string // name of the environment
Repo   string // name of GitHub repo
Commit, Branch, Tag string // Git info
Buildnum   string
IsPullRequest   bool
IsCronJob   bool
}

func (env Environment) String() string {
return fmt.Sprintf("%s env (commit:%s branch:%s tag:%s buildnum:%s pr:%t)", env.Name, env.Commit, env.Branch, env.Tag, env.Buildnum, env.IsPullRequest)
}

// Env returns metadata about the current CI environment, falling back to LocalEnv
// if not running on CI.
func Env() Environment {

```
switch {
case os.Getenv("CI") == "true" && os.Getenv("TRAVIS") == "true":
return Environment{
Name:          "travis",
Repo:          os.Getenv("TRAVIS_REPO_SLUG"),
Commit:        os.Getenv("TRAVIS_COMMIT"),
Branch:        os.Getenv("TRAVIS_BRANCH"),
Tag:           os.Getenv("TRAVIS_TAG"),
Buildnum:      os.Getenv("TRAVIS_BUILD_NUMBER"),
IsPullRequest: os.Getenv("TRAVIS_PULL_REQUEST") != "false",
IsCronJob:     os.Getenv("TRAVIS_EVENT_TYPE") == "cron",
}
case os.Getenv("CI") == "True" && os.Getenv("APPVEYOR") == "True":
return Environment{
Name:          "appveyor",
Repo:          os.Getenv("APPVEYOR_REPO_NAME"),
Commit:        os.Getenv("APPVEYOR_REPO_COMMIT"),
Branch:        os.Getenv("APPVEYOR_REPO_BRANCH"),
Tag:           os.Getenv("APPVEYOR_REPO_TAG_NAME"),
Buildnum:      os.Getenv("APPVEYOR_BUILD_NUMBER"),
IsPullRequest: os.Getenv("APPVEYOR_PULL_REQUEST_NUMBER") != "",
IsCronJob:     os.Getenv("APPVEYOR_SCHEDULED_BUILD") == "True",
}
default:
return LocalEnv()
}
}

// LocalEnv returns build environment metadata gathered from git.
func LocalEnv() Environment {
env := applyEnvFlags(Environment{Name: "local", Repo: "ethereum/go-ethereum"})
if _, err := os.Stat(".git"); err != nil {
return env
}
if env.Commit == "" {
env.Commit = RunGit("rev-parse", "HEAD")
}
if env.Branch == "" {
if b := RunGit("rev-parse", "--abbrev-ref", "HEAD"); b != "HEAD" {
env.Branch = b
}
}
```

```
if env.Tag == "" {
env.Tag = firstLine(RunGit("tag", "-l", "--points-at", "HEAD"))
}
return env
}

func firstLine(s string) string {
return strings.Split(s, "\n")[0]
}

func applyEnvFlags(env Environment) Environment {
if !flag.Parsed() {
panic("you need to call flag.Parse before Env or LocalEnv")
}
if *GitCommitFlag != "" {
env.Commit = *GitCommitFlag
}
if *GitBranchFlag != "" {
env.Branch = *GitBranchFlag
}
if *GitTagFlag != "" {
env.Tag = *GitTagFlag
}
if *BuildnumFlag != "" {
env.Buildnum = *BuildnumFlag
}
if *PullRequestFlag {
env.IsPullRequest = true
}
if *CronJobFlag {
env.IsCronJob = true
}
return env
}

52:F:\git\coin\ethereum\go-ethereum\internal\build\pgp.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// signFile reads the contents of an input file and signs it (in armored format)
// with the key provided, placing the signature into the output file.

package build
```

```
import (
"bytes"
"fmt"
"os"

"golang.org/x/crypto/openpgp"
)

// PGPSignFile parses a PGP private key from the specified string and creates a
// signature file into the output parameter of the input file.
//
// Note, this method assumes a single key will be container in the pgpkey arg,
// furthermore that it is in armored format.
func PGPSignFile(input string, output string, pgpkey string) error {
// Parse the keyring and make sure we only have a single private key in it
keys, err := openpgp.ReadArmoredKeyRing(bytes.NewBufferString(pgpkey))
if err != nil {
return err
}
if len(keys) != 1 {
return fmt.Errorf("key count mismatch: have %d, want %d", len(keys), 1)
}
// Create the input and output streams for signing
in, err := os.Open(input)
if err != nil {
return err
}
defer in.Close()

out, err := os.Create(output)
if err != nil {
return err
}
defer out.Close()

// Generate the signature and return
return openpgp.ArmoredDetachSign(out, keys[0], in, nil)
}

53:F:\git\coin\ethereum\go-ethereum\internal\build\util.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package build

import (
"bytes"
"flag"
"fmt"
"io"
"io/ioutil"
"log"
"os"
"os/exec"
"path/filepath"
"runtime"
"strings"
"text/template"
)

var DryRunFlag = flag.Bool("n", false, "dry run, don't execute commands")

// MustRun executes the given command and exits the host process for
// any error.
func MustRun(cmd *exec.Cmd) {
fmt.Println(">>>", strings.Join(cmd.Args, " "))
if !*DryRunFlag {
cmd.Stderr = os.Stderr
cmd.Stdout = os.Stdout
if err := cmd.Run(); err != nil {
log.Fatal(err)
}
}
}

func MustRunCommand(cmd string, args ...string) {
MustRun(exec.Command(cmd, args...))
}

// GOPATH returns the value that the GOPATH environment
// variable should be set to.
func GOPATH() string {
if os.Getenv("GOPATH") == "" {
log.Fatal("GOPATH is not set")
```

```
}
return os.Getenv("GOPATH")
}

// VERSION returns the content of the VERSION file.
func VERSION() string {
version, err := ioutil.ReadFile("VERSION")
if err != nil {
log.Fatal(err)
}
return string(bytes.TrimSpace(version))
}

var warnedAboutGit bool

// RunGit runs a git subcommand and returns its output.
// The command must complete successfully.
func RunGit(args ...string) string {
cmd := exec.Command("git", args...)
var stdout, stderr bytes.Buffer
cmd.Stdout, cmd.Stderr = &stdout, &stderr
if err := cmd.Run(); err == exec.ErrNotFound {
if !warnedAboutGit {
log.Println("Warning: can't find 'git' in PATH")
warnedAboutGit = true
}
return ""
} else if err != nil {
log.Fatal(strings.Join(cmd.Args, " "), ": ", err, "\n", stderr.String())
}
return strings.TrimSpace(stdout.String())
}

// Render renders the given template file into outputFile.
func Render(templateFile, outputFile string, outputPerm os.FileMode, x interface{}) {
tpl := template.Must(template.ParseFiles(templateFile))
render(tpl, outputFile, outputPerm, x)
}

// RenderString renders the given template string into outputFile.
func RenderString(templateContent, outputFile string, outputPerm os.FileMode, x interface{}) {
tpl := template.Must(template.New("").Parse(templateContent))
```

```
render(tpl, outputFile, outputPerm, x)
}

func render(tpl *template.Template, outputFile string, outputPerm os.FileMode, x interface{}) {
if err := os.MkdirAll(filepath.Dir(outputFile), 0755); err != nil {
log.Fatal(err)
}
out, err := os.OpenFile(outputFile, os.O_CREATE|os.O_WRONLY|os.O_EXCL, outputPerm)
if err != nil {
log.Fatal(err)
}
if err := tpl.Execute(out, x); err != nil {
log.Fatal(err)
}
if err := out.Close(); err != nil {
log.Fatal(err)
}
}

// CopyFile copies a file.
func CopyFile(dst, src string, mode os.FileMode) {
if err := os.MkdirAll(filepath.Dir(dst), 0755); err != nil {
log.Fatal(err)
}
destFile, err := os.OpenFile(dst, os.O_CREATE|os.O_WRONLY|os.O_TRUNC, mode)
if err != nil {
log.Fatal(err)
}
defer destFile.Close()

srcFile, err := os.Open(src)
if err != nil {
log.Fatal(err)
}
defer srcFile.Close()

if _, err := io.Copy(destFile, srcFile); err != nil {
log.Fatal(err)
}
}

// ExpandPackagesNoVendor expands a cmd/go import path pattern, skipping
```

```
// vendored packages.
func ExpandPackagesNoVendor(patterns []string) []string {
expand := false
for _, pkg := range patterns {
if strings.Contains(pkg, "...") {
expand = true
}
}
if expand {
args := append([]string{"list"}, patterns...)
cmd := exec.Command(filepath.Join(runtime.GOROOT(), "bin", "go"), args...)
out, err := cmd.CombinedOutput()
if err != nil {
log.Fatalf("package listing failed: %v\n%s", err, string(out))
}
var packages []string
for _, line := range strings.Split(string(out), "\n") {
if !strings.Contains(line, "/vendor/") {
packages = append(packages, strings.TrimSpace(line))
}
}
return packages
}
return patterns
}
```

54:F:\git\coin\ethereum\go-ethereum\internal\cmdtest\test_cmd.go

```
// along with go-ethereum. If not, see <http://www.gnu.org/licenses/>.

package cmdtest

import (
"bufio"
"bytes"
"fmt"
"io"
"io/ioutil"
"os"
"os/exec"
"regexp"
"sync"
"testing"
```

```
"text/template"
"time"

"github.com/docker/docker/pkg/reexec"
)

func NewTestCmd(t *testing.T, data interface{}) *TestCmd {
return &TestCmd{T: t, Data: data}
}

type TestCmd struct {
// For total convenience, all testing methods are available.
*testing.T

Func    template.FuncMap
Data    interface{}
Cleanup func()

cmd    *exec.Cmd
stdout *bufio.Reader
stdin  io.WriteCloser
stderr *testlogger
}

// Run exec's the current binary using name as argv[0] which will trigger the
// reexec init function for that name (e.g. "geth-test" in cmd/geth/run_test.go)
func (tt *TestCmd) Run(name string, args ...string) {
tt.stderr = &testlogger{t: tt.T}
tt.cmd = &exec.Cmd{
Path:   reexec.Self(),
Args:   append([]string{name}, args...),
Stderr: tt.stderr,
}
stdout, err := tt.cmd.StdoutPipe()
if err != nil {
tt.Fatal(err)
}
tt.stdout = bufio.NewReader(stdout)
if tt.stdin, err = tt.cmd.StdinPipe(); err != nil {
tt.Fatal(err)
}
if err := tt.cmd.Start(); err != nil {
```

```
tt.Fatal(err)
}
}

// InputLine writes the given text to the childs stdin.
// This method can also be called from an expect template, e.g.:
//
//     geth.expect(`Passphrase: {{.InputLine "password"}}`)
func (tt *TestCmd) InputLine(s string) string {
io.WriteString(tt.stdin, s+"\n")
return ""
}

func (tt *TestCmd) SetTemplateFunc(name string, fn interface{}) {
if tt.Func == nil {
tt.Func = make(map[string]interface{})
}
tt.Func[name] = fn
}

// Expect runs its argument as a template, then expects the
// child process to output the result of the template within 5s.
//
// If the template starts with a newline, the newline is removed
// before matching.
func (tt *TestCmd) Expect(tplsource string) {
// Generate the expected output by running the template.
tpl := template.Must(template.New("").Funcs(tt.Func).Parse(tplsource))
wantbuf := new(bytes.Buffer)
if err := tpl.Execute(wantbuf, tt.Data); err != nil {
panic(err)
}
// Trim exactly one newline at the beginning. This makes tests look
// much nicer because all expect strings are at column 0.
want := bytes.TrimPrefix(wantbuf.Bytes(), []byte("\n"))
if err := tt.matchExactOutput(want); err != nil {
tt.Fatal(err)
}
tt.Logf("Matched stdout text:\n%s", want)
}

func (tt *TestCmd) matchExactOutput(want []byte) error {
```

```
buf := make([]byte, len(want))
n := 0
tt.withKillTimeout(func() { n, _ = io.ReadFull(tt.stdout, buf) })
buf = buf[:n]
if n < len(want) || !bytes.Equal(buf, want) {
// Grab any additional buffered output in case of mismatch
// because it might help with debugging.
buf = append(buf, make([]byte, tt.stdout.Buffered())...)
tt.stdout.Read(buf[n:])
// Find the mismatch position.
for i := 0; i < n; i++ {
if want[i] != buf[i] {
return fmt.Errorf("Output mismatch at :\n---------- (stdout text)\n%s%s\n---------- (expected text)\n%s",
buf[:i], buf[i:n], want)
}
}
if n < len(want) {
return fmt.Errorf("Not enough output, got until :\n---------- (stdout text)\n%s\n----------
(expected text)\n%s%s",
buf, want[:n], want[n:])
}
}
return nil
}

// ExpectRegexp expects the child process to output text matching the
// given regular expression within 5s.
//
// Note that an arbitrary amount of output may be consumed by the
// regular expression. This usually means that expect cannot be used
// after ExpectRegexp.
func (tt *TestCmd) ExpectRegexp(resource string) (*regexp.Regexp, []string) {
var (
re      = regexp.MustCompile(resource)
rtee    = &runeTee{in: tt.stdout}
matches []int
)
tt.withKillTimeout(func() { matches = re.FindReaderSubmatchIndex(rtee) })
output := rtee.buf.Bytes()
if matches == nil {
tt.Fatalf("Output did not match:\n---------- (stdout text)\n%s\n---------- (regular
```

```
expression)\n%s",
output, resource)
return re, nil
}
tt.Logf("Matched stdout text:\n%s", output)
var submatches []string
for i := 0; i < len(matches); i += 2 {
submatch := string(output[matches[i]:matches[i+1]])
submatches = append(submatches, submatch)
}
return re, submatches
}

// ExpectExit expects the child process to exit within 5s without
// printing any additional text on stdout.
func (tt *TestCmd) ExpectExit() {
var output []byte
tt.withKillTimeout(func() {
output, _ = ioutil.ReadAll(tt.stdout)
})
tt.WaitExit()
if tt.Cleanup != nil {
tt.Cleanup()
}
if len(output) > 0 {
tt.Errorf("Unmatched stdout text:\n%s", output)
}
}

func (tt *TestCmd) WaitExit() {
tt.cmd.Wait()
}

func (tt *TestCmd) Interrupt() {
tt.cmd.Process.Signal(os.Interrupt)
}

// StderrText returns any stderr output written so far.
// The returned text holds all log lines after ExpectExit has
// returned.
func (tt *TestCmd) StderrText() string {
tt.stderr.mu.Lock()
```

```
defer tt.stderr.mu.Unlock()
return tt.stderr.buf.String()
}

func (tt *TestCmd) CloseStdin() {
tt.stdin.Close()
}

func (tt *TestCmd) Kill() {
tt.cmd.Process.Kill()
if tt.Cleanup != nil {
tt.Cleanup()
}
}

func (tt *TestCmd) withKillTimeout(fn func()) {
timeout := time.AfterFunc(5*time.Second, func() {
tt.Log("killing the child process (timeout)")
tt.Kill()
})
defer timeout.Stop()
fn()
}

// testlogger logs all written lines via t.Log and also
// collects them for later inspection.
type testlogger struct {
t   *testing.T
mu  sync.Mutex
buf bytes.Buffer
}

func (tl *testlogger) Write(b []byte) (n int, err error) {
lines := bytes.Split(b, []byte("\n"))
for _, line := range lines {
if len(line) > 0 {
tl.t.Logf("(stderr) %s", line)
}
}
tl.mu.Lock()
tl.buf.Write(b)
tl.mu.Unlock()
```

```
return len(b), err
}

// runeTee collects text read through it into buf.
type runeTee struct {
in interface {
io.Reader
io.ByteReader
io.RuneReader
}
buf bytes.Buffer
}

func (rtee *runeTee) Read(b []byte) (n int, err error) {
n, err = rtee.in.Read(b)
rtee.buf.Write(b[:n])
return n, err
}

func (rtee *runeTee) ReadRune() (r rune, size int, err error) {
r, size, err = rtee.in.ReadRune()
if err == nil {
rtee.buf.WriteRune(r)
}
return r, size, err
}

func (rtee *runeTee) ReadByte() (b byte, err error) {
b, err = rtee.in.ReadByte()
if err == nil {
rtee.buf.WriteByte(b)
}
return b, err
}

55:F:\git\coin\ethereum\go-ethereum\internal\debug\api.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package debug interfaces Go runtime debugging facilities.
// This package is mostly glue code making these facilities available
// through the CLI and RPC subsystem. If you want to use them from Go code,
// use package runtime instead.
```

```
package debug

import (
"errors"
"io"
"os"
"os/user"
"path/filepath"
"runtime"
"runtime/debug"
"runtime/pprof"
"strings"
"sync"
"time"

"github.com/ethereum/go-ethereum/log"
)

// Handler is the global debugging handler.
var Handler = new(HandlerT)

// HandlerT implements the debugging API.
// Do not create values of this type, use the one
// in the Handler variable instead.
type HandlerT struct {
mu        sync.Mutex
cpuW      io.WriteCloser
cpuFile   string
traceW    io.WriteCloser
traceFile string
}

// Verbosity sets the log verbosity ceiling. The verbosity of individual packages
// and source files can be raised using Vmodule.
func (*HandlerT) Verbosity(level int) {
glogger.Verbosity(log.Lvl(level))
}

// Vmodule sets the log verbosity pattern. See package log for details on the
// pattern syntax.
func (*HandlerT) Vmodule(pattern string) error {
return glogger.Vmodule(pattern)
```

```
}

// BacktraceAt sets the log backtrace location. See package log for details on
// the pattern syntax.
func (*HandlerT) BacktraceAt(location string) error {
return glogger.BacktraceAt(location)
}

// MemStats returns detailed runtime memory statistics.
func (*HandlerT) MemStats() *runtime.MemStats {
s := new(runtime.MemStats)
runtime.ReadMemStats(s)
return s
}

// GcStats returns GC statistics.
func (*HandlerT) GcStats() *debug.GCStats {
s := new(debug.GCStats)
debug.ReadGCStats(s)
return s
}

// CpuProfile turns on CPU profiling for nsec seconds and writes
// profile data to file.
func (h *HandlerT) CpuProfile(file string, nsec uint) error {
if err := h.StartCPUProfile(file); err != nil {
return err
}
time.Sleep(time.Duration(nsec) * time.Second)
h.StopCPUProfile()
return nil
}

// StartCPUProfile turns on CPU profiling, writing to the given file.
func (h *HandlerT) StartCPUProfile(file string) error {
h.mu.Lock()
defer h.mu.Unlock()
if h.cpuW != nil {
return errors.New("CPU profiling already in progress")
}
f, err := os.Create(expandHome(file))
if err != nil {
```

```
return err
}
if err := pprof.StartCPUProfile(f); err != nil {
f.Close()
return err
}
h.cpuW = f
h.cpuFile = file
log.Info("CPU profiling started", "dump", h.cpuFile)
return nil
}

// StopCPUProfile stops an ongoing CPU profile.
func (h *HandlerT) StopCPUProfile() error {
h.mu.Lock()
defer h.mu.Unlock()
pprof.StopCPUProfile()
if h.cpuW == nil {
return errors.New("CPU profiling not in progress")
}
log.Info("Done writing CPU profile", "dump", h.cpuFile)
h.cpuW.Close()
h.cpuW = nil
h.cpuFile = ""
return nil
}

// GoTrace turns on tracing for nsec seconds and writes
// trace data to file.
func (h *HandlerT) GoTrace(file string, nsec uint) error {
if err := h.StartGoTrace(file); err != nil {
return err
}
time.Sleep(time.Duration(nsec) * time.Second)
h.StopGoTrace()
return nil
}

// BlockProfile turns on CPU profiling for nsec seconds and writes
// profile data to file. It uses a profile rate of 1 for most accurate
// information. If a different rate is desired, set the rate
// and write the profile manually.
```

```
func (*HandlerT) BlockProfile(file string, nsec uint) error {
runtime.SetBlockProfileRate(1)
time.Sleep(time.Duration(nsec) * time.Second)
defer runtime.SetBlockProfileRate(0)
return writeProfile("block", file)
}

// SetBlockProfileRate sets the rate of goroutine block profile data collection.
// rate 0 disables block profiling.
func (*HandlerT) SetBlockProfileRate(rate int) {
runtime.SetBlockProfileRate(rate)
}

// WriteBlockProfile writes a goroutine blocking profile to the given file.
func (*HandlerT) WriteBlockProfile(file string) error {
return writeProfile("block", file)
}

// WriteMemProfile writes an allocation profile to the given file.
// Note that the profiling rate cannot be set through the API,
// it must be set on the command line.
func (*HandlerT) WriteMemProfile(file string) error {
return writeProfile("heap", file)
}

// Stacks returns a printed representation of the stacks of all goroutines.
func (*HandlerT) Stacks() string {
buf := make([]byte, 1024*1024)
buf = buf[:runtime.Stack(buf, true)]
return string(buf)
}

func writeProfile(name, file string) error {
p := pprof.Lookup(name)
log.Info("Writing profile records", "count", p.Count(), "type", name, "dump", file)
f, err := os.Create(expandHome(file))
if err != nil {
return err
}
defer f.Close()
return p.WriteTo(f, 0)
}
```

```
// expands home directory in file paths.
// ~someuser/tmp will not be expanded.
func expandHome(p string) string {
if strings.HasPrefix(p, "~/") || strings.HasPrefix(p, "~\\") {
home := os.Getenv("HOME")
if home == "" {
if usr, err := user.Current(); err == nil {
home = usr.HomeDir
}
}
if home != "" {
p = home + p[1:]
}
}
return filepath.Clean(p)
}

56:F:\git\coin\ethereum\go-ethereum\internal\debug\flags.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package debug

import (
"fmt"
"io"
"net/http"
_ "net/http/pprof"
"os"
"runtime"

"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/log/term"
colorable "github.com/mattn/go-colorable"
"gopkg.in/urfave/cli.v1"
)

var (
verbosityFlag = cli.IntFlag{
Name:  "verbosity",
Usage: "Logging verbosity: 0=silent, 1=error, 2=warn, 3=info, 4=debug, 5=detail",
Value: 3,
```

```
}
vmoduleFlag = cli.StringFlag{
Name:  "vmodule",
Usage: "Per-module verbosity: comma-separated list of <pattern>=<level> (e.g. eth/*=5,p2p=4)",
Value: "",
}
backtraceAtFlag = cli.StringFlag{
Name:  "backtrace",
Usage: "Request a stack trace at a specific logging statement (e.g. \"block.go:271\")",
Value: "",
}
debugFlag = cli.BoolFlag{
Name:  "debug",
Usage: "Prepends log messages with call-site location (file and line number)",
}
pprofFlag = cli.BoolFlag{
Name:  "pprof",
Usage: "Enable the pprof HTTP server",
}
pprofPortFlag = cli.IntFlag{
Name:  "pprofport",
Usage: "pprof HTTP server listening port",
Value: 6060,
}
pprofAddrFlag = cli.StringFlag{
Name:  "pprofaddr",
Usage: "pprof HTTP server listening interface",
Value: "127.0.0.1",
}
memprofilerateFlag = cli.IntFlag{
Name:  "memprofilerate",
Usage: "Turn on memory profiling with the given rate",
Value: runtime.MemProfileRate,
}
blockprofilerateFlag = cli.IntFlag{
Name:  "blockprofilerate",
Usage: "Turn on block profiling with the given rate",
}
cpuprofileFlag = cli.StringFlag{
Name:  "cpuprofile",
Usage: "Write CPU profile to the given file",
}
```

```
traceFlag = cli.StringFlag{
Name:  "trace",
Usage: "Write execution trace to the given file",
}
)

// Flags holds all command-line flags required for debugging.
var Flags = []cli.Flag{
verbosityFlag, vmoduleFlag, backtraceAtFlag, debugFlag,
pprofFlag, pprofAddrFlag, pprofPortFlag,
memprofilerateFlag, blockprofilerateFlag, cpuprofileFlag, traceFlag,
}

var glogger *log.GlogHandler

func init() {
usecolor := term.IsTty(os.Stderr.Fd()) && os.Getenv("TERM") != "dumb"
output := io.Writer(os.Stderr)
if usecolor {
output = colorable.NewColorableStderr()
}
glogger = log.NewGlogHandler(log.StreamHandler(output, log.TerminalFormat(usecolor)))
}

// Setup initializes profiling and logging based on the CLI flags.
// It should be called as early as possible in the program.
func Setup(ctx *cli.Context) error {
// logging
log.PrintOrigins(ctx.GlobalBool(debugFlag.Name))
glogger.Verbosity(log.Lvl(ctx.GlobalInt(verbosityFlag.Name)))
glogger.Vmodule(ctx.GlobalString(vmoduleFlag.Name))
glogger.BacktraceAt(ctx.GlobalString(backtraceAtFlag.Name))
log.Root().SetHandler(glogger)

// profiling, tracing
runtime.MemProfileRate = ctx.GlobalInt(memprofilerateFlag.Name)
Handler.SetBlockProfileRate(ctx.GlobalInt(blockprofilerateFlag.Name))
if traceFile := ctx.GlobalString(traceFlag.Name); traceFile != "" {
if err := Handler.StartGoTrace(traceFile); err != nil {
return err
}
}
```

```
if cpuFile := ctx.GlobalString(cpuprofileFlag.Name); cpuFile != "" {
if err := Handler.StartCPUProfile(cpuFile); err != nil {
return err
}
}

// pprof server
if ctx.GlobalBool(pprofFlag.Name) {
address := fmt.Sprintf("%s:%d", ctx.GlobalString(pprofAddrFlag.Name),
ctx.GlobalInt(pprofPortFlag.Name))
go func() {
log.Info("Starting pprof server", "addr", fmt.Sprintf("http://%s/debug/pprof", address))
if err := http.ListenAndServe(address, nil); err != nil {
log.Error("Failure in running pprof server", "err", err)
}
}()
}
return nil
}

// Exit stops all running profiles, flushing their output to the
// respective file.
func Exit() {
Handler.StopCPUProfile()
Handler.StopGoTrace()
}

57:F:\git\coin\ethereum\go-ethereum\internal\debug\loudpanic.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// +build go1.6

package debug

import "runtime/debug"

// LoudPanic panics in a way that gets all goroutine stacks printed on stderr.
func LoudPanic(x interface{}) {
debug.SetTraceback("all")
panic(x)
}
```

58:F:\git\coin\ethereum\go-ethereum\internal\debug\loudpanic_fallback.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// +build !go1.6

package debug

// LoudPanic panics in a way that gets all goroutine stacks printed on stderr.
func LoudPanic(x interface{}) {
panic(x)
}

59:F:\git\coin\ethereum\go-ethereum\internal\debug\trace.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

//+build go1.5

package debug

import (
"errors"
"os"
"runtime/trace"

"github.com/ethereum/go-ethereum/log"
)

// StartGoTrace turns on tracing, writing to the given file.
func (h *HandlerT) StartGoTrace(file string) error {
h.mu.Lock()
defer h.mu.Unlock()
if h.traceW != nil {
return errors.New("trace already in progress")
}
f, err := os.Create(expandHome(file))
if err != nil {
return err
}
if err := trace.Start(f); err != nil {
f.Close()
return err
}

```
h.traceW = f
h.traceFile = file
log.Info("Go tracing started", "dump", h.traceFile)
return nil
}

// StopTrace stops an ongoing trace.
func (h *HandlerT) StopGoTrace() error {
h.mu.Lock()
defer h.mu.Unlock()
trace.Stop()
if h.traceW == nil {
return errors.New("trace not in progress")
}
log.Info("Done writing Go trace", "dump", h.traceFile)
h.traceW.Close()
h.traceW = nil
h.traceFile = ""
return nil
}

60:F:\git\coin\ethereum\go-ethereum\internal\debug\trace_fallback.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

//+build !go1.5

// no-op implementation of tracing methods for Go < 1.5.

package debug

import "errors"

func (*HandlerT) StartGoTrace(string) error {
return errors.New("tracing is not supported on Go < 1.5")
}

func (*HandlerT) StopGoTrace() error {
return errors.New("tracing is not supported on Go < 1.5")
}

61:F:\git\coin\ethereum\go-ethereum\internal\ethapi\addrlock.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package ethapi

import (
"sync"

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

type AddrLocker struct {
mu    sync.Mutex
locks map[common.Address]*sync.Mutex
}

// lock returns the lock of the given address.
func (l *AddrLocker) lock(address common.Address) *sync.Mutex {
l.mu.Lock()
defer l.mu.Unlock()
if l.locks == nil {
l.locks = make(map[common.Address]*sync.Mutex)
}
if _, ok := l.locks[address]; !ok {
l.locks[address] = new(sync.Mutex)
}
return l.locks[address]
}

// LockAddr locks an account's mutex. This is used to prevent another tx getting the
// same nonce until the lock is released. The mutex prevents the (an identical nonce) from
// being read again during the time that the first transaction is being signed.
func (l *AddrLocker) LockAddr(address common.Address) {
l.lock(address).Lock()
}

// UnlockAddr unlocks the mutex of the given account.
func (l *AddrLocker) UnlockAddr(address common.Address) {
l.lock(address).Unlock()
}

62:F:\git\coin\ethereum\go-ethereum\internal\ethapi\api.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package ethapi

import (
"bytes"
"context"
"errors"
"fmt"
"math/big"
"strings"
"time"

"github.com/ethereum/go-ethereum/accounts"
"github.com/ethereum/go-ethereum/accounts/keystore"
"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/consensus/ethash"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rlp"
"github.com/ethereum/go-ethereum/rpc"
"github.com/syndtr/goleveldb/leveldb"
"github.com/syndtr/goleveldb/leveldb/util"
)

const (
defaultGas      = 90000
defaultGasPrice = 50 * params.Shannon
emptyHex        = "0x"
)

// PublicEthereumAPI provides an API to access Ethereum related information.
// It offers only methods that operate on public data that is freely available to anyone.
type PublicEthereumAPI struct {
b Backend
}
```

```
// NewPublicEthereumAPI creates a new Etheruem protocol API.
func NewPublicEthereumAPI(b Backend) *PublicEthereumAPI {
return &PublicEthereumAPI{b}
}

// GasPrice returns a suggestion for a gas price.
func (s *PublicEthereumAPI) GasPrice(ctx context.Context) (*big.Int, error) {
return s.b.SuggestPrice(ctx)
}

// ProtocolVersion returns the current Ethereum protocol version this node supports
func (s *PublicEthereumAPI) ProtocolVersion() hexutil.Uint {
return hexutil.Uint(s.b.ProtocolVersion())
}

// Syncing returns false in case the node is currently not syncing with the network. It can be up to date or has not
// yet received the latest block headers from its pears. In case it is synchronizing:
// - startingBlock: block number this node started to synchronise from
// - currentBlock:  block number this node is currently importing
// - highestBlock:  block number of the highest block header this node has received from peers
// - pulledStates:  number of state entries processed until now
// - knownStates:   number of known state entries that still need to be pulled
func (s *PublicEthereumAPI) Syncing() (interface{}, error) {
progress := s.b.Downloader().Progress()

// Return not syncing if the synchronisation already completed
if progress.CurrentBlock >= progress.HighestBlock {
return false, nil
}
// Otherwise gather the block sync stats
return map[string]interface{}{
"startingBlock": hexutil.Uint64(progress.StartingBlock),
"currentBlock":  hexutil.Uint64(progress.CurrentBlock),
"highestBlock":  hexutil.Uint64(progress.HighestBlock),
"pulledStates":  hexutil.Uint64(progress.PulledStates),
"knownStates":   hexutil.Uint64(progress.KnownStates),
}, nil
}

// PublicTxPoolAPI offers and API for the transaction pool. It only operates on data that is non
```

```
confidential.
type PublicTxPoolAPI struct {
b Backend
}

// NewPublicTxPoolAPI creates a new tx pool service that gives information about the transaction pool.
func NewPublicTxPoolAPI(b Backend) *PublicTxPoolAPI {
return &PublicTxPoolAPI{b}
}

// Content returns the transactions contained within the transaction pool.
func (s *PublicTxPoolAPI) Content() map[string]map[string]map[string]*RPCTransaction {
content := map[string]map[string]map[string]*RPCTransaction{
"pending": make(map[string]map[string]*RPCTransaction),
"queued":  make(map[string]map[string]*RPCTransaction),
}
pending, queue := s.b.TxPoolContent()

// Flatten the pending transactions
for account, txs := range pending {
dump := make(map[string]*RPCTransaction)
for _, tx := range txs {
dump[fmt.Sprintf("%d", tx.Nonce())] = newRPCPendingTransaction(tx)
}
content["pending"][account.Hex()] = dump
}
// Flatten the queued transactions
for account, txs := range queue {
dump := make(map[string]*RPCTransaction)
for _, tx := range txs {
dump[fmt.Sprintf("%d", tx.Nonce())] = newRPCPendingTransaction(tx)
}
content["queued"][account.Hex()] = dump
}
return content
}

// Status returns the number of pending and queued transaction in the pool.
func (s *PublicTxPoolAPI) Status() map[string]hexutil.Uint {
pending, queue := s.b.Stats()
return map[string]hexutil.Uint{
```

```
"pending": hexutil.Uint(pending),
"queued":  hexutil.Uint(queue),
}
}

// Inspect retrieves the content of the transaction pool and flattens it into an
// easily inspectable list.
func (s *PublicTxPoolAPI) Inspect() map[string]map[string]map[string]string {
content := map[string]map[string]map[string]string{
"pending": make(map[string]map[string]string),
"queued":  make(map[string]map[string]string),
}
pending, queue := s.b.TxPoolContent()

// Define a formatter to flatten a transaction into a string
var format = func(tx *types.Transaction) string {
if to := tx.To(); to != nil {
return fmt.Sprintf("%s: %v wei + %v × %v gas", tx.To().Hex(), tx.Value(), tx.Gas(), tx.GasPrice())
}
return fmt.Sprintf("contract creation: %v wei + %v × %v gas", tx.Value(), tx.Gas(), tx.GasPrice())
}
// Flatten the pending transactions
for account, txs := range pending {
dump := make(map[string]string)
for _, tx := range txs {
dump[fmt.Sprintf("%d", tx.Nonce())] = format(tx)
}
content["pending"][account.Hex()] = dump
}
// Flatten the queued transactions
for account, txs := range queue {
dump := make(map[string]string)
for _, tx := range txs {
dump[fmt.Sprintf("%d", tx.Nonce())] = format(tx)
}
content["queued"][account.Hex()] = dump
}
return content
}

// PublicAccountAPI provides an API to access accounts managed by this node.
// It offers only methods that can retrieve accounts.
```

```
type PublicAccountAPI struct {
am *accounts.Manager
}

// NewPublicAccountAPI creates a new PublicAccountAPI.
func NewPublicAccountAPI(am *accounts.Manager) *PublicAccountAPI {
return &PublicAccountAPI{am: am}
}

// Accounts returns the collection of accounts this node manages
func (s *PublicAccountAPI) Accounts() []common.Address {
addresses := make([]common.Address, 0) // return [] instead of nil if empty
for _, wallet := range s.am.Wallets() {
for _, account := range wallet.Accounts() {
addresses = append(addresses, account.Address)
}
}
return addresses
}

// PrivateAccountAPI provides an API to access accounts managed by this node.
// It offers methods to create, (un)lock en list accounts. Some methods accept
// passwords and are therefore considered private by default.
type PrivateAccountAPI struct {
am        *accounts.Manager
nonceLock *AddrLocker
b         Backend
}

// NewPrivateAccountAPI create a new PrivateAccountAPI.
func NewPrivateAccountAPI(b Backend, nonceLock *AddrLocker) *PrivateAccountAPI {
return &PrivateAccountAPI{
am:        b.AccountManager(),
nonceLock: nonceLock,
b:         b,
}
}

// ListAccounts will return a list of addresses for accounts this node manages.
func (s *PrivateAccountAPI) ListAccounts() []common.Address {
addresses := make([]common.Address, 0) // return [] instead of nil if empty
for _, wallet := range s.am.Wallets() {
```

```
for _, account := range wallet.Accounts() {
addresses = append(addresses, account.Address)
}
}
return addresses
}

// rawWallet is a JSON representation of an accounts.Wallet interface, with its
// data contents extracted into plain fields.
type rawWallet struct {
URL      string             `json:"url"`
Status   string             `json:"status"`
Accounts []accounts.Account `json:"accounts"`
}

// ListWallets will return a list of wallets this node manages.
func (s *PrivateAccountAPI) ListWallets() []rawWallet {
wallets := make([]rawWallet, 0) // return [] instead of nil if empty
for _, wallet := range s.am.Wallets() {
wallets = append(wallets, rawWallet{
URL:      wallet.URL().String(),
Status:   wallet.Status(),
Accounts: wallet.Accounts(),
})
}
return wallets
}

// DeriveAccount requests a HD wallet to derive a new account, optionally pinning
// it for later reuse.
func (s *PrivateAccountAPI) DeriveAccount(url string, path string, pin *bool) (accounts.Account,
error) {
wallet, err := s.am.Wallet(url)
if err != nil {
return accounts.Account{}, err
}
derivPath, err := accounts.ParseDerivationPath(path)
if err != nil {
return accounts.Account{}, err
}
if pin == nil {
pin = new(bool)
```

```
}
return wallet.Derive(derivPath, *pin)
}

// NewAccount will create a new account and returns the address for the new account.
func (s *PrivateAccountAPI) NewAccount(password string) (common.Address, error) {
acc, err := fetchKeystore(s.am).NewAccount(password)
if err == nil {
return acc.Address, nil
}
return common.Address{}, err
}

// fetchKeystore retrives the encrypted keystore from the account manager.
func fetchKeystore(am *accounts.Manager) *keystore.KeyStore {
return am.Backends(keystore.KeyStoreType)[0].(*keystore.KeyStore)
}

// ImportRawKey stores the given hex encoded ECDSA key into the key directory,
// encrypting it with the passphrase.
func (s *PrivateAccountAPI) ImportRawKey(privkey string, password string) (common.Address,
error) {
key, err := crypto.HexToECDSA(privkey)
if err != nil {
return common.Address{}, err
}
acc, err := fetchKeystore(s.am).ImportECDSA(key, password)
return acc.Address, err
}

// UnlockAccount will unlock the account associated with the given address with
// the given password for duration seconds. If duration is nil it will use a
// default of 300 seconds. It returns an indication if the account was unlocked.
func (s *PrivateAccountAPI) UnlockAccount(addr common.Address, password string, duration
*uint64) (bool, error) {
const max = uint64(time.Duration(math.MaxInt64) / time.Second)
var d time.Duration
if duration == nil {
d = 300 * time.Second
} else if *duration > max {
return false, errors.New("unlock duration too large")
} else {
```

```
d = time.Duration(*duration) * time.Second
}
err := fetchKeystore(s.am).TimedUnlock(accounts.Account{Address: addr}, password, d)
return err == nil, err
}

// LockAccount will lock the account associated with the given address when it's unlocked.
func (s *PrivateAccountAPI) LockAccount(addr common.Address) bool {
return fetchKeystore(s.am).Lock(addr) == nil
}

// SendTransaction will create a transaction from the given arguments and
// tries to sign it with the key associated with args.To. If the given passwd isn't
// able to decrypt the key it fails.
func (s *PrivateAccountAPI) SendTransaction(ctx context.Context, args SendTxArgs, passwd
string) (common.Hash, error) {
// Look up the wallet containing the requested signer
account := accounts.Account{Address: args.From}

wallet, err := s.am.Find(account)
if err != nil {
return common.Hash{}, err
}

if args.Nonce == nil {
// Hold the addresse's mutex around signing to prevent concurrent assignment of
// the same nonce to multiple accounts.
s.nonceLock.LockAddr(args.From)
defer s.nonceLock.UnlockAddr(args.From)
}

// Set some sanity defaults and terminate on failure
if err := args.setDefaults(ctx, s.b); err != nil {
return common.Hash{}, err
}
// Assemble the transaction and sign with the wallet
tx := args.toTransaction()

var chainID *big.Int
if config := s.b.ChainConfig(); config.IsEIP155(s.b.CurrentBlock().Number()) {
chainID = config.ChainId
}
```

```
signed, err := wallet.SignTxWithPassphrase(account, passwd, tx, chainID)
if err != nil {
return common.Hash{}, err
}
return submitTransaction(ctx, s.b, signed)
}

// signHash is a helper function that calculates a hash for the given message that can be
// safely used to calculate a signature from.
//
// The hash is calulcated as
//   keccak256("\x19Ethereum Signed Message:\n"${message length}${message}).
//
// This gives context to the signed message and prevents signing of transactions.
func signHash(data []byte) []byte {
msg := fmt.Sprintf("\x19Ethereum Signed Message:\n%d%s", len(data), data)
return crypto.Keccak256([]byte(msg))
}

// Sign calculates an Ethereum ECDSA signature for:
// keccack256("\x19Ethereum Signed Message:\n" + len(message) + message))
//
// Note, the produced signature conforms to the secp256k1 curve R, S and V values,
// where the V value will be 27 or 28 for legacy reasons.
//
// The key used to calculate the signature is decrypted with the given password.
//
// https://github.com/ethereum/go-ethereum/wiki/Management-APIs#personal_sign
func (s *PrivateAccountAPI) Sign(ctx context.Context, data hexutil.Bytes, addr common.Address,
passwd string) (hexutil.Bytes, error) {
// Look up the wallet containing the requested signer
account := accounts.Account{Address: addr}

wallet, err := s.b.AccountManager().Find(account)
if err != nil {
return nil, err
}
// Assemble sign the data with the wallet
signature, err := wallet.SignHashWithPassphrase(account, passwd, signHash(data))
if err != nil {
return nil, err
}
```

```
signature[64] += 27 // Transform V from 0/1 to 27/28 according to the yellow paper
return signature, nil
}

// EcRecover returns the address for the account that was used to create the signature.
// Note, this function is compatible with eth_sign and personal_sign. As such it recovers
// the address of:
// hash = keccak256("\x19Ethereum Signed Message:\n"${message length}${message})
// addr = ecrecover(hash, signature)
//
// Note, the signature must conform to the secp256k1 curve R, S and V values, where
// the V value must be be 27 or 28 for legacy reasons.
//
// https://github.com/ethereum/go-ethereum/wiki/Management-APIs#personal_ecRecover
func (s *PrivateAccountAPI) EcRecover(ctx context.Context, data, sig hexutil.Bytes)
(common.Address, error) {
if len(sig) != 65 {
return common.Address{}, fmt.Errorf("signature must be 65 bytes long")
}
if sig[64] != 27 && sig[64] != 28 {
return common.Address{}, fmt.Errorf("invalid Ethereum signature (V is not 27 or 28)")
}
sig[64] -= 27 // Transform yellow paper V from 27/28 to 0/1

rpk, err := crypto.Ecrecover(signHash(data), sig)
if err != nil {
return common.Address{}, err
}
pubKey := crypto.ToECDSAPub(rpk)
recoveredAddr := crypto.PubkeyToAddress(*pubKey)
return recoveredAddr, nil
}

// SignAndSendTransaction was renamed to SendTransaction. This method is deprecated
// and will be removed in the future. It primary goal is to give clients time to update.
func (s *PrivateAccountAPI) SignAndSendTransaction(ctx context.Context, args SendTxArgs,
passwd string) (common.Hash, error) {
return s.SendTransaction(ctx, args, passwd)
}

// PublicBlockChainAPI provides an API to access the Ethereum blockchain.
// It offers only methods that operate on public data that is freely available to anyone.
```

```
type PublicBlockChainAPI struct {
b Backend
}

// NewPublicBlockChainAPI creates a new Etheruem blockchain API.
func NewPublicBlockChainAPI(b Backend) *PublicBlockChainAPI {
return &PublicBlockChainAPI{b}
}

// BlockNumber returns the block number of the chain head.
func (s *PublicBlockChainAPI) BlockNumber() *big.Int {
header, _ := s.b.HeaderByNumber(context.Background(), rpc.LatestBlockNumber) // latest header
should always be available
return header.Number
}

// GetBalance returns the amount of wei for the given address in the state of the
// given block number. The rpc.LatestBlockNumber and rpc.PendingBlockNumber meta
// block numbers are also allowed.
func (s *PublicBlockChainAPI) GetBalance(ctx context.Context, address common.Address,
blockNr rpc.BlockNumber) (*big.Int, error) {
state, _, err := s.b.StateAndHeaderByNumber(ctx, blockNr)
if state == nil || err != nil {
return nil, err
}
b := state.GetBalance(address)
return b, state.Error()
}

// GetBlockByNumber returns the requested block. When blockNr is -1 the chain head is returned.
When fullTx is true all
// transactions in the block are returned in full detail, otherwise only the transaction hash is
returned.
func (s *PublicBlockChainAPI) GetBlockByNumber(ctx context.Context, blockNr rpc.BlockNumber,
fullTx bool) (map[string]interface{}, error) {
block, err := s.b.BlockByNumber(ctx, blockNr)
if block != nil {
response, err := s.rpcOutputBlock(block, true, fullTx)
if err == nil && blockNr == rpc.PendingBlockNumber {
// Pending blocks need to nil out a few fields
for _, field := range []string{"hash", "nonce", "miner"} {
response[field] = nil
```

```
}
}
return response, err
}
return nil, err
}

// GetBlockByHash returns the requested block. When fullTx is true all transactions in the block
are returned in full
// detail, otherwise only the transaction hash is returned.
func (s *PublicBlockChainAPI) GetBlockByHash(ctx context.Context, blockHash common.Hash,
fullTx bool) (map[string]interface{}, error) {
block, err := s.b.GetBlock(ctx, blockHash)
if block != nil {
return s.rpcOutputBlock(block, true, fullTx)
}
return nil, err
}

// GetUncleByBlockNumberAndIndex returns the uncle block for the given block hash and index.
When fullTx is true
// all transactions in the block are returned in full detail, otherwise only the transaction hash is
returned.
func (s *PublicBlockChainAPI) GetUncleByBlockNumberAndIndex(ctx context.Context, blockNr
rpc.BlockNumber, index hexutil.Uint) (map[string]interface{}, error) {
block, err := s.b.BlockByNumber(ctx, blockNr)
if block != nil {
uncles := block.Uncles()
if index >= hexutil.Uint(len(uncles)) {
log.Debug("Requested uncle not found", "number", blockNr, "hash", block.Hash(), "index", index)
return nil, nil
}
block = types.NewBlockWithHeader(uncles[index])
return s.rpcOutputBlock(block, false, false)
}
return nil, err
}

// GetUncleByBlockHashAndIndex returns the uncle block for the given block hash and index.
When fullTx is true
// all transactions in the block are returned in full detail, otherwise only the transaction hash is
returned.
```

```
func (s *PublicBlockChainAPI) GetUncleByBlockHashAndIndex(ctx context.Context, blockHash
common.Hash, index hexutil.Uint) (map[string]interface{}, error) {
block, err := s.b.GetBlock(ctx, blockHash)
if block != nil {
uncles := block.Uncles()
if index >= hexutil.Uint(len(uncles)) {
log.Debug("Requested uncle not found", "number", block.Number(), "hash", blockHash, "index",
index)
return nil, nil
}
block = types.NewBlockWithHeader(uncles[index])
return s.rpcOutputBlock(block, false, false)
}
return nil, err
}

// GetUncleCountByBlockNumber returns number of uncles in the block for the given block number
func (s *PublicBlockChainAPI) GetUncleCountByBlockNumber(ctx context.Context, blockNr
rpc.BlockNumber) *hexutil.Uint {
if block, _ := s.b.BlockByNumber(ctx, blockNr); block != nil {
n := hexutil.Uint(len(block.Uncles()))
return &n
}
return nil
}

// GetUncleCountByBlockHash returns number of uncles in the block for the given block hash
func (s *PublicBlockChainAPI) GetUncleCountByBlockHash(ctx context.Context, blockHash
common.Hash) *hexutil.Uint {
if block, _ := s.b.GetBlock(ctx, blockHash); block != nil {
n := hexutil.Uint(len(block.Uncles()))
return &n
}
return nil
}

// GetCode returns the code stored at the given address in the state for the given block number.
func (s *PublicBlockChainAPI) GetCode(ctx context.Context, address common.Address, blockNr
rpc.BlockNumber) (hexutil.Bytes, error) {
state, _, err := s.b.StateAndHeaderByNumber(ctx, blockNr)
if state == nil || err != nil {
return nil, err
```

```
}
code := state.GetCode(address)
return code, state.Error()
}

// GetStorageAt returns the storage from the state at the given address, key and
// block number. The rpc.LatestBlockNumber and rpc.PendingBlockNumber meta block
// numbers are also allowed.
func (s *PublicBlockChainAPI) GetStorageAt(ctx context.Context, address common.Address, key
string, blockNr rpc.BlockNumber) (hexutil.Bytes, error) {
state, _, err := s.b.StateAndHeaderByNumber(ctx, blockNr)
if state == nil || err != nil {
return nil, err
}
res := state.GetState(address, common.HexToHash(key))
return res[:], state.Error()
}

// callmsg is the message type used for call transitions.
type callmsg struct {
addr          common.Address
to           *common.Address
gas, gasPrice *big.Int
value         *big.Int
data          []byte
}

// accessor boilerplate to implement core.Message
func (m callmsg) From() (common.Address, error)         { return m.addr, nil }
func (m callmsg) FromFrontier() (common.Address, error) { return m.addr, nil }
func (m callmsg) Nonce() uint64                     { return 0 }
func (m callmsg) CheckNonce() bool                  { return false }
func (m callmsg) To() *common.Address               { return m.to }
func (m callmsg) GasPrice() *big.Int                { return m.gasPrice }
func (m callmsg) Gas() *big.Int                     { return m.gas }
func (m callmsg) Value() *big.Int                   { return m.value }
func (m callmsg) Data() []byte                      { return m.data }

// CallArgs represents the arguments for a call.
type CallArgs struct {
From     common.Address  `json:"from"`
To       *common.Address `json:"to"`
```

```
Gas      hexutil.Big     `json:"gas"`
GasPrice hexutil.Big     `json:"gasPrice"`
Value    hexutil.Big     `json:"value"`
Data     hexutil.Bytes   `json:"data"`
}

func (s *PublicBlockChainAPI) doCall(ctx context.Context, args CallArgs, blockNr
rpc.BlockNumber, vmCfg vm.Config) ([]byte, *big.Int, error) {
defer func(start time.Time) { log.Debug("Executing EVM call finished", "runtime", time.Since(start))
}(time.Now())

state, header, err := s.b.StateAndHeaderByNumber(ctx, blockNr)
if state == nil || err != nil {
return nil, common.Big0, err
}
// Set sender address or use a default if none specified
addr := args.From
if addr == (common.Address{}) {
if wallets := s.b.AccountManager().Wallets(); len(wallets) > 0 {
if accounts := wallets[0].Accounts(); len(accounts) > 0 {
addr = accounts[0].Address
}
}
}
// Set default gas & gas price if none were set
gas, gasPrice := args.Gas.ToInt(), args.GasPrice.ToInt()
if gas.Sign() == 0 {
gas = big.NewInt(50000000)
}
if gasPrice.Sign() == 0 {
gasPrice = new(big.Int).SetUint64(defaultGasPrice)
}

// Create new call message
msg := types.NewMessage(addr, args.To, 0, args.Value.ToInt(), gas, gasPrice, args.Data, false)

// Setup context so it may be cancelled the call has completed
// or, in case of unmetered gas, setup a context with a timeout.
var cancel context.CancelFunc
if vmCfg.DisableGasMetering {
ctx, cancel = context.WithTimeout(ctx, time.Second*5)
} else {
```

```
ctx, cancel = context.WithCancel(ctx)
}
// Make sure the context is cancelled when the call has completed
// this makes sure resources are cleaned up.
defer func() { cancel() }()

// Get a new instance of the EVM.
evm, vmError, err := s.b.GetEVM(ctx, msg, state, header, vmCfg)
if err != nil {
return nil, common.Big0, err
}
// Wait for the context to be done and cancel the evm. Even if the
// EVM has finished, cancelling may be done (repeatedly)
go func() {
select {
case <-ctx.Done():
evm.Cancel()
}
}()

// Setup the gas pool (also for unmetered requests)
// and apply the message.
gp := new(core.GasPool).AddGas(math.MaxBig256)
res, gas, err := core.ApplyMessage(evm, msg, gp)
if err := vmError(); err != nil {
return nil, common.Big0, err
}
return res, gas, err
}

// Call executes the given transaction on the state for the given block number.
// It doesn't make and changes in the state/blockchain and is useful to execute and retrieve values.
func (s *PublicBlockChainAPI) Call(ctx context.Context, args CallArgs, blockNr rpc.BlockNumber)
(hexutil.Bytes, error) {
result, _, err := s.doCall(ctx, args, blockNr, vm.Config{DisableGasMetering: true})
return (hexutil.Bytes)(result), err
}

// EstimateGas returns an estimate of the amount of gas needed to execute the given transaction.
func (s *PublicBlockChainAPI) EstimateGas(ctx context.Context, args CallArgs) (*hexutil.Big,
error) {
// Binary search the gas requirement, as it may be higher than the amount used
```

```
var lo, hi uint64
if (*big.Int)(&args.Gas).Sign() != 0 {
hi = (*big.Int)(&args.Gas).Uint64()
} else {
// Retrieve the current pending block to act as the gas ceiling
block, err := s.b.BlockByNumber(ctx, rpc.PendingBlockNumber)
if err != nil {
return nil, err
}
hi = block.GasLimit().Uint64()
}
for lo+1 < hi {
// Take a guess at the gas, and check transaction validity
mid := (hi + lo) / 2
(*big.Int)(&args.Gas).SetUint64(mid)

_, gas, err := s.doCall(ctx, args, rpc.PendingBlockNumber, vm.Config{})

// If the transaction became invalid or used all the gas (failed), raise the gas limit
if err != nil || gas.Cmp((*big.Int)(&args.Gas)) == 0 {
lo = mid
continue
}
// Otherwise assume the transaction succeeded, lower the gas limit
hi = mid
}
return (*hexutil.Big)(new(big.Int).SetUint64(hi)), nil
}

// ExecutionResult groups all structured logs emitted by the EVM
// while replaying a transaction in debug mode as well as the amount of
// gas used and the return value
type ExecutionResult struct {
Gas         *big.Int       `json:"gas"`
ReturnValue string         `json:"returnValue"`
StructLogs  []StructLogRes `json:"structLogs"`
}

// StructLogRes stores a structured log emitted by the EVM while replaying a
// transaction in debug mode
type StructLogRes struct {
Pc      uint64            `json:"pc"`
```

```
Op      string           `json:"op"`
Gas     uint64           `json:"gas"`
GasCost uint64           `json:"gasCost"`
Depth   int              `json:"depth"`
Error   error            `json:"error"`
Stack   []string         `json:"stack"`
Memory  []string         `json:"memory"`
Storage map[string]string `json:"storage"`
}

// formatLogs formats EVM returned structured logs for json output
func FormatLogs(structLogs []vm.StructLog) []StructLogRes {
formattedStructLogs := make([]StructLogRes, len(structLogs))
for index, trace := range structLogs {
formattedStructLogs[index] = StructLogRes{
Pc:      trace.Pc,
Op:      trace.Op.String(),
Gas:     trace.Gas,
GasCost: trace.GasCost,
Depth:   trace.Depth,
Error:   trace.Err,
Stack:   make([]string, len(trace.Stack)),
Storage: make(map[string]string),
}

for i, stackValue := range trace.Stack {
formattedStructLogs[index].Stack[i] = fmt.Sprintf("%x", math.PaddedBigBytes(stackValue, 32))
}

for i := 0; i+32 <= len(trace.Memory); i += 32 {
formattedStructLogs[index].Memory = append(formattedStructLogs[index].Memory,
fmt.Sprintf("%x", trace.Memory[i:i+32]))
}

for i, storageValue := range trace.Storage {
formattedStructLogs[index].Storage[fmt.Sprintf("%x", i)] = fmt.Sprintf("%x", storageValue)
}
}
return formattedStructLogs
}

// rpcOutputBlock converts the given block to the RPC output which depends on fullTx. If inclTx is
```

```
true transactions are
// returned. When fullTx is true the returned block contains full transaction details, otherwise it will
only contain
// transaction hashes.
func (s *PublicBlockChainAPI) rpcOutputBlock(b *types.Block, inclTx bool, fullTx bool)
(map[string]interface{}, error) {
head := b.Header() // copies the header once
fields := map[string]interface{}{
"number":          (*hexutil.Big)(head.Number),
"hash":            b.Hash(),
"parentHash":      head.ParentHash,
"nonce":           head.Nonce,
"mixHash":         head.MixDigest,
"sha3Uncles":      head.UncleHash,
"logsBloom":       head.Bloom,
"stateRoot":       head.Root,
"miner":           head.Coinbase,
"difficulty":      (*hexutil.Big)(head.Difficulty),
"totalDifficulty":  (*hexutil.Big)(s.b.GetTd(b.Hash())),
"extraData":       hexutil.Bytes(head.Extra),
"size":            hexutil.Uint64(uint64(b.Size().Int64())),
"gasLimit":        (*hexutil.Big)(head.GasLimit),
"gasUsed":         (*hexutil.Big)(head.GasUsed),
"timestamp":       (*hexutil.Big)(head.Time),
"transactionsRoot": head.TxHash,
"receiptsRoot":     head.ReceiptHash,
}

if inclTx {
formatTx := func(tx *types.Transaction) (interface{}, error) {
return tx.Hash(), nil
}

if fullTx {
formatTx = func(tx *types.Transaction) (interface{}, error) {
return newRPCTransaction(b, tx.Hash())
}
}

txs := b.Transactions()
transactions := make([]interface{}, len(txs))
var err error
```

```
for i, tx := range b.Transactions() {
if transactions[i], err = formatTx(tx); err != nil {
return nil, err
}
}
fields["transactions"] = transactions
}

uncles := b.Uncles()
uncleHashes := make([]common.Hash, len(uncles))
for i, uncle := range uncles {
uncleHashes[i] = uncle.Hash()
}
fields["uncles"] = uncleHashes

return fields, nil
}

// RPCTransaction represents a transaction that will serialize to the RPC representation of a transaction
type RPCTransaction struct {
BlockHash        common.Hash     `json:"blockHash"`
BlockNumber      *hexutil.Big    `json:"blockNumber"`
From             common.Address  `json:"from"`
Gas              *hexutil.Big    `json:"gas"`
GasPrice         *hexutil.Big    `json:"gasPrice"`
Hash             common.Hash     `json:"hash"`
Input            hexutil.Bytes   `json:"input"`
Nonce            hexutil.Uint64  `json:"nonce"`
To               *common.Address `json:"to"`
TransactionIndex hexutil.Uint    `json:"transactionIndex"`
Value            *hexutil.Big    `json:"value"`
V                *hexutil.Big    `json:"v"`
R                *hexutil.Big    `json:"r"`
S                *hexutil.Big    `json:"s"`
}

// newRPCPendingTransaction returns a pending transaction that will serialize to the RPC representation
func newRPCPendingTransaction(tx *types.Transaction) *RPCTransaction {
var signer types.Signer = types.FrontierSigner{}
if tx.Protected() {
```

```
signer = types.NewEIP155Signer(tx.ChainId())
}
from, _ := types.Sender(signer, tx)
v, r, s := tx.RawSignatureValues()
return &RPCTransaction{
From:     from,
Gas:      (*hexutil.Big)(tx.Gas()),
GasPrice: (*hexutil.Big)(tx.GasPrice()),
Hash:     tx.Hash(),
Input:    hexutil.Bytes(tx.Data()),
Nonce:    hexutil.Uint64(tx.Nonce()),
To:       tx.To(),
Value:    (*hexutil.Big)(tx.Value()),
V:        (*hexutil.Big)(v),
R:        (*hexutil.Big)(r),
S:        (*hexutil.Big)(s),
}
}

// newRPCTransaction returns a transaction that will serialize to the RPC representation.
func newRPCTransactionFromBlockIndex(b *types.Block, txIndex uint) (*RPCTransaction, error) {
if txIndex < uint(len(b.Transactions())) {
tx := b.Transactions()[txIndex]
var signer types.Signer = types.FrontierSigner{}
if tx.Protected() {
signer = types.NewEIP155Signer(tx.ChainId())
}
from, _ := types.Sender(signer, tx)
v, r, s := tx.RawSignatureValues()
return &RPCTransaction{
BlockHash:        b.Hash(),
BlockNumber:      (*hexutil.Big)(b.Number()),
From:             from,
Gas:              (*hexutil.Big)(tx.Gas()),
GasPrice:         (*hexutil.Big)(tx.GasPrice()),
Hash:             tx.Hash(),
Input:            hexutil.Bytes(tx.Data()),
Nonce:            hexutil.Uint64(tx.Nonce()),
To:               tx.To(),
TransactionIndex: hexutil.Uint(txIndex),
Value:            (*hexutil.Big)(tx.Value()),
V:                (*hexutil.Big)(v),
```

```
R:              (*hexutil.Big)(r),
S:              (*hexutil.Big)(s),
}, nil
}

return nil, nil
}

// newRPCRawTransactionFromBlockIndex returns the bytes of a transaction given a block and a
transaction index.
func newRPCRawTransactionFromBlockIndex(b *types.Block, txIndex uint) (hexutil.Bytes, error) {
if txIndex < uint(len(b.Transactions())) {
tx := b.Transactions()[txIndex]
return rlp.EncodeToBytes(tx)
}

return nil, nil
}

// newRPCTransaction returns a transaction that will serialize to the RPC representation.
func newRPCTransaction(b *types.Block, txHash common.Hash) (*RPCTransaction, error) {
for idx, tx := range b.Transactions() {
if tx.Hash() == txHash {
return newRPCTransactionFromBlockIndex(b, uint(idx))
}
}

return nil, nil
}

// PublicTransactionPoolAPI exposes methods for the RPC interface
type PublicTransactionPoolAPI struct {
b         Backend
nonceLock *AddrLocker
}

// NewPublicTransactionPoolAPI creates a new RPC service with methods specific for the
transaction pool.
func NewPublicTransactionPoolAPI(b Backend, nonceLock *AddrLocker)
*PublicTransactionPoolAPI {
return &PublicTransactionPoolAPI{b, nonceLock}
}
```

```
func getTransaction(chainDb ethdb.Database, b Backend, txHash common.Hash)
(*types.Transaction, bool, error) {
txData, err := chainDb.Get(txHash.Bytes())
isPending := false
tx := new(types.Transaction)

if err == nil && len(txData) > 0 {
if err := rlp.DecodeBytes(txData, tx); err != nil {
return nil, isPending, err
}
} else {
// pending transaction?
tx = b.GetPoolTransaction(txHash)
isPending = true
}

return tx, isPending, nil
}

// GetBlockTransactionCountByNumber returns the number of transactions in the block with the
given block number.
func (s *PublicTransactionPoolAPI) GetBlockTransactionCountByNumber(ctx context.Context,
blockNr rpc.BlockNumber) *hexutil.Uint {
if block, _ := s.b.BlockByNumber(ctx, blockNr); block != nil {
n := hexutil.Uint(len(block.Transactions()))
return &n
}
return nil
}

// GetBlockTransactionCountByHash returns the number of transactions in the block with the given
hash.
func (s *PublicTransactionPoolAPI) GetBlockTransactionCountByHash(ctx context.Context,
blockHash common.Hash) *hexutil.Uint {
if block, _ := s.b.GetBlock(ctx, blockHash); block != nil {
n := hexutil.Uint(len(block.Transactions()))
return &n
}
return nil
}
```

```
// GetTransactionByBlockNumberAndIndex returns the transaction for the given block number and index.
func (s *PublicTransactionPoolAPI) GetTransactionByBlockNumberAndIndex(ctx context.Context, blockNr rpc.BlockNumber, index hexutil.Uint) (*RPCTransaction, error) {
if block, _ := s.b.BlockByNumber(ctx, blockNr); block != nil {
return newRPCTransactionFromBlockIndex(block, uint(index))
}
return nil, nil
}

// GetTransactionByBlockHashAndIndex returns the transaction for the given block hash and index.
func (s *PublicTransactionPoolAPI) GetTransactionByBlockHashAndIndex(ctx context.Context, blockHash common.Hash, index hexutil.Uint) (*RPCTransaction, error) {
if block, _ := s.b.GetBlock(ctx, blockHash); block != nil {
return newRPCTransactionFromBlockIndex(block, uint(index))
}
return nil, nil
}

// GetRawTransactionByBlockNumberAndIndex returns the bytes of the transaction for the given block number and index.
func (s *PublicTransactionPoolAPI) GetRawTransactionByBlockNumberAndIndex(ctx context.Context, blockNr rpc.BlockNumber, index hexutil.Uint) (hexutil.Bytes, error) {
if block, _ := s.b.BlockByNumber(ctx, blockNr); block != nil {
return newRPCRawTransactionFromBlockIndex(block, uint(index))
}
return nil, nil
}

// GetRawTransactionByBlockHashAndIndex returns the bytes of the transaction for the given block hash and index.
func (s *PublicTransactionPoolAPI) GetRawTransactionByBlockHashAndIndex(ctx context.Context, blockHash common.Hash, index hexutil.Uint) (hexutil.Bytes, error) {
if block, _ := s.b.GetBlock(ctx, blockHash); block != nil {
return newRPCRawTransactionFromBlockIndex(block, uint(index))
}
return nil, nil
}

// GetTransactionCount returns the number of transactions the given address has sent for the given block number
```

```
func (s *PublicTransactionPoolAPI) GetTransactionCount(ctx context.Context, address
common.Address, blockNr rpc.BlockNumber) (*hexutil.Uint64, error) {
state, _, err := s.b.StateAndHeaderByNumber(ctx, blockNr)
if state == nil || err != nil {
return nil, err
}
nonce := state.GetNonce(address)
return (*hexutil.Uint64)(&nonce), state.Error()
}

// getTransactionBlockData fetches the meta data for the given transaction from the chain
database. This is useful to
// retrieve block information for a hash. It returns the block hash, block index and transaction index.
func getTransactionBlockData(chainDb ethdb.Database, txHash common.Hash) (common.Hash,
uint64, uint64, error) {
var txBlock struct {
BlockHash  common.Hash
BlockIndex uint64
Index      uint64
}

blockData, err := chainDb.Get(append(txHash.Bytes(), 0x0001))
if err != nil {
return common.Hash{}, uint64(0), uint64(0), err
}

reader := bytes.NewReader(blockData)
if err = rlp.Decode(reader, &txBlock); err != nil {
return common.Hash{}, uint64(0), uint64(0), err
}

return txBlock.BlockHash, txBlock.BlockIndex, txBlock.Index, nil
}

// GetTransactionByHash returns the transaction for the given hash
func (s *PublicTransactionPoolAPI) GetTransactionByHash(ctx context.Context, hash
common.Hash) (*RPCTransaction, error) {
var tx *types.Transaction
var isPending bool
var err error

if tx, isPending, err = getTransaction(s.b.ChainDb(), s.b, hash); err != nil {
```

```
log.Debug("Failed to retrieve transaction", "hash", hash, "err", err)
return nil, nil
} else if tx == nil {
return nil, nil
}
if isPending {
return newRPCPendingTransaction(tx), nil
}

blockHash, _, _, err := getTransactionBlockData(s.b.ChainDb(), hash)
if err != nil {
log.Debug("Failed to retrieve transaction block", "hash", hash, "err", err)
return nil, nil
}

if block, _ := s.b.GetBlock(ctx, blockHash); block != nil {
return newRPCTransaction(block, hash)
}
return nil, nil
}

// GetRawTransactionByHash returns the bytes of the transaction for the given hash.
func (s *PublicTransactionPoolAPI) GetRawTransactionByHash(ctx context.Context, hash
common.Hash) (hexutil.Bytes, error) {
var tx *types.Transaction
var err error

if tx, _, err = getTransaction(s.b.ChainDb(), s.b, hash); err != nil {
log.Debug("Failed to retrieve transaction", "hash", hash, "err", err)
return nil, nil
} else if tx == nil {
return nil, nil
}

return rlp.EncodeToBytes(tx)
}

// GetTransactionReceipt returns the transaction receipt for the given transaction hash.
func (s *PublicTransactionPoolAPI) GetTransactionReceipt(hash common.Hash)
(map[string]interface{}, error) {
receipt := core.GetReceipt(s.b.ChainDb(), hash)
if receipt == nil {
```

```
log.Debug("Receipt not found for transaction", "hash", hash)
return nil, nil
}

tx, _, err := getTransaction(s.b.ChainDb(), s.b, hash)
if err != nil {
log.Debug("Failed to retrieve transaction", "hash", hash, "err", err)
return nil, nil
}

txBlock, blockIndex, index, err := getTransactionBlockData(s.b.ChainDb(), hash)
if err != nil {
log.Debug("Failed to retrieve transaction block", "hash", hash, "err", err)
return nil, nil
}

var signer types.Signer = types.FrontierSigner{}
if tx.Protected() {
signer = types.NewEIP155Signer(tx.ChainId())
}
from, _ := types.Sender(signer, tx)

fields := map[string]interface{}{
"root":              hexutil.Bytes(receipt.PostState),
"blockHash":         txBlock,
"blockNumber":       hexutil.Uint64(blockIndex),
"transactionHash":   hash,
"transactionIndex":  hexutil.Uint64(index),
"from":              from,
"to":                tx.To(),
"gasUsed":           (*hexutil.Big)(receipt.GasUsed),
"cumulativeGasUsed": (*hexutil.Big)(receipt.CumulativeGasUsed),
"contractAddress":   nil,
"logs":              receipt.Logs,
"logsBloom":         receipt.Bloom,
}
if receipt.Logs == nil {
fields["logs"] = [][]*types.Log{}
}
// If the ContractAddress is 20 0x0 bytes, assume it is not a contract creation
if receipt.ContractAddress != (common.Address{}) {
fields["contractAddress"] = receipt.ContractAddress
```

```
}
return fields, nil
}

// sign is a helper function that signs a transaction with the private key of the given address.
func (s *PublicTransactionPoolAPI) sign(addr common.Address, tx *types.Transaction)
(*types.Transaction, error) {
// Look up the wallet containing the requested signer
account := accounts.Account{Address: addr}

wallet, err := s.b.AccountManager().Find(account)
if err != nil {
return nil, err
}
// Request the wallet to sign the transaction
var chainID *big.Int
if config := s.b.ChainConfig(); config.IsEIP155(s.b.CurrentBlock().Number()) {
chainID = config.ChainId
}
return wallet.SignTx(account, tx, chainID)
}

// SendTxArgs represents the arguments to sumbit a new transaction into the transaction pool.
type SendTxArgs struct {
From     common.Address  `json:"from"`
To       *common.Address `json:"to"`
Gas      *hexutil.Big    `json:"gas"`
GasPrice *hexutil.Big    `json:"gasPrice"`
Value    *hexutil.Big    `json:"value"`
Data     hexutil.Bytes   `json:"data"`
Nonce    *hexutil.Uint64 `json:"nonce"`
}

// prepareSendTxArgs is a helper function that fills in default values for unspecified tx fields.
func (args *SendTxArgs) setDefaults(ctx context.Context, b Backend) error {
if args.Gas == nil {
args.Gas = (*hexutil.Big)(big.NewInt(defaultGas))
}
if args.GasPrice == nil {
price, err := b.SuggestPrice(ctx)
if err != nil {
return err
```

```
}
args.GasPrice = (*hexutil.Big)(price)
}
if args.Value == nil {
args.Value = new(hexutil.Big)
}
if args.Nonce == nil {
nonce, err := b.GetPoolNonce(ctx, args.From)
if err != nil {
return err
}
args.Nonce = (*hexutil.Uint64)(&nonce)
}
return nil
}

func (args *SendTxArgs) toTransaction() *types.Transaction {
if args.To == nil {
return types.NewContractCreation(uint64(*args.Nonce), (*big.Int)(args.Value), (*big.Int)(args.Gas),
(*big.Int)(args.GasPrice), args.Data)
}
return types.NewTransaction(uint64(*args.Nonce), *args.To, (*big.Int)(args.Value),
(*big.Int)(args.Gas), (*big.Int)(args.GasPrice), args.Data)
}

// submitTransaction is a helper function that submits tx to txPool and logs a message.
func submitTransaction(ctx context.Context, b Backend, tx *types.Transaction) (common.Hash,
error) {
if err := b.SendTx(ctx, tx); err != nil {
return common.Hash{}, err
}
if tx.To() == nil {
signer := types.MakeSigner(b.ChainConfig(), b.CurrentBlock().Number())
from, _ := types.Sender(signer, tx)
addr := crypto.CreateAddress(from, tx.Nonce())
log.Info("Submitted contract creation", "fullhash", tx.Hash().Hex(), "contract", addr.Hex())
} else {
log.Info("Submitted transaction", "fullhash", tx.Hash().Hex(), "recipient", tx.To())
}
return tx.Hash(), nil
}
```

```
// SendTransaction creates a transaction for the given argument, sign it and submit it to the
// transaction pool.
func (s *PublicTransactionPoolAPI) SendTransaction(ctx context.Context, args SendTxArgs)
(common.Hash, error) {

// Look up the wallet containing the requested signer
account := accounts.Account{Address: args.From}

wallet, err := s.b.AccountManager().Find(account)
if err != nil {
return common.Hash{}, err
}

if args.Nonce == nil {
// Hold the addresse's mutex around signing to prevent concurrent assignment of
// the same nonce to multiple accounts.
s.nonceLock.LockAddr(args.From)
defer s.nonceLock.UnlockAddr(args.From)
}

// Set some sanity defaults and terminate on failure
if err := args.setDefaults(ctx, s.b); err != nil {
return common.Hash{}, err
}
// Assemble the transaction and sign with the wallet
tx := args.toTransaction()

var chainID *big.Int
if config := s.b.ChainConfig(); config.IsEIP155(s.b.CurrentBlock().Number()) {
chainID = config.ChainId
}
signed, err := wallet.SignTx(account, tx, chainID)
if err != nil {
return common.Hash{}, err
}
return submitTransaction(ctx, s.b, signed)
}

// SendRawTransaction will add the signed transaction to the transaction pool.
// The sender is responsible for signing the transaction and using the correct nonce.
func (s *PublicTransactionPoolAPI) SendRawTransaction(ctx context.Context, encodedTx
hexutil.Bytes) (string, error) {
```

```
tx := new(types.Transaction)
if err := rlp.DecodeBytes(encodedTx, tx); err != nil {
return "", err
}

if err := s.b.SendTx(ctx, tx); err != nil {
return "", err
}

signer := types.MakeSigner(s.b.ChainConfig(), s.b.CurrentBlock().Number())
if tx.To() == nil {
from, err := types.Sender(signer, tx)
if err != nil {
return "", err
}
addr := crypto.CreateAddress(from, tx.Nonce())
log.Info("Submitted contract creation", "fullhash", tx.Hash().Hex(), "contract", addr.Hex())
} else {
log.Info("Submitted transaction", "fullhash", tx.Hash().Hex(), "recipient", tx.To())
}

return tx.Hash().Hex(), nil
}

// Sign calculates an ECDSA signature for:
// keccack256("\x19Ethereum Signed Message:\n" + len(message) + message).
//
// Note, the produced signature conforms to the secp256k1 curve R, S and V values,
// where the V value will be 27 or 28 for legacy reasons.
//
// The account associated with addr must be unlocked.
//
// https://github.com/ethereum/wiki/wiki/JSON-RPC#eth_sign
func (s *PublicTransactionPoolAPI) Sign(addr common.Address, data hexutil.Bytes) (hexutil.Bytes,
error) {
// Look up the wallet containing the requested signer
account := accounts.Account{Address: addr}

wallet, err := s.b.AccountManager().Find(account)
if err != nil {
return nil, err
}
```

```
// Sign the requested hash with the wallet
signature, err := wallet.SignHash(account, signHash(data))
if err == nil {
signature[64] += 27 // Transform V from 0/1 to 27/28 according to the yellow paper
}
return signature, err
}

// SignTransactionResult represents a RLP encoded signed transaction.
type SignTransactionResult struct {
Raw hexutil.Bytes      `json:"raw"`
Tx  *types.Transaction `json:"tx"`
}

// SignTransaction will sign the given transaction with the from account.
// The node needs to have the private key of the account corresponding with
// the given from address and it needs to be unlocked.
func (s *PublicTransactionPoolAPI) SignTransaction(ctx context.Context, args SendTxArgs)
(*SignTransactionResult, error) {
if args.Nonce == nil {
// Hold the addresse's mutex around signing to prevent concurrent assignment of
// the same nonce to multiple accounts.
s.nonceLock.LockAddr(args.From)
defer s.nonceLock.UnlockAddr(args.From)
}
if err := args.setDefaults(ctx, s.b); err != nil {
return nil, err
}
tx, err := s.sign(args.From, args.toTransaction())
if err != nil {
return nil, err
}
data, err := rlp.EncodeToBytes(tx)
if err != nil {
return nil, err
}
return &SignTransactionResult{data, tx}, nil
}

// PendingTransactions returns the transactions that are in the transaction pool and have a from
address that is one of
// the accounts this node manages.
```

```
func (s *PublicTransactionPoolAPI) PendingTransactions() ([]*RPCTransaction, error) {
pending, err := s.b.GetPoolTransactions()
if err != nil {
return nil, err
}

transactions := make([]*RPCTransaction, 0, len(pending))
for _, tx := range pending {
var signer types.Signer = types.HomesteadSigner{}
if tx.Protected() {
signer = types.NewEIP155Signer(tx.ChainId())
}
from, _ := types.Sender(signer, tx)
if _, err := s.b.AccountManager().Find(accounts.Account{Address: from}); err == nil {
transactions = append(transactions, newRPCPendingTransaction(tx))
}
}
return transactions, nil
}

// Resend accepts an existing transaction and a new gas price and limit. It will remove
// the given transaction from the pool and reinsert it with the new gas price and limit.
func (s *PublicTransactionPoolAPI) Resend(ctx context.Context, sendArgs SendTxArgs, gasPrice,
gasLimit *hexutil.Big) (common.Hash, error) {
if sendArgs.Nonce == nil {
return common.Hash{}, fmt.Errorf("missing transaction nonce in transaction spec")
}
if err := sendArgs.setDefaults(ctx, s.b); err != nil {
return common.Hash{}, err
}
matchTx := sendArgs.toTransaction()
pending, err := s.b.GetPoolTransactions()
if err != nil {
return common.Hash{}, err
}

for _, p := range pending {
var signer types.Signer = types.HomesteadSigner{}
if p.Protected() {
signer = types.NewEIP155Signer(p.ChainId())
}
wantSigHash := signer.Hash(matchTx)
```

```
if pFrom, err := types.Sender(signer, p); err == nil && pFrom == sendArgs.From && signer.Hash(p)
== wantSigHash {
// Match. Re-sign and send the transaction.
if gasPrice != nil {
sendArgs.GasPrice = gasPrice
}
if gasLimit != nil {
sendArgs.Gas = gasLimit
}
signedTx, err := s.sign(sendArgs.From, sendArgs.toTransaction())
if err != nil {
return common.Hash{}, err
}
s.b.RemoveTx(p.Hash())
if err = s.b.SendTx(ctx, signedTx); err != nil {
return common.Hash{}, err
}
return signedTx.Hash(), nil
}
}

return common.Hash{}, fmt.Errorf("Transaction %#x not found", matchTx.Hash())
}

// PublicDebugAPI is the collection of Etheruem APIs exposed over the public
// debugging endpoint.
type PublicDebugAPI struct {
b Backend
}

// NewPublicDebugAPI creates a new API definition for the public debug methods
// of the Ethereum service.
func NewPublicDebugAPI(b Backend) *PublicDebugAPI {
return &PublicDebugAPI{b: b}
}

// GetBlockRlp retrieves the RLP encoded for of a single block.
func (api *PublicDebugAPI) GetBlockRlp(ctx context.Context, number uint64) (string, error) {
block, _ := api.b.BlockByNumber(ctx, rpc.BlockNumber(number))
if block == nil {
return "", fmt.Errorf("block #%d not found", number)
```

```
}
encoded, err := rlp.EncodeToBytes(block)
if err != nil {
return "", err
}
return fmt.Sprintf("%x", encoded), nil
}

// PrintBlock retrieves a block and returns its pretty printed form.
func (api *PublicDebugAPI) PrintBlock(ctx context.Context, number uint64) (string, error) {
block, _ := api.b.BlockByNumber(ctx, rpc.BlockNumber(number))
if block == nil {
return "", fmt.Errorf("block #%d not found", number)
}
return fmt.Sprintf("%s", block), nil
}

// SeedHash retrieves the seed hash of a block.
func (api *PublicDebugAPI) SeedHash(ctx context.Context, number uint64) (string, error) {
block, _ := api.b.BlockByNumber(ctx, rpc.BlockNumber(number))
if block == nil {
return "", fmt.Errorf("block #%d not found", number)
}
return fmt.Sprintf("0x%x", ethash.SeedHash(number)), nil
}

// PrivateDebugAPI is the collection of Etheruem APIs exposed over the private
// debugging endpoint.
type PrivateDebugAPI struct {
b Backend
}

// NewPrivateDebugAPI creates a new API definition for the private debug methods
// of the Ethereum service.
func NewPrivateDebugAPI(b Backend) *PrivateDebugAPI {
return &PrivateDebugAPI{b: b}
}

// ChaindbProperty returns leveldb properties of the chain database.
func (api *PrivateDebugAPI) ChaindbProperty(property string) (string, error) {
ldb, ok := api.b.ChainDb().(interface {
LDB() *leveldb.DB
```

```
})
if !ok {
return "", fmt.Errorf("chaindbProperty does not work for memory databases")
}
if property == "" {
property = "leveldb.stats"
} else if !strings.HasPrefix(property, "leveldb.") {
property = "leveldb." + property
}
return ldb.LDB().GetProperty(property)
}

func (api *PrivateDebugAPI) ChaindbCompact() error {
ldb, ok := api.b.ChainDb().(interface {
LDB() *leveldb.DB
})
if !ok {
return fmt.Errorf("chaindbCompact does not work for memory databases")
}
for b := byte(0); b < 255; b++ {
log.Info("Compacting chain database", "range", fmt.Sprintf("0x%0.2X-0x%0.2X", b, b+1))
err := ldb.LDB().CompactRange(util.Range{Start: []byte{b}, Limit: []byte{b + 1}})
if err != nil {
log.Error("Database compaction failed", "err", err)
return err
}
}
return nil
}

// SetHead rewinds the head of the blockchain to a previous block.
func (api *PrivateDebugAPI) SetHead(number hexutil.Uint64) {
api.b.SetHead(uint64(number))
}

// PublicNetAPI offers network related RPC methods
type PublicNetAPI struct {
net            *p2p.Server
networkVersion uint64
}

// NewPublicNetAPI creates a new net API instance.
```

```
func NewPublicNetAPI(net *p2p.Server, networkVersion uint64) *PublicNetAPI {
return &PublicNetAPI{net, networkVersion}
}

// Listening returns an indication if the node is listening for network connections.
func (s *PublicNetAPI) Listening() bool {
return true // always listening
}

// PeerCount returns the number of connected peers
func (s *PublicNetAPI) PeerCount() hexutil.Uint {
return hexutil.Uint(s.net.PeerCount())
}

// Version returns the current ethereum protocol version.
func (s *PublicNetAPI) Version() string {
return fmt.Sprintf("%d", s.networkVersion)
}

63:F:\git\coin\ethereum\go-ethereum\internal\ethapi\backend.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package ethapi implements the general Ethereum API functions.
package ethapi

import (
"context"
"math/big"

"github.com/ethereum/go-ethereum/accounts"
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/state"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rpc"
)
```

```
// Backend interface provides the common API services (that are provided by
// both full and light clients) with access to necessary functions.
type Backend interface {
// general Ethereum API
Downloader() *downloader.Downloader
ProtocolVersion() int
SuggestPrice(ctx context.Context) (*big.Int, error)
ChainDb() ethdb.Database
EventMux() *event.TypeMux
AccountManager() *accounts.Manager
// BlockChain API
SetHead(number uint64)
HeaderByNumber(ctx context.Context, blockNr rpc.BlockNumber) (*types.Header, error)
BlockByNumber(ctx context.Context, blockNr rpc.BlockNumber) (*types.Block, error)
StateAndHeaderByNumber(ctx context.Context, blockNr rpc.BlockNumber) (*state.StateDB,
*types.Header, error)
GetBlock(ctx context.Context, blockHash common.Hash) (*types.Block, error)
GetReceipts(ctx context.Context, blockHash common.Hash) (types.Receipts, error)
GetTd(blockHash common.Hash) *big.Int
GetEVM(ctx context.Context, msg core.Message, state *state.StateDB, header *types.Header,
vmCfg vm.Config) (*vm.EVM, func() error, error)

// TxPool API
SendTx(ctx context.Context, signedTx *types.Transaction) error
RemoveTx(txHash common.Hash)
GetPoolTransactions() (types.Transactions, error)
GetPoolTransaction(txHash common.Hash) *types.Transaction
GetPoolNonce(ctx context.Context, addr common.Address) (uint64, error)
Stats() (pending int, queued int)
TxPoolContent() (map[common.Address]types.Transactions,
map[common.Address]types.Transactions)

ChainConfig() *params.ChainConfig
CurrentBlock() *types.Block
}

func GetAPIs(apiBackend Backend) []rpc.API {
nonceLock := new(AddrLocker)
return []rpc.API{
{
Namespace: "eth",
Version:   "1.0",
```

```
Service:   NewPublicEthereumAPI(apiBackend),
Public:    true,
}, {
Namespace: "eth",
Version:   "1.0",
Service:   NewPublicBlockChainAPI(apiBackend),
Public:    true,
}, {
Namespace: "eth",
Version:   "1.0",
Service:   NewPublicTransactionPoolAPI(apiBackend, nonceLock),
Public:    true,
}, {
Namespace: "txpool",
Version:   "1.0",
Service:   NewPublicTxPoolAPI(apiBackend),
Public:    true,
}, {
Namespace: "debug",
Version:   "1.0",
Service:   NewPublicDebugAPI(apiBackend),
Public:    true,
}, {
Namespace: "debug",
Version:   "1.0",
Service:   NewPrivateDebugAPI(apiBackend),
}, {
Namespace: "eth",
Version:   "1.0",
Service:   NewPublicAccountAPI(apiBackend.AccountManager()),
Public:    true,
}, {
Namespace: "personal",
Version:   "1.0",
Service:   NewPrivateAccountAPI(apiBackend, nonceLock),
Public:    false,
},
}
}
```

64:F:\git\coin\ethereum\go-ethereum\internal\ethapi\tracer.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package ethapi

import (
"encoding/json"
"errors"
"fmt"
"math/big"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/hexutil"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/robertkrimen/otto"
)

// fakeBig is used to provide an interface to Javascript for 'big.NewInt'
type fakeBig struct{}

// NewInt creates a new big.Int with the specified int64 value.
func (fb *fakeBig) NewInt(x int64) *big.Int {
return big.NewInt(x)
}

// OpCodeWrapper provides a JavaScript-friendly wrapper around OpCode, to convince Otto to treat it
// as an object, instead of a number.
type opCodeWrapper struct {
op vm.OpCode
}

// toNumber returns the ID of this opcode as an integer
func (ocw *opCodeWrapper) toNumber() int {
return int(ocw.op)
}

// toString returns the string representation of the opcode
func (ocw *opCodeWrapper) toString() string {
return ocw.op.String()
}

// isPush returns true if the op is a Push
```

```
func (ocw *opCodeWrapper) isPush() bool {
return ocw.op.IsPush()
}

// MarshalJSON serializes the opcode as JSON
func (ocw *opCodeWrapper) MarshalJSON() ([]byte, error) {
return json.Marshal(ocw.op.String())
}

// toValue returns an otto.Value for the opCodeWrapper
func (ocw *opCodeWrapper) toValue(vm *otto.Otto) otto.Value {
value, _ := vm.ToValue(ocw)
obj := value.Object()
obj.Set("toNumber", ocw.toNumber)
obj.Set("toString", ocw.toString)
obj.Set("isPush", ocw.isPush)
return value
}

// memoryWrapper provides a JS 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 {
return mw.memory.Get(begin, end-begin)
}

// getUint returns the 32 bytes at the specified address interpreted
// as an unsigned integer
func (mw *memoryWrapper) getUint(addr int64) *big.Int {
ret := big.NewInt(0)
ret.SetBytes(mw.memory.GetPtr(addr, 32))
return ret
}

// toValue returns an otto.Value for the memoryWrapper
func (mw *memoryWrapper) toValue(vm *otto.Otto) otto.Value {
value, _ := vm.ToValue(mw)
obj := value.Object()
obj.Set("slice", mw.slice)
```

```
obj.Set("getUint", mw.getUint)
return value
}

// stackWrapper provides a JS 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 {
return sw.stack.Data()[len(sw.stack.Data())-idx-1]
}

// length returns the length of the stack
func (sw *stackWrapper) length() int {
return len(sw.stack.Data())
}

// toValue returns an otto.Value for the stackWrapper
func (sw *stackWrapper) toValue(vm *otto.Otto) otto.Value {
value, _ := vm.ToValue(sw)
obj := value.Object()
obj.Set("peek", sw.peek)
obj.Set("length", sw.length)
return value
}

// dbWrapper provides a JS wrapper around vm.Database
type dbWrapper struct {
db vm.StateDB
}

// getBalance retrieves an account's balance
func (dw *dbWrapper) getBalance(addr common.Address) *big.Int {
return dw.db.GetBalance(addr)
}

// getNonce retrieves an account's nonce
func (dw *dbWrapper) getNonce(addr common.Address) uint64 {
return dw.db.GetNonce(addr)
}
```

```
// getCode retrieves an account's code
func (dw *dbWrapper) getCode(addr common.Address) []byte {
return dw.db.GetCode(addr)
}

// getState retrieves an account's state data for the given hash
func (dw *dbWrapper) getState(addr common.Address, hash common.Hash) common.Hash {
return dw.db.GetState(addr, hash)
}

// exists returns true iff the account exists
func (dw *dbWrapper) exists(addr common.Address) bool {
return dw.db.Exist(addr)
}

// toValue returns an otto.Value for the dbWrapper
func (dw *dbWrapper) toValue(vm *otto.Otto) otto.Value {
value, _ := vm.ToValue(dw)
obj := value.Object()
obj.Set("getBalance", dw.getBalance)
obj.Set("getNonce", dw.getNonce)
obj.Set("getCode", dw.getCode)
obj.Set("getState", dw.getState)
obj.Set("exists", dw.exists)
return value
}

// contractWrapper provides a JS wrapper around vm.Contract
type contractWrapper struct {
contract *vm.Contract
}

func (c *contractWrapper) caller() common.Address {
return c.contract.Caller()
}

func (c *contractWrapper) address() common.Address {
return c.contract.Address()
}

func (c *contractWrapper) value() *big.Int {
```

```
return c.contract.Value()
}

func (c *contractWrapper) calldata() []byte {
return c.contract.Input
}

func (c *contractWrapper) toValue(vm *otto.Otto) otto.Value {
value, _ := vm.ToValue(c)
obj := value.Object()
obj.Set("caller", c.caller)
obj.Set("address", c.address)
obj.Set("value", c.value)
obj.Set("calldata", c.calldata)
return value
}

// JavascriptTracer provides an implementation of Tracer that evaluates a
// Javascript function for each VM execution step.
type JavascriptTracer struct {
vm            *otto.Otto             // Javascript VM instance
traceobj      *otto.Object           // User-supplied object to call
log           map[string]interface{} // (Reusable) map for the `log` arg to `step`
logvalue      otto.Value             // JS view of `log`
memory        *memoryWrapper         // Wrapper around the VM memory
memvalue      otto.Value             // JS view of `memory`
stack         *stackWrapper          // Wrapper around the VM stack
stackvalue    otto.Value             // JS view of `stack`
db            *dbWrapper             // Wrapper around the VM environment
dbvalue       otto.Value             // JS view of `db`
contract      *contractWrapper       // Wrapper around the contract object
contractvalue otto.Value             // JS view of `contract`
err           error                  // Error, if one has occurred
}

// NewJavascriptTracer instantiates a new JavascriptTracer instance.
// code specifies a Javascript snippet, which must evaluate to an expression
// returning an object with 'step' and 'result' functions.
func NewJavascriptTracer(code string) (*JavascriptTracer, error) {
vm := otto.New()
vm.Interrupt = make(chan func(), 1)
```

```
// Set up builtins for this environment
vm.Set("big", &fakeBig{})
vm.Set("toHex", hexutil.Encode)

jstracer, err := vm.Object("(" + code + ")")
if err != nil {
return nil, err
}

// Check the required functions exist
step, err := jstracer.Get("step")
if err != nil {
return nil, err
}
if !step.IsFunction() {
return nil, fmt.Errorf("Trace object must expose a function step()")
}

result, err := jstracer.Get("result")
if err != nil {
return nil, err
}
if !result.IsFunction() {
return nil, fmt.Errorf("Trace object must expose a function result()")
}

// Create the persistent log object
log := make(map[string]interface{})
logvalue, _ := vm.ToValue(log)

// Create persistent wrappers for memory and stack
mem := &memoryWrapper{}
stack := &stackWrapper{}
db := &dbWrapper{}
contract := &contractWrapper{}

return &JavascriptTracer{
vm:            vm,
traceobj:      jstracer,
log:           log,
logvalue:      logvalue,
memory:        mem,
```

```
memvalue:      mem.toValue(vm),
stack:         stack,
stackvalue:    stack.toValue(vm),
db:            db,
dbvalue:       db.toValue(vm),
contract:      contract,
contractvalue: contract.toValue(vm),
err:           nil,
}, nil
}

// Stop terminates execution of any JavaScript
func (jst *JavascriptTracer) Stop(err error) {
jst.vm.Interrupt <- func() {
panic(err)
}
}

// callSafely executes a method on a JS object, catching any panics and
// returning them as error objects.
func (jst *JavascriptTracer) callSafely(method string, argumentList ...interface{}) (ret interface{}, err
error) {
defer func() {
if caught := recover(); caught != nil {
switch caught := caught.(type) {
case error:
err = caught
case string:
err = errors.New(caught)
case fmt.Stringer:
err = errors.New(caught.String())
default:
panic(caught)
}
}
}()

value, err := jst.traceobj.Call(method, argumentList...)
ret, _ = value.Export()
return ret, err
}
```

```
func wrapError(context string, err error) error {
var message string
switch err := err.(type) {
case *otto.Error:
message = err.String()
default:
message = err.Error()
}
return fmt.Errorf("%v    in server-side tracer function '%v'", message, context)
}

// CaptureState implements the Tracer interface to trace a single step of VM execution
func (jst *JavascriptTracer) CaptureState(env *vm.EVM, pc uint64, op vm.OpCode, gas, cost
uint64, memory *vm.Memory, stack *vm.Stack, contract *vm.Contract, depth int, err error) error {
if jst.err == nil {
jst.memory.memory = memory
jst.stack.stack = stack
jst.db.db = env.StateDB
jst.contract.contract = contract

ocw := &opCodeWrapper{op}

jst.log["pc"] = pc
jst.log["op"] = ocw.toValue(jst.vm)
jst.log["gas"] = gas
jst.log["gasPrice"] = cost
jst.log["memory"] = jst.memvalue
jst.log["stack"] = jst.stackvalue
jst.log["contract"] = jst.contractvalue
jst.log["depth"] = depth
jst.log["account"] = contract.Address()
jst.log["err"] = err

_, err := jst.callSafely("step", jst.logvalue, jst.dbvalue)
if err != nil {
jst.err = wrapError("step", err)
}
}
return nil
}

// CaptureEnd is called after the call finishes
```

```
func (jst *JavascriptTracer) CaptureEnd(output []byte, gasUsed uint64, t time.Duration) error {
//TODO! @Arachnid please figure out of there's anything we can use this method for
return nil
}

// GetResult calls the Javascript 'result' function and returns its value, or any accumulated error
func (jst *JavascriptTracer) GetResult() (result interface{}, err error) {
if jst.err != nil {
return nil, jst.err
}

result, err = jst.callSafely("result")
if err != nil {
err = wrapError("result", err)
}
return
}

65:F:\git\coin\ethereum\go-ethereum\internal\ethapi\tracer_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package ethapi

import (
"errors"
"math/big"
"reflect"
"testing"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/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) ReturnGas(*big.Int)                           {}
func (account) SetCode(common.Hash, []byte)                   {}
func (account) ForEachStorage(cb func(key, value common.Hash) bool) {}

func runTrace(tracer *JavascriptTracer) (interface{}, error) {
env := vm.NewEVM(vm.Context{}, nil, params.TestChainConfig, vm.Config{Debug: true, Tracer:
tracer})

contract := vm.NewContract(account{}, account{}, big.NewInt(0), 10000)
contract.Code = []byte{byte(vm.PUSH1), 0x1, byte(vm.PUSH1), 0x1, 0x0}

_, err := env.Interpreter().Run(0, contract, []byte{})
if err != nil {
return nil, err
}

return tracer.GetResult()
}

func TestTracing(t *testing.T) {
tracer, err := NewJavascriptTracer("{count: 0, step: function() { this.count += 1; }, result: function() {
return this.count; }}")
if err != nil {
t.Fatal(err)
}

ret, err := runTrace(tracer)
if err != nil {
t.Fatal(err)
}

value, ok := ret.(float64)
if !ok {
t.Errorf("Expected return value to be float64, was %T", ret)
}
if value != 3 {
t.Errorf("Expected return value to be 3, got %v", value)
}
}
```

```
func TestStack(t *testing.T) {
tracer, err := NewJavascriptTracer("{depths: [], step: function(log) {
this.depths.push(log.stack.length()); }, result: function() { return this.depths; }}")
if err != nil {
t.Fatal(err)
}

ret, err := runTrace(tracer)
if err != nil {
t.Fatal(err)
}

expected := []int{0, 1, 2}
if !reflect.DeepEqual(ret, expected) {
t.Errorf("Expected return value to be %#v, got %#v", expected, ret)
}
}

func TestOpcodes(t *testing.T) {
tracer, err := NewJavascriptTracer("{opcodes: [], step: function(log) {
this.opcodes.push(log.op.toString()); }, result: function() { return this.opcodes; }}")
if err != nil {
t.Fatal(err)
}

ret, err := runTrace(tracer)
if err != nil {
t.Fatal(err)
}

expected := []string{"PUSH1", "PUSH1", "STOP"}
if !reflect.DeepEqual(ret, expected) {
t.Errorf("Expected return value to be %#v, got %#v", expected, ret)
}
}

func TestHalt(t *testing.T) {
timeout := errors.New("stahp")
tracer, err := NewJavascriptTracer("{step: function() { while(1); }, result: function() { return null; }}")
if err != nil {
t.Fatal(err)
```

```
}

go func() {
time.Sleep(1 * time.Second)
tracer.Stop(timeout)
}()

if _, err = runTrace(tracer); err.Error() != "stahp    in server-side tracer function 'step'" {
t.Errorf("Expected timeout error, got %v", err)
}
}

func TestHaltBetweenSteps(t *testing.T) {
tracer, err := NewJavascriptTracer("{step: function() {}, result: function() { return null; }}")
if err != nil {
t.Fatal(err)
}

env := vm.NewEVM(vm.Context{}, nil, params.TestChainConfig, vm.Config{Debug: true, Tracer:
tracer})
contract := vm.NewContract(&account{}, &account{}, big.NewInt(0), 0)

tracer.CaptureState(env, 0, 0, 0, 0, nil, nil, contract, 0, nil)
timeout := errors.New("stahp")
tracer.Stop(timeout)
tracer.CaptureState(env, 0, 0, 0, 0, nil, nil, contract, 0, nil)

if _, err := tracer.GetResult(); err.Error() != "stahp    in server-side tracer function 'step'" {
t.Errorf("Expected timeout error, got %v", err)
}
}

66:F:\git\coin\ethereum\go-ethereum\internal\guide\guide.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package guide is a small test suite to ensure snippets in the dev guide work.
package guide

67:F:\git\coin\ethereum\go-ethereum\internal\guide\guide_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// This file contains the code snippets from the developer's guide embedded into
```

```
// Go tests. This ensures that any code published in out guides will not break
// accidentally via some code update. If some API changes nonetheless that needs
// modifying this file, please port any modification over into the developer's
// guide wiki pages too!

package guide

import (
"io/ioutil"
"math/big"
"os"
"path/filepath"
"testing"
"time"

"github.com/ethereum/go-ethereum/accounts/keystore"
"github.com/ethereum/go-ethereum/core/types"
)

// Tests that the account management snippets work correctly.
func TestAccountManagement(t *testing.T) {
// Create a temporary folder to work with
workdir, err := ioutil.TempDir("", "")
if err != nil {
t.Fatalf("Failed to create temporary work dir: %v", err)
}
defer os.RemoveAll(workdir)

// Create an encrypted keystore with standard crypto parameters
ks := keystore.NewKeyStore(filepath.Join(workdir, "keystore"), keystore.StandardScryptN,
keystore.StandardScryptP)

// Create a new account with the specified encryption passphrase
newAcc, err := ks.NewAccount("Creation password")
if err != nil {
t.Fatalf("Failed to create new account: %v", err)
}
// Export the newly created account with a different passphrase. The returned
// data from this method invocation is a JSON encoded, encrypted key-file
jsonAcc, err := ks.Export(newAcc, "Creation password", "Export password")
if err != nil {
t.Fatalf("Failed to export account: %v", err)
```

```
}
// Update the passphrase on the account created above inside the local keystore
if err := ks.Update(newAcc, "Creation password", "Update password"); err != nil {
t.Fatalf("Failed to update account: %v", err)
}
// Delete the account updated above from the local keystore
if err := ks.Delete(newAcc, "Update password"); err != nil {
t.Fatalf("Failed to delete account: %v", err)
}
// Import back the account we've exported (and then deleted) above with yet
// again a fresh passphrase
if _, err := ks.Import(jsonAcc, "Export password", "Import password"); err != nil {
t.Fatalf("Failed to import account: %v", err)
}
// Create a new account to sign transactions with
signer, err := ks.NewAccount("Signer password")
if err != nil {
t.Fatalf("Failed to create signer account: %v", err)
}
tx, chain := new(types.Transaction), big.NewInt(1)

// Sign a transaction with a single authorization
if _, err := ks.SignTxWithPassphrase(signer, "Signer password", tx, chain); err != nil {
t.Fatalf("Failed to sign with passphrase: %v", err)
}
// Sign a transaction with multiple manually cancelled authorizations
if err := ks.Unlock(signer, "Signer password"); err != nil {
t.Fatalf("Failed to unlock account: %v", err)
}
if _, err := ks.SignTx(signer, tx, chain); err != nil {
t.Fatalf("Failed to sign with unlocked account: %v", err)
}
if err := ks.Lock(signer.Address); err != nil {
t.Fatalf("Failed to lock account: %v", err)
}
// Sign a transaction with multiple automatically cancelled authorizations
if err := ks.TimedUnlock(signer, "Signer password", time.Second); err != nil {
t.Fatalf("Failed to time unlock account: %v", err)
}
if _, err := ks.SignTx(signer, tx, chain); err != nil {
t.Fatalf("Failed to sign with time unlocked account: %v", err)
}
```

```
}

68:F:\git\coin\ethereum\go-ethereum\internal\jsre\completion.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package jsre

import (
"sort"
"strings"

"github.com/robertkrimen/otto"
)

// CompleteKeywords returns potential continuations for the given line. Since line is
// evaluated, callers need to make sure that evaluating line does not have side effects.
func (jsre *JSRE) CompleteKeywords(line string) []string {
var results []string
jsre.Do(func(vm *otto.Otto) {
results = getCompletions(vm, line)
})
return results
}

func getCompletions(vm *otto.Otto, line string) (results []string) {
parts := strings.Split(line, ".")
objRef := "this"
prefix := line
if len(parts) > 1 {
objRef = strings.Join(parts[0:len(parts)-1], ".")
prefix = parts[len(parts)-1]
}

obj, _ := vm.Object(objRef)
if obj == nil {
return nil
}
iterOwnAndConstructorKeys(vm, obj, func(k string) {
if strings.HasPrefix(k, prefix) {
if objRef == "this" {
results = append(results, k)
} else {
```

```
results = append(results, strings.Join(parts[:len(parts)-1], ".")+"."+k)
}
}
})

// Append opening parenthesis (for functions) or dot (for objects)
// if the line itself is the only completion.
if len(results) == 1 && results[0] == line {
obj, _ := vm.Object(line)
if obj != nil {
if obj.Class() == "Function" {
results[0] += "("
} else {
results[0] += "."
}
}
}

sort.Strings(results)
return results
}

69:F:\git\coin\ethereum\go-ethereum\internal\jsre\completion_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package jsre

import (
"os"
"reflect"
"testing"
)

func TestCompleteKeywords(t *testing.T) {
re := New("", os.Stdout)
re.Run(`
function theClass() {
this.foo = 3;
this.gazonk = {xyz: 4};
}
theClass.prototype.someMethod = function () {};
  var x = new theClass();
```

```
  var y = new theClass();
y.someMethod = function override() {};
`)

var tests = []struct {
input string
want  []string
}{
{
input: "x",
want:  []string{"x."},
},
{
input: "x.someMethod",
want:  []string{"x.someMethod("},
},
{
input: "x.",
want: []string{
"x.constructor",
"x.foo",
"x.gazonk",
"x.someMethod",
},
},
{
input: "y.",
want: []string{
"y.constructor",
"y.foo",
"y.gazonk",
"y.someMethod",
},
},
{
input: "x.gazonk.",
want: []string{
"x.gazonk.constructor",
"x.gazonk.hasOwnProperty",
"x.gazonk.isPrototypeOf",
"x.gazonk.propertyIsEnumerable",
"x.gazonk.toLocaleString",
```

```
"x.gazonk.toString",
"x.gazonk.valueOf",
"x.gazonk.xyz",
},
},
}
for _, test := range tests {
cs := re.CompleteKeywords(test.input)
if !reflect.DeepEqual(cs, test.want) {
t.Errorf("wrong completions for %q\ngot  %v\nwant %v", test.input, cs, test.want)
}
}
}

70:F:\git\coin\ethereum\go-ethereum\internal\jsre\deps\bindata.go
"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: %v", name, err)
}

var buf bytes.Buffer
_, err = io.Copy(&buf, gz)
clErr := gz.Close()

if err != nil {
return nil, fmt.Errorf("Read %q: %v", name, err)
}
if clErr != nil {
return nil, err
}

return buf.Bytes(), nil
}

type asset struct {
```

```
bytes []byte
info  os.FileInfo
}

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 _bignumberJs =
[]byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\x94\xbc\x6b\x77\x9b\xc8\x93\x38\xfc\x7e\x3f\x85\
xc4\xc6\x9c\x6e\x53\x20\x90\x9d\x38\x86\x14\x9c\x4c\x62\xe7\xe7\x79\x1c\x3b\x4f\x9c\xcc\xcc\x
ae\xa2\xc9\x91\x51\x23\x75\x82\x40\xe1\x62\xc7\x09\xfe\x7d\xf6\xff\xa9\x6e\x40\xf2\x25\xbb\xb3\
x6f\x2c\xe8\x4b\x75\x75\x75\xdd\xbb\xf0\x68\x77\x70\x29\x17\x59\xbd\xba\x14\x85\xf3\xa5\x1c\x5
c\x8d\x1d\xd7\xd9\x1b\x2c\xab\x6a\x5d\xfa\xa3\xd1\x42\x56\xcb\xfa\xd2\x89\xf3\xd5\xe8\xad\xfc\x
2a\xde\xc6\xe9\x68\x7b\xf8\xe8\xf4\xe4\xd5\xd1\xd9\xab\xa3\xc1\xee\xe8\x3f\x46\xbb\x83\x55\x3
e\x97\x89\x14\xf3\xc1\xe5\xcd\xe0\x87\x48\xe5\x62\x50\xe5\x83\x44\x7e\x7f\x0c\x5c\x91\x5f\x8a\
xa2\xfa\x5a\xc8\x95\xc8\x46\x79\x55\xe5\xff\x59\x88\x45\x9d\xce\x0a\x5b\x7c\x5f\x17\xa2\x2c\x6
5\x9e\xd9\x32\x8b\xf3\xd5\x7a\x56\xc9\x4b\x99\xca\xea\x86\x96\x19\x26\x75\x16\x57\x32\xcf\x98\
xe0\x3f\x8d\xba\x14\x83\xb2\x2a\x64\x5c\x19\x41\xd7\x31\x50\x5d\xfd\xdb\x8c\x09\xc8\xf8\xcf\xa
b\x59\x31\xa8\xa0\x00\x09\x39\xd4\x50\x42\x82\xd5\x52\x96\x81\x4c\xd8\x90\x25\x03\x99\x95\xd
```

5\x2c\x8b\x45\x9e\x0c\x66\x9c\x17\xa2\xaa\x8b\x6c\xf0\xc5\x34\x4f\xd9\xf8\x19\x18\x71\x9e\x95\x55\x51\xc7\x55\x5e\x0c\xe2\x59\x9a\x0e\xae\x65\xb5\xcc\xeb\x6a\x90\x89\x6b\x03\x04\x87\x4c\x5c\xb7\xeb\x10\xc0\xac\x4e\xd3\x21\x66\xa6\xf9\x2f\x96\xc1\x18\x9e\xed\xc3\x5b\x30\x2e\x67\xa5\x30\x38\xff\x49\xfd\xe8\x36\x19\x94\x28\x2c\xc3\x00\xcf\x45\xcc\xba\x15\x13\x6c\x21\xdd\x41\x28\x12\x7e\xc9\xe1\x23\x4b\xe0\x9d\x95\x38\xc2\xf2\xe0\xab\x5a\x87\xe5\x68\xe8\xa3\x30\x10\xab\x9b\x35\x0d\x16\xdc\x34\xdd\x5d\x31\x44\xb7\x69\x86\x04\xec\xbd\x58\x1c\x7d\x5f\x33\xe3\x6f\x3b\x32\x2c\x56\xa1\x31\x31\xac\x73\xa7\x4c\x65\x2c\x98\x0b\x19\xb7\x8c\xa9\x65\x70\xcb\x60\x91\xff\xe9\x93\x63\x58\x95\x65\xf0\xe8\x89\x01\x7b\x07\x61\x16\x19\xd2\xf0\x0d\x83\x3b\x95\x28\x2b\x56\xf6\x84\x59\xb0\x04\x4a\xc8\x69\xbb\x79\xc4\x12\xa7\x44\x37\xf4\x46\x22\x62\x25\x96\x2d\x68\x8f\x83\xed\x71\xdf\x83\x2f\xa6\x59\x3a\x85\x58\xa7\xb3\x58\xb0\xd1\xdf\xee\x27\xc7\xdd\x6d\x3e\x39\x23\x20\xb8\xa9\xc8\x16\xd5\x32\xf4\x9e\x12\xa5\xdf\xc2\x25\xd1\x32\xc7\xa1\xc7\x7d\x02\xba\xff\x14\x11\x4b\x27\x5e\xce\x8a\x57\xf9\x5c\xbc\xac\x98\xcb\x1f\x5d\xa3\xc4\xd7\xac\x04\xcf\x85\x0c\x12\xa7\xe4\xb7\x22\x2d\x05\x11\xfa\x2e\x19\x7b\x22\x3b\x25\x0a\xa7\x84\xc4\x11\x28\x1c\x01\x89\x13\x23\xa3\xc7\x98\x47\xa2\x05\xcd\x7d\x01\x57\xb9\x9c\xb3\xb7\xe8\xfe\x6f\xb4\x46\x74\xd5\xb1\x6e\xd1\x41\xa0\x2d\x5a\xdc\x04\x22\xfe\xfb\xdf\xc4\x90\x79\xc1\x0a\x74\x41\xa2\x08\x64\x88\x9e\x1b\xc8\x11\x7a\x2e\x14\x96\xc5\x83\x1e\x35\x81\x85\x42\x68\x22\xa6\x1b\x04\x6e\x35\xaf\xf4\xfb\x1a\xae\xdb\x13\x51\xcd\xf7\x8f\x85\x07\xff\x17\xe2\xdd\xde\x12\x62\xac\xc0\xd2\x91\xd9\x5c\x7c\x3f\x4f\x98\xe1\x18\x9c\x87\xb6\x67\x9a\x6a\x7c\x77\x78\x86\x63\xd0\xa1\x71\x60\x92\xa0\x88\x59\x11\x2f\xd9\x48\x8c\x24\xe7\xa1\x1b\x31\x37\x2c\x4c\x93\x15\x28\x39\x14\x16\x5a\xdd\x3a\xd2\xf2\x38\xa8\x65\xeb\x4b\x92\xd4\x6c\xc1\x5c\x90\x9c\xfb\xdd\xf8\xb2\xe5\x02\x0e\x12\xdd\x60\xff\xf9\x7d\xb4\x25\x0f\x24\x91\x88\xd0\xac\xfb\xd1\x8f\x0c\xb4\xed\x9a\x07\xea\xb0\x36\xbb\x94\x50\x5b\x1e\xe7\x32\xd9\x9a\x0a\xb9\x69\x7e\x31\xcd\x7a\x8b\xed\x12\xa7\xdc\x15\x1c\x0a\x2c\x6c\x69\x7b\x50\x84\x3f\x38\x1d\x02\x1d\x07\x09\x73\x40\x84\x1f\xc8\x84\xbd\x09\x0b\xd5\x31\xa1\x1e\x77\x1a\x74\x07\xb2\x75\x6e\x53\x90\xc8\x0a\xcb\xe3\x3b\x37\xa0\xb7\x28\x2d\xbc\xe1\x50\x87\x52\xf3\x80\x34\xcd\xc4\x89\x9d\x75\x5d\x2e\x59\x4f\x25\x45\x12\xa8\x6d\xbc\x09\xea\x50\x06\xfc\xe1\x08\x09\x0a\x0e\x0f\xb6\x36\x47\x24\xbb\xb1\xbb\x7d\xdd\x6a\x2c\x6d\xac\x15\xad\x02\x69\xdb\x41\x69\xa1\xe1\x1a\xc4\x11\x3d\x3c\x2d\x1e\x83\xed\x6d\xbc\x45\xf7\xb6\xd7\x97\xaf\x49\x8f\x41\x05\x52\xeb\x4c\xd2\x96\x09\xc4\xb0\x84\x05\xac\x61\x8e\xe2\x0e\x9b\xc0\x0a\xdf\xc1\x35\x7e\x55\x2b\xee\x1d\x84\x95\x69\x2a\x51\xaa\xf2\xd3\xfc\x5a\x14\xaf\x66\xa5\x60\x9c\xc3\x3c\x44\xd7\x34\x59\x82\xbf\xc3\xef\xe8\x02\x8d\xb8\xc7\x55\xb0\x6e\x55\x5f\xc5\x61\x89\x6b\x67\x9d\x5f\x33\xd1\x6e\xcc\x9e\x73\xf8\x1d\x13\x58\x3b\x31\x96\x2c\x65\x05\x5b\x3a\x31\x87\xa5\x23\xb8\x12\x7a\x0e\x6b\x47\xe0\xda\x89\x7b\x4e\x5a\x60\xc9\x04\x54\xd4\x55\x63\x82\x8b\x8e\x69\x5c\xc4\xc5\xc4\xb6\x93\x69\xb0\x70\xd6\xf9\x9a\x71\xc5\x2e\xc3\xc5\xc4\x9d\xb6\x42\x64\xb8\x06\x35\xb9\xe1\x3c\xb2\xed\xda\xa7\x95\x70\x41\x4b\x61\x0d\x4b\xa7\x44\x09\x4b\x7c\xc5\x96\xb0\x86\x15\x5c\x13\xfc\x05\x2e\x9d\x18\x62\x5c\x3a\x05\xd4\xa8\x70\xca\xb1\xb6\x56\x96\x07\x73\x5c\x4c\xf2\x29\x24\x98\x8d\xc6\x10\x63\xdc\x34\x6e\x98\x37\x8d\x36\x0f\x8b\x49\x6e\x79\x53\x88\x71\x3f\xbc\x8e\x5a\x93\x31\x6f\x9a\x98\x9b\x26\x73\x11\xaf\x9b\xe6\x1a\x91\x2d\x9d\xf2\x85\x1b\xed\xf9\x63\xce\xfd\x79\x98\x34\xcd\x1c\x31\x31\x4d\xb6\xaf\x46\xc4\x4d\xf3\x0c\xf1\xda\x34\x3d\x73\x31\xc9\x6d\x6f\xba\x3d\xe9\xb9\x7f\xc0\x39\x78\xb4\xa2\xde\xa0\xc0\x38\x4a\x99\xe1\x19\x60\xaf\xb8\x4f\x1b\xed\xd8\xb7\xa3\x0f\xe6\x10\x73\x3a\x49\xdb\xce\x02\xcb\x22\x52\xe5\xd3\x30\x0b\x38\xed\x03\x5d\xc8\x9b\x86\x59\x56\x0d\x0b\xa7\xce\xca\xa5\x4c\x2a\xe6\x71\x2d\x98\x5b\x34\x1e\xb6\x14\xd6\x1d\x73\x75\xdc\x86\x11\x2

4\x21\xce\x03\x61\xe1\xb9\x12\xd9\x97\x15\x5b\x4c\xe6\x96\x35\xe5\x3c\x10\x98\x32\x01\x35\xbf\x6d\xd5\x98\xd8\xf0\xe2\xe7\x87\xbc\x58\x12\x2f\xd2\x11\x55\xa8\x89\x56\x91\x9d\xad\xc0\x85\xe7\x20\xe1\x8a\x47\x6e\x53\xf9\x5f\x61\x48\xea\xbc\x03\xe8\x54\xf9\x85\x56\x3d\xea\xbc\x73\xd2\xf5\x13\x77\x4a\x26\xd8\x11\x40\x60\xc8\x06\x2f\xb1\x60\x42\x31\x16\x7a\x87\x88\xb2\x69\xc6\xfb\x88\xd2\x34\x7f\x0b\xb1\x8c\x12\xb6\x84\x92\xfb\xa9\xfa\xe9\x15\x82\xc0\x8f\xac\x35\xd9\x9c\x30\x25\x7e\x23\x98\x3d\x2c\x62\x8c\x56\xed\xdc\x05\xca\xea\x10\xb3\xa6\xf9\x2d\xc4\x9a\x6b\xc5\x10\x64\x61\x1c\x2c\x95\xc0\x42\x4c\x1a\x6f\x89\xb4\x68\xdd\x0a\x2c\x39\x0e\x36\x96\xb0\xc4\x54\xb5\x92\x66\x0b\x63\x65\x79\x6c\x3b\x0b\x5d\x75\x70\x34\xdd\x31\x82\xcc\xb6\x5b\x48\x3c\xd8\xcc\xb6\xb0\xb6\x63\xe8\x86\xd6\x96\x87\x18\x9b\x66\x3b\x87\xdf\x99\xd4\x53\xae\x7c\xe1\x9a\x66\x1e\x19\xb6\x61\x2d\xfd\xe5\xe6\x64\xbe\xdf\xf3\xaa\xd0\xd5\x0a\x9a\x09\x62\x35\xad\x05\xe8\x09\xaa\xce\xa5\xa1\xb7\xc0\xb2\xe4\x8b\x4e\xac\x03\x85\x7b\xd1\xf7\xcb\x29\x87\x61\xe1\x94\xfc\x67\x85\x45\x70\x59\x88\xd9\xd7\xdb\xcc\x21\x7f\x8b\x55\x50\x10\xcc\x0a\x8b\x9e\x4b\xaa\x0d\x2e\xc7\x2d\x97\x14\xc4\x27\xba\x9b\x65\xa1\x68\x1a\x11\x56\x4d\x23\x86\x18\x33\xc1\x39\xe9\xfa\x02\x98\x6c\x1a\x63\x2e\x62\xb9\x9a\xa5\x03\xa5\x81\x4a\x83\x5b\xfd\xf0\xc8\x18\x90\x5f\x97\x27\x83\x62\x96\x2d\x84\xe1\x1b\x83\x2c\xaf\x06\xb3\x6c\x20\xb3\x4a\x2c\x44\x61\x70\xf2\x51\x86\x5b\xfa\xf2\x44\xaf\xae\xcf\x90\xe8\x51\xa0\x07\x12\xb3\x5e\x1e\xb2\x89\x6d\xcb\x69\x90\x75\x1a\x47\x19\x01\xcc\x26\xee\xf4\x57\x7e\x00\x6d\xd4\xaa\x76\x6f\x6c\x8f\x87\x3f\x22\xe1\xc4\xc4\x53\x8a\xdd\xfd\x37\x61\xa5\x1a\x26\x42\xa9\x6e\x9f\xd1\x6f\x05\xd4\x94\x71\xd8\x12\x9d\xd3\x0e\x2d\x8d\x12\x11\xf9\xa8\x28\xf2\x82\x4d\x0c\x7a\xfe\x4d\x2e\xce\xb4\x3b\x03\x46\xbc\x5a\x1b\xca\xc9\x4d\xe4\xc2\x00\x63\x2e\xaf\xf4\xdf\x0f\xf9\x49\x56\x19\x60\x88\x6f\x06\x18\x8b\x4a\xfd\x11\x06\x18\x69\xa5\xfe\xd0\xe3\x4a\x66\x75\x49\xbf\xf9\xdc\x00\x63\x9d\xaa\x97\x75\x21\x62\x49\xfe\xbb\x01\x46\x31\xcb\xe6\xf9\x8a\x1e\xf2\x3a\xa3\x31\x4a\x6f\x18\x60\x54\x72\x25\x68\x70\x95\xbf\x96\x0b\x59\xe9\xc7\xa3\xef\xeb\x3c\x13\x59\x25\x67\xa9\x7a\x3f\x96\xdf\xc5\x5c\x3f\xe5\xc5\x6a\x56\xe9\xc7\x62\xa6\xb6\x48\x2b\xe5\xd7\xaa\xe9\xdd\xd6\x8a\x9d\xac\x1b\x60\x6c\x36\x39\x9d\x88\xa9\x65\x30\x3e\x30\xac\xcc\x32\xfc\x81\x61\x55\x3c\xa8\x96\x45\x7e\x3d\x28\x9c\x6c\xb6\x12\xb8\x19\xac\xe9\x64\xc0\x5b\x74\xa1\xd8\x10\xf4\x63\xc7\x65\x9a\xa4\x7d\x1c\x01\x29\xc4\x30\x23\x95\x02\x4b\x7c\x4f\xfa\x65\xc6\x7f\x0a\x5f\xdb\x7a\x24\xe7\x74\x46\x47\x5d\xaa\xa3\x2e\xd5\x51\x2b\x7f\x46\x29\xa2\xcc\x96\xe0\x86\x39\xcf\x2d\xbc\x81\x1a\x33\x48\x70\x36\x49\xd1\x25\xc3\x90\x8c\x96\x13\x69\xd7\xb6\x37\xdd\xf1\xdc\xc6\xed\x75\x4e\x8a\x73\xc6\x72\xcb\xe3\xa3\x1b\x0e\x69\x88\xb3\xce\xec\x29\xd7\xb0\xe0\x4a\x72\x06\x42\x3b\x01\x5d\xe7\x0b\x4c\x83\x99\x76\x01\x5c\xe2\x41\x8c\x95\x2b\xea\x41\xbe\xa3\x56\xce\xed\x1b\xcb\xd3\x0e\xa6\xd6\xe7\x84\x76\x4a\xce\x8c\xf7\x10\xf5\xad\x39\x12\x62\x74\xc3\x3a\x72\xfd\x7b\xe8\xde\x2a\xd9\x2e\xc8\xe6\x65\x9d\xcd\x9b\x4d\x52\x8b\x8c\x14\xa3\x19\x89\x9f\xec\x74\x33\xc8\xf5\xda\x0f\xab\x88\xc5\x4d\x53\xb4\x16\xb0\x6a\x9a\x0a\x91\x89\x2d\x0b\x18\x87\x4f\x9b\xe6\xa9\xd6\x5a\xfb\x6a\x44\xa1\x2c\x20\x79\x1d\x79\xe8\x46\x75\xe8\x46\x2d\x1a\x53\xdf\xf5\x67\x93\x94\x60\xef\x78\xae\xe9\x6d\x03\xeb\x2c\x63\xd6\x34\xc3\xd9\xc6\xf4\x0f\x3a\x5a\xd1\xb9\x47\xa4\x6c\x85\x0a\xb6\x68\x08\x2e\x27\xd9\xce\xcd\x14\x48\xda\xec\xac\x69\x5c\xee\xab\x66\x25\x85\x20\x94\xcb\x80\x98\x47\xac\x87\x91\x42\x89\x1e\xa4\xb6\xcd\xfd\xad\x46\x8b\xf8\x61\x39\xb9\xb1\xf3\x29\x10\x7d\x91\x50\x5e\xb1\x0e\xe9\x9d\xe5\xa4\x9e\xf2\xdd\xd2\x77\x39\x14\x4a\x4b\x07\x5a\x4b\xba\x88\xa9\xd6\x30\x39\x7a\x50\x6b\x96\xaa\xd5\xb9\xd4\xea\x5c\xf2\x8d\x8b\x4c\x7d\x16\x96\xb4\xfe\x9d\x21\xa5\x3a\xba\x21\x96\xa4\x9d\x1d\x61\x59\x7a\x67\x78\x66\x9a\x4c\x3d\x91\x31\xd7\x6a\x97\x98\x78\x92\x2a\x28\xf4\x3b\xc4\x33\xcd\x55\x01\x91\xd4\x26\x57\xa0

\x44\xef\x56\xa3\x33\xdb\x72\xae\x70\xa6\x5c\x06\xe2\x34\xad\xeb\x6e\x85\x23\xee\xab\x30\xe1\x88\x17\x6f\x14\x0e\xbd\x1a\xdb\xb2\xfd\x24\x5b\xaf\x94\xec\x7d\xc0\x99\xb3\x2e\xf2\x2a\xa7\x70\x0b\xbe\xb5\x76\xc2\xe3\xf0\x0e\xc7\x2e\x7c\xc5\x7d\xf8\x0d\xed\x03\x78\x82\x63\x0f\xde\xa0\xed\x89\x03\xf8\x81\xf4\xf7\x0b\x0e\x5d\xf8\x17\x1e\xc3\x1f\x38\xf4\xe0\x4f\xf4\xe0\x77\xf4\x5c\x17\xfe\xc2\x9f\xad\xe6\xbf\x10\xeb\x59\x31\xab\xf2\xc2\x27\xf7\x73\x51\xe4\xf5\x7a\xab\x09\xba\x26\xf9\x43\xf8\x7b\x50\x8a\x38\xcf\xe6\xb3\xe2\xe6\x4d\xdf\xe8\x42\xd2\x2a\xa1\x37\xf7\xe6\x0e\x8c\x7b\x5d\x6a\xf8\x6d\xd0\xb3\xd8\x2c\xcb\xab\xa5\x28\x30\x83\x99\xf3\xfe\xfc\xe3\xd9\xeb\xcf\x1f\xdf\xa1\xdb\xbf\xbc\x3e\xff\xf3\x0c\xbd\xfe\xf5\xd5\xd1\xc9\x29\x8e\xfb\xd7\xe3\xd3\xf3\xf3\xf7\xb8\xd7\xbf\xff\xeb\xe5\xe9\x31\xcd\xdf\xbf\xdb\xa2\x80\x3c\xbd\xdb\x76\xf4\xc7\xd1\x19\x3e\xbb\xdb\xa6\xa0\x1f\xdc\x6d\xd3\x4b\x3c\x87\x99\x73\xf4\xf1\xd5\xe9\xc9\x6b\x3c\x84\x99\xa3\x6d\x03\xf6\xa9\x17\xad\x02\x95\x3e\x24\x61\xc1\x9f\xb7\x20\x71\x56\x2c\xea\x95\xc8\x2a\xe2\x3c\x49\xee\x55\x42\xac\x66\xe4\x97\x5f\x44\x5c\x6d\xa2\xe6\x32\xda\x02\xd3\x92\xa5\x74\x96\xb3\xf2\xfc\x3a\x7b\x57\xe4\x6b\x51\x54\x37\x2c\xe3\x91\x56\x19\x4c\x60\x39\xc9\xa6\xdc\xa7\x60\x78\xe0\xde\xfa\x0f\x27\xcb\x2e\x8d\x50\x6d\xe6\xc8\x49\x45\xce\x65\x37\xab\x8f\xaf\x59\x86\xc6\xeb\xa3\x57\x27\x6f\x5f\x9e\x7e\x7e\x77\xfa\xf2\xd5\xd1\x85\xc1\xc9\x7f\x14\xe0\xc2\x11\x8c\x21\x23\xe5\xf3\x0e\xdd\x86\xa2\xc1\x49\x36\xc5\x77\xa0\xe6\x28\x02\x9d\x9c\xbd\xf9\xfc\xf6\xfc\xf5\xd1\x66\xca\xf3\x6e\xca\xd7\xad\x29\x5f\xf5\x94\xa3\xbf\xde\x9d\x9f\x1d\x9d\x7d\x38\x79\x79\xfa\xf9\xe5\x07\x9a\x43\xde\x11\x8f\xfe\xa5\x5c\x21\xb0\x8f\xc0\x6d\x67\x53\x8b\x37\xdd\xc6\xe0\x37\x02\x47\xa3\x9e\xa8\x07\x6f\xca\x7d\x5a\xd0\x3e\xda\x1e\x62\x33\xea\x65\x6e\x28\x22\x5b\xf8\x82\x73\xde\x22\x30\xf9\x0d\x9e\x4c\x5b\xbc\x5f\x9e\xbd\x39\x7a\x6c\x6d\xdb\xbb\xbb\xb8\xb7\x81\xfc\xa6\x5b\xfc\xc7\x2f\x17\x77\x1b\x11\xbd\x41\x9b\xfd\xb8\x8b\x80\xaf\x33\x66\x90\x59\xc6\x20\x9e\x65\xe4\x39\x5d\x8a\xc1\x0f\x51\xe4\x06\x88\x0d\x7a\x6f\xe0\x47\x8b\xde\xd1\xfb\xf7\xe7\xef\xd5\x11\x30\x81\x88\xc3\xa1\x68\x1a\x0f\x11\x45\xd3\x90\x36\x11\x11\x23\x45\xf0\x2f\x64\x5f\xa8\x8f\x47\xc7\x7e\xbe\xb5\xc8\x35\x01\xd5\x30\xbf\x68\x78\xaf\xde\xff\xd7\xbb\x0f\xe7\xff\x13\xbc\x3f\x70\xc8\xa8\x75\xb8\x6c\x9a\x8e\x35\x87\x1d\x6b\x2e\x39\x08\xd3\x1c\xfe\xa1\xf2\x03\xb4\x86\x11\x17\x37\xeb\x2a\x1f\xd4\xd9\xec\x6a\x26\xd3\xd9\x65\x2a\x0c\x58\xf2\xc7\x71\xf8\x43\xe3\xf0\xf6\xfc\xf5\xc7\xd3\xf3\x7b\x8c\x72\xd8\x51\xee\xcf\x2d\x46\xf9\x53\x4f\x78\x77\xfe\xe7\xe7\x77\xef\x8f\x5e\x9d\x5c\x9c\x9c\x9f\x3d\xc2\x8e\xbf\x6f\x4d\xf9\x5d\x4f\x39\x3e\x7f\xff\xb6\xe5\xa9\x07\xf2\x25\xa2\xbf\x50\x6c\x9f\x44\xeb\xc0\xb6\xe3\x36\xf8\xfe\x05\xc5\x2d\xcc\x9c\xd5\xec\x3b\x3e\x14\xaa\xef\x6c\x23\xce\x1f\x9c\xb4\xe2\x6a\xa8\xcc\xfe\xd7\xa1\x0b\x3d\x54\xfb\x7d\x0f\x34\x06\x1e\xba\xee\x81\x77\x78\x38\x7e\xba\x7f\xb0\xef\x1e\x1e\x8e\x21\xc3\xb7\xb3\x6a\xd9\x8e\x67\x7c\x57\x98\x63\xf7\xf0\xc0\x7b\xea\x3d\xa2\x26\x56\xec\xde\x58\xfe\x98\x3e\x78\xbe\xf7\xfc\xf9\x33\xf7\xf9\x2e\xf3\xdc\x83\xbd\x83\x7d\xef\xf9\x78\x7f\xf7\xce\xbc\xc6\xe5\x16\xeb\x46\xdd\xef\xd9\xe8\x8a\xad\x3c\xf3\xbd\xe4\x31\xba\x90\xe0\x64\x0a\x69\x6b\x93\xbe\x29\x6f\x4e\xb4\x01\xa9\xd8\x9c\xa0\xb7\x4f\xf1\xa8\xf0\xdf\x41\x8e\x73\x26\xc8\x61\xfb\x83\xcb\x84\x2d\x4d\x73\xe9\x2c\x44\xf5\x5e\xad\xfb\xc7\x2c\xad\x45\xa9\xcd\x7b\x85\x0f\x3a\x54\x80\xf9\x51\x66\xd5\xde\xf8\x65\x51\xcc\x6e\x58\xbe\x8b\x63\xce\x83\x3c\x2c\x03\x5e\xa3\xb7\xe7\xb9\x07\xe3\xdd\x6a\x52\x4e\x2d\x56\x4d\x4a\xcb\x9b\x86\x61\xe8\x79\x1c\xea\x10\x0f\x85\xf7\x34\x62\xc5\x3f\x00\x3a\xe6\x1c\x08\x06\x16\x24\xfa\x1a\x0e\x16\x4a\xfa\x59\xa2\x1d\xc7\x7a\xc7\x13\xde\x3e\x87\xd2\xc2\x31\x0f\x4a\xcc\x47\xe3\x3e\xb8\x54\x3b\xd2\x64\xfc\xed\xa6\xda\xde\xcd\x56\x23\x61\x7e\xd0\x23\x3e\x7e\xee\xed\x1f\xec\x1f\x1e\x3c\x3b\xf0\xdc\x67\x4f\x9f\xed\xb2\x3d\xcf\x24\x0c\xb8\xe5\xb9\x87\x87\x4f\x3d\xef\xd9\xf8\xe0\xe0\xe0\xd9\xae\xc6\xc5\xda\x1f\x1f\xee\x1f\x3e\x3

b\x18\x1f\xea\x96\xf1\xd4\xf2\x9e\x1d\x1c\x1c\x8c\x3d\xfd\xbe\xd7\xee\x7e\x7f\xfa\xe2\x85\xf7\x8c\xeb\x97\xa7\xd3\x17\x2f\x9e\x73\x8b\x1e\x9f\x4d\x7b\x7a\xdc\xc5\xe9\x80\x3b\x71\xbe\xbe\x61\x15\x85\xf7\x8f\x6c\xf5\x40\x6f\xf5\x40\x6f\x55\xc9\x95\xb7\xff\x2b\xcd\xa0\xd2\x49\xa5\xf6\xdc\xda\x6d\x66\x8c\x03\x2d\x1b\xd6\xa6\xc9\x92\x49\x69\x59\x53\x6c\xc1\x07\xda\x83\x4a\x26\xb6\x5d\x4e\x41\x90\x57\x9d\x9b\xa6\x20\x6d\x8d\xef\x27\x37\xb6\x98\x42\x42\x47\xb2\x62\xf9\xa8\xe6\xbb\x35\x57\x3e\x16\x35\x05\x89\xf6\xb0\xa0\xb4\x6d\xae\x13\x56\x25\x4f\x70\x22\xfb\xac\xa4\x0e\x3f\x6c\xaf\x9d\xe2\xd2\x14\x9d\xb3\xe1\x20\x6d\xbc\xd1\x8b\x97\xca\x9b\x4c\xee\x7b\x93\xca\x55\xbc\x09\xc9\x53\xa4\xb1\x76\xd9\x3b\x68\xa9\x23\x50\x42\xea\xc4\x98\x40\x7a\x7b\xcb\x38\xbc\xda\x16\xf2\x3e\x5a\x12\x77\xc2\xcf\x3b\x82\xd3\xc5\xff\x24\x3e\x3b\x2f\x21\xc6\x6c\xf4\xb2\xd1\xe9\x03\x81\x7d\x02\x3e\x48\x6c\x3b\xe0\x39\x8a\x49\x32\xdd\x79\x09\xb5\x7a\xa0\x81\x50\x60\xbc\x9b\x5b\xf5\x6e\x0a\x12\xd3\xdd\xdc\x2a\x76\x5e\xee\xbe\xb4\xc8\xeb\x60\x72\x54\x29\xe1\x2e\x68\x20\xb7\xe2\xdd\x1a\x68\x1a\xca\x9d\xaa\x13\xeb\xd2\x34\x45\x9f\xbe\x2a\xef\x84\xcc\xd9\x83\x08\x4f\xe5\x99\x86\x58\xf0\x1c\xab\xb0\x88\x3c\xdf\xf6\x74\x18\xa6\xa9\x9b\xa3\x1b\x54\xa1\x54\xf9\x69\x52\x00\x13\x39\x1d\x62\x36\x91\x53\xfe\x93\x10\x97\xd3\x90\x5e\xf4\x34\xed\x58\xb7\x48\xe4\x9b\x45\x8b\xcd\xa2\x5d\x02\x41\x12\x58\xda\xbd\x98\x54\x53\x1b\x25\x48\xa4\xa7\x17\xd9\xa4\x22\x60\x2e\xd0\x1b\xca\xdd\xc2\x52\x03\xa8\x59\x07\x7b\x43\x32\xdb\xb4\xbf\xee\x5e\x25\x10\xdd\x99\xf3\xe0\xf6\xbe\x5e\xeb\x23\x58\xbd\xdd\x74\x93\xe4\x85\x6b\xb8\x82\x4b\x38\x87\x0b\x78\x0f\x2f\xe1\x08\x5e\xc3\x67\xf8\x0e\xc7\x28\x9d\x12\x31\x77\x4a\xb5\x25\x38\x41\xe9\xc4\x70\x8a\xb9\x13\xeb\x7b\xb4\x13\xd3\x3c\x51\x18\x9c\x9a\xe6\x29\x05\x56\x5d\x64\xa5\xd5\xa4\x74\x4a\xd3\xcc\xe9\x0f\x3b\x89\x86\xa7\x4d\x43\x83\x87\x48\x23\xfd\x53\x1e\x9d\x98\xa6\x8b\x48\x6d\x4d\x33\x3c\x8d\xdc\xdd\x63\xff\x78\xe4\xfa\xee\xc8\xd5\xbc\x7a\xd5\x6a\xdb\x63\x0e\x97\x78\xa5\x73\xed\x31\x4a\x47\xd8\xb9\x23\xe0\x18\x6b\x2b\xb6\x3c\x48\x9a\x86\x25\x78\x06\x31\x56\x4c\x3a\xa4\x72\xed\x8a\xe5\xea\x01\x8e\xf1\x78\x74\xd3\xb8\x1c\x96\xe8\x06\xa7\x93\xe5\x14\x91\x9d\x4c\x96\x53\x8a\xe7\x82\x65\x1b\x94\x53\x7b\xd8\x37\x9b\x66\x6c\xdb\xe0\x86\xc7\xfc\x52\x6b\x06\x8f\xc3\x02\x87\xee\x46\xc8\x8e\xf0\xa4\x63\xe8\xcf\x78\xda\x3d\x52\x10\x79\x6c\xe1\x18\xd6\x48\xe1\x1d\xa3\x4d\x5a\x1e\xe7\xb0\x0e\x3d\xd3\x64\xa7\x28\xd8\x29\xac\x21\xe1\x70\x82\x82\x9d\xe8\xc7\xad\xf9\x1b\xa8\x1c\x5e\xe2\x67\x38\xc7\x93\xfe\xaa\xe0\x33\x87\x0b\x3c\xef\xc2\xae\xcf\xe1\x45\x70\x3e\xb9\x20\xb5\xe2\xf2\xe0\x3b\x9e\x76\x12\x04\xdf\x7b\x3e\x77\x39\xbc\x56\x74\x86\xd3\x89\x37\x0d\x31\x19\x8d\x4d\xf3\xb5\x65\x05\xf3\x7c\xb0\x46\x97\x24\x91\x9d\xc2\x39\x7c\x86\x0b\x0e\x6e\x98\x46\xec\x3d\x9e\xd3\xf0\xcf\x43\xbc\x30\x4d\xf6\x1e\xdf\xef\x26\x16\x3b\x9f\x78\x8a\x28\x5c\xed\xea\xfd\xe8\xb5\xda\x4e\xc4\xd6\xa1\x4a\x4a\xaf\x31\xb1\x3d\x0e\xf3\xcd\xde\xae\x71\xde\x6d\x68\x83\xb1\x5a\x6d\x0e\xe7\x70\x4d\xab\x79\x88\x29\xcd\xb5\x6d\x28\xd8\x1c\xae\xc3\xcf\xd1\x77\xff\x14\xae\x21\xe1\x9c\xfb\x14\xf8\xae\x4d\x93\xa5\xb8\x46\x05\xba\xdf\xdd\x5d\xe0\xe1\xb5\x69\xce\xb7\xb7\x5b\xb0\x73\x98\xc3\x05\x21\x61\xb7\x4b\xdc\xc3\xa0\xdf\xaf\x17\x2a\x04\x2c\x4b\x4d\xba\x68\x11\xb8\x50\x08\x6c\xa1\xcd\x7d\xd2\xa4\xdd\xd0\x73\x54\xd9\xcd\xcb\xc9\x92\x08\xbf\x86\xd4\x34\x89\x60\x51\x7b\x12\x27\x93\x97\x44\x29\x9f\x9d\xe3\x84\x9e\xa7\x70\x81\x1e\x0f\xae\x97\x32\x15\x8c\xbd\xb4\xac\x17\x47\x5d\x52\xe4\x5c\x27\x4c\x8f\x49\x91\x2f\x70\xd3\x06\x97\x4a\x12\x2e\x3b\x09\xa6\xa0\x3c\x41\x3c\xd3\x7a\x62\x89\x1e\x1c\x23\x0d\x09\x8e\x95\xe2\x3e\x56\x8a\x5b\x31\xf1\x47\x76\x05\xb5\xc5\xae\x1c\x81\x4b\x2b\x56\x69\x44\xcb\x83\x12\x16\x6d\x26\x99\x3a\x62\xb8\x72\x0a\xb4\x16\x9d\x5a\xbc\x52\xba\xfc\x61\x88\x87\xa3\xbf\x99\x1d\x71\x97\x4d\xbe\x5f\xe6\x53\xce\x3e\x5d\x4f\x3e\x5d\x3b\xd3\xdd\x27\x7c\x24\x21\xa3\xde\xc9\xdf\xce\xd4\xe2\x9f\x9c\x27\x23\xa8\x

70\xf4\xf7\x27\xa7\x6d\x79\x32\x82\x02\x47\x7f\xdb\x11\x3b\xc9\x12\x99\xc9\xea\xa6\x39\x9b\x9
d\x51\xb3\xa4\x61\xe5\xee\x27\x8b\x29\x58\xbc\xf9\xfb\x53\x69\x35\x9f\x4a\xeb\xc9\x68\xf1\xc0\x
fb\xba\xaf\xa3\xb0\x8c\x6a\xbf\xee\xaf\x8f\x24\x18\x4f\x3c\x43\x09\x6e\xa1\x2f\x45\x63\xce\x73\x
a7\x44\x59\x9e\xcd\xce\x58\xac\xe3\x48\xdf\x0d\xe3\xc8\xf6\x7c\xaf\xbf\xf2\x18\x92\x16\x8a\x31\
xee\x01\x09\xd8\x38\x7c\xda\x72\x75\x16\x0f\x8d\xef\x06\x22\xab\xb0\xba\x77\xad\x15\x79\xcf\x7
c\xe3\x92\x3c\xef\x68\xec\x3f\x87\xc4\x34\x93\x21\xa6\x91\xf0\xb3\x5b\x4e\x6f\x2c\xc5\x04\xb6\x
d7\xc8\x34\xb2\xfd\x7b\x05\x86\xeb\x50\x0b\x87\x7a\x88\xf1\x3d\x75\x19\x43\xca\x83\x2f\xfa\x8a
\xd2\x50\x4e\xbc\x61\xb1\x24\x32\x06\x97\xb3\x52\x0c\x0c\x2b\xf1\x0d\x83\x93\x7f\xdf\xe6\x71\x
6b\x0e\xb4\x71\xda\xef\x6d\xee\xc4\x98\xb7\x09\x17\x78\x8b\xae\x3a\xdd\x0f\xce\xec\xb2\xcc\xd
3\xba\x12\xca\x07\x44\xf5\xfe\xf0\xc4\xdb\x7b\xb8\xa5\x2c\xef\xdf\x03\x30\xe1\x94\x24\x86\xe2\x
16\x3e\x38\xb1\x90\xe9\x23\xd1\x40\x77\x1f\xa2\xe6\x03\xfd\x55\x49\xb4\x31\x57\x73\xf2\xd5\x7
a\x56\x88\xf9\x87\x1c\x3f\x38\xf1\x6a\x8d\xdb\x34\xef\x41\xbc\x45\x0f\xa4\x02\xb0\x55\x58\xa1\x
e6\xb7\xe9\x9b\x77\x2a\x6f\x8f\x1f\x9c\xf9\xfa\xb1\x9c\x44\xa1\x4a\x3b\x5a\xa3\x54\xf4\x44\xad\x
d3\x54\xbb\xe9\x8c\x65\x58\x74\x77\x8b\x1e\xd9\x07\x8d\xe6\xe8\x86\xf3\xdd\x1b\xc8\x90\xc2\x2
3\xed\xc3\x65\x3b\x9e\x8b\xe8\x06\x99\x92\x2e\x41\x32\xda\x82\x73\x43\xa1\xa2\x4c\xb7\x25\xc
7\x5c\x5e\xc9\xb9\x98\xff\x76\x83\xea\xf9\x57\x3b\xdb\x83\x57\xf7\x77\x06\xef\xe0\x2b\xdf\x02\xa
1\xd2\xee\x62\x21\x8a\x0e\x96\x6a\xf8\x15\xc0\xfd\x47\x00\xba\xe0\x29\x80\xe2\x5b\x3d\x4b\x89
\x4e\xe2\xdb\xaf\xa6\x3f\x05\xd2\x6a\x8f\x53\x3b\x49\xf3\xbc\xf8\xe7\x47\xbc\xa7\x26\x2d\x0a\x3
1\xab\x44\xf1\x61\x39\xcb\x90\xa2\xc1\x5f\x2d\xfc\xec\x91\x23\x0e\xdd\x7b\x10\xce\x8b\x23\xda\
x82\x62\x97\x45\x25\x7e\x05\xeb\x80\xac\x08\xb2\xec\x91\x7d\x70\x1d\xf9\x67\x04\x58\x96\xc7\x
a4\x87\xc4\xc3\x2d\x0d\x87\x9a\x63\xf4\xa8\x96\xfc\xd8\x3e\xff\x7a\xb8\x69\x6e\xb1\x4e\xa8\xdb\
x3a\xbe\x1a\x6b\x58\x67\xb3\xb3\x47\xe6\xab\xa1\x65\x3b\x42\x2c\x66\x95\xbc\x12\xd8\xbe\x3c\
x42\x70\x3d\xfc\x85\xab\x27\xfc\xb7\x28\xf2\xff\x09\x27\x17\x5b\xfe\x9f\xb8\x53\x9a\x91\x8a\xb2\
x6c\x8f\x23\xfd\xe5\x71\x3c\x7f\xe4\x38\xf4\x82\xdd\xf4\xed\xb3\x48\x7f\x7d\x16\x87\xca\xde\xfe\
xef\x87\xa1\x6e\x8e\xf0\x83\x53\xd6\x97\xf7\x40\xdd\x8d\x18\x14\x8c\x04\x4b\x47\xd5\x6a\xbd\x
55\x62\x88\x5b\xbc\x9e\xa9\x5a\x9e\x61\xd2\x34\xc3\xec\xae\xfe\x54\x8e\x23\x19\xcd\xe1\xa6\xc
0\x8a\x14\x98\x9d\x41\xe9\xac\xd3\xba\x64\x82\x07\xca\xaa\xa0\x3a\x41\x50\x39\xea\xd1\x0d\x2
c\xb1\x74\x62\x58\xa0\x68\x55\x48\xda\x34\x43\x7d\xd1\x3a\x5c\x36\xcd\x70\xd1\x01\x5b\x46\xa
c\x85\x27\xb8\xaf\xd7\x5c\x44\xa5\xdf\xad\x3b\x5c\x6a\x57\x76\xab\xba\x60\x40\xcf\x0f\x67\xd1\x
c0\xa8\xf4\xf7\x10\xbf\x46\xb6\xeb\xbb\xca\xd6\xa7\x58\xb1\x94\x2b\x3f\x56\xdd\x49\x2f\x7b\xbf\x
2e\xc1\xd4\x8e\xb5\x1b\xc0\x6a\x74\xc3\x84\x47\x2c\x41\x3b\x81\x1c\x97\xdc\x67\x31\xa6\x90\x
e3\x82\xac\x41\x21\xae\x44\x41\xb6\x0a\x32\x4c\xd4\x05\x6f\xbe\xb9\x03\xda\xea\xbe\xdd\x0a\x
6a\x58\x8d\x2c\xe9\x6f\xad\xf9\x0b\x96\xf5\x77\xfb\x9c\x47\x89\x9f\x41\x82\x19\xba\x81\x0c\xb3\
x20\xd3\x81\xcf\x72\x92\x4d\x87\xb8\x20\xad\xf9\xb3\x46\x7a\x7b\x41\x2f\x9b\xcb\x04\x0a\x7d\x7
3\x24\xaf\x78\x01\x0b\xcc\x41\x11\x40\x38\x25\xe1\xc5\xe4\x06\xbe\xad\x52\x15\x9d\xdf\xdb\xdd\
x54\xeb\x9b\xe9\x49\xd1\xba\xb8\xd4\x94\xe1\x99\xed\x05\x32\x4c\xf4\xf5\xc8\x52\x5d\xb1\xbe\x
58\xa8\xd0\x4b\x17\x5a\xc9\xa0\x30\xcd\x21\x75\x14\x53\x9a\x3c\xc5\x8c\x07\xb6\x4d\x4f\xb0\x9
c\xc8\xa9\x85\x67\xb7\xf4\x6b\x23\xcd\x52\x77\x19\x14\x2a\xd3\x51\x04\xcb\x3e\x52\xb6\xed\xb8
\xd7\xf8\xea\x94\x4e\x98\x80\x25\xc4\xdc\x57\x87\xa8\x4f\xcc\xf3\x3d\xd8\xba\xcc\x00\xa1\x14\x
e1\x2a\x9f\xd7\x29\x09\xcb\x2a\x9f\x3f\xc2\xe1\xfa\xd6\x5c\xd5\x20\x6e\xcc\x9e\x77\x97\xb7\x87\
xd2\x89\x9b\x66\x28\x9c\xb2\x69\x04\x89\xf6\x50\x17\x2e\x44\x1b\x06\xf7\xa9\xa9\x69\xa4\xea\x
95\xdb\xbd\x92\xfb\xec\x10\xf1\xcf\x88\x15\x4a\x44\x94\xed\x86\x0a\x5f\x31\x09\x02\x5c\xd8\xe3\

xaa\xa9\x80\xca\x29\x77\xb1\xe0\xfe\xa6\xeb\x4f\x0e\x52\x0b\x28\xab\x1c\x75\x51\xcb\x04\xd7\x36\x21\x23\x6d\x25\xe6\xa8\x9e\xfe\xa9\xef\xa0\xce\x5a\xfb\xbb\xda\x58\x92\xf4\x91\xfb\x31\x7f\x8c\x32\x1d\x5d\x20\xa7\x78\xb3\x95\xfa\xf1\xa3\x52\x9f\xff\x5a\xea\xf3\x87\x52\xdf\xed\xa9\x15\xfb\x1a\x55\x7c\xa8\xab\x40\x46\x37\x90\xa8\x70\x36\xed\xc5\xbe\x6e\x9a\x61\xa9\xc5\x9e\xb4\x4b\x7a\x77\x9d\xbc\x93\xf2\x44\x4b\x79\xba\x25\xe5\xf4\x4c\x6e\xa0\x1a\x48\xfd\x91\xf4\xdd\xdd\x5c\x89\x75\x8d\x15\xab\x39\x29\x36\x56\x92\x28\x27\xbd\x58\xe7\x58\xdb\x6d\xde\x2c\x0f\xdd\x88\x95\x58\x43\x81\x29\xf7\x59\x8e\x76\x0e\x05\x26\x1c\x8a\x8d\xcc\x06\xb9\x6d\x07\xc5\x46\x9c\xb7\xba\xda\x9b\xb9\xa4\x0b\x77\x32\x4c\xbb\x47\x37\xcc\xed\x4c\xd5\xdd\xa5\x40\xee\x69\x82\x05\x64\x98\xd3\xea\x6e\x90\x05\x3c\x47\x96\x4c\x6c\x3b\x9b\x62\x32\xc9\xa6\x56\x4a\x7f\x72\x3e\x3a\x6b\x5c\xa0\x86\x1d\x3c\xeb\xce\x35\x37\x4d\x96\xf4\x21\x57\xce\xc1\xb2\x4a\x0e\x24\x1f\x09\x94\x8a\x57\xfa\x3a\x00\x52\xf3\xdb\x27\xad\xcf\x59\x65\x3d\xf4\x49\x4b\x2c\x34\xd1\xfb\x0c\xaa\x18\xaa\xf4\xbd\x69\x7a\x43\xa4\x77\x57\xff\x30\x9d\x7f\xdb\x03\xa3\xcb\x39\x1b\x2a\x05\x0f\x62\xa8\x87\xb7\x59\x58\x4e\xc2\x73\xdf\xf3\xab\x50\xf6\x5e\x1f\x64\x58\xed\xde\x58\x24\x10\x72\x52\xb5\x5a\x23\xa8\x5a\x77\xaf\x52\xee\x5e\x46\xee\x9e\x4e\x63\x4a\x52\x0b\x95\x0a\xb4\xda\x3e\x0a\xb4\xfa\x5b\x4b\xd3\x2c\xc8\x05\x0a\x89\xb2\xe4\x5b\x0a\xcb\xe3\xa0\xcc\x9c\x2a\x7b\x78\x4c\xfc\x1f\x11\x15\xa6\x2b\x91\x44\xd3\xf4\xf9\xe3\xa7\x9c\x9b\xe6\x47\x56\xc1\xbf\xff\x2d\xac\xde\xd3\xba\x53\x60\xec\xc2\x73\xf0\x9e\xea\xca\xa7\xcc\xff\xca\xa1\xa2\x75\xd5\xa9\x3c\x24\xf9\x1d\x85\xa3\x6e\x75\x2e\xe0\x02\xbc\x67\x5b\xf4\xe4\x51\xd6\xca\xbc\xe1\x09\xc3\x52\xb5\x33\x2d\x2b\x67\xa4\x65\x32\xa5\x64\x4c\x93\xd9\x17\xba\x68\xe6\x82\x66\x94\xbb\xea\x1e\xc8\xf5\x3d\x52\x4a\x99\x3a\xff\xf2\x5b\x3d\x2b\xc4\xfb\x3c\xaf\x88\x01\xbe\x15\xd5\x63\xce\xfa\x03\x3b\x4f\x22\x58\x3a\x25\x45\x7a\xaa\x90\xea\x9d\xb5\x0f\x8b\x96\x5a\x86\xeb\x3c\xd5\xc1\x1e\xb1\x05\xd9\x65\x92\xcc\x64\x4b\xf4\xf4\x38\x32\xd9\xae\x0a\xeb\x69\x80\xea\x8f\xdc\x91\xeb\x27\x51\xa9\x10\x0c\x94\x7d\x55\xa9\x7f\xc2\x8b\x11\xe7\xba\x0a\x60\x8a\xe8\x8d\xdc\x88\x4e\x91\x25\x1c\x58\x57\xc6\x63\xc5\x7c\x67\x8c\xaa\x8a\x31\xd3\x35\x52\xb0\x0d\x20\xd3\x86\x9a\xc5\x96\xc7\x47\x63\x6e\x33\x37\x8c\x9b\x26\xde\x19\xd3\x30\x05\x31\x43\x4d\x4e\x9f\x91\x34\xde\x29\x75\x51\xe6\x39\xdb\xd4\x64\x6f\x2a\x2c\x85\xc1\x2d\x8f\x5b\x31\x07\xd9\x52\x20\xe3\xdc\xef\x9e\x53\xcb\x30\x48\x53\xd3\x79\x28\x43\xa9\xb2\x61\x90\x62\x6c\x2d\x61\x4f\x6d\x3f\x25\x83\x19\xe8\xfa\x57\x09\x64\x69\xf5\xd1\xd6\xda\x01\x7a\xc5\x4a\xa8\x61\x09\x9e\xba\x9c\x63\xb5\x13\xf3\x1e\x8d\x94\x6b\x37\xae\x60\xd2\x89\xf9\x76\xbb\xd2\x89\xd2\x11\x2f\x62\xd3\xb4\xed\x74\x0b\xf9\xd4\xde\x83\x94\x78\xdf\x38\x3c\x3c\x3c\x34\x14\x8f\xb2\xbc\x69\x8c\xfd\xf6\x95\xf3\x9f\x6c\x68\x65\x4d\x33\xb4\xb2\xbe\x10\xd9\x34\x8d\xa7\x06\x62\xd6\x55\x06\xba\xc4\xf4\xec\x23\x93\x20\x1d\x61\xbd\xb3\xc6\x40\x31\x27\x0e\x65\x8b\xbc\xe4\x8e\xf8\xc6\xca\xed\x6a\x85\x61\xae\x66\xd4\x50\xb7\x33\x5c\x0e\x75\xb7\xd7\x6e\x38\xff\x29\xb1\x6e\xe7\x2c\x2d\xdc\x87\x94\xfe\xe4\xe8\xdd\xf6\x81\x4d\xb7\xa4\x07\x5f\x5b\x33\xae\x60\x90\x15\xaf\xd3\xff\xc9\x4f\x6d\xeb\x80\xba\x04\xea\x4a\xa7\x50\x35\x57\x9f\xe3\xa5\x13\xc3\x05\x92\x1d\x3b\xb8\x63\xc7\x78\x97\x39\x3d\x37\xcd\x0b\x9d\x41\x32\xcd\x8b\xad\xcc\xe9\xf0\x92\x0c\xa7\xf6\x00\xce\x4d\x73\xa8\x47\x0c\x2f\x9a\xe6\x82\x7e\xf4\xdb\x79\x5f\x5f\x21\xda\xf8\x5f\x79\x27\xbb\x78\xe9\x94\x40\x90\x23\x5d\x6b\xe1\xea\xfa\x15\x97\xfb\xdb\xf5\x18\x1c\x44\x5b\x92\x56\xb1\x4b\x15\xc9\x58\x15\x13\x3a\x61\xda\x43\x49\x37\xb9\xb3\x05\x5e\xf4\x8f\x8a\xc7\x56\x78\x0e\xe7\x78\x01\x17\xb8\x82\x5c\x99\x15\xe5\xe4\x91\x49\x49\xad\x05\xac\x70\x32\x55\xb6\x6a\xb5\x55\x7e\x94\x17\xec\x1a\xcf\xe0\x0a\x5f\x92\xab\x1a\xd8\x76\x1e\xa2\x1b\x6c\x8a\xe4\xd7\x78\x31\xc9\xa7\x3b\x57\x30\x57\x0f\xa3\xab\xc6\x85\x12\x53\xa8\x31\xb7\xca\xa0\x0e\xf3\x80\xc7\x78\xae\xee\x4d\x76\xae\x

60\x89\xe7\x93\x52\x0f\x4a\x70\xbe\x1b\x5b\xcb\xdd\x35\xc4\xb8\xde\x8d\xad\x64\xe7\x6a\xf7\xc
a\x5a\x4d\xea\xa9\x55\x40\x81\x2c\x1e\x5d\xab\x1b\x82\x84\x46\x73\x6b\xbe\xbb\x84\xd5\xa4\xb
6\xed\x29\xc6\x3b\xd7\x01\x8d\xc3\xa2\x63\x87\x22\xb2\x2c\xe9\xaf\x7a\x67\x90\x6c\xdb\x0a\xa4
\x66\x8b\xb6\x6c\xed\x1f\xaa\xf6\xc1\xbd\xcb\x41\x8f\x94\xfb\xf3\xed\x52\x39\x7d\x51\xa8\x5c\xa
4\x0c\x1f\x2a\xf8\xe7\xbd\x82\x07\x11\x91\x41\xa0\xe5\xfc\x4a\xa3\xb2\xa5\x4b\x1e\x0f\xcb\x3e\x
b7\xa1\xd8\x83\xfb\xc9\x43\x1e\x91\x65\xf1\xda\x85\xa9\x41\x83\x54\x95\x77\xff\x37\x60\x63\x57
\x03\xeb\xcc\x54\x07\x73\xec\x76\x30\x55\x0d\xdf\xa3\x14\xfb\x25\x4c\xef\x17\x30\x3d\xa5\xc3\x
75\x9c\xbb\xe5\x36\x3a\xe5\x3a\x95\x95\x2e\x4d\xcf\xd1\xfa\xcb\xe9\x0b\x79\xa0\xa6\xd7\x87\xb
5\x3c\x50\x62\x37\xaa\xab\xe2\x21\x4f\x90\x84\x25\x45\x39\x51\x25\xda\x5d\xfc\x0d\x33\x8c\xa3\
xa4\xd7\x5b\x7e\x02\xcb\x4d\xf9\x53\x1b\xe6\x14\x98\x93\x27\x07\x35\x16\xb0\xb4\xb1\xe0\x90\
x87\xae\x69\x2e\x43\xb7\xe3\xee\xe5\x4e\xde\x34\x39\x24\x38\x6b\xbf\x89\x60\x2e\x14\x3c\x58\
x86\x45\x50\x58\x98\xf3\xc4\xc2\xd2\xea\xfb\x0a\xc8\x79\x50\x87\xaa\x7c\xbe\xed\x50\xcb\x17\x
9c\x43\xac\x6a\xea\x0d\xdb\xb0\x12\x7e\x5b\x61\x1a\x25\xd6\x5f\xce\xfd\x12\x27\x8b\x82\x44\xe
b\x2f\xe7\x41\x59\x12\x8f\xd2\x4d\x66\x72\xeb\x4b\xa1\x4f\x9f\xe6\x3f\x0d\xab\xb6\x8c\xdb\x4f\x
9f\x7e\x33\xc0\x58\x18\x1c\x8c\x27\xa6\xf1\x00\x46\xb7\x02\xf7\x53\xee\x27\x9b\xc2\x5c\x7d\xd8
\xed\xd0\x47\xdd\xbe\x7b\x4a\x13\xbf\xc0\x42\xab\xca\x35\x2e\x9c\x18\xe6\xfd\xbd\x3a\xac\xb0\x
da\xbc\x5c\x63\x72\xe7\xc6\xbd\x67\x17\xf6\x05\x87\x1e\x94\xd8\x97\x62\x7f\xc1\x25\xb0\x21\xa
3\x48\x5e\xe5\x70\x18\xe7\x4d\x53\x3a\x69\xc5\xbe\x29\xe3\xa2\xcb\x23\xc6\x60\xac\x66\xdf\x07
\x73\x91\xe5\x2b\x99\xd1\x56\x06\x86\xc5\x96\x91\x71\xaf\x06\xf8\xb1\x12\x60\x81\xc3\xa5\x69\x
aa\x84\xcb\x47\x56\x82\x76\xcc\x3c\xee\x2c\x2a\xc1\xbe\xf1\xa8\xf4\x3b\x37\x74\xdd\xc7\xfe\xdb
\x65\xe8\xda\x5c\x17\x6c\x4d\x7c\x3a\x77\x04\xf6\x89\xa3\x85\x23\x6c\x0f\xe6\xca\xaa\xe3\xfb\x
09\xab\x31\xdf\xb9\xe1\x2f\xdc\xe8\xc6\xaa\xfd\x7a\x4a\x0b\x0b\xda\x4b\xbc\x5a\xb3\x39\x0f\xdd\
x88\x82\x85\xb9\xbf\xf2\x4b\xa8\xf1\x07\xfc\x20\x6f\xa3\x27\x45\xcc\x21\xd1\x90\xdc\x20\x45\x32
\xf7\x73\x95\x1d\x54\xb2\xa2\x5c\x80\xb4\xb5\x92\xd7\x9c\x83\x37\xa4\x10\x68\xb5\xa6\x08\x89\
x57\x78\x0d\xd7\x28\x61\x85\xc9\xdd\x91\x12\x57\x9c\x22\x17\x09\x73\x2c\xdb\x90\x6a\xd3\x37\
xe7\x14\xdc\xc8\x4e\xef\x49\x7c\xc5\x44\x17\x4b\x72\xb8\xd6\xab\x27\x1d\xcc\xce\xa4\x13\xc4\x
aa\x43\x49\x6e\xa1\x94\x38\x25\xae\x9c\x12\x17\x4e\x09\xf9\x2e\x8e\x21\xc3\x57\x8c\xac\x6b\x0
e\x5f\x79\x0b\x77\xc1\x9d\xd9\x65\xc9\xb8\x42\xfd\x15\x4b\xa0\x7a\xac\x97\xbf\xf0\xa2\xc9\x6a\x
eb\x0c\xe0\x7a\xeb\x65\xea\x4f\x92\xed\xbe\x6a\xbb\x0f\x7e\x60\xad\xdd\xf9\x2a\xd7\x35\xc2\x0f
\x23\xdf\x2d\xc7\xda\x12\x4d\x43\x06\x38\x72\x77\x85\xa3\xf3\x41\x7a\xee\xbb\xfc\x5a\xa5\x15\x
d7\xf9\xf5\x2f\xa2\xa1\x55\x57\x4d\x65\x09\xde\xa5\x07\xc8\x41\xe8\x5d\xf5\xf1\x1e\x18\xa2\x55\
xf7\xaa\xfe\x67\xd8\x65\x35\x99\xe0\x4d\x53\x84\x17\x14\x03\x8d\xd0\xe5\x4d\xb3\x9e\x15\xa5\x
38\x4e\xf3\x59\xc5\x04\x57\x72\x32\x64\x02\x09\x9d\x7b\x37\x0d\xca\x8f\x5d\xe7\xd7\xcc\x92\x2
0\x78\x97\x61\xf9\x3d\x9a\xb3\xdf\x47\x37\xd6\x98\xfb\x2e\x6c\xa4\xb0\xad\x48\x2d\x76\xc6\xea\
x57\x5d\x8b\xb4\x6e\x19\x0c\x2b\x27\x6e\x2b\x45\x33\xd3\xac\xfa\x6c\xa8\x0a\x8c\x36\xaf\x98\x
71\x5d\x1e\xbc\x62\xc5\x68\xcc\xa1\x2b\x5a\x0e\x24\x6e\x7c\x3c\xc8\x4c\x53\xa5\x35\xe4\x5d\x3
0\xf2\x0e\x98\x3b\xd9\xf8\x0a\xbf\x39\x73\x79\xc5\x2a\xce\x21\x53\x56\xf2\x77\xf8\xda\x5b\xc9\x
be\x48\xfc\x9f\x9b\x35\x55\x15\xb7\xff\x2b\x33\x0d\xe3\xfd\xf6\x60\x35\xa7\x3c\x76\xa6\x5d\x7c\x
5b\x11\xff\x62\xe5\x88\x60\x2b\x28\x45\xc4\x3c\x92\x14\x6c\x18\xdd\x1d\x99\x01\x6e\x28\x55\x1
4\x49\x6a\x9d\xbc\xfd\x0c\x8d\xb3\xd9\x99\xe1\x2b\x57\x9c\xe8\xdb\xfb\x07\x2d\x92\xea\x0b\xd3\
xf1\xd3\xee\x13\xd3\xe8\x35\x4b\x59\x06\x39\x07\xb7\x11\xe0\xb9\x20\xb9\xff\x5b\x88\x64\x73\x
42\x7c\x12\x25\xaa\xcf\xef\x86\xd0\x62\x55\x17\xd1\xf5\x8b\xb6\xcc\x5e\xd4\x59\xdc\x66\x7b\xd4

\xf3\x3f\xbf\x0b\xd0\xf7\x0f\x57\xb3\xb4\x16\xe7\x09\x4d\xcf\x7f\xbf\x38\x7f\x24\x13\xae\x53\xdb\x1b\x51\xbb\xdd\xd0\xbf\xab\x3a\x25\x75\x3e\xdb\xd4\x4b\x54\x9b\x58\xd6\x6d\x7a\x6a\x8a\xd0\x6d\x1a\x81\x88\x59\x94\xf9\x99\xed\xdd\xa9\xaf\xd8\x54\x56\x68\x21\xf3\x40\x6e\x8a\x50\x72\xf5\x9d\x8a\x65\x18\x81\x0c\x8b\xd6\x03\xcd\x50\xa8\x6c\xa3\x65\x18\x50\xe1\x8d\xdd\x7f\xcb\x51\xd9\x76\x90\x51\xf4\x67\x65\x3c\xc8\x2d\xcc\x6e\xdb\x42\x90\x3b\x5f\x25\xe6\x77\xbf\x4a\x94\x3c\xe8\xdd\xc0\x7c\xf3\xbd\x9f\xe5\x35\x8d\xc7\x37\x88\xca\xfb\xb9\x41\xe1\xc4\x90\x53\x54\xa4\xbe\x29\x2a\x49\xa7\x3b\xa5\xaa\x9f\xa1\x18\x2f\x73\xc4\x56\x96\xea\x61\xa6\xc3\x34\x87\xca\x89\x29\x30\x37\xcd\x61\xae\x8a\xba\x9a\xa6\xbf\x0d\xab\xa2\x22\x72\x7d\xbb\xf4\x6b\xe5\xb8\x0c\xb1\x87\x51\x6b\x00\x6e\x58\x43\x81\x09\x62\x0a\x43\xd9\x34\xc3\x9c\xf7\x5e\xb1\xeb\x0f\xe5\xdf\x95\x2e\x6b\xb9\x73\xc5\x96\x84\x69\xd7\xae\x8b\x8b\x58\xd2\xa7\x5c\xf8\x0b\x96\xf6\x74\xe2\x51\xe2\x93\x33\xef\x06\x65\x58\x07\xb5\xce\x22\xcb\x49\x3d\x1d\x62\x3e\xa9\xfb\x60\x9e\x5a\x42\x6a\xe8\xa0\xf6\x9f\x49\x63\x1a\xb9\xfe\x66\xb9\x0d\x15\xf3\xbb\xb7\xb7\x4c\xe8\x8f\x7f\x42\x72\xa6\xab\x10\xb7\xaa\x7d\x6a\x62\x8c\xf6\xa3\xbf\x89\x2e\x8e\x1c\xa8\x52\xb8\xa9\x81\x78\xae\xde\x37\xe5\xe7\x3d\x8b\xea\xef\x91\xc4\xd6\xb9\x95\x0f\xbe\xff\x21\xf7\x46\x45\x5b\xb5\x2a\x94\xef\xbf\x77\xa2\xbd\xb6\xdf\x80\x6e\x38\x46\xda\x76\x90\x4f\xe4\x74\x17\xb3\xb6\x1e\x6c\x52\xa0\x3b\xb5\xf0\xbc\x4f\x03\x88\x2e\x30\x26\x42\xf1\xa0\x78\xd1\x4f\x2e\x2c\x8b\xe7\x93\x62\x1a\x56\xea\x6b\x5d\xad\x53\xf2\x49\x61\x79\x24\xce\xfa\x01\x5d\x0e\xfa\xc9\xa2\xae\xe9\xa8\x6a\x5c\x6a\x98\xee\x60\xd5\xeb\xcf\xed\xbb\x80\x7e\x67\xc9\xb6\x7e\x64\x9b\xaa\xa2\x48\x6c\x22\x75\xcb\x70\x0c\x4b\x6c\x5c\x62\xc1\x2d\xe6\x86\x59\x64\x90\xdf\x24\x2c\x83\x5b\xd9\x06\x60\x7a\x87\xc5\x75\xd9\x5a\xd6\xb9\xc5\x86\xeb\x18\x81\x65\x65\xe4\x04\xab\x6f\xd0\x04\x16\x96\xe8\x0b\x0c\xab\x8d\xc8\x5a\x56\x16\x56\x9b\x69\x06\x64\x36\x56\x81\x6d\x6f\x4d\xb5\xb0\xd0\x33\x2b\x65\x33\x36\x75\x65\xfa\x93\xf7\x2d\x9c\x33\xbe\x89\xd1\x36\x98\xc6\x1b\xe6\x18\x08\xbc\x63\x48\x81\x2c\xf4\x9c\x09\xee\xaf\x88\x0f\x68\x33\x33\x1d\xf7\xeb\x6a\x87\x4f\x73\x8b\x7d\x72\x3e\xcd\x77\x79\xd4\xd0\xaf\xc5\x99\x98\x58\xf6\x34\xa2\xc7\xe8\xc9\x88\xdc\x26\x65\x70\x63\x21\x53\x58\xe9\x67\x75\xd5\x0a\xd7\xd8\x56\xeb\x0e\x2e\xf3\x3c\x15\xb3\x6c\x90\x17\x83\x4b\x99\xcd\x8a\x9b\xc1\x9c\xc2\x4d\x03\xae\x50\x7f\x49\x25\xb3\xc5\x60\x95\xcf\x85\x01\x97\xdd\x87\xe9\x03\x62\xd4\xc1\x72\x56\x0e\x56\x79\x21\x06\xd5\x72\x96\x0d\xbc\xa7\x83\x52\x2e\x32\x99\xc8\x78\x96\x55\x1a\x48\x69\xc0\x39\x1a\xae\x37\xde\xdb\x7f\xfa\xec\xe0\xf9\xe1\xec\x32\x9e\x8b\x64\xb1\x94\x5f\xbe\xa6\xab\x2c\x5f\x7f\x2b\xca\xaa\xbe\xba\xfe\x7e\xf3\xe3\xe5\x6f\xaf\x5e\x1f\x1d\xbf\xf9\xd7\xc9\xef\xff\xdf\xe9\xdb\xb3\xf3\x77\xff\xff\xfb\x8b\x0f\x1f\xff\xf8\xf3\xaf\xff\xfa\xef\x27\x9f\x0d\x38\x43\x4f\x78\xfb\x70\x83\xde\x3e\x5c\xdc\x2f\xec\xf5\xe0\x3d\x4e\x3c\x32\x3f\x9e\xeb\x82\x27\xf6\xc0\x13\xfb\xe0\x89\xa7\xe0\x89\x67\xe0\x89\x03\xf0\xc4\x73\xf0\xc4\x21\x78\x82\x06\x09\xcf\xa3\x3f\x63\xfa\xb3\x37\x85\x97\xea\x43\x8e\x23\xf4\xc4\xa1\xfa\xa2\x4a\x55\x51\x1a\xdd\xf1\x6c\x8a\x9d\xe7\x22\x91\x99\x30\x4d\xfd\xeb\xcc\x56\x73\xae\x1f\xd9\x43\x53\x33\xbb\xdd\x7c\xb7\x69\xd4\x99\x1e\x37\xdf\x54\x7f\xab\x0b\x1b\x61\x9a\xfa\xd7\x21\x2f\xab\xa8\xf4\x05\xc0\xdd\x26\x9c\xc1\x70\xc9\xab\xe2\xe6\xe7\x12\x0b\xf1\xad\x96\x85\x60\x6d\x3d\xa8\xc1\x6f\xe3\x59\x15\x2f\xd9\x6b\xfe\xf3\x56\x73\xa0\x70\xfa\x2f\xcb\x70\x76\xdb\x66\x05\xfe\x63\x34\xfa\xcf\x41\x99\xd7\x45\x2c\xde\xce\xd6\x6b\x99\x2d\x3e\xbe\x3f\xc5\x79\x1e\xdf\xf9\xf7\x1a\xce\x6a\xb6\xfe\x8f\xff\x17\x00\x00\xff\xff\x2f\x88\x72\xca\xa2\x43\x00\x00")

func bignumberJsBytes() ([]byte, error) {
return bindataRead(

```
_bignumberJs,
"bignumber.js",
)
}

func bignumberJs() (*asset, error) {
bytes, err := bignumberJsBytes()
if err != nil {
return nil, err
}

info := bindataFileInfo{name: "bignumber.js", size: 0, mode: os.FileMode(0), modTime:
time.Unix(0, 0)}
a := &asset{bytes: bytes, info: info}
return a, nil
}

var _web3Js =
[]byte("\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\xff\xec\xbd\xf9\x7a\xdb\x38\xb2\x38\xfa\xbf\x9f\x02\x
d6\x3d\x37\x92\x62\x46\xf2\xd6\xe9\x34\xdd\xee\x8c\xb3\x74\xc7\x73\x92\x38\x5f\x12\x4f\xcf\x1c\
x8f\x4f\x3e\x4a\x84\x24\x74\x28\x52\x3f\x92\xf2\xd2\xb1\xdf\xe5\x3e\xcb\x7d\xb2\xdf\x87\xc2\xbe\
x70\x91\xed\xf4\x36\xf6\x1f\x89\x08\x14\xb6\x42\xa1\x50\x28\x14\xaa\x72\xfc\x7f\x96\x24\xc7\xfb\
xbd\xc9\x32\x1d\x97\x24\x4b\x11\xee\x95\x41\x1a\xe4\xfd\x2f\x32\xa5\xe8\x65\xc1\xb2\xff\x85\x4
c\x7a\xeb\xe9\x49\x76\xca\x7e\x95\xf0\xeb\x2c\xca\x51\xb4\x5f\x5e\x2e\x70\x36\x41\xa2\xae\xfd\
x8e\x28\xda\x79\xf0\x80\x27\xee\xd1\x32\xcb\x07\x0f\xa2\x7e\x8e\xcb\x65\x9e\xa2\xa8\x97\x05\x
eb\x9b\x7d\x9a\x4e\x44\x1a\xe1\x69\xb4\xd6\xc9\x7e\x8a\xcf\xd1\xcb\x3c\xcf\xf2\x5e\xe7\x79\x94
\xa6\x59\x89\x26\x24\x8d\xd1\x3c\x8b\x97\x09\x46\xdd\xce\x46\xb6\xd1\xe9\x76\xfa\x7b\xe5\x2c\
xcf\xce\xd1\x64\x30\xce\x62\xbc\xdf\x79\x73\xf4\xe2\xf8\xf5\xcb\x4f\x6f\x8f\x3e\x7e\xfa\xf1\xe8\xf
8\xed\x8b\x4e\x30\xb9\xa6\xf5\x25\xfb\xb4\xef\xfb\x5f\xf0\xc5\x22\xcb\xcb\x22\xfc\x72\x7d\xbd\x4
7\xc7\x70\xb2\x79\x3a\x18\x47\x49\xd2\x4b\x06\x3c\x2b\x10\xbd\xef\x61\x36\xc0\x74\x1f\x00\xb7\
x4e\x4f\xf0\xe9\x1e\xef\x6a\xd1\x4b\x9f\xa6\x21\xee\x5f\x07\x49\xa0\x4a\xe2\x80\xe1\xee\x9a\x4
3\xd1\x26\x45\x26\xf4\x82\xb4\xc2\xd5\x24\xcb\x7b\x14\x3a\xdb\xdf\xdc\xcb\xbe\xcf\x07\x09\x4e\
xa7\xe5\x6c\x2f\xdb\xd8\xe8\x17\xbd\x9c\x22\x5e\x76\xe3\xba\xdf\xfb\xb2\x15\x9e\xc8\x2e\xf3\x2
a\x02\x86\xa5\x80\xb7\xdd\xff\xb2\xc6\x12\x44\x67\xf6\x4f\xd6\x10\xfa\xb2\x86\x10\x42\x9d\x71\x
96\x16\x65\x94\x96\x9d\x10\x95\xf9\x12\x07\x2c\x95\xa4\x8b\x65\x59\x74\x42\x74\x02\xdf\x02\x1
a\xf2\xd2\x68\x8e\x3b\x21\xea\x7c\xca\xce\x53\x9c\x77\x02\x95\x43\x47\x47\x73\xa2\x38\xce\x71
\x51\x74\x78\xce\x35\xfc\x7f\xca\xab\x16\xc5\xe1\x7f\x9e\x96\x2d\xcb\xe6\xf6\xb2\x4f\x5a\x11\xa
3\xbd\xd1\x65\x89\x8b\x9d\x6d\x7f\x7b\x02\x48\x62\x7a\x0d\xa1\xeb\xe0\x4e\x10\x70\xa3\xfe\xc8
\xe1\x68\xd8\x6b\x87\x80\x95\x51\xfd\x47\x1d\xfa\x38\x4b\x4b\x9c\x96\xb7\x1e\xfc\x9f\x72\xde\xe
9\x8c\xfd\x61\xa6\x7d\x12\x25\xc5\x6f\x37\xf4\x1c\x17\x38\x3f\xf3\xad\xfa\x3f\xfa\xa4\x15\xcb\xd1
\x7b\x3c\x25\x45\x99\x47\xff\x01\x93\x17\xd4\xd5\x81\xcf\x8f\x6e\xc5\xf7\xcb\x3c\x4a\x8b\x89\x9
7\xf5\xfd\x59\x70\x90\x5b\xa4\xb0\x3a\x12\x0a\x5c\x7e\xa8\x27\xa9\x3b\xc3\x85\xdd\xf4\x6f\xd2\x
```

e8\x57\x9e\x80\xa8\x0d\xe2\xeb\x2a\x58\xe4\x64\x1e\xe5\x97\xde\x7e\x64\x59\xd2\x38\x79\x07\x
bc\xad\x3f\x2f\x0a\xcd\x3d\xb8\xb6\x9a\x2a\x24\x3c\xaf\xdc\xc6\xff\x48\x48\xf0\xf6\x3e\x26\x45\x
76\x9e\xde\xa2\xe7\x51\x9a\xa5\x97\xf3\x6c\x59\xac\xd0\x75\x92\xc6\xf8\x02\xc7\xc6\xde\x75\x6
7\x13\xab\x2a\xd7\xba\x63\xd6\x7e\x4e\xd2\xdb\x30\xee\x83\x25\x60\xe2\x65\x1a\xe3\xb8\x63\xa
1\x09\x9f\x51\x42\xf8\x0b\xe0\x68\x44\xe2\xb8\x1d\x8e\x6e\x56\xff\x59\x94\x2c\xbd\xdd\x5f\x92\x
b4\xdc\xfe\xe6\x71\xfd\x14\xbc\xc5\xe7\xcf\xc8\xef\x88\xfc\x5b\xad\xb9\xe7\xb3\x28\x9d\xfe\x9e\x
a4\x73\x27\x94\x53\x51\xb7\x26\xd5\xd7\x52\x8d\x17\x33\xef\xd8\x6e\xd4\x88\xa0\xb5\xd3\xb5\x
b5\xeb\xe0\xcb\xf5\x69\xb0\xfd\xbb\x1d\xfa\xff\x42\x67\xde\xdf\x49\x76\x9c\x2c\xd3\xf8\xc6\xa4\x
72\xeb\x8d\xeb\xfe\xd8\xfb\xe7\x3e\xf6\xde\x1f\xfa\xfe\xc8\x67\x0e\xef\xe0\xf9\x79\xe1\x8f\x26\x6
d\x7e\xdd\xcd\x5c\xed\x55\x3b\x77\xb6\x57\xad\x3a\xef\x93\x3c\x9b\xdf\x72\xda\xcb\xec\x96\x47\
xcd\xdb\x09\x7c\xbf\xef\xba\xf9\x23\xe0\x8f\xa4\x31\xc9\xf1\xb8\x3c\xf4\xee\x99\x2b\xf4\xe4\x76\
x13\x41\xc6\xd1\xe2\xe3\xef\x3a\x19\x7e\x4c\xb6\x3b\xed\xe2\x45\x56\x90\xba\x83\xfa\x22\xba\x
8c\x46\x09\x36\x85\x82\xdf\x85\x2b\x55\xd1\xdc\x9d\x1c\xbf\x6e\x47\x03\x07\x62\xbc\x2f\x4c\x7c
\xfe\xf6\x27\x99\x3b\x41\x52\x45\xdd\xed\xe8\xec\x77\x40\xff\x1f\x16\xeb\x77\x71\x7e\xbc\x31\x9f
\xfc\xda\x58\xb7\x99\xde\x3d\xda\x5b\xa2\xfd\xd6\x1b\xd7\xd7\x9e\xd9\x43\xcf\x96\x56\x27\xc7\x
ed\xb6\x91\xe3\xc0\x78\x03\xed\x0b\x0b\x87\x5e\x77\x30\x9c\x64\xf9\x3c\x2a\x4b\x9c\x17\xdd\xf
e\x1e\x00\x7c\xc8\x12\x12\x93\xf2\xf2\xe3\xe5\x02\x9b\xb0\xb4\x7d\x0a\xb5\x36\x7c\xf8\x70\x0d\
x3d\x34\x20\xb9\xce\x1d\x91\x02\x45\x68\x91\x67\x19\x05\x46\xe5\x2c\x2a\x51\x8e\x17\xf4\x90\x
95\x96\x05\xe2\x73\x87\x68\x26\xad\xe1\xb0\x44\xf3\xa8\x1c\xcf\x70\x11\xd2\x4f\x9e\xad\xfd\x3c\
x39\xd5\x3f\x76\x8d\xaf\x53\x33\x73\xc7\xfa\x3e\x3d\x79\x7c\x7a\x72\x1a\xa0\xc1\x60\xb0\x86\x1
e\x0e\x9d\xb1\x89\x1e\xef\x23\x69\x4d\xd3\xeb\xf3\x29\x2e\x67\xa4\x18\x7c\x82\x85\xf1\xa3\x40\
x10\x05\x1c\x30\x74\x1d\xd2\x8c\xc3\xb4\xdc\xd3\x80\xd9\xbe\xed\x83\x3e\x82\x1c\xde\xdc\xde\
xda\xf5\xde\xda\x9a\xa7\x1f\x83\x45\x9e\x95\x0c\x6b\xfb\x28\xc5\xe7\x46\x5f\x7b\x5f\xae\xfb\x7b\
xf5\xa5\x06\x20\xbd\xe4\xcb\x71\x99\xd1\xc6\x3d\xb0\x4d\xed\x0e\x48\xc1\xe7\x5c\x21\x84\x92\x
a3\x40\x0a\xb7\x6b\x59\x5f\xa7\x89\x03\x98\xb7\xde\x90\x63\xbb\xf7\xef\x93\xde\xc9\xe6\xa3\xef
\x4e\x1f\xf6\xff\x7d\xda\x7f\x3a\xec\xb3\x71\x9a\x07\x87\xca\x6e\x5d\x07\x5f\x3a\x3a\x29\x76\xc2
\xef\x82\x0e\xa3\xb7\x4e\xb8\xb5\x7b\x7d\x1a\x7c\xf3\x3b\x93\xf7\xb3\x2c\x4b\x1a\x68\x7b\x44\x
41\x2a\x08\x9b\xe6\x89\xff\x19\x95\xc2\xaf\x5d\xf5\xf3\x54\x4b\xde\xd1\x3f\x9a\xc8\x18\x7a\x76\x
53\x1a\xa6\x85\x57\x21\x62\x06\x6f\x53\x30\x4d\x5d\x91\x7c\xcd\x22\x35\xb4\xcb\x5a\xac\x2b\x7
b\x13\xaa\xfd\x5f\x8a\x5a\x93\x66\x1f\xfe\x57\x2b\xa2\xe5\xfd\x69\xa6\xd8\xc7\xbf\x37\xc5\xd2\x3
d\x4c\x92\x6c\xe9\xa7\xd9\x72\x86\x11\x6c\x76\x40\xb8\x03\x1f\xe5\xd2\x5c\xf9\x83\xd3\x25\xfc\
xdc\xd5\x7e\x9f\xea\x19\x3b\xc6\x97\x49\xbf\x88\x6f\xad\xf2\xe7\x13\xa3\x1e\x5e\xd4\x43\xe5\xd
0\xc9\x1b\x93\x39\x2d\xbd\x12\x9d\xb3\x02\x0e\xa1\xd3\xe4\x55\x29\xdd\x2c\x53\x47\xea\xac\xd
1\xda\xd2\x37\x23\x76\x5a\x09\x23\xf5\x2f\x5b\xc1\x75\xff\x66\x84\xcf\x7b\xd7\x4c\xf9\xdf\xb6\xa
1\xfc\xe1\x43\xe8\xf0\xc7\x19\x29\xd0\x84\x24\x98\x52\xea\x22\xca\x4b\x94\x4d\xd0\x39\x1e\xed\
x0c\x7e\x29\x06\x6b\x00\xc2\xbf\x28\xc0\x24\xc7\x18\x15\xd9\xa4\x3c\x8f\x72\x1c\xa2\xcb\x6c\x8
9\xc6\x51\x8a\x72\x1c\x93\xa2\xcc\xc9\x68\x59\x62\x44\x4a\x14\xa5\xf1\x30\xcb\xd1\x3c\x8b\xc9\
xe4\x12\xea\x20\x25\x5a\xa6\x31\xce\x81\xe0\x4b\x9c\xcf\x0b\xda\x0e\xfd\xf8\xe9\xed\x31\x7a\x8
d\x8b\x02\xe7\xe8\x27\x9c\xe2\x3c\x4a\xd0\xbb\xe5\x28\x21\x63\xf4\x9a\x8c\x71\x5a\x60\x14\x15
\x68\x41\x53\x8a\x19\x8e\xd1\xe8\x92\x53\x11\x46\x3f\xd2\xce\x7c\xe0\x9d\x41\x3f\x66\xcb\x34\x8
e\xe8\x98\x03\x84\x49\x39\xc3\x39\x3a\xc3\x79\x41\x67\x68\x47\xb4\xc5\x6b\x0c\x50\x96\x43\x
2d\xbd\xa8\xa4\x63\xc8\x51\xb6\xa0\x05\xfb\x28\x4a\x2f\x51\x12\x95\xaa\xac\x8b\x02\x35\xd2\x1

8\x91\x14\xaa\x9d\x65\x62\x65\x93\x12\x9d\x93\x24\x41\x23\x8c\x96\x05\x9e\x2c\x13\x26\x38\x8e\x96\x25\xfa\xf9\xf0\xe3\xab\xa3\xe3\x8f\xe8\xe0\xed\xbf\xd0\xcf\x07\xef\xdf\x1f\xbc\xfd\xf8\xaf\x3d\x74\x4e\xca\x59\xb6\x2c\x11\x95\x28\xa1\x2e\x32\x5f\x24\x04\xc7\xe8\x3c\xca\xf3\x28\x2d\x2f\x51\x36\x81\x2a\xde\xbc\x7c\xff\xfc\xd5\xc1\xdb\x8f\x07\xcf\x0e\x5f\x1f\x7e\xfc\x17\xca\x72\xf4\xe3\xe1\xc7\xb7\x2f\x3f\x7c\x40\x3f\x1e\xbd\x47\x07\xe8\xdd\xc1\xfb\x8f\x87\xcf\x8f\x5f\x1f\xbc\x47\xef\x8e\xdf\xbf\x3b\xfa\xf0\x72\x80\xd0\x07\x4c\x3b\x86\xa1\x86\x66\x44\x4f\x60\xce\x72\x8c\x62\x5c\x46\x24\x11\xf3\xff\xaf\x6c\x89\x8a\x59\xb6\x4c\x62\x34\x8b\xce\x30\xca\xf1\x18\x93\x33\x1c\xa3\x08\x8d\xb3\xc5\x65\xeb\x89\x84\xca\xa2\x24\x4b\xa7\x30\x6c\x49\x65\x08\x1d\x4e\x50\x9a\x95\x01\x2a\x30\x46\xdf\xcf\xca\x72\x11\x0e\x87\xe7\xe7\xe7\x83\x69\xba\x1c\x64\xf9\x74\x98\xb0\x0a\x8a\xe1\x0f\x83\xb5\x87\x43\xc1\x6c\xff\x06\x64\x3b\xce\x62\x9c\x0f\x7e\x01\x16\xf9\xb7\x68\x59\xce\xb2\x1c\xbd\x89\x72\xfc\x19\xfd\x77\x56\xe2\x73\x32\xfe\x15\x7d\x3f\xa7\xdf\x7f\xc3\xe5\x2c\xc6\x67\x83\x71\x36\xff\x01\x80\xe3\xa8\xc4\x68\x7b\x73\xeb\x1b\x60\x78\xcd\x5b\x41\x8d\x00\xab\x95\xe1\xf2\x98\x6f\xef\xe0\x92\x82\x06\x4c\x77\x41\x1f\xe4\x61\x5a\x9a\x80\x24\x2d\x7d\x70\xc7\x0e\xe0\xb2\x02\xf2\xc5\x65\x1a\xcd\xc9\x58\xb0\x71\xad\x44\xcc\x72\x80\x47\xf9\x4a\x7e\x28\x73\x92\x4e\xcd\x32\x05\xa4\xf9\xa0\xdf\xe3\xc8\x1a\x63\x8e\x23\xef\x18\x8f\x5d\xd0\x65\x15\xac\xa7\xdb\xb2\xbf\x00\x4c\x0a\x3e\x40\x83\x33\x17\x5a\x15\x01\xec\xb0\x9c\x4f\x0b\x0b\x71\x2d\x7f\x20\xab\x80\x6d\x84\x01\x5f\x5d\xc9\xd3\x23\xaa\x80\x3e\xc8\xf3\xe8\x92\x81\x33\x26\x6e\x89\x02\xcf\x29\x7d\x6a\x12\x00\x5f\x49\x8c\x43\xc4\xa8\xcc\x10\x4e\x29\x0d\x0f\x63\x4c\xff\x93\xad\x50\x66\x1c\x31\x36\x49\xb9\x12\x97\x6b\xcd\x8d\x99\xd5\xad\x8f\x98\x82\x15\xe6\xce\x0c\x49\x68\x1f\x6a\x28\x8c\x2e\x02\xef\x9f\xe3\x72\x96\xc5\x9e\x6e\x31\xe5\x7a\x96\xcf\x11\x93\x5c\x32\x63\x46\xd6\x10\x5b\x83\xbc\xf8\x27\x3e\x33\x3c\x0b\xfd\x0d\x7a\x8f\xbe\x30\xe2\xb9\x96\x62\xf9\xdf\x18\xe6\x0b\xf4\x45\xaf\xec\x1a\xb2\xe0\xad\x42\x81\xbe\xc0\xbb\x86\x6b\xc4\x3f\x09\xe5\x0d\x4c\x22\xa2\x64\x08\x7d\xa1\x3b\x11\x65\xf7\x80\x10\x03\x19\xda\x4e\xad\x77\xc9\xc1\x91\x40\x11\xc5\x66\x61\x8a\x77\x1a\xd6\x06\x13\x92\x94\x38\xef\x69\x65\xfb\x9a\x0e\x82\x53\x51\xc9\x85\x02\x41\x04\xa0\x53\xe8\x9f\x6c\x9e\xee\x31\xfe\x49\x26\xa8\xb7\xae\x37\xa2\xd7\xc1\x1e\x68\xb0\xa7\x1c\x5d\x92\x9e\x45\x09\x89\x15\x0d\xd0\x1a\xd7\x43\xd4\x45\x1b\x48\xaf\x7c\x4d\x97\x35\xf4\x9a\x4d\x0a\xac\xa0\x34\xb4\x48\x22\x92\x32\xfa\xb2\xa6\x91\x01\xbc\xe3\x39\xd5\xb3\xc8\xd3\x8f\x46\xbf\xe0\x71\x79\x6d\x55\x28\x26\x59\x95\x63\xd5\xc6\x16\x5c\xf5\xd4\x69\xdd\x70\x66\x2e\x60\xe5\x2d\x81\x0b\x26\x4d\x2b\x56\xf4\x4e\x28\xf0\x69\x80\x4e\x00\xfc\xb4\xdf\x0e\x35\x09\x29\x40\x02\x62\x8b\xaf\x1a\x3b\x85\x8e\x06\x60\x01\x0c\x3b\xbe\xf4\x85\x2a\x50\x85\x18\xa7\xd9\x56\xb8\x29\xdc\xa5\xcf\xb1\x53\x54\xd1\x77\x21\x08\x7c\x8a\x4b\x7d\x05\x16\x9c\x73\x70\x92\xa5\xc5\x78\xdf\x68\x09\xa3\x86\xc1\x3c\x5a\xf4\xaa\x78\x2c\x68\xe5\x3c\x6b\xc4\xe0\x9d\xac\xe6\x1e\xeb\xe9\x09\x14\x39\x65\xec\x59\x7c\xc9\x55\xa4\xf5\x87\xef\x53\x47\x93\x49\x81\x4b\xa7\x53\x39\x8e\x97\x63\xac\xf5\x2b\x1a\x8f\x03\xd4\xd0\x39\xc0\x4e\x19\x95\x64\xfc\x2e\xca\xcb\xd7\xf0\x92\xc8\xaa\x79\x60\xe7\xf7\x3c\xfd\x14\x75\xe5\x94\x29\xe1\xf8\x83\x5b\xe5\x9b\xa8\x9c\x0d\x26\x49\x96\xe5\xbd\x9e\xd3\xe2\x06\xda\xd9\xea\xa3\x21\xda\xd9\xee\xa3\x87\x68\x67\x9b\x0f\x5a\x43\x5f\x34\x1e\xa3\x0d\xd4\x93\x9b\x8e\x81\xf5\x0a\x14\xa2\xa7\xda\xde\x85\xd0\xce\x36\x0a\x8d\x84\x8a\xce\x0a\xd4\x07\x68\x53\xc7\x7e\x8e\x8b\x65\x52\x0a\xea\x61\x33\xf8\x66\x99\x94\xe4\x67\x52\xce\xd8\x9c\x08\x0a\x34\xfa\x16\x48\x3a\x0a\xcc\x19\x14\x95\xf3\x11\xb2\xfa\xcd\x13\x9f\x9f\xf4\xad\x56\x7d\x6b\xa0\x65\x0f\xb4\x35\x22\x87\xd7\xe9\xec\xa9\x85\x83\x93\x09\x1f\x31\xef\x2c\xdf\x15\xb2\xfc\x65\x34\x9e\xf5\x6c\xc6\x44\x74\xda\xa2\x5c\xbf\x72\xbe\xd4\x5c\x9d\xf6\xf5\x42\x0c\x21\xd0\x95\x0d\x57\xdb\xd9\x33\xbb\x2f\xd6\x9

1\x46\x84\x72\xed\x52\x2a\xc6\xc9\x84\x83\xd8\x73\x04\x1d\x70\xbb\x24\xf0\x04\x1f\xf6\x64\xe9\x4d\x98\x4b\x71\x63\x1f\x61\xfe\x0c\x0f\x0d\xd1\xb6\x02\xbd\x46\x38\x29\xb0\x35\xbc\xe1\x10\xc5\x59\xda\x2d\x51\x14\xc7\x88\x97\x2a\x33\xb3\xca\x01\x22\x65\xb7\x40\x51\x92\xe3\x28\xbe\x44\xe3\x6c\x99\x96\x38\xae\xc0\xd2\x57\x1a\xe7\xb5\x5a\x84\xc3\x21\xfa\x78\xf4\xe2\x28\x44\x13\x32\x5d\xe6\x18\xd1\x03\x5b\x8a\x0b\x7a\x02\xa4\xa7\xb4\xcb\xc2\x64\x56\xbf\x05\x91\xfc\x71\x26\xd9\x9c\x0c\xac\x23\x50\x60\xa5\x62\x99\x4b\xb4\xe6\x78\x12\x81\x3a\xe6\x7c\x96\x25\x98\xf5\x90\xa4\xd3\xf5\x06\x46\x50\xc3\x03\x6c\xce\xcf\x07\x1d\xa0\xcc\x59\xf9\xc6\x22\x17\x73\xd2\x28\xea\x7b\xb6\xb8\x9e\xab\x1a\xd3\x08\x88\x35\x8c\xce\x23\x45\xd6\x05\x2e\x9d\x39\x65\x64\xf5\x36\x9a\x63\x7b\x1f\x52\x39\xba\x9c\xe9\x96\xf5\x6c\x3e\xf5\xfb\x99\xaa\xd8\x53\xa7\xe4\x8b\x1c\x83\x4a\xaa\x15\x7f\x35\xc3\x16\x95\x2c\x72\x7c\x46\xb2\x65\x21\x3b\xb4\xbd\x47\x51\x42\x52\x44\xd2\xd2\x29\xd1\x84\x7f\xad\xbf\xbe\x06\xe9\xdf\x24\xcb\x11\x3c\x12\x26\x68\x1f\x6d\xed\x21\x82\xbe\x17\x03\x10\xef\x85\x11\xd9\xd8\xa8\x2a\x4e\xff\xac\x3e\x6f\xec\xa3\x8d\x9e\xc0\x01\x41\x8f\xd0\xd6\x29\x95\xf0\xd1\xd5\x15\xda\xdc\xab\xac\xa4\x86\x95\x73\x7a\xd8\x40\x04\x3d\xac\x9a\xb9\x0d\xbb\x17\x54\x38\xa8\x62\xfb\xe2\xef\xda\x49\x35\x53\xae\xfb\xbd\xbe\x35\x85\xc3\x21\x9a\x90\xbc\x28\x11\x4e\xf0\x1c\xa7\x25\x3d\x5f\x31\x34\x05\xa8\xf8\x4c\x16\x88\x94\xab\x4c\xb9\x81\xfd\x4d\x1f\xf6\x29\xfe\x6a\x67\x00\x9e\xce\xc7\x31\xa1\x8d\x44\x89\x5c\xe4\x1c\x9f\x0e\xff\x71\xf1\xed\xe7\x8b\x8a\x74\x2a\x18\xc4\x09\x41\x1b\x68\xeb\x54\xf0\x09\xb4\x81\x9c\x6e\x78\xd0\xde\x88\x60\x8b\xf9\x79\x20\xf9\x56\xe9\xa1\x7d\x46\x15\x37\x66\x3d\x7f\x68\xa6\x42\x85\x2d\x13\x53\xb7\x5c\xfc\x0d\x94\x89\xaa\x18\xd2\x66\x1d\x43\x42\xad\x68\xba\x91\xa3\x0c\x87\x68\x1c\x25\xe3\x65\x12\x95\x58\x08\x3e\xf4\xc8\xc7\xfb\x82\x48\x89\xe7\xb7\x60\x47\x94\x15\x9d\xfc\x89\x98\x52\xdf\x86\xbd\x5e\x69\x5f\xb9\xe5\x84\xfc\x7e\x0c\x46\x67\x2e\x5f\x9d\xb7\x20\x47\x5b\xc4\xfb\xd1\xa0\x0d\xe1\xba\x48\x7e\x33\x99\xd5\x68\x8c\x18\x64\x6b\x8d\x91\x48\x97\xb7\x9a\x52\x25\xe2\xd7\x25\x55\xeb\x41\xb4\x86\x3d\xe2\x1f\xd4\xef\xd3\x11\x69\xc5\x94\x8e\x88\x41\x83\x6c\xd3\x06\x2d\xb5\x4a\xa2\x0a\x84\x54\xe9\x88\xaa\x11\xc2\x4b\xc0\x09\x03\x5a\x53\x88\xa9\xd7\x10\xe9\x43\xf4\x9d\x8e\x0d\xdc\xac\xae\x20\x12\xa5\x18\x15\xeb\xf0\x8c\x88\x0b\xef\x29\xdc\x3a\xee\xdf\xb1\x46\x89\x0d\xb9\x07\x23\x13\xeb\x4b\xa9\x45\x0c\xbd\x88\xa8\x51\x69\x98\xea\x54\x0e\x6a\x54\x8d\x7a\x06\x1d\xa3\x8c\x03\xd1\x32\x77\x3d\xd2\x36\xea\x28\x79\x12\xf5\xc9\xc1\xbc\x6b\x95\x4c\x72\x38\x44\xc5\x72\xce\x6e\xe8\x3c\xbb\x14\x17\x11\x25\x3c\xaf\xee\x84\x9c\x52\xae\x28\xbf\x60\x4b\xf2\xf1\x1f\xd1\xbc\x89\x08\x21\x6d\x3a\x28\x18\x0e\x51\x8e\xe7\xd9\x19\x5c\x63\xa2\xf1\x32\xcf\xa9\x7c\x2a\x85\xd3\x0c\x92\x79\x37\x49\x01\x3d\xf7\xf4\xb6\x58\x45\xe3\x27\x90\xd9\x5a\xf3\x67\x8c\x0c\x3d\x72\xea\x6f\x4d\x69\x1f\xac\x75\x58\x71\xad\xe3\x3d\xb5\x0a\x1e\xe7\xa1\xb2\xd2\xba\x72\x10\x64\x45\x77\x30\xfd\x92\xc4\xbc\xbf\x60\xbd\xa5\x6d\x8d\xf9\x2d\x93\x6e\x6a\x01\xbd\xef\x31\x7b\x55\xdb\x04\x83\x5f\x8b\xf6\xfa\x81\x37\xfb\x59\x96\x25\x55\x79\x54\x08\xa9\xc8\x3a\xae\xc9\xd3\x2f\x37\x2b\x9b\xad\xcb\x64\x5c\xb8\x2a\xf7\x3d\x8e\x2a\x7b\x7c\xcc\x32\xd7\x28\x41\xb8\xf6\x1b\x80\x3a\x69\xb3\x21\x0c\x67\xc3\xdd\xa0\xc3\xee\x7e\x3b\xe1\x37\xf0\x93\xf6\xad\x13\x3e\xa6\xbf\xf5\xeb\xd8\x4e\xf8\x24\xf0\xd9\x7a\x90\xb4\xec\x84\x5b\x9b\xf4\x67\x8e\xa3\xa4\x13\x6e\x6d\xd3\xdf\xec\x56\xb6\x13\x6e\xed\xd0\xaf\x25\x83\x82\x06\x96\x1c\xec\xf1\xf5\x69\xf0\xe4\xb7\xb4\x8b\x6a\xb8\x86\xbe\x99\x35\x91\x5e\xc9\x2a\x46\x45\x66\x39\xdb\xb6\x48\xcf\x5d\xd1\xc4\xc8\x5f\xb4\xc6\xd2\xc8\xec\x49\x9b\xba\x6e\x61\x77\x54\x61\x6c\xd4\xaa\x51\xed\x4a\xdc\x3b\x5d\x82\xed\xe4\x4b\xdc\xc2\x84\xc9\x1a\x76\xb3\x25\xd3\x77\xf7\x96\x4c\xf7\x96\x4c\xff\x29\x96\x4c\x6a\x21\xdc\x95\x39\xd3\x33\x32\x7d\xbb\x9c\x8f\x80\x15\x4a\xee\x3c\x22\xd3\x14\x12\x07\xbf\x48\x4e\xbe\x2c\x49\x62\xda\xd7\x

0c\x86\x90\xc6\xfe\x15\x60\x63\x2f\xc8\x38\x4b\x27\xc4\x31\x06\x12\x27\x33\x6d\x57\x80\xb3\x0
b\x6c\x0b\x62\xe0\x8c\x57\x17\x08\xf8\x3d\x82\x07\x1b\xf4\x9c\x45\xf9\x96\xb2\x92\x85\xa5\x40\
xe7\x06\x94\x33\x0f\x29\x8e\x19\x24\x29\x50\x8a\xa7\x51\x49\xce\x70\x20\x38\x11\x5c\x1c\x95\x
e7\x59\xb7\x40\xe3\x6c\xbe\x10\xd2\x2a\x94\xa2\x73\x2b\x4b\x4e\x92\x2c\x2a\x49\x3a\x45\x8b\x
8c\xa4\x65\xc0\xae\x43\x29\xd9\xc7\xd9\x79\x6a\x9d\xe9\x4c\x35\x89\x7b\x7c\xbb\x62\x58\xbe\x
92\xf8\xbe\x16\x63\xa1\x4b\x29\xc5\x38\x86\x53\xf4\x48\xcd\x71\xec\x37\x86\x01\xa4\x5d\x4b\x3
b\x1f\xb3\x5d\x83\x01\x43\xfd\x82\x0b\xcb\x76\x07\x6c\x2e\x7a\xe3\xc1\xcb\x8f\xaf\x3e\x3d\x3b\x
fc\xe9\xed\xf1\x9b\x67\x2f\xdf\x7f\x7a\x7f\x74\xfc\xf6\xc5\xe1\xdb\x9f\x3e\xbd\x39\x7a\xf1\x52\x3b
\xc3\x49\x4d\x1c\xcc\xe4\x60\x11\xc5\xaf\xf1\xa4\xec\xb1\xaf\x32\xfb\x78\x9e\x15\xcf\x25\x16\x79
\x9b\x83\x32\xe3\xe2\xd2\xd6\xe3\x7e\x80\x1e\xef\x9a\x37\x3c\xfa\x6e\x09\xc3\xe9\xb1\x46\x4c\x
03\x0c\x73\xe2\xc5\xe1\xb7\x02\xe7\xcf\xe4\xd9\xd8\x3c\x34\xaf\x8a\x43\x57\xea\x30\xb0\xe8\x4
1\x48\x99\xbd\xc2\x17\x62\xdc\xc5\x72\x54\x94\x79\x6f\x5b\xc3\x5f\x62\x5d\xed\xb3\xe2\x42\xcb\
xbd\x81\x1e\xef\xf4\xd1\x50\x47\x91\x8d\xee\xf7\x64\x3a\x2b\x79\xb1\x00\x25\xe8\xe1\x57\xc6\x
27\xdf\x81\xef\x14\xad\x95\x32\xdd\xad\xb1\x2b\x8e\x67\x26\x5a\xa5\x76\xee\x77\x9b\x01\x4b\x6
d\xca\x1a\xeb\x0f\xd8\x9a\xdf\x40\xcd\x13\xd4\xc4\xe9\x98\x24\x5f\xbd\x22\x3e\x88\xfc\xdb\xce\x
9d\x34\xee\x6c\x3f\x6b\x93\x3c\x9b\x1f\x97\x93\x27\xf7\x13\xe7\x99\x38\xfe\xce\xa8\x8a\x91\xf1\
x57\x48\x62\xd2\xe8\x37\x8e\xd2\xd5\x19\x99\xfd\xe4\xa8\x7a\xce\xba\x9b\xb7\xfb\xeb\xa2\x0d\x
5e\x3d\x7a\x8a\x50\x77\xab\x8b\x42\xd4\xdd\xec\xde\x9e\x47\x35\x61\x92\x9e\x58\x69\xa9\x7f\x
50\xb8\x02\x51\xc1\x78\xbe\x4c\x4a\xc2\x84\xca\xd1\x25\xda\xfe\xdf\x39\x15\xcf\xa5\x0d\x5d\x44
\x6b\x2e\xf1\x14\xe7\x35\x5b\xc9\x7b\x5e\x6b\xd3\xfe\xbd\xea\x8c\x70\x5b\xe6\x8a\x19\xe1\x68\x
b2\xa8\x8f\x62\x4d\xb6\x28\x37\x57\x32\xc7\x85\x95\xb5\xdd\x1f\x2c\xb2\xf3\xde\xd6\xf6\x93\x7e
\xdf\x44\xe9\xf3\x19\x1e\x7f\x46\x64\x62\xe0\x54\x13\x8b\x2c\x44\x14\x64\x9a\xe2\xf8\xb0\x78\x
ab\xb2\x1d\x45\xb4\xac\x63\x86\x2f\x78\x8f\x4d\x64\x08\xa2\x85\x43\x1f\xb4\x5d\x9a\x92\x58\x4
6\x8f\x2c\xe7\x84\x8a\xe1\x51\x52\x28\xab\x65\xbb\xf5\x46\x7c\xf9\x30\x24\xd8\xcd\x66\x80\xb6\
xfa\x01\xda\x7a\xac\xc9\x23\xdb\x7d\x23\xb7\x8f\xf6\xf7\xf7\x29\xc9\x7a\xa9\x30\xa7\xec\xe3\x51
\x94\x40\xa7\x10\x53\x1d\xa8\x0b\x0f\x26\x6a\xba\x44\xc4\x14\x09\xb6\x10\x68\x90\x87\x63\x07\
x4b\x71\xa6\x04\xc3\x9a\x76\xa5\x70\x08\xcb\x82\x4c\x11\x93\xd3\x2d\x7a\x93\x5d\x30\xf0\x67\x
18\xc5\x52\x60\x36\x8f\xfb\xac\x37\x9a\x2e\xb3\xd7\x47\x57\x57\xa8\xb3\xd9\xe1\x3a\xe2\xe1\x1
0\x8d\x25\x15\x51\xe1\x59\x4c\xa4\x6c\x9d\x01\x91\x92\x4d\xb4\x94\xb4\x5d\x21\x5b\xdc\xdf\x5a
\xf3\xcc\xe7\xd6\xa3\x82\xf4\xcc\x2f\x9b\xd2\x39\x49\x97\xf6\x2a\xe8\x4e\x6e\xf9\xd7\x85\xba\x4
5\xe5\x5b\xf2\x7a\xac\x45\x87\x6e\x40\x41\xcb\x7a\x12\x3a\xae\xa5\x21\x1f\xf5\xe0\x95\xc8\x87\
x37\xef\x12\xce\xf1\x5d\x50\xce\xd7\x41\x19\x67\xf9\x55\x28\x73\x78\x77\x23\xca\x00\x63\x9a\x4
8\x6c\xa2\x88\x37\xe7\xa2\xc8\x61\xe6\x3e\x8b\x73\x6b\x31\x72\x98\x41\x4c\xce\x48\x8c\xe3\x6
7\x97\x35\x3c\xfc\x26\xd4\xd4\x80\x9b\xe3\xbb\x46\xce\xb2\x12\x3b\xc7\x2b\xa3\xe7\xf8\x36\xf8\
x71\x6f\x61\x59\xd5\x12\x45\x55\x12\x97\x7a\x30\xdd\x1a\x2f\x62\x67\x33\xe7\xa2\x12\x47\xbc\x
69\x17\x45\x8e\x7c\xe6\xc3\x90\x67\x79\xc1\x7e\x75\x4b\x81\x6d\xab\x8b\x9e\xb2\xad\x99\x7b\x
c6\x58\x0d\x9b\x95\x27\x47\xed\x5d\x6e\xcd\xde\x97\xe0\x89\x42\x1c\x95\x20\x6a\xce\x36\x8e\x
e8\x91\x46\x73\xcc\x1e\xf8\xd0\x5f\x96\x08\xc6\x61\x68\x9d\xb2\x06\x0f\xe6\x9d\x43\x28\xb4\x11
\x20\x5d\x59\x4e\x0b\xf1\x27\xd6\x68\x1f\x55\xbd\xd4\x7d\xd8\x1f\x6a\x47\x9a\x82\xfc\xca\x79\x6
2\x01\xb7\x54\xbc\xfc\xc9\xd6\xa9\x29\x0a\x77\x37\x2f\xa8\xc8\xec\x4e\xee\xa0\x48\xc8\x18\x53\
xc9\x64\x1b\x3d\x84\xea\x56\xa4\xf3\x86\x99\xd1\x4f\xe1\x77\x36\x41\xab\xa2\xbf\x52\x15\xe0\x
6c\x32\xf2\x88\x68\xf1\x01\x86\x38\x7e\x09\x66\x63\xee\xf1\x6e\x9f\xef\xe1\x65\xc6\xe1\xfb\xe8\x

a1\x38\x55\xfa\x66\xc0\xaa\x88\x49\x87\x8f\x77\x03\xde\xfe\x6a\x53\x50\x73\x2a\x67\xc3\xf7\x1c\xcb\xef\x14\xfb\x51\x31\x26\xa4\x0e\xff\x9e\xe3\xfc\x6f\x88\x79\xa1\xd5\x01\xed\x40\x3b\xfc\xaf\x36\x01\xca\x3d\x4d\xd5\x0c\x1c\x28\x07\x36\x15\x53\x50\xc9\xdb\x2b\x50\x2e\x2b\x74\xb1\xed\x73\x60\xb3\x82\x34\x65\xe0\xae\xb3\x79\xd1\x41\x1b\x88\x9f\x71\x00\xed\xec\xb7\x34\x2b\xd8\xdd\x0c\x90\x9e\x54\xe5\x33\xe0\x8b\x30\xfd\xd0\xce\x9a\xa1\xf5\x1d\xd8\x30\xb0\x62\x43\x27\xc5\x81\xd3\x17\x78\x58\x95\xe1\x94\x62\xc8\x0c\xdd\x24\xb7\x1f\x59\x96\x84\x76\x82\x03\x45\x25\x90\xd0\x4e\xd0\xa1\xa4\x58\x16\xda\x09\x2e\xd4\xb1\x03\x76\xec\x85\xd3\x1b\x55\x29\x9e\xfa\x5c\xc0\x63\x3f\xa4\x3e\x58\x95\xe2\x81\xd3\xb1\xad\x25\xb9\x90\xbe\xe9\x71\x73\xdc\x72\xe6\x04\xe9\x69\x2e\x2c\xa7\xfa\xd0\xbb\xee\xae\xc5\xb5\xae\x79\x39\xd4\x09\xb7\x9e\x04\x1d\xf3\x52\xa9\x13\x6e\x83\x05\x03\x2c\x8c\x4e\xb8\xb5\x15\x74\xf4\xab\xa9\x4e\x68\x7e\x5e\x9f\x06\x5b\x9b\xbf\xb3\x4b\x97\x43\x66\x1b\x5f\xe3\x83\x88\xa4\x65\x95\x0b\x22\x7e\x7b\x45\xd2\x92\x79\x67\xa1\x3f\x76\xe5\xaf\x53\x95\xb8\xa3\xfd\xb6\x9c\xb7\x90\xb4\x64\xae\x5b\x48\x5a\x3e\xde\x95\x60\x4f\x54\x45\xdb\xdf\x3c\xae\xa8\x8b\xc2\x37\xb8\x32\xb2\x8f\x86\x5f\xd1\x1b\x17\x80\xdb\x66\x08\x87\x69\xb9\xa2\xe5\x85\x51\xa2\xc6\xe0\x02\x9a\xab\x29\x79\x23\xf3\x0a\x92\x96\x42\x54\x7c\x7a\x23\x97\x2e\xac\x57\xcd\x66\x10\x5b\xad\xa2\xd8\xdd\xdb\x41\xdc\xdb\x41\xfc\x79\xed\x20\x90\x32\x84\x60\xa2\xd2\x1d\xd9\x40\xb4\x30\x6d\xb0\x59\x3d\x33\x5d\xc8\xc0\x20\x5d\x79\xee\x18\x78\x24\xd4\xf3\x19\x4e\xe5\x7b\xc5\x80\xd9\x7e\x53\x01\x5c\x3a\x70\x10\x92\xe5\xd0\x6b\x1b\x61\xa9\xbf\xed\xe7\x89\xc0\x49\x85\xfc\xc8\xfe\xbf\xba\x42\xdd\xae\xc6\x67\x33\xf1\x72\x81\xfd\xd8\xd3\x9e\x1a\x92\x94\xb7\xde\xda\xe3\xc7\x14\x97\xba\xc9\x2f\x18\x90\x77\x0b\xf1\x10\x14\x78\x09\xad\xc4\xb0\x76\x57\xf2\x3d\x33\x76\x35\xa5\x68\xa1\x66\x52\xb5\xea\x95\xa1\x9e\xe8\x63\xdf\x30\x68\x07\xf4\xe8\x06\xed\x76\x23\xb5\xa6\x68\x60\xe5\x6f\x1c\x3b\xf4\xeb\xc7\xd6\xc8\x18\xe7\x98\x12\x93\x58\x0f\xa6\x5b\x16\x46\xee\x31\x99\x4c\x30\x18\x24\x33\x94\x5b\xe7\x92\x73\xf9\x2e\x44\x3f\x8e\x08\x94\xf0\x59\x12\xb6\xcb\xa9\xf7\x10\x62\x1e\x5d\xe8\x76\xe8\xeb\x47\xb4\x60\x1c\x46\xf6\xa2\x1a\x95\xe7\xfe\x37\xb3\x26\xdd\x55\xde\xea\x29\x82\x94\xa4\xba\x0a\x46\xb3\xf9\x88\xa4\xae\x87\x9b\x32\x9b\x62\xca\xdd\x69\x0d\x78\x3a\x60\x8b\x2a\x5a\x2c\x70\x0a\x6b\x29\x4a\xd9\x1b\x08\x0b\xbb\xbc\xb6\xa6\x7b\x18\xce\x98\x66\x64\x4c\xd9\x93\xe8\x55\x73\x61\x7e\x81\x9a\x4d\x38\x2c\xec\x43\xb5\xa8\x15\xc3\x6b\xd2\xfb\xd5\xa1\x55\xea\x2d\xd8\x95\xc9\x1e\x6a\xc6\xee\x38\x4a\x12\x8e\x5f\x71\x8d\xc3\x46\x34\x8b\xd4\xd2\x2d\xc8\xaf\xdc\xb9\x20\x5c\xd7\xcd\xa2\x22\xa0\xff\x0b\x42\x03\xf7\xbf\x9e\x7b\x3b\x1d\xdf\xd2\x16\xd4\xaf\x33\xad\x45\x8d\xdf\x3b\x93\x6f\xe1\xf2\x55\xb1\xbe\xbf\x0f\xd2\xc5\x84\xa4\xd6\x5b\xa5\x26\x24\x28\xaf\x45\xbc\x2a\x7e\xc3\x6c\x2b\x0d\x58\xee\x41\xf1\xac\xfa\xe8\xcf\x34\xbe\xae\x86\xa6\xc5\x32\x33\x6a\xaf\x1b\xf4\x3a\x8c\x5a\xb9\x00\xe8\xa3\xa7\xa8\xdb\x45\x61\x3b\x83\x2c\x0d\x65\x5e\xb3\xac\x15\xf0\x46\x79\x3f\x53\x4e\x48\x99\xd1\xf7\xdc\x4b\xe9\x2f\xfc\x38\x13\x7b\x8f\xb8\x15\x8e\x74\x86\x1f\xcd\x75\x22\x03\x12\xaf\xc5\xa2\x6a\xcc\x8b\x42\xf0\xab\x64\xe3\xcf\xe7\x9f\x49\x2e\xaf\x3d\xc4\xae\xfc\x50\x05\xdd\xf1\x09\xeb\xad\x8e\x3a\x63\x5b\xab\xc0\x9d\xb6\x29\xf9\x91\x27\x12\x22\x71\x09\xdf\x02\x8b\x78\xbe\x28\x2f\x75\x95\x60\x8b\x4d\xb4\x71\x15\x9a\xf4\xa8\xb1\xa7\x10\xa4\x8f\x15\x70\x23\x3c\x4e\x55\xfa\x9a\xf2\x62\xa2\x76\x20\xbc\xca\xa6\x31\x18\x17\x2b\x1b\x1e\xb1\xe0\x26\xe3\x50\x8f\xf1\xaa\xfd\x43\xbd\x26\x45\xe9\xbc\xfc\x3b\x31\x46\x73\xea\x71\x0a\x55\x3b\x7a\x55\xb3\xbb\xbd\xc8\x77\x41\xe2\xa6\x7e\xb9\x88\x99\x65\x2b\x7f\x07\x27\x55\x91\x65\x56\x6a\x6f\x5d\x59\x61\x21\x1c\x31\xbf\x43\xc8\x78\xdb\x27\x9f\x10\x72\x50\xf3\x59\x91\xb1\xb7\xc9\xf5\xc8\xb6\xaf\x8a\x05\x69\xdf\x7e\xd9\xce\x42\xcc\xe6\xd1\xbe\xde\x63\x05\xab\x0f\x63\x63\xdf\x55\xf4\xf3\xd7\x5a\xee\x0b\x2d\x06\xa9\x44\xa0\x5e\xa6\xbf\x

ba\x95\xaf\xe6\x86\x43\x31\xdd\xf8\x0c\xe7\x97\xe5\x0c\x7c\x91\x68\xf5\xe8\xd8\x71\x1d\x4f\x09\
x8b\x34\x07\x3f\xc6\x4b\x5d\xff\x0d\x85\xf4\xbd\x74\xa7\x4d\xb8\x4a\xe7\xeb\x00\x75\xbb\x42\xf
9\x5e\xa3\xa4\x78\xc7\x66\xc9\xd2\xe9\x49\xf5\xdd\xf5\x69\xb0\xd5\x2a\xd6\xde\x57\xd4\xc9\xc1\
x6d\x74\xbd\x52\x2e\xa7\x20\x15\x5a\x39\x61\x66\x46\xff\x67\xaa\x32\xf8\xb5\xab\x7e\x9e\x6a\x
c9\x3b\xfa\x87\xa5\x9b\xa3\x69\x4c\x39\x47\x7f\x09\xed\x1c\xfd\xfd\x44\xab\x4e\xd3\xcf\x39\x35\
xb6\xd0\xd0\x39\x77\xef\xab\xa8\xe8\x68\xe1\x55\x74\x74\x0c\xde\x56\xd2\xd1\xd4\x15\xb5\x74\
x66\x91\x1a\x35\x1d\x6b\xb1\xae\xec\x4d\x14\x75\x14\xb7\x15\x8a\xba\x76\x8e\xf2\x79\xb7\x5a\
x28\xea\x5a\x45\xf3\xfa\x5a\x8f\xeb\x3c\xb7\x7f\xab\x90\x07\x2b\xbe\x0a\x81\x88\x12\x36\x89\xb
0\xf4\x15\x89\xc4\x2e\x54\x43\x26\xa2\xdd\xfa\xf2\x37\xd2\xe9\x32\x49\xaa\xcd\x9b\x39\x4f\x7b\x
77\xfb\x5a\x4e\x8e\xb2\x05\xdd\xdd\x7d\xf4\x91\xda\xf7\x3b\x1e\x3e\xac\xb9\xb8\x25\x45\x7b\xdf
\xb6\x63\x9c\x97\x11\x49\xfd\xfe\x6d\x1d\x44\xb2\xdb\xa4\x06\xa2\x66\x40\x03\x33\xbd\x9e\xac\x
79\x11\x2b\xa3\xd1\x1b\x44\x89\xf3\x39\x3d\xf2\x93\x09\xd4\x6c\xf6\x3b\xe6\x5e\x6b\xd1\x94\x9
c\xe1\x54\x98\xb4\x98\x47\xea\x2a\x77\xb9\x96\xfd\x0b\x3b\x66\x2b\x8b\x5b\xc0\x32\xab\xdc\x69
\xd7\x6f\x7f\xab\x43\xb4\x5f\x22\xcc\x39\x6d\xa7\xf4\x0a\xc7\xd9\x19\xce\xf3\xf3\x9c\x94\x25\x06
\x73\x2f\xd6\xab\x0e\xda\x80\xde\xb7\xc6\xdd\x39\x68\xd9\x0b\xfd\x21\x3f\x58\x41\xa8\xa3\x28\x
49\x39\x0a\x4b\xd7\xef\xb0\xfd\xd6\xbe\x15\x32\x5d\xad\xa4\xd5\x9c\xd2\xda\x56\xe0\xcd\xe3\x4
2\xc0\x8f\xc1\xe1\x10\x54\xe1\xd1\x9c\xae\x0a\xf0\x7a\xc8\xb5\x59\x74\xbc\x94\x13\x60\x76\xc7\
x90\x90\xcf\x18\x45\xa8\x20\xe9\x34\xc1\xd2\x0f\x17\x40\x0e\x0c\x93\x68\xa0\x60\xe6\x66\x86\x
b9\xe5\x60\xad\x5d\x5d\xa1\x93\xee\xc9\xd6\x69\xf7\xb4\x2f\x85\xc1\x06\x37\x00\xbc\x7b\x26\xd
e\xe9\x97\xee\xda\xb0\x42\x74\x67\x36\x50\x0c\x15\x60\xab\xb0\x15\xa0\x47\x60\x8f\xbd\x09\x7
d\xd9\xd2\x1d\xd1\xa8\x0e\x39\x82\xac\x70\xd4\x10\x08\xd7\x0e\x55\xa7\x05\xe1\xd0\xe1\xa1\x0
0\x54\x0d\x0c\x87\x28\x4a\x12\x34\x8a\x0a\x32\x66\xfe\x0f\xe0\xb1\xc0\xce\x36\x57\xe0\x24\x19\
x3d\x19\x8b\xde\x04\x68\x67\xbb\xc9\xe8\xc4\x5c\xd8\x9c\xa3\x89\x13\xb8\xd0\x45\x22\x3c\x05\
x01\x12\x82\x42\x9d\x9c\x76\xd0\xfe\x0f\xb0\x3e\x55\xda\x2e\x4b\xac\x55\xa6\x1d\x88\xda\x56\x
e5\x00\x33\x5c\xd9\xb3\x9a\xd5\xae\xb7\x5a\x49\xb3\xca\xed\x97\xe1\x10\xc6\x21\xba\x3d\x6b\x
1b\xd5\x8a\x3c\x78\x80\xf4\xef\x13\xed\xb7\xe6\x02\xee\x54\xec\xba\x32\x32\xc6\x70\x7a\xa3\xb
9\xe1\xcb\xb7\x6e\x6a\xc4\x2c\x98\x73\xc3\x27\xcc\x9c\x1a\xcd\xe3\xda\x2d\x67\xc6\xea\x57\xcd
\xc4\x68\x6d\x7e\xed\x79\xb9\xcb\x89\x31\x5d\x9f\x28\x46\xaa\xcd\x04\x9c\x8d\x3a\x60\x8b\xb0\
xcd\x90\xce\x0e\x49\x1d\x6e\xac\xb0\xc5\xa7\x62\x6b\x57\x02\x6e\x9f\x9e\xec\x70\x50\x91\xc6\x
40\x24\xc4\xd6\xa9\x95\xa0\xbe\xdd\xdd\x01\xb0\x7a\x83\xed\x41\x1f\x0b\x1f\x62\xf3\x9e\xa0\x3
5\x76\x47\x13\x49\x26\xa8\xa7\x65\x69\x1c\xd2\xe6\xc7\x37\x9c\x58\x60\xd8\xbe\xd7\x10\x5b\x3
5\x53\xce\x37\x09\x71\xaa\xf6\xcd\x33\xcc\x9b\x6f\xaa\x3b\x32\xfe\x9e\x33\xe1\xfc\xb3\x63\xcc\x
bb\x51\xd1\x89\x59\xb9\x3e\xdd\xca\xfb\x5a\xab\x79\x96\x19\x6c\x28\x3c\xbf\x72\x7e\x0d\x2f\x8a
\x95\xbb\x3d\xf7\x56\x94\x44\x45\x89\x4e\x4e\xa9\x30\xc1\xea\xbd\xd1\xb4\xaf\xfb\xe7\x5d\xce\x
01\xc8\x59\xc8\xf1\xb1\x04\x07\x1a\xf5\x12\x0a\x3e\x25\x0d\xb4\x21\x92\x1a\xe3\x58\xed\x08\x2
3\x39\xb0\x7d\xd3\x84\x46\x97\x28\xc6\x93\x68\x99\x80\x22\xb4\x58\x52\x39\x55\x6e\xcc\x1d\xe
e\xa6\x26\xe0\x61\x1e\xed\x59\x34\x8e\x51\x37\x60\xc0\x6a\x47\x5c\x51\x14\x6e\x79\x7a\xab\x3
4\xaa\x17\xbe\xda\x85\x8e\x58\x5b\x22\x85\xbd\x46\x80\xe2\x39\x29\x9f\x74\x28\xc5\x07\xa8\x4
3\x17\x01\xfd\xef\xb4\x73\xaa\xa8\x9d\x43\x68\x69\x50\x28\x5d\x26\xf6\xb3\x07\x6d\x36\x5b\xa1\
xcd\x76\x30\x67\xf5\xb7\x61\x21\xb8\x4e\xaa\x9c\x95\xc0\xf6\x06\xee\x2c\x8f\xcd\x7a\x01\x37\xb
c\x74\x38\xc6\x78\xe9\xbf\xb0\xea\x2d\x22\xe6\xdc\xaa\xf7\xef\x13\x76\x1a\xff\xf7\x69\xbf\x59\x4
4\xe0\xca\x5b\xe9\xed\xa1\xfa\xde\xc1\x0a\x63\x21\xa0\xdb\xb3\x0e\xf1\xf6\xd4\xbd\xcb\xb2\x70\

xe6\xb9\xb4\xe0\xf7\xe8\xf6\xc6\xe0\xf5\x47\x6d\xde\xca\x70\x57\xa8\xc2\x09\xaa\xcd\x16\x1a\xb
c\xc1\x4a\xfb\x6f\xdd\x98\x78\x0f\x55\xfe\xf9\x1d\xa3\xba\x7e\x65\x71\x32\xd1\xfd\xc9\x72\x56\xe
6\x14\x92\x2f\x93\x4f\x4e\x7d\x4e\xc4\x07\x8b\x65\x31\xeb\x39\x9e\x49\xc5\x4b\x6d\xe1\x66\xd4
\xad\x99\x8e\xc5\xf5\xb9\x7e\xe6\x73\x00\xaa\xb7\xa4\xf9\xf1\xec\x9d\x05\x48\xf7\x2f\x6b\xb9\x2
7\xbd\x95\x53\x5f\x3e\x81\xba\x33\xdf\x5b\xcf\x1f\x74\xdd\x91\x3a\x38\xe2\x7f\xfb\xf9\xf3\x79\x6
4\x6d\xf0\xc4\x5a\x39\x11\x74\x36\xc1\x55\x6a\xcd\x7c\xac\x3c\x1b\x6b\xce\x1d\xa1\xa5\x3b\x32
\x96\xa4\xe6\xd1\xb6\x8d\x4f\x50\x76\x3f\x3a\xc9\xb3\xb9\xd7\xdc\x80\x41\xf9\x78\xcb\xc8\x7e\x
b0\x63\x19\x08\x19\x96\x41\x2b\x3c\x98\x12\x4c\x8d\xb5\xdc\x82\x45\xf1\x81\xe8\x2c\xca\xf0\xa
7\xd9\xc0\xaa\xbe\x0a\xaf\x82\xbd\x49\xbf\xb1\x64\x82\x2e\x7f\xe2\x03\xdd\x13\x82\x0e\x47\xd7\
x43\xb4\x0d\xc6\x0f\x7d\xe1\xd1\x99\x23\xaf\x6a\x11\xd5\xd6\xa9\x37\xef\x54\xec\x5b\x51\x50\xe
0\x43\xc9\xee\xd8\xf5\xd2\x1b\x68\x87\x39\xbd\x67\xbb\x6d\x41\x41\x0a\x14\x4d\x4a\x9c\xcb\x45
\xa2\xf7\xf7\x46\x6b\xd5\x5f\xc6\xe7\xbb\x5b\x71\x8e\x0a\x9f\xdd\xa8\x16\x7b\x3c\x74\xcc\xdb\xa
a\xfa\x75\xbf\x1e\x95\x6e\xa4\xed\x98\x37\xb5\x8c\xa6\x25\xa7\x41\x0f\xeb\xfb\x46\x61\x37\xf6\x
eb\x61\x5a\x31\x2a\xd3\xe1\xac\x36\xed\x1b\x88\xdc\x2d\xd7\xfa\x43\xec\x21\xfa\x5f\x4b\xea\x17
\x06\xa9\x2d\xff\xfe\x50\xc4\x7f\x4f\xfb\xda\xdf\xef\x42\xfb\xc8\x4b\xfa\x7a\x80\xc6\x9b\x92\xbe\x
1d\x46\x6c\xc5\x4d\xc5\x21\x56\xbb\xfe\x76\x3b\x8b\xd9\x8b\x55\xea\xe7\xf3\xe7\xa5\xb7\xc4\xa
1\x2f\xff\xfa\xab\x5e\xc2\x0b\x7e\xeb\xe7\x1a\xa9\x36\x75\xbf\x87\xb6\xd0\x86\xd9\xbb\x3e\xf3\xc
9\xc4\x22\x89\x79\xa6\x9e\x79\x20\xb6\x2e\xdd\x8c\x07\xdb\x35\xfe\xec\x0d\x5c\x5b\x16\x5f\x06\
x17\x5b\x5b\x71\x6c\xfa\x9c\xcb\x95\xb5\xdd\x37\xd5\xaa\xde\x8b\x44\xab\xeb\x8d\x17\xbc\xd5\x
57\xbb\xf2\x4d\xdc\xf5\x69\xb0\xf5\x7b\x87\xde\x3f\x6e\x7e\xf6\xb6\xac\x79\xf7\xc6\x3d\x91\xc0\x
ff\xcc\xd6\x65\xa9\x9e\xbe\x2d\xb5\xb7\x6f\x4b\xfd\xc1\xda\xd2\xf3\xfa\x6d\x29\x9f\xbf\x2d\xb5\xf
7\x6f\x4b\xed\x01\xdc\xd2\x7c\x01\xe7\xd4\xd8\xc2\xc2\xc6\xf1\x8f\xf2\x15\x1f\xc1\x1d\x7b\x5f\xc
1\x1d\xaf\xfe\x0c\xee\xb8\xed\x3b\xb8\x63\xf7\x21\xdc\xf1\x1d\xbc\x84\x5b\xde\xfa\x29\xdc\x71\x
eb\xb7\x70\xbf\x77\x5c\xff\xe3\x16\x16\x67\xcb\x3a\x93\x33\xe1\x5a\x85\xfd\xe0\xc4\xa9\x59\x9d\
x2d\x75\xb3\xb3\xa5\x61\x25\xb6\xf4\x19\x9e\x2d\x95\xe5\xd9\x52\x37\x3d\x5b\xea\xb6\x67\x4b\
xcb\xf8\xcc\x53\x6f\x9b\xc5\xf1\x9b\xda\x9f\x1d\xfb\x0d\xd0\x8e\x6f\x60\x81\x76\xdc\xda\x04\xed\
xd8\x63\x83\x66\x97\xbe\xd9\x1a\xa9\x31\x43\x6b\xbb\x48\xda\x1b\xa2\x7d\xdb\x66\x95\x74\x97\
x05\x06\xc5\xec\xb8\xec\xb2\x80\x7c\xd3\x0c\xe1\xf4\x0c\xc5\x19\x06\x6b\x05\x78\x1d\x18\xa5\x
31\xf8\xb0\x45\xff\x7c\xf3\xfa\x55\x59\x2e\xde\xe3\xff\xb3\xc4\x45\xb9\x06\x82\xd9\xe5\x02\x67\x
13\x2b\x87\xf9\xb1\x91\xef\x37\xba\x02\x2f\xbc\xe1\x81\x0d\x8d\xbe\x5c\xef\xad\x19\xc1\x22\x2b\
x21\xcd\x04\x90\xd4\x7f\x29\x66\x74\xf7\x21\xd3\x34\xcb\x71\x98\x90\x14\xaf\x5d\x33\x8b\x55\x8
a\x87\x56\xde\xee\xef\x5f\xce\xde\xbf\x9c\xfd\x13\xbf\x9c\x65\xaf\x66\xb9\x0d\x9b\xf1\x6c\x96\x6
d\x38\xe8\x66\xaf\x67\xf9\xde\x77\x5c\x92\x04\xea\x64\xfa\x4c\x58\x3b\xec\x79\x92\x03\x46\xca\
x4b\xc9\x12\x55\x91\x71\x12\x15\x05\x3a\x81\x22\xa7\xbc\x9b\x2c\x43\x31\x61\x56\xd5\xda\x10\
xee\x8d\x60\x95\x72\xe5\x2a\xe5\x20\xa8\xc6\x99\x75\x7b\x3f\xe7\x00\x49\x6b\x3a\x7e\x7b\xf8\x
f1\x03\x3d\x5b\xc3\x24\x74\xcf\x31\xe9\x32\xd2\xec\x7e\xd6\x7e\xbf\xd1\x7e\xff\xa4\xfd\x2e\x7e\x
8d\x46\x99\xf8\x98\x90\x34\xc5\x97\xf2\x0b\xcf\xcb\x0c\x9e\x32\x8a\x94\x05\x19\x9b\x09\x69\x94
\x9a\x09\x73\x32\xce\xed\x94\x24\x21\x4e\x21\x03\xde\x00\x15\x1f\x46\x91\x69\x1e\xa5\xb1\x1c\
x8a\x91\xf5\x93\xf1\xf5\xd1\xf8\x7a\x67\x7c\xbd\x34\xbe\xfe\xc7\xf8\xfa\x97\xf1\xf5\xd6\xf8\x7a\x6
1\x7c\xfd\xc3\xf8\x3a\x66\x5f\x6b\xa7\xd5\xae\x6b\xe8\x1c\xbd\x3b\x78\x41\xa7\x38\x44\x3b\xdb\
x81\x4c\xfc\x70\xf8\xd3\xdb\x83\x8f\xc7\xef\x5f\x7e\x7a\xfd\xf2\xed\x4f\x1f\x5f\x85\x68\x57\x65\xc
2\xac\x86\xea\xa7\xca\xa9\xa0\x9c\x10\x7d\x41\x56\x82\xf2\xa3\x0e\x19\x9f\x5e\x1c\xfd\xfc\x16\x

5d\xab\x9a\xde\x1d\xbd\x7e\x4d\xa1\x3f\x1e\xbe\x79\x79\x74\xfc\x31\x44\x5b\x9b\x9b\x9b\x43\xde\x43\x7e\xe3\xfd\x2c\xc9\xc6\x9f\x43\xd4\xa5\xac\xb3\x28\xbb\x46\xde\xc1\x18\x42\x19\x87\xea\x6d\x23\x7b\x80\x41\xf7\xf3\x26\xdf\x27\xf7\xa1\x30\xee\x37\xb2\xbf\xfa\x46\xb6\x26\x5d\x40\x14\xb3\x68\xe7\xae\x3c\x40\x3c\xcf\x2f\x17\x65\xf6\xf7\x0f\xfa\xe6\x30\x86\xb4\x47\x2a\x02\x06\x6d\xd0\x0b\x30\xa4\x39\x5d\x6f\x74\x27\xd7\x7d\x03\x50\x5c\xa1\x3f\x50\xe5\x49\xe8\xc1\x03\x91\x3b\x10\xfe\x22\x98\x98\x3c\xc3\x17\x5d\xfb\x15\x9d\xe1\xf9\xeb\x07\xb4\x4d\x4b\xdb\xde\x8f\xb7\x85\xbb\x48\xb3\x38\x12\x97\xe1\xf2\x82\xdf\xf2\xcf\x8e\xac\xd7\x76\x0c\x54\xe0\x88\x76\x6e\xf0\x0a\x5f\x0c\x40\x7b\xc9\x3d\xf7\xfa\x6c\x8c\x28\x56\xc4\xb0\x55\xeb\xec\x44\xc7\xd4\x6f\x21\xda\xfe\xe6\x31\x2b\xa9\x3d\x4e\x16\x6f\xce\x28\xcb\x93\x38\xee\x84\xdf\x7c\x17\x74\x4c\x94\x77\xc2\x27\x9b\xd7\xa7\xc1\x76\x2b\x9f\x4f\xf7\x7c\xef\x9e\xef\xfd\x79\xf9\x9e\x62\x7b\xec\x9d\xff\x1d\xf0\x3d\x4b\x76\x5f\x5d\x74\xf7\x48\xee\xa2\xa0\x4f\x70\x5f\x29\xda\x90\xcd\x6b\x07\x43\xce\xee\x55\x38\xa2\xc9\x13\x1d\x80\x7e\x4b\x11\x7e\x99\x92\xf2\x4d\xb4\x90\xe2\x62\x57\x48\xd4\x21\xe3\x41\xdd\xcd\x6e\x80\xc4\x73\x68\x90\xee\x43\xc5\x1a\xbb\x5b\x86\xac\x1f\x6a\x19\x9b\x9b\x9b\x22\xef\xbf\x6b\xf2\x46\xd1\x68\x14\x4d\xb1\x6c\x4d\xcf\xd3\x0e\x00\xa1\x9d\x37\xf7\xd4\xa9\x65\xbf\xa9\xcf\x4e\xb2\x33\x9c\x44\x63\xd1\xac\x9d\xad\xce\x19\xa1\x2f\x7b\xea\xaf\x5c\x83\xf8\xa9\x11\xa2\x98\x45\x69\x9a\xa5\xc6\xb8\x4d\x08\x75\xb6\x09\x6b\x20\x1a\x5a\x81\xd3\x55\xe8\x81\xd0\x51\xa9\xce\x4c\x61\x3d\x50\x53\x4d\xfc\xfc\x16\x7a\x81\x8c\xca\xe4\x99\xcc\x1e\x9b\x07\xd0\x3f\x44\x13\xd0\x20\x59\x0f\x9c\x06\xfa\xd9\x84\xf5\x81\xea\x73\x0d\x27\xbf\xda\x8a\xf5\xfe\xb6\xaa\x5b\xaf\xbe\x6d\x01\xad\x4c\xb9\x42\x19\x5a\xcc\x6f\x70\xa5\x9c\x31\x2c\xa2\x98\x9b\x93\x82\xb9\xe7\xc5\x02\x8f\xe9\x06\x26\x4d\xf4\x75\xc3\x2b\xee\x41\xc5\x67\x3d\xa5\xaa\x18\x61\x0a\x17\xf3\xa8\x5c\x96\x1d\xd6\x78\x16\xe5\xd1\xb8\xc4\x79\x21\xd4\xfc\x70\x37\xcf\x4b\x6b\x7b\x89\xb7\x0d\x32\x4d\x03\xcd\x1e\x1a\x6d\xae\xf9\x5d\x7f\x90\xe9\xac\x44\xc2\x2b\xad\xe5\xe1\x97\x8f\xc1\x90\x38\x19\x48\x00\xbd\x2b\x02\x68\xc7\xe3\x67\x88\x59\x89\x00\x0c\x04\xa6\x85\x17\xab\xf2\x96\x78\xab\x3f\xf8\x25\x23\x29\x04\x6c\x40\x4f\xa1\x0e\x14\xa2\xce\x66\xa7\x8f\x36\x38\x70\x85\xf1\xdb\x8d\xe7\x02\x82\xf6\xfc\xd9\x27\x03\x06\xb1\xe2\x6c\xf0\x1e\x6e\x30\xaf\xcb\x37\x9d\x97\x2a\x63\x44\xd3\x19\x0d\x6c\x9f\x60\x8a\x08\x01\x3d\x5c\x3f\xd3\xd6\xbc\x30\x8f\xcd\x35\xb3\x42\x52\x5a\x89\x1f\x59\xba\x4f\x6a\x8f\xb3\x24\xda\xb8\x32\x3d\x64\x5e\x48\x8e\xd9\xf6\x2e\xc5\xfa\x19\x8b\xf9\x3c\x1c\xa2\x1f\x49\x1a\x23\xf6\xc0\x8b\x77\x54\xc6\x6c\xa6\x52\x45\xa7\xa3\x6e\xf3\xc1\xfe\x25\x80\x30\x52\x33\x7c\x21\xcc\x98\xe5\xb9\x8b\xa6\xb1\x93\x0f\x3d\x75\x54\x9f\x97\x68\x35\xdb\xfa\xdb\x17\x30\xb0\xe1\x76\x35\x7b\x88\x6c\xec\x6f\xeb\xe0\x22\x1e\xb2\x6e\xdf\xa1\x9a\xea\x11\xda\x0e\x0f\x7f\x21\x5b\x98\xa0\x1e\x2b\xb2\xbf\x8f\x36\xfb\xc6\x49\x6d\x94\xe3\xe8\xb3\x02\xa5\xa3\xdc\xd8\x47\xfc\x65\x39\x9d\xc1\xe7\xb3\x28\x7f\x9e\xc5\x18\x6a\xf0\x1e\xc4\xe8\x64\x0b\x93\x9c\xa2\xcc\xdb\x51\x08\x9b\xb4\x95\x48\xe4\x80\x16\xf9\xed\x68\x04\x9a\xfb\xcf\x21\x92\x9b\xcc\x7c\x51\x56\xbd\x50\x37\x27\xdb\xe3\x67\xbe\xb7\xc8\xf1\x84\x5c\xb0\x40\x5a\x9b\x17\x7d\x3a\x0b\xc0\x35\xfc\x2e\xee\x79\xc4\xb7\xea\xd9\xf7\xda\x2f\xc3\x31\x34\x4a\x80\x9b\xd7\x06\x14\xf0\x45\xfa\x34\xfc\xed\x73\xd7\xeb\xbc\x1b\x3a\x55\x50\x8a\xe7\x98\x67\xb3\x0f\xcb\x81\x9b\x6e\xb3\xe5\x20\x66\x84\xb6\xa4\xa8\x63\x92\xe5\xb6\x19\x5d\x51\xe6\x55\x51\xf1\xb5\x19\xa5\x50\x63\x3e\x37\x07\x65\x8f\xdc\x6c\xa5\x83\x85\x22\x0f\x10\x6e\x78\x6e\x53\x20\xb4\xbf\x1b\xfb\x28\x15\xfb\xc2\xf7\x68\x1b\x3d\xa5\xa7\x1b\xb4\x81\xe8\x7e\x90\xfa\x68\x82\xbb\x91\x9f\xe1\x8b\xbb\x24\x0d\x2b\xee\x80\x4d\x1b\x0d\xac\xe1\x37\x23\x0e\x87\x67\x68\xd4\xf1\xdb\x50\xc0\xef\x36\xad\x96\xd7\xd2\xc9\x32\x49\x24\x1a\x86\xf8\x0c\xa7\x25\x7b\x2c\x00\x2c\xff\x97\x22\x4b\x51\x34\x22\x36\x8f\x17\xae\x13\x3f\x66\x3f\x2e\x93\xc

4\x7e\x47\x29\x1e\x14\xd0\xd2\x8f\x58\x69\xf7\x41\x14\x6b\xd8\x69\x57\x31\x76\xb7\x0d\x43\x90\xa2\x95\xeb\xea\x53\xfa\x3d\x00\x33\x0a\x92\xc6\xf8\xe2\x68\xd2\xeb\xf6\xba\x7d\xf0\x0f\xf9\x68\xcb\xf3\x24\x52\xc2\x3b\xb6\x82\xe5\xe5\x02\xf3\xe6\x00\x08\xa8\xc8\xf4\x69\xd6\x23\xfd\x2f\x22\x8c\xf0\x80\xc2\xef\xa1\x6b\x2e\x8a\x99\xd6\x7f\xb2\x15\xb4\x81\xba\x3d\x3a\x73\xb2\xf6\x0d\xd4\xed\x77\x5b\xad\xbd\x98\x14\x8b\x24\xba\x64\xf3\x02\x7e\x46\xd3\x92\xca\xb6\x12\x1b\xf6\xbb\xb5\x0b\xc8\x7e\xc1\x8a\xd5\xbd\x72\xa5\xb5\x99\x93\xef\x5f\x5e\x46\x0f\xe8\x96\x66\x51\x0c\x9e\x0e\x44\xcc\xc5\xcb\x1e\x37\xad\xeb\xa3\x47\x3f\xc8\x44\x39\xad\x6e\xdf\x6a\x1f\x3f\x4b\xbb\x4d\x67\x66\x0d\x34\x73\x30\x36\xd9\xe8\xa9\xfd\xae\x95\xbf\x09\xa3\x6b\x46\x39\x1c\x19\x0e\xd5\x40\xb3\x33\x9c\x27\x59\x14\xe3\x58\x2a\x83\x3d\x6b\x42\x1f\xc0\x47\x45\x24\x55\xef\x1a\x87\xe8\xe3\xd1\x8b\xa3\x10\xcd\xa3\xcf\xa0\x1e\x26\xe9\xd9\x32\x49\x71\x1e\x8d\x12\x7c\x97\x03\x54\xa7\x01\xfb\x05\xef\x16\x7a\x84\xb4\xec\x7e\x7f\x90\xe3\x45\x12\x8d\x71\xaf\x8b\xba\xe0\xd8\x8d\x9e\x16\x3a\x66\xa0\xc8\x2c\x3d\xc3\x79\x59\xa8\xb0\x9b\x20\xf7\xc5\x78\x4c\xe6\x51\x62\x33\x59\x92\xfa\x99\x7d\x99\xbd\x60\x05\x5c\xca\xab\x0d\xa1\x69\xba\x36\x64\x02\x1e\xaf\xa9\x31\x08\x64\x99\xb9\x31\x32\x65\x08\x9a\x36\x63\x6c\x94\x6d\x29\x4f\xbc\xab\x71\x69\x75\xd5\x07\x68\x4d\x85\xa6\xd4\x1d\x9f\x27\x3c\x37\x57\xa1\x9a\x3b\x8a\x71\xd8\x67\x00\x09\x2e\x8a\x8f\xb3\x28\xed\x6d\x82\x23\xd9\x47\xcc\xf2\x9c\x5b\xf0\x73\xc2\xda\xea\x43\x08\x57\x2d\xc7\xc0\xe2\xc1\x12\x5c\x35\x73\x54\x46\xe9\x25\x77\xbe\xc3\x5d\x92\xa6\xd5\x68\x1d\x70\xbc\x1e\xa4\x31\xbb\x02\x60\x34\x44\x26\x97\x05\x77\xa6\x5e\xa0\x11\x9e\x64\x39\x1e\x38\x74\xf5\x8a\x1f\x1d\xea\x71\x7f\xc5\xf7\xa0\x06\xd2\x7a\x05\xfb\xbc\x81\x7c\xb9\x7e\x1f\x72\x73\xb1\x79\x74\xc1\x42\x57\x5e\x90\xf2\x32\x44\x4f\x40\x8d\x2d\x76\x1d\x52\x70\xb7\xc6\x50\xb4\x6f\x6f\x32\xda\x24\xf7\x36\x28\xc4\x9e\x51\x54\x9f\xce\xfa\xc2\x4e\x59\x36\xbe\xea\x82\x10\x59\xe9\xef\x1f\x8e\xde\x0e\x24\x6e\x19\xb0\x72\x5d\x09\x4e\x63\x0b\x14\xd9\x71\x3c\x03\xb4\x88\x8a\x82\x72\xac\x72\x96\x67\xcb\xe9\xcc\xa4\x7b\xd9\x05\x4e\x61\x50\xab\x7b\x2d\xa9\x78\xd9\x23\x38\x23\x79\x24\xdd\xca\x71\x0a\x00\xfe\xaa\xc3\xac\xae\xa1\xb6\x33\x61\x39\xaa\x55\x80\x7a\xeb\xa4\xf8\x91\xa4\xa4\xc4\x16\xc6\xac\x6e\x80\x5c\xa8\x75\xc2\x94\xad\xdc\x8e\x6a\xab\xe1\x3d\xdf\x4a\x18\xf5\xd3\x53\x52\x0a\x3c\x1f\xfd\x8c\x6d\xf1\x69\x8a\x4b\x88\x55\x7c\x34\x39\x4e\x89\x57\xc7\x05\x65\xcb\x19\xe6\x3f\xe4\x32\x43\x65\x16\x48\x9d\x94\x74\x85\xee\x8d\xd7\x28\xfb\x21\xab\xe9\xb1\xce\xf4\xa1\x08\x38\xec\x2a\x10\xce\xf3\x2c\x17\xce\x68\x58\x8f\x0b\x94\x66\x25\x1a\x67\x79\x8e\xc7\x65\x78\x2e\x57\x8b\xd9\x6b\x63\xd9\xd0\x82\x82\x04\x96\x2c\x13\xfe\x7b\x0a\xff\x0d\xca\xec\x75\x76\x8e\xf3\xe7\x51\x81\x7b\xc0\x52\x98\x96\x57\x71\x2f\x0a\xf5\x0f\x7e\xbf\xcc\x2f\x6d\x4e\xe8\xff\xa7\xea\x00\xae\x81\xe8\x1e\xbf\x75\xc2\x63\x3e\xc8\x52\x7c\x8e\x5e\xd2\x51\xf5\xba\x70\xc9\x0b\x1d\x01\x2b\xd5\x7f\x77\x4b\x84\x2f\x48\x51\x16\x01\x5a\x24\x38\x2a\x40\x18\x86\x91\x67\xa9\x44\xd5\x24\x4b\x92\xec\x9c\xa4\x53\x28\x59\x50\xde\x67\x2d\x23\xde\xc3\x00\x3c\x2b\x04\xea\xc1\x47\x4d\x7c\x58\xd9\x7b\xf0\x7b\x65\xfa\x13\x8e\x3e\x63\x58\x84\x8c\xcd\xc3\x35\x34\x01\x4b\x5a\xc9\x5a\x19\x09\x50\x06\x0b\x5e\x2a\xd8\xc4\x33\xd4\x72\xca\x7a\x97\x15\x05\x19\x25\x6c\x0a\xc1\x79\x06\x37\xe7\xfb\x70\x48\xa5\xca\xbc\x64\x3f\xa9\x20\x2d\xb0\xf5\x72\x32\x21\xd3\x4b\xfe\x71\x24\x48\xe9\x11\xfa\x4c\x9b\x67\x7f\xea\x92\x0a\x3e\xf9\x7d\x16\x03\x9b\x2b\x30\x79\xa5\xc4\x3e\xc5\x05\x14\x83\x9b\x2a\x38\x79\xeb\xc3\x3e\xf9\x35\x91\xca\x63\x05\x1e\x3d\x92\x0b\x53\xdd\xde\xb0\x02\xbf\x46\xa3\xcc\xc8\xf3\x94\x10\xb7\x2f\x6c\x00\x70\x69\xa3\xe7\xb1\x12\x5a\x2f\xb4\xc2\xec\x93\x63\x41\x03\x41\x16\x84\xf6\x01\x57\x28\x1c\x21\x58\xe1\x70\xaa\xfd\x2e\xc5\x6f\x5b\x90\x60\x7c\xc1\x3a\xef\x5e\x49\xe9\x9c\x91\xc3\x38\x4a\xe9\x79\x20\x92\xac\x99\xa7\x73\x0d\x59\x96\xa3\x08\xbd\x7a\xf9\x4f\x38\x7a\x0b\x19\xed\xce\x

18\x8a\xdc\x5d\xc5\x81\xee\xe7\x19\x16\x1e\xf6\x22\xed\x12\x97\xc7\x3f\xd1\xc2\x04\xd0\xf5\x14\x15\xe8\x1c\xd3\x05\xa2\x5c\xab\x88\x61\xac\x69\x32\xd0\xcf\xd8\x38\x88\x8b\x71\xea\x2c\x85\x09\x38\xb4\x66\xc1\x24\x74\x51\x88\x95\xd0\xe3\xc5\x9a\x9c\x8a\x71\x27\x4b\x0a\xd2\x37\x5f\x5e\x01\x7a\x6a\x34\x12\xea\x5f\x9a\x3c\xd5\xb8\x7c\x23\x86\x63\xcf\x0a\x3e\xc7\xe4\x7e\xc1\xfe\xa7\x2c\xf1\x32\xab\x5b\xe0\xda\x29\xe1\x37\x5b\xea\x74\xb5\xfd\x8e\x8b\x1d\x10\x72\x37\x4b\xbd\x24\x73\x5c\xfc\x1e\xcb\x3c\xe5\x2a\x45\xba\xb8\xa5\x82\xaa\x60\x87\x7b\xd8\xa2\x91\xb4\x62\x71\xc8\x41\xf6\xa4\x15\x51\x28\x32\x10\x37\x86\x74\xee\x15\x2d\x98\xb5\x49\xff\x56\xaa\x02\x05\x20\xf1\xaf\x9b\xdd\x58\xb3\xd0\x70\xea\xf9\x86\x0a\x81\xb0\xec\x45\x79\xfe\xe3\xea\x0a\x6d\xee\x79\x8f\x34\xbc\x5e\xe7\x70\xc2\xd2\x8d\xb3\x0c\xc7\xb9\xe8\xc9\x83\x07\x88\xff\xf6\x09\xfd\xb4\x49\x3b\x57\x3f\x61\xf8\xbc\x9f\x19\xb2\x18\x2f\x2c\x35\x21\x9b\x17\xdd\xa0\xdb\xd5\xaf\x59\x2c\x1f\x69\xbe\xd2\x3a\xa1\x54\xca\x74\xa9\x88\x1a\xeb\x21\x15\x49\x27\x0c\x4c\xc4\xef\x90\x47\x31\x6e\x30\x09\xb0\xe5\x79\xd6\x2d\xd0\x58\x46\x73\x71\x48\xcb\x0c\xf6\xd2\x86\xbe\x2a\xa8\x46\x3b\x1a\x9b\x75\x9a\x6a\x2e\x83\x64\x28\xf8\x48\xa3\x2c\xdf\x82\x85\xc7\xde\x3d\xcd\x5f\x9d\x2c\xa0\x2b\x22\x1a\xa7\xae\x33\xb9\xe5\x5f\x07\x66\x79\xb0\x48\x96\x85\xea\x02\xff\xf6\x3a\x36\x94\x40\xa6\xfe\x68\x86\xc7\x9f\x0b\x71\x6c\x62\x3c\x52\x5c\x6e\x16\xfc\x99\x5c\x72\x09\x1e\x7c\xbd\x71\x88\x19\xc9\x8f\xbd\x31\x88\xcd\x68\xc2\x5a\x03\x74\xfd\x47\x0a\x5e\x77\x69\x07\x61\x95\xf8\xcc\x59\x75\x1b\x13\xc7\x2b\xb5\xf4\x66\xc3\xff\xdd\xbc\x38\xd9\x7c\xf4\x5d\xf4\x68\x72\xfa\x65\x77\xf3\xfa\xbf\x86\x64\x50\xe2\xa2\x94\xe0\x2b\x8c\xbd\x66\xc8\x5f\x67\xb0\x2d\x86\x09\xe7\xff\xe1\xff\xf6\x36\x2f\xfa\x4f\x6b\xc7\xa9\xd3\xdf\x70\xa8\xa2\x64\xb1\x38\x58\xd0\x3b\xe6\x3b\x98\x9b\x1b\xce\xe1\x05\x2f\xdd\x8f\xb5\x51\x9b\xf4\xcb\x5d\x00\x22\xd3\x49\x85\xb7\x33\x66\x5f\x28\x9b\xd3\xc0\x0e\x1e\xfd\xe8\x05\xb3\xba\x0c\x41\xbb\xba\x05\xb8\x39\x2e\xe6\xf4\xdf\x71\xb4\x28\x40\x76\x48\x12\x24\xbe\x03\xdd\x37\xa3\xdd\x63\xe6\x72\x5e\xeb\xb0\xd1\xc0\x91\xdc\xde\x19\x76\x70\x34\x9e\xa1\x71\x54\x38\xd5\x90\x82\x11\xca\x72\xce\x67\x48\xa3\x26\xb6\xca\x58\x40\x91\x76\x54\xc5\x5a\x2b\x96\xf3\x39\x8e\x2b\x09\xcc\x6a\xf0\x8e\x09\xcd\xaa\xbd\x82\xe0\x90\x16\x5d\xe7\xb9\x07\x43\x91\x2c\xcd\x7f\x39\xfb\x90\xd2\x8a\x70\x88\x57\x51\x01\xce\x68\x66\xd1\x8e\x68\xc8\xd4\xa8\x08\x99\xc7\xe7\xf0\x65\x77\x13\xee\x27\x11\xed\x9a\x3e\x8f\xe0\xbd\xbb\x9c\xa1\x04\xc3\x7b\x6a\x2d\x04\xdf\x62\x81\x73\xda\x5d\x31\x17\x29\x84\x2f\x9c\x12\x16\xe1\x2e\x2a\xf0\x3c\x5a\xd0\x39\xd9\x32\x14\x7e\x3d\x69\xbe\xa0\xf5\x1a\xfc\xb2\x6d\x3d\xee\xa3\x1f\xd0\xb7\x74\x57\xe7\x59\x27\xe4\x74\x50\x66\xc7\xb4\x21\xae\x12\x5a\xdf\xdf\xd7\x32\x81\xf6\xeb\x2b\xfc\x7e\xdf\x53\xa3\xae\x64\xb2\x6a\xac\x70\x16\xae\xad\x4e\xc5\xf9\x0d\xfe\x0f\xab\x01\x26\xd5\x20\xd7\x37\xfc\xd4\x27\xc8\xb2\x82\x26\xcb\xec\x2e\x69\x52\xa8\xaf\xe5\x16\xdd\x92\x24\x25\x13\xe4\x4f\xb0\x25\x0d\xda\x6f\xaf\x79\x43\xdd\x2e\x27\x28\x97\x58\x0d\x24\xdf\x88\x74\x35\xa0\xb1\xd3\x7d\x5a\x51\x0d\x31\x8b\x5e\x68\x17\xef\x0e\x61\x03\x07\x9c\x29\xeb\x3f\x4a\xaa\xdf\xd1\x53\xd0\x84\xf9\xd1\x17\x97\x71\x8a\xce\x0d\x3a\x6e\x22\x63\x93\x90\xec\x11\x6c\xec\x57\xd2\xb8\x46\x65\x36\x5b\x6d\xac\xa9\x96\x42\xad\x92\xa6\x1c\xaa\xe4\x4e\x83\xa5\x96\x19\x95\x2f\x49\x8c\xb6\x37\x99\xef\xa0\x47\xfc\x92\x90\xb5\xc9\xde\x29\x6c\x5e\x20\x66\xe0\xe1\x1a\x78\x35\x12\xb3\xff\xc6\x9f\x7b\x21\xd0\x39\xb8\x34\xe2\x6a\xb7\x8d\x5b\xc2\x8d\x77\x1b\x14\xce\x75\x05\x3e\x34\x89\x9e\xed\xbd\x75\xdb\xae\xa7\x22\x7e\x01\xe6\xab\xcf\x84\x10\x21\x18\xe1\x5a\x49\xd6\xa8\x5e\x55\x05\x68\x77\xd3\x7f\x67\x20\x1c\x12\x8b\xb3\x75\xa1\x64\xde\xe6\x60\x9b\xde\x73\xa5\xef\xfa\xcb\x08\xc0\xc9\x36\x35\xdf\x89\x10\xf5\x58\x37\x2c\x29\x51\xf4\x2d\x2d\xca\x28\x1d\x53\x3e\xa2\x0a\x5f\x5d\x49\xa4\xf1\xc2\xf0\x8a\x0d\x7e\x19\x0e\x34\xbc\xa9\xcc\x3e\x02\xb8\x91\xac\xb2\xdb\x16\x51\xe2\x74\xdc\x84\xa5\x0f\x8e\x8

1\x51\x4b\x14\x79\x42\x25\x79\xf1\xc3\x99\x2b\xef\x19\x8c\x86\xf5\xad\x7b\x77\xe8\x61\x7d\x69\x8d\x1b\xd1\xe3\x66\xec\xfc\xa8\x0c\x49\x56\xc5\x8f\x28\x7a\x23\x0c\x89\x12\xdd\x96\x23\xa2\x7d\x2a\x9b\x87\xc3\xba\x7e\x83\xc1\x1c\xf1\xbe\xdd\x60\x28\xec\x77\xdb\x81\x8c\x58\xdb\x2d\x56\x37\x03\xbc\xc9\xda\x66\x49\x37\x1a\x0c\xef\x5e\xdb\xd1\x80\x67\xbf\xe6\xb1\x38\x81\x32\x5a\x8e\xc4\x0d\x76\xd1\x92\x43\x41\xc1\xda\x31\xd8\x27\x0d\xb6\x2d\x82\xd9\x5b\x26\x88\xc8\x1c\xc4\xdf\x0b\x73\x99\x28\xa3\x82\xda\x31\xd0\x62\xf6\x43\x00\xe9\x11\x29\xbf\x74\xb7\x9d\xf5\x75\xb8\x76\x64\xaf\x6b\x95\x7d\xea\x35\x1a\x43\x34\xa7\x1e\xf6\x6c\x55\xfa\x69\xd3\xef\x23\x88\x15\xe1\x3b\x55\x28\x7e\x04\x22\x15\xde\x2b\x84\xf2\x17\x4b\x87\xfd\x2c\x64\xff\x89\x14\x7e\x0d\x1e\xaa\x9f\x2c\x47\xbb\x22\x0f\xf5\x0f\x51\xee\xb8\x9c\x3c\x09\xf9\xff\x22\x0d\x2e\xdd\x43\xf1\x43\xd5\xc3\x60\xc5\x2f\x95\xce\xe1\xe5\x4f\x5e\x8f\x6b\x2f\x18\xfa\x12\x19\xb4\x6b\x86\x16\x7a\xd2\x0c\x58\x61\xf1\x15\xda\x09\x62\x1c\x3f\x63\x18\xc5\xcf\x58\x1b\x03\xa4\xf1\x1f\x02\x4e\x6e\x72\xa1\xfe\x21\x72\x4d\xbd\x5b\xe8\xa4\x48\xac\x31\xf9\x22\x54\x3f\x59\x8e\xb6\xa9\x87\xfa\x87\xc8\x35\x04\xa8\xd0\x4e\x10\x50\x5a\xbe\x95\x63\x1d\x3a\x42\x37\x49\xf4\xd0\x81\x74\x92\x44\x9d\x62\x0f\x09\xb5\xdf\x7a\x7f\xd3\x69\x28\x7f\x89\x74\xc6\x3b\x42\xf9\x4b\x8e\x9e\xad\xbd\x50\xfd\x94\x63\xa2\xdc\x20\x14\x3f\x44\x2a\x5d\x98\xa1\x58\xd7\xd7\xd2\x65\x16\x7f\x67\xda\x09\xb7\xbe\x0b\x6a\x3d\x6e\x04\x9d\x65\x39\x79\xd2\x09\x9f\x7c\x73\x7d\x1a\x6c\x6f\xb5\x79\x83\x6e\xae\xca\x7d\xb6\x26\x3b\xfc\xe9\x75\x27\x44\x9d\xcd\xc1\xd6\x93\xc1\x56\x67\xed\x5a\x38\xa7\xda\x6e\x15\x3b\xf5\xfe\x6d\xfb\xfd\xdb\xf6\xbf\xc2\xdb\x76\x5e\xcb\x9a\xeb\x9d\xea\xef\x78\x32\xc9\xf1\x25\xfa\x99\x24\xe3\xcf\x18\x7d\xff\x0b\x9e\x4c\xec\x07\xee\x2d\x7d\x58\x01\x18\x89\x52\x74\x44\x85\x85\x08\xa0\x48\x94\xba\x60\x3f\x46\x23\x0a\xf6\x8f\x6c\x8a\x93\xa2\xc4\x49\x82\x73\xf4\xfd\x04\x12\x5d\xe0\x9f\xa2\x33\xf4\x73\x96\xc5\xe8\xfb\x69\xe5\xc3\xfb\x5d\xe5\x70\x84\x7b\xa7\x7b\x13\xa5\xd1\xd4\x7c\x0d\x3f\x18\x52\x2c\x0c\x73\x06\x30\x67\x00\xe2\xd5\xfb\xe1\x08\xe4\x3a\x1b\x98\x8c\xa2\x54\x80\xbc\x04\xa3\x62\x1b\x82\x49\x31\xc5\x10\x97\x33\x01\xf8\xe2\x59\x0d\x5c\x3c\x92\x1e\x30\x67\x75\xf5\x15\x33\x59\xdf\x5b\xf0\x95\x5c\x05\x98\xe2\x52\x00\xbe\xc3\x79\x01\x0f\x3b\xaa\xa1\x17\x1c\x44\x76\xe2\x3c\xca\xe7\x75\xdd\xa0\xf9\x12\x18\x97\x25\xc4\x91\x71\xe1\x0b\x9e\x25\x40\x05\x57\x31\x20\x05\xbb\xa0\xc2\xa0\x72\x37\x40\x12\xab\x42\x2d\xd0\x75\xb5\xd7\x02\x06\x24\xfc\xc3\x70\x33\x72\x9c\xc6\x9e\xbe\xb1\x0c\x01\xf6\x0c\x84\x3d\x17\x6a\x44\xd3\x25\x26\xf3\x6c\x81\xf3\xf2\xd2\x03\xb7\xe0\x59\x02\xf4\x55\x59\x2e\xde\xe5\xd9\x19\x89\xbd\xe4\x46\x17\xea\x82\x67\x4b\x62\x5b\x8c\x6b\x4a\x90\xc5\xd8\x2e\xd0\xce\xc7\xda\xda\x9a\x94\x85\x7f\xc6\xa3\x1d\xd4\x13\xd5\x98\x7e\x42\x73\x7b\x85\xa4\xf8\xdc\x5a\x36\xaa\xa4\xe6\x32\x94\x07\x7f\xd4\x7a\x2e\xa0\x34\x20\xcc\x2c\xef\xf1\x39\x5d\x2e\xe0\x3a\x5c\xaf\x22\x1e\xf1\xcc\x17\xcf\x9c\xbc\x62\x26\x4a\x7e\x98\xb9\x25\x53\x58\x03\x34\xf7\x2d\x2e\x9d\xdc\x85\x22\x7c\x0a\x22\xd6\x81\x03\x37\xfa\xf5\x57\xd1\x06\xa5\x6b\xb7\x0f\x8a\xc0\x01\x88\x7f\xf6\x74\x18\x45\xd9\xea\xb0\x10\x2d\x48\x28\x37\x43\xfe\x3f\x3b\x33\xe8\x9d\xe4\xd8\x2a\x8c\xa2\x3a\xf9\x84\xc6\x57\x20\x61\x34\x7a\x09\xf5\x0f\xa7\x89\x4f\x72\x0d\xb0\x1f\xce\x00\x39\x40\x4f\xb5\xcf\xc9\x99\xe0\x22\xd4\x7e\xf7\x98\x91\xc1\x75\x7f\x8f\x4a\x4c\xc3\x21\x38\x05\x2d\x30\x52\x63\xc8\xd8\x4e\x0c\x5e\x4a\xd6\x28\xb9\x79\xc6\xd7\x34\xb6\xca\x71\x51\xa1\x51\xd4\x29\x22\xfc\x61\x9d\xf2\xf4\x28\xa6\xcd\x34\xae\x17\x5e\x99\xb4\x3d\x7d\xc9\x31\x73\x5f\xaf\x7a\xf1\x19\xe3\xc5\x61\xf1\xe1\x32\x1d\x93\x74\x5a\xdb\x15\x28\x6b\xc1\xb7\xa3\x40\x4f\x47\x74\xbe\xf0\x4c\xdd\xab\x5b\x50\xc2\x28\x9f\x39\xb8\x81\x2f\x0f\x8c\x78\xc0\x27\xa0\xe0\xdb\x03\xc7\x5f\x81\x0a\x30\xfa\xe9\x40\xe9\x0f\x02\x19\xa0\x4c\xf1\xc2\x1a\x75\x8a\x04\x4f\xdb\xea\x75\x87\x68\x9e\xa7\x78\x6b\xb5\xa1\xb5\x34\x4f\xdd\x3a\x2e\x45\xed\x75\x38\x65\xa6\x57\x02\xf2\

x67\xec\x1f\x99\x0e\xc5\xbf\x1d\x38\xfd\xca\x9d\x41\xca\x14\x0f\xac\x7b\x45\x25\xca\x3c\xb7\xef\
x2d\x9c\x3e\x57\x95\x75\x72\x3c\xed\x1e\x3e\x3b\x78\xab\x35\x46\x3f\xe9\x9e\x63\x2f\x53\xb6\x
51\xeb\xaf\x40\x99\x32\xd9\xf4\xce\x66\xaa\xf9\xe1\x36\xcb\x86\xac\xba\x76\x89\x49\x0e\x3a\xa9
\x71\xb4\x00\x1b\x6e\xed\xe2\xc3\x83\xff\xc3\xe7\x07\xef\x8c\x95\x4a\xcb\xe9\x76\x36\x84\x09\x
7e\x74\xb1\x51\x19\x90\xe5\x1b\x0f\xc5\x28\xc4\x80\x37\x23\xd6\x21\x38\xa3\x90\xdc\xb2\x2e\x6
2\xe1\x89\xe4\xb4\xb0\x33\x71\xe1\xa1\x67\xde\x55\xa5\xa0\x04\xe9\x8a\x1d\x24\xcd\x62\xdc\x0
d\x0c\x88\x29\xdc\x2a\x87\xa8\x4b\x45\x84\x4f\xe3\x84\xe0\xb4\xfc\x07\x03\xef\xaa\xbb\xac\x7e\
x70\x93\xd6\x70\x79\x9e\xe5\x9f\xab\x1a\x4c\x71\xf9\x89\x83\x5a\x20\xa6\xc3\xf1\xd0\x5e\x93\xb
7\xec\x16\x58\x53\xe2\xe5\xbc\xaa\x5f\xb8\x9c\x7d\x82\xb9\x1e\x67\xc9\x3f\x7e\x87\xfe\x9d\xcf\x
48\xb1\x90\xbe\x55\x9d\xee\x15\xb3\xd9\xad\xd1\x06\x3f\x4f\xbd\x9c\x9f\x14\xcf\xb3\x34\x65\xfe\
x5e\xb4\xe5\xd6\x37\x68\xaf\xe7\xdd\xdc\x1e\x3c\xf0\x6e\x7a\x7a\x95\xbd\xbe\x7f\xbf\x61\x2f\x9c
\x85\x04\x5d\x49\xf3\x60\x62\x06\x9e\xd7\xb9\xfc\xe1\x55\x9c\xd2\xba\x85\x37\x42\x5d\x9e\x67\x
0a\x22\xe3\x18\xd0\x09\xb7\x37\x69\x92\x7e\x80\xe8\x84\xdb\x5b\x34\x4d\x09\xef\x9d\x70\x7b\x
57\xa6\x30\x41\xa7\x13\x6e\x3f\x91\x49\xba\x28\xde\x09\x77\xb6\x65\x06\x5d\xe1\x9d\x70\x67\x
47\x25\x28\x11\xbc\x13\xee\xa8\x4a\xd5\x21\xae\x13\xee\x7c\xeb\x24\xe3\x72\xd6\x09\x77\x9e\x
38\xe9\x29\x2e\x3b\xe1\xce\x77\x4e\xba\x10\x5b\x3b\xe1\xee\xa6\x93\x59\xcc\x66\x9d\x70\x77\x
cb\x4d\xa7\x92\x6b\x27\xdc\x55\xdd\x17\x27\x92\x4e\xb8\xfb\x8d\x4c\x34\x8f\xb9\x9d\x70\xf7\xb1
\xcc\x12\x32\x46\x27\xdc\xfd\xb6\x5e\x13\x77\x7d\x1a\x6c\xef\xdc\xeb\xc9\xee\xf5\x64\x7f\x6d\x3
d\x99\xe6\xfb\x36\x4a\x12\x78\x9d\x7e\x3b\x47\x90\x9a\x3e\xca\xd1\x5c\xf8\x54\x17\x22\xce\xc4\
xcb\x33\x66\x18\xac\xa9\x04\xa0\x37\x02\x4e\x45\x9d\x68\x0a\xaf\xe2\xaa\x55\xbc\x7a\x95\x1f\x
49\x52\xda\x4a\x88\x09\xa4\x09\x88\x73\x16\x3c\xc5\x04\x11\xbc\x88\x67\x4a\xf7\x90\x07\x49\x6
2\x0c\xc5\x94\x8c\xcc\x93\x50\x00\x77\x82\x01\xb2\x6c\x52\x2a\x74\x14\x66\x82\x7e\xa2\xfd\x85
\x5d\x03\xd2\xff\xf4\x64\xc7\xd2\x8a\xed\x42\x4e\x0f\xeb\xe3\x04\x5b\x62\xab\x70\x28\xbc\x2f\x7f
\x5d\x5d\x41\x00\x0d\x64\x3f\x1a\xa7\x89\x90\x7a\xd2\xa5\x62\x28\x38\x26\xef\x06\xa8\x5b\x66\
xec\xe7\xe9\x80\xa1\x59\x0b\x98\x36\xf1\x5c\x3f\xf2\x66\x4e\x26\xa7\x60\x7f\x27\x4d\xcb\xf8\x8d
\x64\xdf\x13\x75\xd7\xaa\x86\xf6\x87\x16\xdf\xd7\x88\x87\xf9\xdf\x80\x8e\xb0\xe3\x8d\x8a\xa2\xa
5\x1a\x14\xb7\xa3\xea\xfd\x07\x3c\x64\x57\x78\x35\xf0\x6c\x3e\x12\xd1\x9f\xb6\xd7\x61\xdc\x13\
xf6\x34\x8e\xca\x48\x8c\x80\xfe\x1e\xd0\x7f\xd0\xbe\xf6\xfb\xea\x0a\xec\xe9\x24\x40\x99\x2d\xc8
\xb8\x10\x20\xfc\xeb\xea\x4a\x85\xef\x03\xe5\x20\x6d\xfa\x23\xcd\x33\x01\x4f\x36\x4f\x07\x05\xe
5\x08\xd2\x47\x33\x85\x9e\x73\x09\x47\x51\x98\x3b\x5d\xbf\x78\xa6\x4b\x6f\x65\x9f\x5b\xe9\x71\
xf1\xce\xbd\x36\xed\xfd\x22\x9f\xb9\xf6\x4f\x36\x4f\xb5\xf7\x1b\xeb\xd0\x7e\x1f\x7d\x01\x9b\xe9\x
28\x4d\xb3\x12\x4d\x48\x1a\xb3\x7e\x91\x74\xca\x1a\x7a\x2a\x9b\x1f\x67\x69\x91\x25\x78\x70\x
1e\xe5\x69\xaf\xab\x97\x60\xae\x36\x28\x2f\x4e\xb2\x69\x57\xb3\x99\xe3\x3d\xa6\xa8\x70\xdc\xb
5\x60\xce\x86\xf4\xd0\x3e\x30\x77\x3d\xdf\xea\x0c\x58\xb7\x02\x93\x20\xcc\x33\x14\xd4\x28\x3c\
xa5\xc1\x14\xb7\x58\x8e\x17\x78\x4c\x45\x00\xcf\x7a\x0c\xc0\x9d\xcb\x28\x1a\x7f\x96\x41\x08\x
e1\x45\x33\x3f\x9b\x8a\x0b\xcf\x5e\x94\x4f\x97\x60\x52\x7e\x22\x7f\xe9\xb1\xf6\x0d\xdb\x34\x51\
x23\x04\x8f\xad\x2d\xa6\x3b\x9d\xea\x39\x10\x74\xe2\xb7\xcc\xe7\xf0\x8a\x6d\xa4\xcb\x24\x71\x
d0\x9d\x09\x4a\xe3\xae\xb3\xd4\x09\x58\x40\x4c\xb4\x30\x4d\x4c\x91\x0a\x98\x1c\x8c\x88\xa9\x
e3\xd3\xe4\x6f\xc6\xd9\x2b\x26\x2c\x0b\x04\x5f\xa7\xc7\xac\x5e\x3f\x50\x2d\x68\xa8\x6d\x9e\xa2
\xa8\x2c\xa3\xf1\xec\x63\xf6\x5c\xb8\xcf\xd1\xe7\x4a\xf8\xd4\xd1\x4f\xdb\x6a\x4e\xd9\x80\xd9\xa
7\x33\x0e\x51\x74\x10\x25\x89\xdc\x48\x38\x70\xc5\x69\xc2\xe9\xa6\x3c\x5a\x78\xce\x16\xde\xc
3\x05\xd0\x68\x27\xdc\x06\xb9\x9e\x2d\xf7\x4e\xb8\x0d\x52\xbb\x1e\xed\x69\x07\x80\xad\x1d\xb

0\x13\xee\xee\x50\x61\x79\xf7\x5e\x58\xbe\x17\x96\xff\x63\x84\x65\x38\x75\xdf\x55\xa4\x88\xbf\x17\x59\x9a\x2f\xc6\xa6\xa0\xf9\x0b\x4b\x94\x57\x7c\x79\x9e\xd9\xb2\x2f\x4b\x93\x22\xa8\xab\x9c\xa0\x83\x35\xa4\x4b\x47\xb8\x04\x74\x7c\xaa\x14\x31\x79\x46\xc1\x43\x02\x37\xb8\x17\x8b\xe2\x58\x78\x82\xa3\x7c\x98\x17\x06\xe7\xba\xd0\x35\x9e\x60\x59\xc1\x45\x71\xec\x31\xe3\x43\x7c\xfc\xac\x50\xa9\x0c\xe8\x86\x6b\x30\x4e\x9d\x15\xc7\xb1\x4f\xd8\xf6\x0d\xbc\x60\xf1\x84\x05\x44\xe3\x88\x04\xd3\xae\xeb\x3f\x87\xf1\x76\xcd\xb7\x91\x9b\xaf\x93\x25\x7e\x8d\x6e\xba\x53\xa0\xee\x73\xd2\x98\x29\x98\x04\x6c\xa0\xd5\x8d\xf3\x3c\xe0\x22\x68\xe1\x0a\xc3\x8c\x7c\xd8\x2f\x2e\x25\x2a\x00\x8e\x1f\xdd\x31\x9d\x44\x65\x80\xe0\x75\x6c\xc5\xc3\x17\x5e\xe5\x09\xc0\x9c\xea\xe7\x82\x4a\x49\x9d\x15\xa9\xa8\x96\xca\x33\xa2\x3f\xbc\xd2\x81\x23\xf4\xd8\x05\xd6\xf9\x22\x1a\x90\xe2\x1f\x51\x42\xe2\xf7\xb8\x58\x64\x69\x81\x79\x53\xce\xa3\x1d\x67\x0c\xfe\xf6\x7a\x6c\x8d\x0d\x0e\xd3\x33\x6f\xad\x7b\x4e\xa5\xd7\x6e\xff\x2a\x2b\x67\x3e\x5f\x9c\xc1\xb2\x3d\x17\xde\x96\xfb\x32\x78\xe3\x03\xde\x07\x78\x75\xae\x27\x38\xf1\xaf\xd5\x54\xc8\x83\x0d\xf2\x8b\x12\x40\x59\x4a\x33\xc9\x06\xdf\x09\xb7\x41\x83\xc6\x57\x64\x27\xdc\x01\xeb\xb4\x56\xf1\x81\xef\x37\xfc\xfb\x0d\xff\xcf\xbb\xe1\xab\xfd\x5e\x8a\xe5\x77\xa4\x1b\x6b\xa9\xa4\xa2\x47\x9d\xdc\x02\x2b\xb8\xac\x3f\x84\xcc\x55\xf5\x68\x02\x4e\x7b\x69\xa1\x2b\xc0\xc4\x13\x0a\x0e\x7d\xa0\x1d\x42\x34\x30\xa9\x2a\x34\x82\x15\xfb\xf6\x4f\xa6\x57\xd2\x9f\xa5\xc0\x36\x6f\xbf\x6d\x64\x70\xcf\x15\xd8\x3b\x01\x25\xe5\x02\xb0\xb3\xbd\x46\xc2\x03\xac\x99\xea\x6d\x80\xfb\x08\xf5\x57\x6d\x3e\x0e\x1b\x91\x80\x97\xb3\xee\x73\xa2\x11\xf1\xe8\x3f\x34\x6f\xb1\xc8\x72\xcf\xca\x42\x03\xef\xef\xa3\xae\xd6\xa7\x2e\x7a\xf0\xc0\x70\xff\xaa\x1d\x98\x59\xb3\x86\x8f\xf0\xeb\xbe\xb5\x0d\xd7\x35\xe8\x71\x28\x8b\x7a\x90\x58\xb1\x5d\x43\x1e\xf3\x33\xeb\xd9\x19\xac\x8a\x28\x58\xe1\x69\x1a\x68\x8f\x9f\xda\x19\x42\x19\xa8\x44\xa3\xa6\xde\x11\x6a\xab\x16\xd2\xa3\x0c\x05\xc4\x5d\xcd\xb0\xa3\xb5\xf7\xf1\x44\x14\xc7\x82\x86\x0b\x75\x0c\xd7\x69\x43\xa4\x5d\xcb\x9a\x2a\xe9\x89\x91\x8a\xbf\xca\xda\x93\xbd\x3a\xae\xdf\x9c\x50\xb4\x77\x4b\xab\xcc\xbe\xae\xa2\x92\x6a\x1f\xd9\x9f\x4f\xb8\x9c\x09\x3d\xb3\xea\xa4\xf9\x6a\xbe\x51\x87\x3a\x71\xd4\x1c\x0a\x01\x4a\x47\xda\x62\x5e\x19\xb7\x68\x35\xa9\x8c\xdf\xdc\xdd\x8c\xda\xf5\x35\x2b\x6a\x04\xc3\xbb\x8b\xb9\x65\xbc\xd7\xd2\x27\x73\xce\xca\xd5\x8c\x92\xc7\x9a\x93\xe7\xaa\xae\x58\xc7\x2a\xa7\xf3\x20\x49\x6a\xa7\x0b\x80\xf8\x0d\xcf\xca\x04\xc6\x74\xa0\x0d\x1d\x5c\x9d\xda\x8c\x77\x47\xae\x52\xad\x8a\xda\xea\xc8\x4d\x3a\xda\x00\x1b\x3d\x31\xe9\x53\x5c\x16\xdc\x6c\x25\xb9\x44\x31\x5e\x24\xd9\x25\x8e\x85\x29\xdf\x28\xc9\xc6\x9f\xc7\xb3\x88\xa4\xb6\x8b\x58\xa8\xed\xc7\x2c\x17\x3d\xf2\xbc\x56\x16\x07\x56\x1f\x49\x8a\x75\x79\x2d\x55\x8b\x6b\x86\x8b\xcd\x63\x71\xab\xa1\x5e\x77\x55\xb4\xa8\x9b\x3a\x88\x96\x34\x85\xa5\x22\x5f\x08\xf5\x0a\xf1\x27\x1c\xfd\xee\x8f\x10\xdf\x78\x5f\xbc\xec\x82\xfc\xe1\x10\x9d\x47\x84\xe9\xc9\x41\xe4\x5a\x94\x4a\xf7\x2a\xae\xc8\xcc\x79\xe7\x4b\x41\x86\x9a\x55\x1d\xc3\x7d\xd3\x73\xeb\x3a\xa6\x1b\xdf\xba\xd1\xbe\xbd\x2b\x41\x7f\x37\x36\xf6\xcc\x63\xd3\x70\x88\x8a\x32\x5b\x30\x5d\x2d\x49\xa7\x28\x9a\xd0\xae\x7c\xb3\xc9\xe6\xaa\x40\xbd\x92\xcc\x71\xb6\x2c\xfb\xce\xd1\x91\x21\xe0\x07\xf4\xcd\xa6\xf7\xb0\xc8\x7a\x3f\xa0\xb5\xff\xcc\x2b\x57\x9e\xd8\xfb\xe8\xcb\xb5\xe7\x4c\x67\x23\x90\xb9\x35\xf1\x9e\x43\xe5\x8c\x78\x4f\x9b\xea\xe4\xa7\x1c\x8b\x4a\xc6\x04\x17\x25\x11\x5b\x19\x63\x4a\xd8\xe0\x64\x74\x44\x25\xe6\x65\x1a\xdb\x18\xe8\xfa\x0e\x9f\x38\xd1\x9c\xa7\xe8\x7f\x8e\x3b\xd3\x1b\xb7\x4a\x97\x9f\x5e\xb3\xf4\x40\xe0\x62\xcd\xa0\x9a\x29\x2e\x3f\xaa\xa6\xde\x33\x52\x53\x1c\x45\xeb\xc6\xab\xa8\x98\xe9\x44\x15\x08\xc2\xec\xfb\x8f\xf0\x64\xd2\xe3\x00\x7e\x6a\xf3\x16\xf2\x76\x10\x02\x9f\xf0\xba\x06\x63\x73\x01\x9a\x3d\x82\xe8\x28\xfe\xee\x88\xbf\x2a\x9f\xcf\x8f\xa5\xcf\xe7\xaa\x3f\x32\xe9\x99\x14\x77\x75\x85\xd6\xa1\xc5\xda\x62\x48\xb2\x6e\x0f\x6d\xea\x7f\x37\x59\x02\xfa\x5f\xcb

\xe5\x60\x0f\x29\x8b\xb5\xe0\xb2\x3b\xb5\x33\x23\xfe\x86\x43\x79\xc1\x97\x64\x53\x8d\x6a\xe1\x58\x21\xd8\xf8\x7a\xb7\xdf\xd0\x3c\x32\x44\x35\xc9\x51\x2b\xa6\xba\x45\x65\xc3\x21\x62\x9b\x95\x10\x17\xa2\x34\x46\xfc\x66\x04\x45\xd3\x88\xa4\x7c\xe5\x9c\x63\x1e\x17\xac\xe1\xcf\x2f\x7b\xda\x1b\x60\x43\x0d\xb6\xac\xe3\x6c\xff\x0d\x43\x1a\x33\xa7\x4e\xfc\x36\x90\x6e\x09\x74\x77\x2c\xf0\x38\x4b\x63\x44\x19\x6e\x63\x25\x1a\xe9\x36\x13\x2b\x32\x38\x22\xe8\xc2\xda\x76\xd8\xeb\xf7\xe4\x8e\x3b\xa4\xfb\x7e\xd6\x44\x09\x7e\xa2\xd5\x38\x65\x51\x66\x39\x8e\xa5\x1f\x68\x26\x81\x80\xc6\x67\x1a\x15\x28\x9a\xd3\x0d\x69\xe0\xe5\xd7\xf6\x5f\x25\xff\xb6\xff\x3c\xee\xa9\xef\xa2\x8b\xf5\x3d\xbc\xae\xcc\xad\xe2\x18\x6e\x09\x1b\x52\xd3\x4e\xb6\x3d\x50\x68\x57\x0c\x82\xd0\x7f\x8c\xe8\x31\xfb\x52\x3e\xd7\xb7\xa4\x38\x0b\xac\xe1\xd0\x60\x57\xaa\x1f\x18\xe0\x54\x15\x8d\x88\x71\xb9\xc0\x5e\xfe\x60\x71\x7c\x87\xb4\x68\x44\xd0\x3e\x85\x14\x72\xd6\x43\xa6\x09\x6d\x1e\x93\x3a\x21\xa5\x28\xd2\x44\x53\x5e\x5c\xd4\x22\xc6\x96\xe2\x73\x99\x24\xc6\x94\x5e\x5e\xeb\xc4\x60\xe9\x46\xb6\x84\x31\x41\x94\xf4\x57\x2c\xba\x5d\x53\xd4\x96\x83\x0d\xc9\x82\xbb\x53\x10\x8a\xe2\xd8\x29\xed\x93\x94\x39\x84\x94\x96\xd5\xf1\x4f\x24\xc9\xb6\xd4\xc4\x43\xa1\xa1\x9a\x08\x8a\x52\xdf\xf5\x0b\x92\xf2\xb2\x3c\x17\xea\x14\xd5\x13\x33\x1b\xc8\xf9\xd4\x79\xd2\x70\xc8\x62\xac\x29\x83\x09\xa3\x52\x65\xf6\xf0\xe5\x7a\x8f\x02\x8b\x71\xaf\x9b\x6d\xf3\xa1\x6a\x15\xc3\xa9\x35\x87\x57\x30\x40\x9a\xfa\x03\x83\x84\x8c\x31\x5c\x1e\x28\xd3\x0b\x2b\x0c\x98\xcf\x0c\x04\x4c\x39\xaa\x8d\x3f\x90\x63\x00\x52\x0c\x16\xd9\xc2\x70\x32\x65\x76\x2f\x89\x8a\x92\x43\x3a\x55\xfb\xbb\xc3\xe3\x4b\xd0\x82\xe0\x91\x75\x5d\xbe\xf5\x80\x80\x94\x90\x6e\xf7\x49\xa1\xb0\xa1\x4b\x5a\xb8\xfb\x01\x8b\x53\xf0\x03\xda\x74\x63\xd3\xe7\x3a\x35\x1f\x88\xd5\xd9\x7c\xae\x17\x7f\xb7\x52\xf2\x69\x68\xb2\xd8\x1f\x57\x90\xfd\xff\xff\x9f\x34\x9b\xd3\xc7\xb5\x6e\xf6\x79\xb0\x88\x2e\xa3\x51\x82\x7d\xfd\x73\x25\x7c\x66\x0b\x55\xe0\x34\x56\xa1\x69\xd2\x2c\x7d\xc4\x2b\xd1\xf1\x61\x73\xfe\xeb\xaa\xb9\x07\x07\x5f\x94\xd9\xf9\xb5\xaa\x3d\xb1\x56\x02\x18\xb2\x56\xab\x98\x21\xb0\x63\xdb\xd8\x67\x15\xed\x99\xb3\x58\x79\xc9\xa7\x1f\x52\x8d\x63\xbd\x10\xe5\xa2\x38\xb5\x0f\xfe\x31\x46\xe7\x51\x21\x65\xc4\x35\x13\x57\x6c\x6d\xc3\x6d\xaa\x76\x2a\x51\x46\x56\xd6\x95\xea\x2c\x2a\x66\x3e\xa4\xd3\x5e\xe3\x3c\xaf\xba\x5c\xd4\x6f\x11\x7d\x57\x85\x75\x42\x0c\x95\x30\xe3\x98\xdd\x64\x69\x8c\x94\xf6\xc4\xdf\x56\xc5\x49\x0a\xed\x43\x99\x0a\x79\xaa\x52\xe8\x9b\x90\xbc\x28\xab\x65\xbe\x15\xc5\xb6\x0a\xa5\x86\x4f\x93\xe1\xbb\x51\x35\xbe\x9a\xbc\xdf\x41\xc8\x3d\x36\xf0\xa6\x79\xb6\x1a\x6b\x8b\xf2\x46\x54\xaf\x32\x74\x3f\x53\x93\x6a\x76\x06\xc4\xd5\x5f\x1c\xbb\x62\x5f\xa3\x47\xd6\x17\xcc\x46\x14\x92\xf8\xa7\x61\x53\x76\x63\x59\xca\x56\x84\x35\x29\x5c\x3d\xfb\x34\xaf\xe9\xda\x14\x6b\x26\xb2\xfe\xe1\xda\x70\x68\x6d\xc1\xc6\x9d\x8c\x8a\x7e\xa6\x69\x24\xad\xca\x7b\x6c\x63\x1e\x0e\x0d\x97\x9a\x95\x01\x68\xc7\x63\xf0\x8e\x99\xb1\xd8\x2d\x24\x9d\xd6\x88\x5b\xa6\x66\xda\x1c\x39\x9b\xc4\x6b\x97\x13\xe9\x12\x4e\x9d\x74\x83\xbe\x68\x82\x54\x5b\x21\x67\x82\xd2\x4c\xd5\x40\xd9\xdb\x22\x2a\x0a\x1c\x07\xb4\x0a\xe5\x30\x8b\x42\x14\xda\x92\x36\x79\x99\x24\x3c\x98\x01\x0b\x9d\x86\x85\xa3\xcf\x81\xa2\x69\x7f\x8a\x56\x16\xa2\x94\xd5\xbb\x52\x40\x96\x33\xcd\xbf\x1c\xc4\xaf\x80\xa8\x41\xc2\x4e\x40\x76\x29\x10\x05\x46\x78\x1c\x2d\x0b\x4c\x0f\xd7\x71\x96\x96\xe8\x3c\x4a\xc1\xcc\xa8\x58\x64\x24\x61\x17\xdc\x69\x89\xf3\x49\x34\x96\x8e\x72\x5b\x1c\xae\xdb\x1c\xa0\xed\x6d\xaa\x99\x1f\x22\xc7\xc7\xa6\x5c\xd3\xda\xda\xfc\x09\x97\xcc\x69\x2b\xdd\x1f\x03\x74\x3e\x23\xe3\x19\xd8\x01\xd0\xe5\x5d\x66\x7c\x1b\x43\x8b\x64\x59\x34\xdf\xa6\x72\x3e\xd0\x30\xbf\x8a\x79\xf8\x6d\x93\x1a\x64\xd8\xd5\x05\x55\x59\xac\x59\x80\xbc\x8d\xf0\x58\x2d\x38\x6a\x86\xc7\x37\x92\x63\xea\x64\x18\xf3\xd5\xc2\x80\x19\x97\xb7\x67\xbe\x9e\x83\x8c\xf7\x04\xdb\xe2\x46\xbc\x8a\x35\x39\xe7\x5b\xef\xc1\xb6\xe2\x55\x8a\xef\x88\xeb\xee\x7e\xca\xc6\x9b\xe1\xcf\x7d\x88\x8

2\x3c\xe7\x63\xaf\x25\x92\x45\xb7\x7b\xd2\xa2\xd9\x34\x7f\xe8\x84\xdf\x56\x19\x35\x4b\x23\x85\x4e\xb8\xbd\xe3\x5a\x39\xf3\x91\x77\xc2\x9d\xad\xeb\xd3\x60\xfb\xf1\xbd\x35\xd3\xbd\x35\xd3\x5f\xdb\x9a\x49\x33\x5f\xe6\x56\x8d\x77\x60\xbf\x5c\xe1\x14\x92\xdb\x4b\xb2\x37\x56\x47\x13\xc6\x55\xc3\x0a\x35\x8c\x26\xdd\xf1\xb3\x2b\x2f\xae\x05\xc2\x62\xb1\x4f\x80\xc7\x61\xf1\xda\x0f\x78\x7a\xa0\xb7\xc7\x9f\x71\x83\xf7\x3f\xd5\xd6\x2c\x2b\xf4\xbb\x23\xb7\xb9\xe7\x47\x6f\xdf\xbe\x7c\xfe\xf1\xf0\xe8\x2d\x7a\xf9\xfe\xfd\xd1\xfb\x10\x3d\x97\xca\xd3\x31\xab\x92\x1d\x9e\x63\x8c\xba\x1b\x88\xd6\x87\x36\xba\x03\x7f\x1f\x94\x5f\x97\xb6\xa3\x95\xef\xd3\xd9\x79\xbd\xa4\x84\x4a\x58\x65\xfe\x26\x84\x19\x6a\x88\x6c\x93\xda\xbe\xa9\xdc\x9a\xe3\xa2\x88\xa6\x18\xed\xa3\xf5\x75\xfe\x40\x8f\xee\xa0\xfc\xf7\x80\x05\x6c\x74\x52\x06\xa2\xd8\x53\xe4\x4d\x0e\x91\x9c\xa0\xbf\x7f\x38\x7a\x8b\xde\xbf\x7b\x4e\x01\x79\x97\x3c\x41\x0e\x79\xdf\x9c\x27\x58\x0a\x07\xbc\x6a\x73\xb4\x6a\x36\x3f\xb2\xcb\x5e\x7d\xbc\xf3\xa2\xed\x94\x7e\x3c\x7c\xf3\xf2\xe8\xf8\x63\x88\xf8\x95\x31\x25\x27\xda\xc9\x79\x81\x36\x50\x97\xfe\x17\x8d\x67\x74\x71\x76\x8d\x70\x12\xdc\x59\xe2\xb7\xf7\x1b\xc3\xfd\xc6\xf0\x9f\xb3\x31\xc0\x6b\xc5\x3f\xaa\x91\x6b\xfb\x47\xe0\xad\xde\x9e\xdf\xe1\x13\x70\xe1\xac\x87\x32\x00\x79\x10\xd2\x23\xa1\x14\x86\xc8\xcf\xdf\xa6\x42\x5b\x4a\x30\xb7\x6d\x78\xbf\xf6\xfb\xf1\x85\xb0\x84\xd5\xb4\xd6\x7a\x3e\xf3\x15\x8f\x6a\x9e\xf1\x16\x59\xda\x6f\x78\x7a\xae\x65\xa6\x59\x7a\x39\xcf\x96\xb2\x45\x99\x50\x71\x52\x12\x48\x9b\x62\x81\x2b\x16\x52\x3f\x9a\x83\x9b\x71\x27\x40\x0a\x4f\x93\x47\xa1\x67\x59\x96\x5c\x43\x74\xc3\x18\xfc\x73\xb3\x5d\x02\x33\xc8\x58\x9b\x1d\x78\x5e\x81\x63\xc3\xed\xb6\x38\x5d\x81\xbb\x70\xba\x2a\x79\xed\xc3\x35\x63\x9a\x74\x27\x53\x14\xc2\xf4\xb8\xc4\xea\xb5\x5d\xa4\x6b\xc8\x77\xef\x1f\x88\x47\x56\x20\x03\x5e\x13\x5c\x26\xf0\xdf\x15\xd6\xa2\xfe\xf2\xca\xdc\xb7\xf2\x82\x55\xc7\x36\xa3\xcf\x98\x39\xaf\x06\xf7\x3f\x16\xae\x63\xe5\xd7\xda\x1b\x9e\xc3\x5b\x41\x35\xea\xb4\xea\xea\xbc\xeb\x30\x4a\x74\x5d\x6b\xf7\x14\xbd\xb6\x1f\x1d\xac\x50\xcf\xd0\x4a\xee\x89\xbb\x66\x5c\x7a\xd1\x7a\x7a\x58\x69\x44\xc2\x07\xf8\x8d\x86\x53\x90\x69\x1a\x95\xcb\xdc\x1e\x8e\x9e\x5e\x35\x1e\x1d\xa6\x7a\x3c\x12\xaa\x6e\x40\xf0\xf0\xbf\x7d\xff\xf9\x03\x01\x41\xde\x9c\x23\x45\x69\x2c\xd5\x38\x65\x06\x31\x41\x27\x24\x8d\x12\x65\x34\x8c\xdc\xd7\x03\x3e\x9b\x4c\x7d\x61\x5b\x59\xbc\x7e\x03\x2b\x22\x0f\x9f\xe1\xfc\xb2\x9c\x31\xf5\xf0\x7c\x44\x80\x67\x64\x2c\x4a\x2b\x74\x8e\xb5\x18\xd7\xa2\xcb\xe3\x53\x83\x77\xc7\xf1\x09\x27\x57\xb7\xfc\xa5\x3d\xa2\xbb\xf7\xbc\xa1\xfc\x5c\x48\xc7\x16\x4d\x2e\x39\x84\x12\x71\xdd\xda\x7a\xdc\x7e\xf2\xca\xd9\xc3\x50\xee\x95\x8f\xf9\xa3\x14\xe4\xde\xeb\xbb\xfa\x43\x3e\x4f\x1f\x45\xc7\x6e\xcb\xd3\xb5\x40\x79\xb5\x0c\x1d\x1c\x0c\xf3\x60\xb6\xbc\xfc\x09\x81\xa0\x2e\xd6\xd5\xfb\x99\x1b\xde\x9e\xd2\x8d\x4a\x4e\x97\x49\x52\xf1\x42\x44\xa9\xf1\x90\x71\xfd\x66\x34\x60\x6a\x61\xa1\xde\xaa\xc8\x68\x90\x69\x8d\xea\xac\xe6\x8e\x9d\xcf\x82\xf3\xc6\xa4\xc7\xf6\xb1\x00\x9d\xd9\xd7\xd5\x7d\x5f\x77\x5b\xd7\x06\x7d\x6f\xa0\x3c\x93\x58\xc6\x59\x3a\x8e\xca\x9e\x41\x05\xfd\x6a\x47\x30\x95\xec\x8f\x7b\x81\xa9\x66\x7f\xf6\xb6\x8b\xab\x38\x5d\xcc\x14\xfe\x2e\x2f\xe3\xdc\x81\x1b\xe0\xc0\x59\x81\xd5\x12\xcb\x66\x1f\x3c\x00\xcd\x83\xd9\x8b\xfa\xfd\xba\xc6\x7b\x0d\x20\xe1\x0e\xfd\xd7\x44\xf9\xd4\x5a\x66\x4a\x90\x7c\x6a\x94\x0c\xf5\x2f\xee\xdb\x66\x4b\xf3\x25\xc2\x07\xc8\x6f\x3d\x64\xbd\xf6\x93\x27\x36\x9b\xe8\x8b\x94\xd7\xf4\xfa\xb6\xfb\x7b\x74\x89\xfe\x92\x91\xb4\xd7\xe9\xb8\x95\xcb\xd7\x65\x8c\xde\x18\xa2\xf4\x4b\x05\x90\x12\x7b\x74\xbd\xf7\x03\xbd\x47\xfd\x3d\xa4\xc6\x9c\x66\xe5\xa1\xd1\x59\x89\x43\x8f\xcb\x1e\x05\xdc\xb2\x71\xb0\xff\xef\x07\x56\x2b\xbc\x46\x77\x5b\xd1\x78\x78\xb6\x2c\x17\xcb\xf2\x75\x36\x55\xcc\x3b\x56\x45\x85\xb2\x88\x1f\x5f\x98\xbb\x16\x4d\x4a\x33\xc1\x14\xef\x86\x71\xd9\xde\x94\x18\x0c\xbb\x60\x32\xb8\x6b\x8e\xe3\xe5\x18\x6b\x13\x16\x8d\xc7\x01\xe2\x2e\x1d\x75\xae\x12\x8d\xc7\x27\x3c\x99\x71\x48\x8a\x18\xfe\x2d\x68\xfd\xa9\x3

9\x6f\x83\x62\x46\x26\x65\xaf\x8f\x42\x07\xab\x22\xcb\x51\x62\x45\xe3\xb1\xd0\x5a\x31\xd3\x69\x46\xdf\x38\xc1\x25\x16\xe3\x50\xbe\x86\xcc\x74\x46\x5a\x37\x60\x1c\xda\xd5\x11\x7f\xa5\xc1\x17\x38\xdd\xf8\x99\x54\x57\xe9\xa6\xe0\xae\xa4\x24\xa3\xe1\x7a\x51\xc8\xe3\x06\xc1\x96\x85\xfe\xe8\x8e\x8d\xb6\x9b\x1d\x1b\xd5\x15\xdf\xaa\xb6\x6f\x33\x2b\x40\x86\x3c\x68\x78\x52\xd0\xee\x92\xd5\x2d\xad\xe5\x41\xc9\x11\x31\xff\x18\xae\x94\x2a\x09\x59\xb7\xa2\x6f\xf1\x3e\xd0\x7a\x20\xe6\x7d\x1c\x58\x4b\x8a\x5f\xcb\x6f\x13\x05\x35\x4f\xb1\x55\xec\x4f\xd8\xf5\x41\x4b\x27\x1a\xc0\xa9\x41\xba\x3e\x00\xdd\x15\x94\xa2\x05\x2f\xe8\x89\xe4\xf7\xac\xed\xd3\xca\x01\x18\xd6\x0a\xde\xbb\x58\x03\x97\x9a\x73\xa9\xba\xab\xd8\x26\x97\x53\x37\xf4\x32\xf5\xa4\x8d\x36\xfe\xb6\xfe\x22\x87\x7e\x3d\xe5\x1b\x46\x83\x1e\x5d\x60\x7d\xec\x0c\x3d\x6c\xc6\xda\x70\x88\x3e\x1e\xbd\x38\x0a\x51\x8e\x99\x21\x54\x80\x8a\x8c\x9b\xac\xc8\x1b\x2e\x65\x03\x13\x31\xad\xd7\x80\x96\x83\x60\xdc\x29\x1e\xe3\xa2\x88\xf2\x4b\xba\x58\x20\xfe\x6c\x41\xc9\xad\x0b\x4e\x7f\xc1\xe5\x32\x3a\xcf\xf2\xcf\x4c\xd0\x9b\x2f\x93\x92\x2c\x12\x2d\x78\x81\x19\x2e\xc4\xef\x29\x68\xf8\x10\x79\x5f\x2e\x7d\x23\xec\xaa\x59\x1d\xa6\xf9\x80\x68\xde\xb0\xdd\x54\x8d\xe1\x98\xed\x1a\xe6\x21\x45\x96\x1a\x08\x1c\xf9\x7c\xc1\xac\xd3\xce\x9d\xb8\xb0\xa7\xbe\x23\x44\x15\xac\xc5\x4b\x91\x63\x57\x68\xf6\x93\xbb\x46\xf2\xd5\xd4\x60\x7e\xe8\xad\xa7\xd2\x70\x59\xd5\xcf\x09\xde\x1e\x93\x03\xe0\x39\x7d\xb3\x1c\x1f\x36\x58\x8e\x64\x7a\xdc\x94\xc6\xec\xa2\xc7\xe2\x92\x17\x2b\x70\x69\x05\x47\xf1\xb9\x8b\xaa\x3d\x8b\xd5\x4f\x37\xc1\x35\xe3\x55\x30\x9e\x21\x57\xd1\x0b\x52\x71\x3d\x2e\x57\x1e\xb6\x2c\x78\x07\x03\x47\x9a\xbd\x26\xbe\x18\x18\xec\x48\x7d\xec\x21\x01\x20\xb8\x10\xfc\xbf\x27\x52\x25\xcb\x61\x3f\x64\xba\xc6\x68\xc4\x4f\x53\x88\xc4\x17\xfc\xa5\xb4\xcb\xcd\x19\x1a\x94\x93\x9f\x0a\xfe\x5c\xc1\x91\x3b\xe1\x0e\x38\x03\xd2\x3d\x6f\x53\xc6\xfc\xdd\xfd\x35\xe9\xfd\x35\xe9\x5f\xfc\x9a\x94\x5d\x91\xf2\xd7\xb3\xff\x11\x21\xe5\xee\xd4\xeb\x36\x1c\x02\x1e\xa2\xe7\x59\x7a\x86\x29\x2b\x8a\x78\xc0\x5c\x38\x04\xc3\x59\x00\xcd\xf0\x85\x0c\xc2\x4d\x09\x38\x4a\x8a\x0c\x45\x49\x92\x9d\x17\x70\x4c\x62\xba\xba\x62\xb0\x46\x2b\x12\x82\xff\x1b\x72\x81\xe3\x6b\x96\xb5\xe6\xde\x71\xac\xc9\xb8\xf0\xc2\x09\xb2\x64\x54\x4c\xf3\x27\x4f\x9b\x3d\x53\x3b\x8a\xae\xae\x44\x38\x63\x95\xd1\x95\xea\xd4\x6e\xdf\xd6\x04\xb0\x83\x1c\x17\x91\x98\x8e\x96\xf5\xa1\x27\x54\x8c\x46\x43\x4c\x09\x71\x34\x01\xad\x73\x1f\x6a\xdf\x74\xea\x04\x48\xce\xf7\xf5\xc7\xa1\xc6\xfd\x91\x88\x1b\x24\xdb\x81\x23\x17\x15\x35\x29\xa7\x15\x17\x41\xb6\x05\x6a\x26\x55\xfd\xfc\xb0\x15\x40\x2c\x7f\x9c\x93\x09\xb8\xc8\xc8\xf1\x38\xa2\x1c\x47\x8b\xf6\xf2\xe0\x01\x4a\xa2\x5f\x2f\x51\x92\x45\x31\x8a\x2f\xd3\x68\x4e\xc6\x28\x4b\x71\x01\xad\xf1\x09\x51\x0d\xf1\x50\xc8\x99\x54\x12\x00\x94\xb0\x6b\x17\x8d\x3b\x50\x74\xb6\xa6\xb8\x3c\x92\x27\x64\xaf\x57\x72\x47\x45\xc0\x71\x2d\x40\xaa\x6f\x36\x0c\x6d\x7e\xe5\xf5\x0a\x93\x85\x87\x5c\xc8\x5e\xe6\x98\x2b\x02\x03\xb8\x74\x1b\x33\x2a\x66\x87\xd8\x19\xbe\xd0\xeb\x52\x7a\x4d\x2b\x41\x73\xba\xd8\x00\x6a\xe8\x24\x99\x4a\xd2\x35\x77\xb2\x74\xf0\x0c\xaa\x8f\x9e\xb2\x8e\x42\x15\x9c\xe8\xfb\x28\x14\xf4\xcf\xc1\x1c\x75\x37\x5b\x33\xb2\x09\xd6\x8d\xd0\xea\x96\xf2\x29\x42\x49\x4e\xe5\xd2\xaf\x40\x2b\xab\x97\x53\xe9\x91\x88\xdd\x2b\x32\x23\x3d\x72\xaf\xa8\x15\xae\x88\xc2\x95\x6e\x8e\x06\xb2\x5c\x5f\xef\xc5\x4d\x6a\xe2\xa5\xfa\x42\x80\x13\xca\x8b\x83\x38\x66\x06\xfa\x52\xa7\x14\xa5\x31\x2a\x70\x59\xa0\xe5\x02\x32\xb8\x7c\x0e\x8b\x88\x94\x38\xa7\xdc\x34\x3b\xe3\xe2\x07\xf7\x91\x39\x58\x5b\xd3\x8c\xf4\x5f\x67\xd3\xe2\xa0\xfc\x50\x46\x79\xb9\x66\x2b\xde\x0a\x9c\x4c\x64\x22\x25\x04\x52\x66\xa9\xe4\x65\x66\x61\x23\xea\x14\x4e\x26\x8e\x5f\x18\xf1\xca\x6b\x8a\x4b\xa6\xd1\xa1\x85\xad\xa7\x5e\x70\xd0\x56\xa3\x2b\xa0\x57\x62\x89\xad\x5b\x6b\x8c\xb6\x32\xf0\x2d\x34\xc8\x98\xe2\xb2\x67\x3d\x3a\xe1\xf6\x7d\x8e\xb8\x3f\x1c\xa2\x38\x4b\xbb\xfc\x99\x22\xed\x23\xc7\x16\x18\x13\xc2\xed\xaf\x48\x14\xb6\x38\xe0\x5d\x61\x3

0\x18\xa0\x5f\x96\xcc\xb9\x2c\x6d\x93\x32\x21\xe7\xe0\x58\xf1\x32\xaf\xe6\x55\xde\xb5\xfd\x04\x
d3\x5a\x62\x72\x18\xfe\xc3\x16\xcb\xf4\x9e\xd0\x98\x79\x63\xd3\x3b\x41\xf6\x7a\xc4\x34\x86\x34
\xfa\xd7\xec\xdb\xf3\xeb\x51\xec\x22\x4b\x12\x46\x3e\x7e\x6a\xe5\xb4\xa9\xc0\x6c\xba\x94\x5c\x
1b\x14\x97\xe9\x1b\x69\x9c\x6a\x10\x4b\x56\x41\x2e\x7c\x46\x33\x67\x4e\x85\xe5\x01\x25\x3d\x
31\x56\xdf\x24\xf8\xde\xed\xf8\x68\x22\x6b\x7d\xa4\x6d\x4b\x1d\x37\xa3\x0c\x65\xbc\x0b\x43\x5
3\xea\xdb\xa7\x56\x82\xaa\x24\x14\x85\x5c\xd2\xb9\x15\x7a\x6e\x47\xa4\x95\x07\x63\xe8\x93\xe
d\xe0\x98\x32\x9e\x77\x59\x92\x50\x3e\xa3\x7a\xc2\x68\x30\x64\x45\x88\x88\x5f\x0f\x67\xaf\x01\
xa5\x38\x18\x9a\x62\xfe\x0b\x6e\x70\x7e\xc2\x30\x05\xe4\x78\x18\x9f\x06\xe2\xa2\xc6\x48\x0e\x
14\x31\xf2\x1c\xdd\x35\x0e\xd3\x94\x02\xfd\xd2\xdd\x52\xc4\xc0\x73\x48\xdc\xb5\xd2\x93\x37\x1
b\x72\xa1\x79\x14\xe4\x01\x3f\x50\x3c\xcf\x31\x8c\x06\xec\x97\x9f\x7b\xde\xd8\x01\x9b\x63\x4a\
x5c\xed\xd6\xd1\xc0\x84\xa6\xe4\x5a\x45\x5b\x54\xa9\xee\xa9\xd4\x77\xf8\xf5\x4a\x66\x67\x34\x
11\x03\x55\xfa\x1f\xaf\x34\x4e\x54\x62\x99\x92\x04\xad\x67\xbb\x00\xe7\x20\xc0\x4c\xd0\x20\xc5
\x6c\xbb\xef\x94\xe4\xaa\xe0\x16\x26\x32\x83\x6f\xb3\xcf\xab\xf2\x15\xab\x73\xb2\xf4\xcb\x16\xf9
\xbb\xb2\xdf\x83\x14\x9f\xeb\x77\x2d\xce\x3b\x74\xc1\x18\x49\x6c\xb8\x58\xf3\x33\xc4\x86\xa5\x
de\x1b\x8f\x3c\xae\xab\xc6\xa3\x46\xde\x87\xa4\xef\x28\x2f\xb1\x3a\xe5\xbd\xfa\x51\x63\xe5\xd1
\xf9\x8a\x5d\xd7\x6f\xfa\x53\xfe\x1c\x83\x32\x9c\xca\xdc\x0b\x9c\xc6\x60\x93\x25\xe7\x23\x2a\x4
0\x21\x90\x16\x94\x8c\xa4\xfb\x0f\x55\x51\x36\x01\x60\x5a\x88\x0a\x25\x7d\xa6\x04\x90\xad\x2f\
xd3\xa8\x28\xc8\x34\xc5\xf1\xc0\xed\xa3\x3d\xf9\x3e\x96\xe9\x43\xa4\x14\x81\xc6\xa3\x06\x5c\x
7a\x9b\xd1\xad\x9c\xb4\x91\x28\x1b\x58\x94\xe8\xc2\x5b\x94\xe4\x38\x8a\x2f\xd5\x7b\x66\x25\xc
7\x15\xb7\x27\x0a\x53\xce\x14\xc2\x65\xd3\xb8\xc8\xa4\x67\xb5\x26\xdd\x7e\x6d\xba\x4e\x98\xd
4\x22\x62\x4c\xd6\xe7\x09\x90\x0a\xb9\x65\xc6\xc7\x46\xe6\x73\x1c\x93\xa8\xc4\xc9\xa5\xdd\x2
c\x3f\xae\xab\x7b\x5f\xc3\xbf\x6a\x8d\xc9\x11\xf4\x17\xaa\xef\x55\x78\x22\xf0\x39\x2a\xd2\x8f\x4
e\x8c\x2f\xd3\xdd\x81\x0d\x46\xbb\x73\x3c\x77\x02\x33\xd8\x7b\xad\xc9\x86\x98\x2d\x9a\xd6\x0f\
xa1\x36\x30\xf8\x98\x3e\x1a\x6b\x9e\xf8\xb5\x9e\x3b\x10\x0d\xd7\xda\xdd\xe5\x75\xdb\x81\xe8\x
db\x62\xf3\x78\x9c\x8d\x3d\x5b\x88\x7d\xdd\x1c\x48\x03\x2b\x86\xa7\xc7\xf3\xec\x4c\xa8\xde\x5
0\x54\x5c\xa6\x63\x79\x36\xf1\xc9\x2d\x3e\x16\xbb\x4c\xe1\x6d\xad\x81\x00\x4d\x04\xb0\xb0\xe5
\xf0\x2e\xdd\x78\x7b\x95\x9a\x0d\xb9\xdc\xc1\xe8\xd4\x8a\xa6\xed\x7b\x5c\xef\x6c\xfc\x5e\xbb\x
08\x59\xd2\x96\x99\xad\xcd\xaf\xc2\xf4\x6f\x38\x44\x87\x13\xc5\x19\x49\x21\x1f\xa3\x5d\x62\xee\
x9e\x03\x91\x12\x29\xc7\x4c\xaa\xdc\xf9\x0c\x83\xb5\x00\x1f\x7d\x1f\x31\xa6\x5a\x20\x52\x9a\x6
c\xd5\xbb\xa7\x3a\xc4\x2e\x97\x99\x6f\xf7\xf0\xa1\x9f\xd7\x68\x4f\xa8\xbe\x75\x42\x50\x0c\x0f\x7
f\xfb\x8a\xfe\x5b\x2c\x71\x39\xc7\xb6\x9d\x59\x92\x4d\xab\xda\x45\x16\x63\xaa\x11\xfd\xa1\x96\x
90\xee\x09\x15\x1e\xd8\xfc\x31\x2a\x4c\x10\x47\x3e\xb7\x07\xd6\x9e\x8e\x1c\x37\x44\x5c\x4e\x3
e\x7c\xc1\x12\x42\x4e\x63\xbd\xfe\x80\xed\xc8\xe3\x48\xf8\xa8\x03\xb7\x1b\x38\x46\x74\x75\xcf\
xf2\x2c\xcd\x96\x85\x74\x58\xc7\x2f\xb0\xe9\x6e\x6f\x7b\xaa\x61\xd5\x70\x89\xb4\xeb\xb5\x04\x0
5\xa7\x03\x99\x32\x25\x6b\x43\x40\xae\xa1\x17\xad\xa1\x79\x0e\x6f\x31\x6f\xd7\x0d\xfc\xd8\xb9\
xca\x63\xb8\x75\xc2\x7d\xd5\x5c\xe4\x5d\x9f\x06\x3b\x9b\xf7\x57\x75\xf7\x57\x75\x7f\xed\xab\x3
a\xf5\xa0\x51\x53\xfe\xde\xe4\x55\x23\x07\x5e\xe1\x8e\xcd\x17\xe0\xab\xf5\x43\xc8\x74\x42\xa6\
x5e\x38\x96\x25\x00\x0f\x59\xe4\x7e\xed\x42\x8e\x8c\xa2\xd4\x13\x8c\x03\xd4\xbb\x2c\x9a\x10\x
33\xdd\x65\xd7\x6c\x23\x32\xe5\x4f\xeb\x2d\xfb\x3a\x06\xf4\x8c\x4c\x2d\xf5\xb8\x6e\x67\xc7\x54\
xc0\x57\x0c\xe2\x4a\xc2\x5e\x9b\x6e\x8c\x54\xba\x6e\x20\x0a\x8a\xbf\x8a\x36\x0c\x39\x88\xf5\x
ce\xfb\x58\xaa\xcc\x64\x59\x01\xb6\x27\xb5\x32\xa4\x78\x97\x63\x7e\x41\xa7\xe9\xf9\x8d\xba\x4
7\x2a\xdd\x6a\x60\xa4\x97\xa0\x47\x07\xee\xe2\x1c\x5d\x5d\xb9\x79\xfc\x34\xea\xcf\xc4\x51\x9e\

x10\x5a\x54\xeb\x5a\xba\x58\x96\x2f\xf0\x24\x5a\x26\xde\x2b\x88\xa6\x3e\xd2\x3d\xd8\x6e\x47\x5e\x46\x7a\x43\x74\x50\x92\x19\xc4\x5a\x8b\x1e\x6f\x44\xd5\x37\x22\x7a\x17\xac\x51\xfc\x16\xdd\xb7\x9f\x1d\x31\x91\x84\xd6\x52\x31\xc7\x46\xa3\x9e\x0a\xb5\x6c\x0f\x1e\x04\x6d\xbd\xc2\x17\x9e\x91\xf3\x55\xc5\x06\x5b\x68\xe6\x7a\xd9\x04\x45\x86\xb3\xb8\x28\x8d\xc5\xdd\x60\x01\x57\x17\xec\xc6\x9a\xae\xbb\x57\x2f\xff\x69\x2d\x37\xa8\x83\x4a\xc2\xde\x85\x26\x94\xeb\x86\x1f\x55\xc7\x1e\x5b\x5c\xde\x0a\xf5\xbb\x5b\xa7\xf7\x02\xf5\x8b\x71\xfb\x09\x17\x68\xda\x25\x24\x7c\x5e\x5d\x59\x34\x74\x30\x06\xdf\xfa\x9a\x2b\x2c\x1d\xde\xe3\x83\x49\x54\x0b\x7d\xe2\x8e\x89\xfc\x97\x77\xa6\xe4\xa3\x57\x5d\x66\x3c\x1a\x30\x29\xd1\x9c\x4c\x67\x4c\x54\x94\x1e\x6a\xb9\x22\xca\x69\xb9\xcc\x1a\xdb\x2d\x33\xb3\xd5\x93\xee\x34\x2a\xde\xe5\x64\x8c\xbb\x01\xa2\xbf\xe9\x7f\x30\x7d\xf4\x47\x9a\xa5\x63\xec\x7b\x42\xf7\x19\x5f\xd6\x3c\xa2\xfb\x8c\x2f\xdb\x3e\xa3\x83\x9a\x1c\x1c\xb2\x1a\xf6\xb5\x9b\xfe\x17\x78\x4c\xe6\x51\xd2\xd3\x01\x2a\xee\x73\xe5\x85\xf7\xd7\x26\x62\xcd\xb9\xe2\x5d\xd3\xb2\xaf\xea\xbb\x27\xe9\x9b\x52\xed\x3d\xbd\xfe\x96\xf4\xca\x85\x18\x87\x60\xe1\x02\x53\x04\x78\xe1\xd4\xea\x15\x6d\x5a\xd3\xe9\x85\x29\xce\xf0\xf4\x35\x43\x86\x69\xa4\xcc\xf2\xa2\xff\x45\xea\xee\x2e\x06\xfa\xf6\xb7\x2e\xce\xcf\x4a\x67\x65\x02\x48\xbf\x0b\x99\xc0\x9f\x09\x20\x5f\x13\xd0\x74\x0d\x17\xf0\x68\xc9\x5f\xbd\x03\xe5\x6d\xc3\x86\x12\xea\xb9\x8b\x01\x90\x94\xbf\x10\x64\x29\xc8\x69\x54\xf8\xe1\xa6\x51\x61\x40\x01\xf9\x6a\xa0\x4a\xb4\xd3\xf2\x55\x09\x61\x56\xe5\x05\xd7\xdf\x71\x8a\xb3\xf3\xc5\xca\xa4\x24\x42\x89\xdc\x84\xa4\x78\x54\x93\x5a\xca\x92\x41\x76\x56\x21\x2f\xbb\x62\xeb\xda\x51\x8f\x8e\xa2\xa2\xa1\x34\xd0\x9b\x0f\xca\x9d\x33\x0f\x94\xa2\x3c\x91\xd9\x82\xfc\x2a\x41\xab\x9b\xac\x20\x44\x19\xb4\x63\x39\x5f\x26\x51\x49\xce\xf0\x4f\x51\x71\x5c\xc0\x73\xa9\xaa\xaa\x1c\x58\xab\xae\x69\x63\x0d\x53\x59\x4e\x0c\xde\xbc\xfb\x17\x70\x49\x36\xb5\x4d\xcf\x54\x86\x16\x72\xc4\x51\x26\x81\x46\xc8\xab\x4a\xf2\x3c\xba\xa4\xb0\x4d\x7a\x23\xde\x52\xab\x05\x00\xb3\xdb\x9e\xe4\x41\xea\xae\xa5\x72\xa8\xb0\x05\x8d\x9b\x35\xe9\x06\x23\x50\x83\xb4\x18\x19\x0e\x91\x74\x40\x03\xae\xd8\xf8\xe9\x15\x21\xd6\x14\x9d\x9f\xd7\x64\x4e\x4a\xcf\x14\x9a\x00\x1c\x57\x32\xb1\x62\xde\x8d\x7c\xa3\x4c\x41\x7e\xf5\x31\x41\x95\x69\x40\x97\x64\x8e\x8b\x32\x9a\x2f\x2a\x8b\x48\x08\xb5\xae\x58\x46\x5a\xb5\x72\x8d\xec\xaa\x6a\xe5\xd1\x58\xeb\x4c\x4c\x26\x13\x32\x5e\x26\xf0\x70\xc0\xe5\xa1\x36\x90\x39\x90\xac\x8c\x92\x17\x6d\x2a\xb0\x20\x75\x21\xc9\x5c\x33\x1c\x5c\x2d\x73\x73\xe5\xb8\xd9\xae\x08\x42\x4a\x3c\xef\xdb\x4f\x86\x1c\x73\x35\x80\x72\x6f\x1e\x8d\xf5\xe5\xdb\xcc\x59\xc1\xa6\x85\x36\x62\x47\xeb\x16\xcb\x2c\xc9\xa6\xde\xf5\xa4\xaf\x6d\xdf\x6a\x4a\xb2\xa9\xe6\xfa\xc4\x59\x52\x50\xaf\xb1\xac\xf4\x0a\xf5\x45\xa5\xe9\xab\xc9\x84\x7e\x35\xec\x11\x36\x84\x4b\x6c\x16\x84\xa2\x61\x9a\xd1\x62\x5f\xf0\x82\xf9\x9b\xa9\xd8\x0f\x78\x5b\x49\x36\xad\x6b\x43\x66\xfb\xeb\x16\xd9\x96\x3c\x0a\xea\xf9\xe6\xb3\xd3\xf9\x8c\x14\x94\x65\x2e\xb2\xa2\xbc\xc1\xe1\xe9\x5d\x56\xd4\xcb\x0c\x6e\x24\x95\x5a\xd6\xea\x56\xaa\xd3\x00\xed\xa4\xce\x57\xe9\xf7\x60\x11\x5d\x82\x4d\xf7\xbe\xa1\x08\xd1\xb3\x38\xb6\x21\xa9\x2c\x13\xaf\x84\x2f\x32\x75\xd8\xf3\x2c\xff\xfc\x31\x7b\x97\x67\x67\xb8\xba\x8c\x06\xa4\x97\x5d\xe4\x24\xcb\x89\xc6\x6c\x9c\x82\x02\x42\xf3\x26\x3e\xd1\xe3\xc7\x18\xc6\xab\x8c\xeb\xb0\x4e\x32\x27\x0f\x3a\xbb\xd1\xd2\xd1\xbe\xf1\xf5\x14\x9d\x68\x9f\xa7\x28\x94\x57\xd3\xd7\xaa\x55\xa6\x65\x65\x0a\xd7\x24\xc9\xce\xc1\xa4\x5d\x9c\x70\xeb\xaa\xaf\x37\xf5\x66\x11\xd0\x28\x31\xa1\x2c\x4d\x2e\x99\x0f\xf8\x52\x5a\x9c\x6b\xb4\xca\x8a\x7a\x1f\x15\x08\x93\x70\x14\xda\x0f\x06\xea\x8c\xc1\x69\x1f\x5b\xb1\x35\xa9\xf1\x07\xfa\xe7\x86\x81\x5e\x46\xd7\x44\xe9\x7e\xb2\x36\x35\xc7\xf5\x84\xcd\xe9\x1a\xf0\x8b\x2f\x16\x24\xbf\xf4\xac\x78\x2d\x57\x27\xb7\x82\xf9\xad\xf0\x42\xd3\xbc\xaa\x25\x60\x81\x7a\x16\x00\x50\xb6\x4f\xcc\xb0\x20\xfa\x7b\xbe\x55\xf9\x3e\x3a\x17\x24\xc3\x53\xbc\x60\x5a\xf5\x07\xc5\x98\x

10\x7b\xf9\x8a\x32\xfa\x46\xfc\xf7\x82\x23\x4e\xc2\xa9\xa0\x0b\x6a\x55\xa8\x06\xc0\xb1\x2b\x44\x3e\xf2\x31\x87\xe1\x70\x95\x15\x01\x6b\x53\x5f\x8d\x95\x8b\x51\x2d\xb7\x5b\xac\x24\x4b\xd1\xcb\x50\xd4\x8e\xfe\x25\x53\xb5\x75\x33\xbe\x08\x09\xba\xe9\x06\x61\x17\x35\x29\x3e\x87\x3b\x9b\x9e\x19\x12\x17\x94\xd9\xa3\x28\x1d\x90\xe2\x1f\x51\x42\xe2\x1e\xf8\xb7\xe7\x29\x2f\x48\x8e\xc7\x65\xcf\xa7\xc9\xe6\x1e\x8d\x00\x90\xd7\xd8\xeb\x3b\x6a\x72\x5d\x06\x52\x61\x47\x44\x0f\x3c\xd5\x1a\xce\xb3\x3c\x15\xb5\xa8\x82\xf7\xcc\xac\x89\xdf\xc8\x5a\x76\x01\xdc\xc1\xb1\x80\xed\x8a\xd0\xbe\x6a\xa1\x7f\xb8\x4c\xc7\x24\xf5\x0b\x32\xdc\xa3\xb0\xd8\xc6\xb8\x3b\x15\x30\x14\x23\xe9\x14\x4e\x15\xde\xc3\x9c\x0b\x66\x3a\xa5\xe1\xfe\x61\x1a\x2a\xd0\xa1\xcc\xf2\x33\x32\x9d\xe1\xa2\xa9\xbc\x0e\xa5\xd1\x02\xcf\xfd\x9c\x66\xe7\xe9\x87\x32\x2a\xed\xc8\xeb\x76\x6e\x75\x03\x7a\x15\x7b\x76\x0d\x8b\x65\x92\xe0\xb8\xa9\x0a\x1d\xaa\xe2\x7c\xa9\xfc\xd4\x54\x78\x01\x6f\xba\xf2\x0a\x1b\x21\x02\x55\x4f\x4d\x05\x0d\x25\x8d\xdb\x90\xd0\x93\xa6\xc1\xfa\x8e\x01\x61\x75\x96\x56\xd2\xe6\x0d\xa1\x3f\x59\x2b\x61\xec\x64\xa1\x27\x8d\xc1\x56\x5d\x98\x86\x95\x39\x7a\x39\xff\x80\xaa\xf3\x2a\xca\xda\x1a\x28\x4f\x15\x36\x88\xd1\x7b\xe3\x9c\x1f\x7a\x53\x75\x78\xfd\x00\x13\x7a\xd2\x74\x58\x0b\x8d\x9e\x44\x1d\xda\xe6\x2a\x61\x45\x3a\xe3\x46\x86\x5d\x0d\xbb\x3a\xe8\x84\x5b\x4f\xaa\xbc\x9a\x50\x96\xdc\x09\x77\x76\xae\x4f\x83\x9d\xad\x7b\x33\x9b\x7b\x33\x9b\xff\x18\x33\x1b\x4e\xe9\x77\x11\x52\x62\x35\x9f\xe0\x2d\x6d\x6b\xee\xd0\x75\x78\x7b\x67\xdf\x51\x92\x0c\xad\xd0\x73\xf0\x60\x91\x0f\x44\xc5\xbc\x72\x5c\x80\x0b\xb3\x5e\x37\x46\x4f\x8d\xeb\x6f\x5f\x90\x9e\x4f\x6c\xeb\xe2\xae\xa9\xf5\xf0\x94\xab\xbb\x8d\x56\x95\x72\xde\xaf\xd7\xca\x92\x6e\x57\x2d\x44\xb6\x8a\xe0\x10\x06\x75\x8a\x6f\x1d\x46\x44\xaf\xe4\x20\xfc\x53\x87\xb8\x13\xb7\xe5\x94\xfb\xdb\x93\x61\x78\xb4\x83\xfb\xdc\xe7\xca\xc1\x9c\x76\x0e\xc8\xa7\x85\x6e\xf5\x72\x83\xa0\xac\x42\x22\x57\x01\xac\xe0\x25\x3a\x70\x72\xf6\xec\x23\x9f\x16\xcc\x37\xf9\x3a\x17\xcd\xda\x75\x58\x17\xb5\x6a\x3b\xad\x77\xef\x07\x87\x94\x44\x8e\x1e\x8e\x8b\x3f\x02\x73\x07\xe7\x1f\x9b\xfd\x2a\xbb\x46\x08\xec\x29\x3c\xb4\x44\x44\x93\xdf\x44\x71\xa0\x85\x9b\x7e\x2d\x84\xaf\xe9\x08\x3c\x7b\xc7\xc0\x3c\x41\xb7\x9c\x25\xac\xbc\xa6\x56\xf8\x50\x8c\x12\xd9\xaa\x32\x7c\x18\xd6\x4d\x99\x6c\xbf\x72\xa2\x9c\x30\x6d\xab\x4f\xdf\xaa\xb3\x67\x46\x57\xab\x08\xdf\xc6\x8e\x7c\x86\x69\x83\xc7\x3f\xb3\x11\x4e\x6d\x1f\x55\xf8\x5e\xf6\xb9\xee\xe5\xa6\xda\xda\x88\x8c\x38\x6d\x35\xc6\x04\x15\xc1\x05\x04\x76\x6f\xe6\x44\xdd\x5b\xbc\x76\x6a\x6f\xe4\x4a\x9d\xbb\x7e\xdd\x0c\xd0\x13\x71\x58\xae\x69\x62\x99\x2e\xa2\xf1\xe7\x23\xa6\xa6\x33\xcc\x65\x20\x49\x5f\xeb\xeb\x66\x92\xea\x82\xe9\x28\x43\x54\xc5\x7e\x48\xea\xda\x47\xdb\xe8\xa9\x48\x14\xde\x69\x91\x10\xab\xd5\x53\x55\xe9\x4f\xb6\xca\x39\xad\xbe\xab\x04\xbc\xb8\x39\xa3\xfc\x68\xaa\xbb\xd5\x94\x31\x8a\x4e\x36\x4f\x51\xe8\x73\x9e\xfa\x1c\xa2\x61\x46\x5a\x00\x52\x81\x2c\x3b\xc4\x69\x94\x24\xfa\xfa\x1d\x0c\x06\x62\x09\x3f\xb7\xcb\x1a\x4c\x03\xe9\x7e\x1a\x4a\x08\xf3\x79\xc8\xa4\x31\x88\x74\x28\x40\xa9\xdc\x1a\xc9\x1a\x02\x33\x92\xb1\x48\x66\x6e\x7a\xe0\xb9\x8d\x90\x61\x23\xe3\x41\x44\x94\xc6\xe6\xab\x7e\x01\xc6\x42\x81\x32\x41\x93\xd6\xc1\x82\x3b\x51\x70\x8e\x36\x2f\xed\xf2\x59\x85\x50\x86\x4d\x54\x0b\xbd\xaa\x0a\x17\xb9\x4a\x2c\x48\xd7\x28\x29\xea\xa3\x2f\x62\xd3\xb3\xcc\x8f\xa4\x8c\xa0\x47\x4c\x57\xde\x2c\x8d\x7d\xb8\xa7\x71\x01\xf0\xc8\x69\x6e\x77\x7a\x11\x7d\xbf\xb1\x8b\x29\x7d\x27\xf3\xa2\x29\x18\xb1\x80\x53\x17\x9a\xeb\xbe\xd7\x67\x4a\x85\x25\x19\x1f\x2e\x67\x0c\x09\xbc\xea\xc0\xe8\x9a\xfb\xd6\x04\x4a\xe9\x4b\xb8\x67\xac\x07\xcd\x13\xbd\xf3\x3a\xac\x4d\x83\x81\xeb\x72\xc1\xe5\x01\x9a\xc3\x05\x61\x52\x6c\xbc\xf4\x0d\xd8\x35\x9b\x15\xbc\x9e\x75\x1a\xc7\x8e\x63\xdb\x32\xbf\xb4\x9e\xca\x68\xa0\xf0\x3a\xa6\x7a\xbc\xc8\x78\xce\x33\x86\xf7\x94\x3d\xc7\x69\x02\xa3\xf8\x7d\x84\xbd\xee\x1a\xec\xce\x8b\xd6\x35\xed\x6f\xed\x46\xd1\x46\x90\xb7\xb7\x0d\xb3\x48\xe3\xae\x60\xb5\xf0\x

a7\x5a\x6a\x8d\x6b\x46\x90\x14\x07\x54\xb1\xfa\x41\x8a\x34\x84\x7b\xf3\x29\xe7\xaa\x71\xed\xd
5\x7b\x5f\xc7\xa8\xd8\xbb\x5c\x8d\xb0\x3c\xbe\x68\xe1\xe6\x25\xc7\xf5\x9a\xb5\xcc\x2a\xc0\xdb\
xfb\x40\xc6\x45\x49\xe6\x51\x89\x7f\x8a\x40\x1f\xd3\x44\x55\x1a\x78\x13\x45\xe9\x35\xdf\x05\x3
5\x7d\x7d\xea\x68\x37\x43\xda\xb8\x9a\x66\xc7\x03\x5a\x35\x33\xef\x45\x33\x58\x04\x78\x61\xf1
\x97\xb9\x6a\x85\xcb\x07\xfe\x48\xcb\xee\xce\x6a\x57\xd3\x34\x57\x4d\xf1\x98\x6f\x36\x4f\xad\x1
0\x2f\x2e\xbc\xf8\xca\xac\x09\x5e\xec\x95\x9a\x6f\x11\x27\x4a\x2f\x2a\xf0\xac\x91\x7d\x2d\xc2\x
7e\xdb\xa0\x51\xb2\xfe\x1b\xc5\x8d\x92\x85\x56\x1d\xe4\xd7\x0c\x22\x65\xc6\xc6\xcf\x17\x63\xe
1\x8e\xa5\x60\x07\xe3\x26\x46\xc4\xa1\xab\x2f\xed\x6b\xc6\xc5\xcb\xfe\xb1\xb9\x12\x32\x3c\xe7\
x0b\x30\x5d\x4c\x31\xfc\xc0\xeb\x73\xe2\x7a\x2e\xb2\x54\x5c\x4f\x51\x17\x97\xb3\x4f\xb4\xc7\x5
d\x14\xb2\x0f\x6b\x27\xe9\x06\x8e\xf0\x12\x2a\xbf\x41\xea\xae\x5c\xf8\x3a\xe2\xc3\x39\xd5\xfc\x
e9\x41\xc7\x85\x0b\x3c\xc6\x20\x1b\x89\x41\x78\xbc\xaf\xda\x7e\x64\x20\x89\xfa\xad\xc7\x13\xf3
\x41\x9f\xe0\xd2\x10\x74\xd6\x4d\xec\x28\x63\x05\xd8\xe6\x4b\x5d\x86\x12\xcf\xd1\x55\x5a\xd5\x
56\x61\xa1\x73\x10\x2d\x16\xc9\x25\x77\x8e\xd1\x8a\xb0\xfa\xb6\x95\x0f\xdb\x02\xac\x66\x68\xe
2\x8d\xea\x6e\x98\x07\x1e\x1a\x41\x31\x1e\x15\x1d\xe1\xd6\x61\x11\x3c\x13\xf6\xb5\x22\x23\x8
8\x74\xb5\xe2\x75\xef\x20\x95\xe0\xfc\xb0\xa9\x30\x5c\x05\xe8\x4a\xcd\xde\xc9\xaf\x2a\x6e\x8a\
x48\x6c\x24\x2a\xa9\xb2\x98\xda\xad\x85\x57\x0b\xfa\xf9\xa7\x0c\x0b\x21\xca\x02\x81\x93\x7c\x
bc\x4c\xa2\x7c\x7d\x7d\x7d\xbd\x3e\x18\x84\xa0\xa0\xbb\x8a\x07\xe1\xb8\x29\xd8\xbe\xbf\x3f\xb
d\xbf\x3f\xfd\x6b\xdf\x9f\xf2\xcb\x53\x0a\x2b\x22\x74\xf8\xfd\x8a\xff\x6e\x1e\xc3\xed\xbb\xd9\xce\
xb2\x60\x6e\x9d\xc7\x65\x87\x1b\xc6\x81\x3a\xc5\xba\x55\x65\x69\x22\xe0\xc9\x79\x96\x7f\x8e\x
72\x4a\xd9\x74\x0f\xa3\x35\xe8\x23\xa6\xf4\x17\x93\xc9\x04\xe7\x38\x2d\x11\x4e\xcf\x0a\x5a\x8
6\xd6\xfc\xcf\x37\xaf\x5f\x95\xe5\x82\x3b\x25\x04\xfe\xc4\x03\x93\x90\x69\x9a\xd1\xa5\x9b\x90\x
14\x43\x13\xa3\x3c\x3b\x2f\x70\xbe\x06\x92\x01\x73\x02\x76\x4e\xd2\x38\x3b\x07\x0d\x87\xe6\x
b4\x1b\x3d\x78\xc0\x73\x06\x66\xf5\x82\xcb\x9a\xa9\x68\xdf\x0f\x5d\xd9\x99\xe1\x10\xa5\x59\x8c
\xd7\x0c\x17\x3a\x4e\x9d\x12\x55\x17\xf3\x84\xe2\x82\x6f\x86\xdd\x7e\xdb\x66\xae\x19\xee\xff\xf
9\xea\xfd\xb6\x51\xdd\x2c\xdf\xee\xf6\x6b\x30\xc5\xa4\x0d\xda\xc2\x3b\x81\x7e\xf7\xde\x1a\x64\x
2e\x7a\x1c\x00\x2f\x72\xcc\x25\x2c\xed\xa5\xbc\xa5\x36\xca\xeb\x62\xc2\x2c\x2b\xca\x40\xc4\xd
9\x37\x2e\xa7\x69\x0e\xda\x47\xf0\xdf\xd5\x15\xea\xf2\xf5\x91\x64\xe3\x28\xa1\x89\xe1\x93\x6f\x
76\xbf\xe9\x6a\x1a\x4c\x11\xac\x7f\x5f\x86\xed\xbf\xba\x42\x9b\x8d\x42\xd3\x22\xc7\x0b\x08\x9e\
x83\xcf\x2d\xb4\x5b\x32\x13\x07\x7c\xaf\x9d\x61\xb4\xb0\xd1\x38\x4a\xaf\x21\x28\x27\x8b\x1b\xc
d\xe6\x4d\x61\x8a\x7b\x20\xd3\x0e\x7a\x66\x5b\x86\xd3\x0b\x1d\x5b\x9a\xbc\x65\x76\xc0\xbc\xe
3\xa3\xd5\xeb\xa2\x16\xef\x81\xa6\x27\x31\x40\x10\x52\xc2\x15\x0c\xfc\xd5\xfb\x6d\x15\xa9\x4e\
x48\x5a\x1a\x46\x35\x04\x73\x91\xc1\x70\xf8\x64\xd5\x66\x0c\xad\x67\x1b\x03\xb2\xca\xb3\x05\x
4e\x7b\xdd\x77\x47\x1f\x3e\x76\x03\x35\xe3\x01\xc3\x94\xbc\xc5\x61\xb0\xca\xd1\xe8\x2b\x1c\xc
5\x38\xef\x75\xa9\xcc\x89\xd3\xf2\x11\x3d\xbc\x76\x83\x2e\x15\xa6\xc9\x18\xb6\xc1\xe1\x2f\x85\
xd2\xe4\xc9\x6b\x20\x8e\x8d\x06\x5a\x60\x9e\xdb\x2f\xd3\xb1\x76\x52\xb5\xd5\xb3\xbe\x0b\xdb\
x85\x76\xed\xeb\x0f\xa7\x59\x37\xaf\x85\x1d\x5b\x49\x28\xa8\xdc\xf9\x14\x53\x61\x92\x02\x8f\xcf
\xc8\x27\xdb\x54\x9b\x2b\x14\xa6\x71\x0f\x8c\xb1\x99\xf7\x7c\x32\xb9\x94\xed\x48\x53\x60\xd0\x
93\xbb\x2e\x42\x99\x46\x95\xb1\xe8\xc1\x21\x33\xc3\x7d\x9e\xa5\x29\xe6\x76\xd3\x62\xf2\xdc\xb
b\x06\x79\x85\x27\xba\x21\x1c\x75\x7f\xc4\x17\x65\x45\x7f\x79\x09\xcd\x70\x9c\xdb\xee\x5a\xbd\
x6c\xea\xe1\x7b\xde\x54\xcf\xd7\x76\xa3\x81\x6a\xa5\x6a\x07\x68\x24\x6a\x20\x92\x03\xb9\xdc\x
6b\x28\x85\x67\xfd\x28\xef\x00\xa5\x32\xa2\xcc\xc9\x74\x0a\xc1\xeb\xb3\x14\x51\xea\x00\xc1\x4
1\xfa\xa5\xa3\xc8\x68\x22\x28\xe8\x81\x8f\xaa\xcc\xb0\x89\xed\xe8\x0b\x82\x8f\xee\x59\xeb\x37\

x05\x3f\xa2\x45\x19\x95\x78\x3c\x8b\x52\xc3\x2f\x77\xcf\xbe\x16\x52\x73\x10\xc5\x97\x60\x23\x0
c\xb7\xb1\xbb\x74\x77\xb5\xd9\xcd\xba\x2f\xfa\x61\x0b\x6a\xb2\xc1\xc5\xed\x8e\x16\x65\x43\xd2\
x8b\x73\xb5\xd4\x8a\xf6\xc4\x9f\x97\x06\xc5\x9f\xbc\x54\x5a\x99\x1a\x65\xf5\x66\x67\xed\x7b\x2
8\xa7\x4b\xa6\x8f\x6c\x35\x41\x8a\x7f\x7b\xe6\xc5\xac\xb5\x18\xa8\x05\xfd\x91\x15\xeb\xe9\x4c\
x5f\xf2\x88\xaf\xc6\x64\xec\xfe\xd4\xf1\x19\xc7\x75\x3d\x5d\xa9\xea\xce\x3b\xb9\xa4\x2b\x88\xbd
\xa0\x83\xe5\x4a\xd7\x49\xc5\x6a\x25\x05\x6f\x01\x9b\xec\x5b\xee\xe9\x42\x4f\x69\xef\xed\xf4\x4
c\x90\x76\x4b\x34\x89\x48\x82\xe3\x01\x3a\xa2\x67\xaf\x73\x42\xcf\x13\x11\xc4\x5d\xa9\x5e\x9f\
x5a\x9b\xbe\xb9\x31\x71\x2b\x75\x13\x3d\xeb\xd5\x6a\x1c\xa2\xef\xe4\x5f\x60\xe4\xd1\x2d\x30\x
5f\x8c\x43\xd4\xdd\x1e\x6c\x76\xcd\x3c\xa1\x5d\xec\xa6\xb8\xfc\x94\x90\xa2\xc4\x29\x49\xa7\x1
6\x90\xd4\x10\x9e\x2a\x22\xf3\xdc\x39\xab\x18\xe4\xbe\x05\x21\xec\xb6\x28\x3a\xf4\x09\x73\x34
\x04\x3a\x9a\xf6\x44\x94\x31\x46\x06\x9d\x70\xfb\x71\xd0\xa1\x52\x6a\x27\x7c\x42\x7f\x19\xe2\x
6f\x27\xdc\xfa\x96\x9e\xfc\x77\xee\x4f\xfe\xf7\x27\xff\xbf\xf8\xc9\x5f\x99\x4e\xc3\x33\xae\x3b\x32\
x9b\x96\x4f\x41\xf4\x43\xe1\x88\x4c\x99\x33\x80\xc1\x2f\xec\x48\xce\xae\x3e\xe2\xd7\x78\x62\x1
e\x3b\x64\xe4\xad\x4b\xed\xe9\x91\xb1\x53\x33\x08\xb6\xfa\xcf\x67\xb4\xf7\x3d\xd3\x26\xeb\x7b\
x56\x18\x3d\x44\xdb\xee\xdb\x25\x30\xf2\xeb\xa2\x0d\x29\x99\x21\xdb\x0b\x9b\x47\x64\x7b\xc7\
x0f\x74\x51\x8a\x0e\x9f\x1d\xbc\xe5\x93\x1c\xa3\xef\xbe\x45\xe3\x6c\xbe\x58\x72\xc7\xff\xa3\x4b
\x34\xcf\xce\x48\x3a\xd5\x42\xda\xec\xa2\xf1\x2c\xca\x61\xdf\x60\x97\xb1\x31\xb3\x9e\x12\x26\x
c1\x02\x3a\xc1\xcc\x30\xbc\xcc\x68\x83\x0c\x57\x05\xea\x1d\xa0\x7d\xb4\xb5\x19\xa0\x67\xf4\xff\
xad\x00\x0d\x06\x83\x00\xfd\x0f\xda\x47\x3b\xdf\xf4\xe9\xc1\x06\x15\x0b\x3c\x26\x13\xc2\x16\xd
2\xe1\x87\xa3\xad\x9d\xc7\x5b\x8f\x6d\xab\x32\x52\x64\x90\xce\xc7\xe1\x7a\x52\xbc\x66\xaf\x04\
x69\x47\xe8\x00\xcd\xdb\x34\xfd\x32\x99\x4b\x71\xb1\x00\xe3\x8f\xbd\xcd\xfa\xcd\x18\xdb\xa3\x2
8\xd5\xe7\x91\x8e\xa8\x7b\xd0\x1d\x50\xb4\x3c\xcf\x62\x7c\x50\xf6\x36\x35\x45\x35\x1d\x5b\xf7\
x7f\x9c\x6c\xc6\x00\xd9\x4b\x46\x20\xd6\x32\x3b\x5e\x2c\x70\xfe\x3c\x2a\x94\xf6\x5a\xcb\x2e\x9
6\xa3\xa2\xcc\x7b\xbb\x7d\xf1\x46\x91\x27\x6c\x06\xbb\xd6\x25\x19\xcb\x5d\x24\xa4\xec\x75\xbb
\x7d\xf3\xf9\x66\xda\x37\x0d\xaa\xc6\x59\x4c\x07\x97\xfa\x3a\x8f\x84\x2b\x70\x0a\xf3\xc3\x3e\x3
a\xa0\x72\x28\x7c\x7c\xbf\x8f\xfe\xa7\xef\x78\xba\xf6\xcc\x2c\x9f\x58\x03\x52\x3a\x52\x8c\x31\x7
a\x84\x0e\xd0\x06\xda\xda\xd4\xc4\x34\x9f\xf7\x67\x11\x09\xce\x91\xe6\xfa\x83\x5f\x32\x92\xd2\x
61\xda\xc6\x89\xe3\x25\x78\x99\x84\x29\x7e\x73\xf4\x82\x12\xf6\xd6\xa6\x60\x4a\xdc\xa8\x0f\x2
8\xdf\x43\x71\xdf\x6e\x3e\xde\xb5\x09\x6e\x9e\xc5\xdf\x7d\xbb\xb5\x59\x45\x68\x26\x7d\x29\xdf\
x9d\x8c\x9a\x78\xe1\x5a\x2a\xca\xf1\x3c\x22\x29\x53\xfd\xd0\x3c\x25\x6b\x70\x4f\x18\x26\x7b\xe
0\xc0\xca\x1a\x79\xbb\x6f\xf9\xf8\x00\x66\x25\xc0\xa4\x91\xeb\x77\x86\x84\xa2\x9a\x04\x41\xfe\x
30\x2d\x99\xfb\x90\x00\x6d\x6d\xf6\xd1\xff\x4b\xb1\xb6\xe1\xd4\xc2\x3c\x88\xb0\x86\xbd\x47\x44
\x59\x97\x2c\xa9\xea\x33\xe6\xa9\xf9\xad\x07\x33\x2c\x87\x75\xa0\x87\xe2\x87\x84\x71\x96\xe7\
x34\x85\xb1\x4f\x31\x5f\xfe\xc9\x19\xea\x1e\x5e\xfd\x93\xc0\x8d\x9f\xd5\x92\x73\xbb\xea\xc4\xfc\
x6b\xea\x27\x86\x50\x1b\xcb\xb9\x78\x01\x61\x11\x15\x85\x39\x90\x39\x4e\xdf\x23\x2d\x4b\x88\
xdc\x74\x08\xd7\x92\xad\xe9\x0a\x31\x9a\x33\xd0\x6a\x6c\x7a\x45\x1d\x15\xcf\xc4\x2b\x6a\xe5\x
6b\x46\xc4\x8e\x43\x5b\x8f\x35\x16\x36\x8a\x0a\xbc\xf3\x18\xed\x43\x99\x41\x99\x71\x07\x30\x3
b\x8f\x8d\x5b\xff\x38\x06\x51\x9d\xef\x81\x3d\x56\x28\x40\x5b\xdf\x98\x7a\x26\xd9\xcf\x67\xa3\x
28\xed\xb1\x62\x26\xf3\xb3\x16\x33\x77\xd4\xa0\x2d\xdc\x67\x74\xe8\x65\x66\xec\x5e\x74\xfa\x1
0\xb8\x1d\xcc\x2f\xc5\x8a\x66\xca\x2e\x30\xd1\x7d\xc7\x3c\x9e\xa7\x59\xc9\x85\xb2\xef\xc9\x0f\x
9d\x29\x48\x24\xcc\x01\xc8\x44\x21\xb5\x98\x45\x4c\x5a\x83\xfd\xed\x62\x9c\x2c\x0b\x72\x26\xe
3\xa7\x91\x11\x49\x48\x29\x05\x9c\x51\x94\x7e\x1e\x8e\xf2\x28\x1d\xcf\x50\x81\xf3\x33\x32\x16\

x1b\x60\xc4\xbc\x20\x76\xbe\x1f\x92\x1f\x06\x36\x0d\x49\x67\xe9\x85\xd8\x85\x26\x38\xa7\xdb\x50\x94\x4c\xb3\x9c\x94\xb3\x39\x8a\x71\x31\xce\xc9\x88\xb1\x25\x2e\xff\xe0\x74\x70\x4e\x3e\x93\x05\x8e\x49\x04\x42\x10\xfd\x1a\x1e\xa6\x25\xce\xd3\x88\x3d\x88\xf8\xf4\x2c\x4a\x3f\x7f\xe2\x2e\x18\x3f\xb1\x79\xfd\x7f\x7e\xe2\x23\x4d\xa7\x9f\xe8\x10\x3f\x41\x2c\x98\x4f\x31\x99\x12\xe7\x81\x86\x98\x1a\x1f\x45\x8e\xc4\x9e\x2a\x66\x40\x78\xd1\x28\x33\xcf\x36\xdb\x82\x56\x9f\xd9\x2b\x72\x64\xb1\x45\x3e\xa3\xcf\xd9\x3e\xd5\xfd\xe7\xcb\xee\xde\x9a\x97\x67\x72\x1e\xdb\xb3\x76\xee\x9e\x5e\xc1\x06\xea\x6e\x82\xa8\x04\xad\xe8\x16\x2e\x14\x1d\x2f\x28\x36\xd0\x3e\xea\x31\x71\xaa\xf7\xdd\x13\xf4\x48\x35\xd1\x17\x2f\x05\x1e\x6d\x5b\xfb\xad\xf4\x43\x60\x36\xa5\xd5\xc9\x1b\x6c\x50\x9a\x71\x26\xa2\xe1\x0a\x08\x9b\x85\x73\x25\x69\x51\x92\x72\x59\x0a\x47\xa2\x24\xc6\x69\x49\x37\x2d\xdb\xd9\x34\xab\xe5\x30\x8d\xc1\xd3\x41\xf5\xcb\x99\x22\x10\xb2\xac\x7c\x3a\x03\x21\x81\x3a\x5a\x4b\x1d\x68\xaa\xa3\xda\xea\xac\xc2\x8b\xcc\x9e\x54\x05\x1c\xad\xe4\x0c\xdd\x97\x1f\x5f\xd1\x79\x10\x0f\x5a\x74\x0c\x68\xa9\xb2\x6f\x7d\x8b\x5f\x67\x75\xfc\x5a\x84\x42\x62\xc8\xe5\xb1\x52\x49\x81\x98\xf7\x04\x8d\x8f\x3b\x72\x27\xf8\x94\xa8\x94\x37\xe5\x5e\xe4\x51\x92\x08\xe5\x08\x29\x55\x4b\x52\xe8\x3c\xd4\x3c\x56\xd4\xca\x09\x44\xf7\x7c\x41\x18\x59\xe9\xc2\x9f\x72\x7b\xd1\xa8\xc9\x97\x58\x80\xae\x03\xfb\xcc\xf3\xfa\x31\x2b\x4f\xe4\xde\x51\x05\x28\xf3\x68\x78\x60\x6c\xba\x66\xc7\x1d\xa5\x45\x09\xc3\xff\xfd\xe7\xcb\x93\xcd\x47\xdf\x9d\x7e\xd9\xbe\xee\xbd\xfc\xf8\x8a\xfe\x3e\x78\xf4\x3f\xa7\x5f\xb6\x76\xae\xaf\xe4\xc7\xce\x66\xb0\xb3\x75\xdd\xff\xaf\xe1\xa0\x04\x05\xac\xdc\xc0\xfb\xe8\xc1\x03\x29\xe5\x54\x31\x06\x0d\x9c\x79\xf4\xd9\x5a\x11\x61\xdc\x55\x1c\x9c\xfe\xbd\x68\x7b\xa1\x96\xe0\xdd\xe0\xed\x85\xbb\x92\x2c\xc4\xa9\x41\xe9\xef\x79\x76\x76\x21\xea\xb2\x3f\xef\x9b\x1b\x0e\x7b\x82\x48\x5a\x31\x70\x83\xfb\xdc\xcd\xd0\xbd\x6c\xa4\xd5\xe0\xb7\x9b\xaf\x37\xa7\xb8\xe4\x22\x25\x1d\x69\xb1\x9c\x53\xc0\xe3\x82\x1f\x1f\xe6\x59\xfc\xe8\xbb\x6f\x1f\x6d\x6d\xca\x6c\x38\xe3\x42\xef\xc6\x59\x82\x7a\x87\x1f\x8e\x86\x87\x2f\x9f\x23\x7a\x6e\x08\xb7\x37\x37\x77\xfa\x36\x4f\xd6\xaa\x75\x4f\xa1\x5a\xae\x33\x70\x91\xd7\x72\xd8\xfc\x4c\xb8\x1d\xa0\xed\x76\xe6\xa9\x3a\x53\x35\xb6\x14\x84\xa7\x03\xf4\xcf\xf7\x2f\x7f\x72\x5c\xab\xc9\x02\xfe\xd1\x54\xd6\xe8\x4e\xaa\x06\xd9\x34\x3c\x45\x00\x3d\xf0\xbb\xe5\x0c\xf9\xdb\x00\xed\xf6\x51\x88\xba\xdd\x56\xe3\x1e\x27\x04\xde\x8e\xc9\x0e\x82\xf2\x89\xa4\xf6\xf8\x28\x16\x7e\x3a\xf8\xc7\xd1\x8f\xff\x3a\x7a\xff\xdf\xf6\xac\x42\x1d\x15\x73\x6a\xd7\xef\x9d\x5c\x06\x74\xeb\xb1\x6f\x6d\xad\x3e\x72\xbe\x9a\xfc\xe7\x12\xf7\xe0\xe1\x0e\xcd\xa9\xc0\x19\x5e\xe4\x39\x87\xe8\x0f\x24\xf9\xe0\x7c\x2e\x99\x8c\x43\x87\x3b\xe0\x5d\xed\x10\x5b\x79\x94\x11\xe7\x0f\x79\x4a\x31\x4e\xa8\xec\x8c\x62\x9e\x67\xb6\x1e\xf7\x03\xb4\xbd\x29\x2f\x61\x0c\x29\x4f\xa0\xd7\x1a\xa4\x28\xdc\x6e\x81\x56\xf8\xf5\x39\x84\x2c\xa6\xd4\xd7\xf5\x8a\x9d\xd0\xfc\xbc\x3e\x0d\x76\x76\xef\xd5\xf8\xf7\x6a\xfc\xbf\xb8\x1a\x9f\xab\xf0\x17\xe3\x7a\xfb\xbd\xbb\xb5\xb8\x6b\xe9\xf2\xa4\xc1\x30\x8f\xe9\x99\x16\x63\xaf\x21\xd7\x22\x2a\x67\x01\x4a\xb1\x61\xf0\xfd\x09\x34\x17\xce\x5b\x53\x71\xf5\xad\x87\x30\x15\xbe\x08\x98\xf9\x41\x04\x8e\x49\xe8\x7f\x2c\x55\x65\x8d\xe5\x7d\x30\x70\xc5\x52\x24\xf4\xbe\x50\xe8\x50\x95\x97\xce\xf9\xac\x62\x83\x2c\xed\x75\x61\x54\x5d\x3d\xe4\x5f\xdf\x30\x99\x2e\x32\xca\xc4\xd8\xf3\xc1\xc3\x77\xcf\x91\xba\x85\x66\x8f\x0a\xbb\x01\xd2\x03\x59\x7f\x62\x6c\x90\xdf\x95\xf7\x6c\xf7\x80\xde\x1e\xa4\xb1\xde\xbe\xd6\x7c\x65\x65\x68\x4d\x3e\x2b\x78\x7d\xf8\xe1\xe3\xcb\xb7\xb0\x82\x9e\x1f\xbd\x7d\xfb\xf2\xf9\xc7\xc3\xa3\xb7\xe8\xfd\xcb\x0f\xef\x8e\xde\x7e\x78\xf9\xa1\xb2\xd5\x38\x2a\x23\xbd\x59\xfa\xad\x6f\x4e\xc3\x87\xdc\x0a\x70\x1e\x5d\x8c\xb3\xf9\x22\xc1\x17\xa4\xbc\x0c\xd1\x63\xa0\x2c\xab\x87\xa0\x0b\x95\x76\x0d\xb4\x2a\xb5\xdf\xf4\x3d\xd1\x1f\xb8\xdd\xc2\x97\x35\xc7\x52\x83\xc4\x7e\x33\x0d\x1e\xc6\x16\xf8\x4b\x8c\xce\x67\x64\x3c\x43\x73\x1e\xd5\x9f\x85\x9c\xa7\x9

b\x10\x65\x68\xb1\x79\x37\xee\x3a\x5b\x87\xa6\xfd\xf1\x5c\xe1\x3a\xca\xe9\x2d\x18\x2d\xf8\xa3\x2d\x92\x49\xef\x93\x9f\x90\x4f\xe0\x39\x1c\x89\x4f\x5d\xdf\xd1\xb2\x30\x1d\x2b\x07\xdb\x73\xa0\x9c\x50\xe8\x55\x11\x23\xa1\x1a\xde\x77\xbb\xa2\x6b\x07\x8b\x13\x92\x63\xc3\x09\x80\x8d\xae\xaa\xf1\xd0\xa1\x78\x5a\xaf\x01\x57\xe1\x2b\x5d\xbb\x19\xfa\x17\xe3\x04\x97\xb8\xae\x06\x7b\x30\x36\x6e\xf4\x87\xd7\x3f\xd3\x5d\x0b\x08\x91\x13\x04\xab\x0f\x94\x3b\xcc\x52\x2b\x65\xce\x5e\x50\xc6\x9c\xcf\x92\x72\xb0\xb6\x26\x84\x41\x93\x84\xd7\x6c\xb5\x07\x3c\xc2\xa4\xc2\x9f\xe2\x79\x9a\x78\x64\x16\xd6\x0d\x39\xf4\x55\x65\xb3\xc1\xc0\x92\xd7\xfe\xc1\x7c\x3d\x2b\x97\xa5\x62\x89\xbf\x78\xf9\xe8\xf9\xab\xe3\xb7\xff\xfd\xf2\xbd\xac\x27\xc6\xe3\xd9\x32\xfd\x8c\x63\xfe\x90\x84\x3d\x12\xe5\x7f\x83\x1c\x2f\x92\x68\x8c\x7b\xc3\x7f\x5f\x9f\xfc\x3b\xfd\x77\x7e\xfa\xf4\xdf\x5f\x86\xd3\xa0\x7b\x7d\xf5\xe8\xd1\xd5\x97\x6e\x1f\x9c\xad\x7e\xf1\xc2\xff\xfb\x54\x94\x38\xe1\x65\x4e\x69\xa1\x13\x51\xea\xf4\xc4\x5f\xce\x2e\x65\x14\xaa\x28\xa3\xda\xd2\x5a\x92\x0d\x69\x65\xf8\x35\x1f\xcd\xee\x0a\x4e\x6a\x60\xc0\x5d\xb3\x80\x78\x8d\xbf\x0c\x87\x70\x07\x8a\xb9\x07\x0c\x70\xae\x01\x15\xac\x39\xa4\x4f\xf3\x9e\xd3\x2c\x73\xe5\x72\x57\x33\x16\x0c\xda\x40\xec\xc9\xab\x21\xaa\xcb\x3b\x6b\x8b\x93\xb9\xc6\x66\x3e\x43\x33\xe8\xbb\x56\xca\x30\xa9\x59\x73\x17\x9f\xea\xcc\xbe\xdd\x19\x64\x44\xc1\xe6\x46\x60\xe0\x5e\x2c\x1d\xe3\x04\x5c\x8c\x8b\x77\x9b\x46\x99\x71\x82\xa3\x5c\x58\x7f\x59\xad\xf0\x64\x6b\x41\xfb\x81\xc0\x3d\x43\x29\x2a\xf2\xed\x71\x66\x79\x7b\xaf\xd3\xff\x6a\x2d\x3b\x39\xce\x74\xf8\xeb\x00\x6d\x6d\x6e\x6e\xa2\x87\xec\x72\xc6\x73\xd7\xea\xf5\xf5\x00\x4f\xf5\x00\x3b\x02\x5f\x94\x83\x14\x98\xd3\x0b\x8b\x20\xc1\x9f\xf2\xad\x8e\x2a\x77\xc6\x2c\x12\x81\xd0\x28\xdc\xae\xd3\xe9\x30\x63\x11\x2c\xec\xb1\x69\x0a\x6b\x69\xeb\x75\x70\xee\xef\x87\xf2\xc8\x9f\xf8\x16\x1a\xc5\x71\xa1\xc7\xc3\xe5\x56\x0e\xae\x34\xc6\xd4\xc3\xc1\x1a\xdb\x70\xc5\xc1\x80\x9f\xb5\x09\x73\xe0\xcd\xb9\xde\x5c\x04\xf7\x36\xb8\xef\x61\xcc\x4a\x45\x79\x4e\xce\xb0\xce\x70\xa3\x58\xce\x9e\x68\xaf\x86\xc3\x7a\xa0\x0d\xff\xdd\x75\x16\xad\x84\x5d\xd8\x5d\xf2\xad\x16\x5d\x5d\x89\xaf\x93\xcd\x53\xb9\x65\xc2\x15\x36\xeb\x9b\x82\xe6\x09\x66\x09\x96\xa8\x4b\x74\xde\xcd\x0b\xed\xcb\xde\xd4\x49\xbc\x14\x74\x20\x1b\x16\x75\x8b\x5d\x4d\xac\x23\x7d\xa5\xb2\xa8\xd9\xdc\x2c\x85\x89\xe5\x70\xfa\x02\x8d\x3b\xdd\xdf\x63\x0d\xcd\x9c\x88\x6b\x50\x5b\x63\x1b\x3a\xc9\xf2\x1e\xc5\xcb\x67\x7c\xc9\x4e\x8a\xbe\x01\x98\x06\xbe\x3d\x3f\xd0\x60\x16\x15\x47\xe7\xe9\x3b\x88\x0f\x53\x5e\x42\xfc\xaf\xbe\x1d\xfe\xd9\x8b\x9e\xcf\xf8\xf2\xb4\xda\x12\xb4\x9b\xa5\xe8\xf0\xdd\xf3\xae\x1d\xaa\x9a\xcb\x16\x35\x75\x3a\x66\x16\x6a\x99\x3c\x17\xaa\x60\x90\x93\x98\xc3\x66\xa4\x1d\x37\x48\x81\x8a\x92\xb0\xf0\x0c\x24\xd6\x88\x5a\x37\x21\xad\x44\x78\x83\xcd\xa7\x7b\x5a\x12\x72\x00\xdd\x3d\x72\xcc\xfb\x11\x30\x2a\x30\x7b\x35\xcd\x52\xcc\x35\x4f\xbd\xf5\x4f\xb6\xd8\x7f\x9e\x93\x12\x5c\xa4\x58\xdc\x48\x03\xb1\x8e\x50\x9f\xdc\x33\x14\x67\x30\xeb\xeb\x55\xb5\x73\x05\x92\x77\xe8\x75\xef\x1b\xd6\x74\xfa\xb1\xea\xc5\x1f\x8c\x17\x2b\xfa\x26\xbb\x67\x70\xee\x15\x50\x24\xd0\xd4\x8c\x25\xe4\x39\x42\x35\x9e\x35\x45\x2f\x63\xed\xa5\xaf\x6f\x54\x35\x46\xd2\x37\x13\x1b\x24\x55\x57\x59\xa6\xb7\xd8\x47\x91\xf5\xe7\xdb\x27\x2d\xb3\x3b\xae\x4d\xb4\xce\x28\x8e\x07\x9e\x7f\x65\x4b\xb0\xc8\x56\x8f\xc5\x3a\xdd\x0d\x9b\xdd\x6e\x74\x3b\x7a\x24\xf7\xe4\x72\xa0\xdb\x74\x2b\x3e\x08\x8f\xb5\xb2\x12\x15\xcb\xc5\x22\xcb\x4b\xd0\xad\xb1\x9b\xda\x77\xcf\x91\xd4\xaa\x74\x0d\x1b\xf1\x6a\xc2\x5c\xe1\x9d\xc4\xea\x8b\xb1\x99\xca\x56\xa2\x30\xef\xb1\x1e\x68\xaa\xc1\xe8\x5e\xfa\x52\xb4\x77\xd3\x4a\x07\x37\xae\x1e\x57\x61\xb0\x3e\x7c\xbc\xb2\xdb\xbe\x3e\x0d\x76\xbe\xb9\x57\xe9\xde\xab\x74\xff\x23\x54\xba\xfc\xcd\xc5\xad\x9e\x63\x1f\x44\x79\x96\xa2\xff\x5e\xce\xa3\x33\x52\xa0\xef\x23\xfa\xf9\xb7\xcf\xec\x73\x30\xc7\x5e\x75\xef\x70\x88\x0e\x53\x52\x92\x28\x21\xbf\x62\xf4\x77\xd6\x0b\x4a\xa8\x11\x2a\xc0\x12\x4b\x18\xdc\xc0\x40\xe9\x52\x35\x1c\x49\x0f\x40

\xab\x2b\x8a\x89\x38\x0c\x3c\x24\xcf\x61\x1c\xa2\xcd\xa6\x9b\x37\x66\xed\x41\x87\x6f\x3b\xcb\x
f5\x9a\x99\x78\x9d\xe4\xaa\x37\x70\x22\xfa\xcf\x44\x20\x14\x5a\x52\x06\x3d\x1e\xd7\xba\xec\xb
1\x4a\xa0\xa9\x7a\x26\xa2\x1a\x91\x25\x3c\xea\x7a\x3d\x0f\x69\x23\xa0\xed\x39\xbd\x1f\xae\x71
\xf4\x54\x38\xd8\x65\x6d\x05\xbc\x31\xc3\x4f\x2a\xcb\xea\x57\xa9\x96\x45\x93\x8e\x31\x8f\x34\x
db\x5d\xef\x6a\x71\x78\xa2\xf8\x8c\x9e\x51\xc5\xec\xa0\xc3\x17\x90\x23\x7a\x27\x27\x6d\x63\xa
3\xca\xb5\x50\xd5\xc3\x20\x12\x87\x6e\x35\x2a\x5b\xbc\x19\xe2\x23\x95\xe9\xe2\x99\x10\xfb\x9f\
x1e\x98\xf8\x83\xa1\x66\x9f\x41\xd2\xf0\x42\xe0\x40\x1e\x1e\x85\x01\x91\xdf\x54\x47\x2a\xeb\x9
a\x62\x41\x79\x1e\x65\x5b\x0d\xf8\xcd\x33\x04\x1a\xac\xf6\xac\x00\xaa\x2c\xd1\xba\x0c\x65\x6e\
x7c\x34\x9d\x33\x07\x7a\x2a\xdb\x1e\xe0\x33\x9c\x5f\xf6\xa0\xf9\xa8\xc4\x1f\x48\x3a\x4d\xf0\x1b
\x86\xf0\x3e\x0a\x91\x37\x43\xd5\xc4\xa7\x55\x76\xc4\x0f\xce\x27\xb0\xaf\x1e\x67\x73\xe1\x5d\x
d0\x8d\x66\x41\x24\xd2\x18\x45\x1a\xb6\x45\x3c\x43\xcc\xcf\xfe\xfe\x3e\xa3\x1a\x1d\x88\x7b\x4e
\x10\xb0\xf4\xcc\x4d\xc1\xd8\xb5\x6e\xd7\x57\x1d\x97\x61\x2d\x37\x92\xc3\x21\x0b\x56\x26\x93\
xe8\x24\x51\x21\x4f\x63\x2e\x62\x3d\x56\xfb\xec\xd6\xa8\x8b\x31\xa2\x11\xf8\xfd\x6c\x60\x47\xcf
\x28\x50\xb5\xe3\x6e\xde\x71\x8b\xbf\xb0\xba\x0a\xc6\x54\x79\x55\x42\xc0\x89\xfb\xa0\x3c\xe2\x
8b\xa2\x27\x78\x4f\x1f\x4d\x08\x4e\x62\xcb\xf4\x80\xb7\x62\xf4\xd4\xe2\x39\x7a\x07\x2d\xc6\xc3\
xba\x66\x91\xa1\x48\xb6\x3c\xeb\x0b\xb2\x70\x5f\xe8\x39\xec\x4d\xc0\x0e\x04\x6b\x13\xdf\x9c\x
c5\x99\x7a\x78\x47\x56\xe4\xf5\x71\x39\x91\x8a\x81\x8f\xef\xc5\xc0\x7b\x31\xf0\xaf\x2d\x06\xaa\
xf7\x79\x6c\xd1\xdc\xd5\x0b\xbd\xbb\xb9\xbb\xa7\x20\x6f\x84\xba\xb1\xd2\x58\x19\xce\x89\x3c\x
1a\x86\xb0\x42\xa6\x9f\xda\x29\x92\x7b\x59\x13\xb9\xf4\xd3\xb8\xb8\x07\x9e\xa7\xf2\x95\x64\xb
0\xa9\x81\x81\x1b\x7e\x3d\x4c\x9b\x32\x84\xd6\x33\xb4\x12\xcc\xb9\xb3\xaf\x88\x95\x63\x28\x5d
\x41\x63\xf0\x26\x4a\xa3\x29\x56\x8e\x00\x28\xcb\x62\xa8\x30\x54\x01\xc2\xc3\x88\x02\xd7\xf6\x
fb\xb9\x81\x21\xa7\xe2\x7c\xde\x60\xff\x1e\x63\xca\x61\x48\x6a\xba\xf4\xb4\xc4\xbf\x51\x54\x30\
x8f\x0f\x55\xf1\x25\xa6\x18\x1c\x53\x7a\x36\x29\xd3\xb9\xbc\xed\x4b\x54\xb4\x69\xb6\x07\x24\xe
6\x20\x82\xb7\x51\x19\x41\xc2\xf0\x20\xaa\x85\x28\x91\xc4\x21\xed\xf8\x84\xfb\xc2\x82\x0a\x36\
x32\xa5\xc9\xb3\x31\xf3\xbf\xa9\x2e\x29\x78\xc0\x0d\xbe\xed\xca\x71\x0e\xd0\x1b\xca\xca\x09\x
2e\x78\x58\x5d\xc0\x87\xe3\x78\xd2\x70\xe6\xd9\x1a\x6f\x62\x50\x57\x6f\x97\x49\xa2\xdc\x72\x0
4\x54\x8a\xc4\x17\x04\xae\xcd\x7c\xb8\xfb\x63\xc6\x78\x69\x23\x84\xd3\xd2\xdc\xf8\x59\xc7\xb2\
x69\x38\x8f\x74\xdc\x0a\xd1\xf3\x20\x9f\x16\x8d\x88\x05\x85\x60\x81\xbe\x80\x3a\xf0\x5a\x3d\x1
4\x48\xb3\xd2\x8f\x49\xbd\xf6\xd6\x51\x67\xa8\x4c\xa9\x71\xa1\x26\xff\x30\xcc\x96\xf2\x68\xc2\x
23\x4a\x78\x7d\x4a\x78\x70\xc6\x5d\xbb\x32\xaa\x03\x8c\xcb\xe3\xa6\x8d\x23\x06\x7a\x48\x21\x
5d\x14\x19\x14\x27\x93\x3c\xb8\xd0\x6a\xa9\x45\xc5\xba\x87\xb5\x56\x90\x8f\xef\x65\xa3\xa7\xb
4\x25\x40\x4a\x8f\x8b\x01\x82\x98\xc1\x75\x31\x7a\xd0\x53\xf5\x9b\x91\x36\x14\x39\xa5\xbc\x40
\xfb\x6c\xf0\xa4\xef\x60\x9d\x31\x7b\x19\xcc\x53\xc7\xbc\x8b\x78\xe6\x70\xb7\xfe\xa0\x68\xba\x1
f\xae\xc0\xbd\x27\x26\x8a\x0a\x27\x6a\xa3\xd0\xde\xa9\xc0\xc1\x0d\x9c\x79\x9e\x7a\x01\x64\x55
\xde\x58\x24\x1c\x17\xbe\x28\x44\xe2\xf1\x94\xa0\xc3\x15\x82\x11\x45\x62\xd1\xb6\x42\x42\xbb\
xb0\x42\xba\xfb\x55\xbe\x89\xd8\x5e\x91\x57\x76\xb6\xcc\x85\x09\x00\xd6\x96\x81\x0e\x08\x79\x
3a\xbb\xe8\xc9\x33\x8a\x5f\x05\x22\x74\x19\xa0\x56\xaa\xd0\x64\xd4\xb9\x51\xd6\xf5\x1b\x0e\xa
a\x84\x4b\x5c\x86\x4f\x53\xd4\x1a\xfd\xa2\xa3\x8b\x66\x88\xa1\x8d\x96\x24\x89\x01\x61\x7c\x50
\x34\xd3\xf1\x67\x0b\xdc\xfe\xe3\xd1\x8b\xa3\xf5\xf5\x75\x90\xed\xbb\x05\x5a\x4e\x93\xcb\x01\x6
2\x11\x1b\xe8\x69\x60\x59\xd0\x0d\xb1\x94\xad\xa4\x9a\x0b\x59\xfa\x5b\x18\xd5\xc8\xeb\x11\xc
a\x38\x20\x43\x3e\xb6\x36\x3c\x2f\x65\xa3\x5f\x4e\x68\xf6\xc9\xe6\xe9\x29\x95\xb9\xf4\xcf\xab\x
2b\x69\xb4\x69\x83\xb2\x1f\x5b\x50\x86\x8e\x65\xcf\x7f\x4f\x64\xd5\x0e\x90\x48\xe3\xc2\x0e\x7a

\x25\xa2\xaa\xae\x50\xe5\x8d\xba\xb2\x38\x65\x21\x4f\x52\xff\x9b\x2c\xe4\xf8\xf5\xe6\xc2\xbb\x3a\x0a\xaf\xe2\xf7\x19\x59\x11\x2b\x7c\xa1\x09\x8c\x83\x3a\xb4\x65\x8a\x93\xea\x56\x4a\x5d\xce\x18\xb1\x57\xa4\x6d\x9d\xc7\x2e\xcf\x6e\x98\xc1\xf3\x76\x74\x66\x26\x2d\x22\x2d\xeb\x19\x6f\xf8\x14\xb3\xbb\x46\x35\xd5\x43\x70\x1c\x3f\xb1\xff\x68\x56\x5b\xcf\xce\x22\xac\x15\x4e\xe3\x66\x7d\x22\xe7\x90\xcb\x1c\xc3\xf5\xe8\xfb\x77\xcf\xa5\xab\x26\x66\xc7\x32\x8e\x52\x29\x69\x92\x94\x6b\x5c\xfc\x4e\xa1\x72\xd7\xc7\xe3\x60\x30\xb8\xd6\x43\xb6\xd9\x6e\xfe\x94\x1a\x53\x14\xf5\x70\xd2\x26\x1f\xf6\x95\xee\xe5\x57\x21\x42\x41\x03\xa6\x0f\x7a\x7d\xd6\xaa\x10\x2d\x63\xc5\x7b\xb5\x3a\x6f\x84\x01\x4c\xeb\xcb\xbf\x6f\xef\xb5\x3e\xf7\x5a\x9f\xbf\xb6\xd6\x87\xab\x7c\xe2\xd1\x2d\xee\xfd\x7c\x5a\x1f\xa9\xab\xd1\xd5\x3e\x8c\x3b\x49\x7d\xce\x8b\x67\x06\x23\xa1\xc3\x30\x1d\x7e\x38\x7a\x0a\x18\xa9\x95\xbc\x57\x13\x19\x6c\x4d\x09\x4c\x45\xcf\x63\xd1\xcf\xaf\xb7\xd0\x17\x64\x89\x57\x96\x20\xd4\xa3\x35\x6b\x5b\x0b\x07\x72\x94\x2e\x3d\x5f\x07\x2d\x6d\xb3\xda\xe6\xab\x23\x14\x2d\x96\xa5\x7c\xba\x96\xe2\x73\x8e\x4d\xcd\x81\x1e\x95\x3a\x42\xd4\x95\x70\x56\xe0\x8c\x10\x75\xe3\xd1\x27\x5f\xae\x90\x13\x77\x64\x9f\x64\xa3\x53\xdc\xae\x51\x09\xe7\x6d\xd4\x97\x2b\x1a\xdd\x76\x1b\x5d\x2c\xcb\x57\xf8\xa2\x79\x98\xaf\xf0\x45\xd5\x18\xcd\xac\xfa\x01\x36\xb7\xc5\x80\xaa\x86\xe6\x6f\xcb\x1a\x17\xdf\x8d\x4e\x14\x9c\x98\x88\x40\x21\x39\xe0\x43\x0f\x78\xb7\x00\xf8\xb4\x62\xeb\x7a\xf1\x6c\x4f\xee\x5a\x8c\x76\x3a\xe1\x0e\x6c\x51\x4f\xee\xb7\xa8\xfb\x2d\xea\xaf\xbd\x45\xa9\x8b\x09\x5c\xce\x6e\x74\x2b\xc1\x81\xef\xf6\x4d\x62\x45\x7c\x75\x5f\x80\x75\xdf\x15\x88\xff\x16\xa4\x61\xdb\xa4\x20\xc2\x18\xd9\x02\x5a\xf0\x64\x01\x36\xae\x6a\x6f\x9c\xa5\x13\x32\x15\x60\x5a\xec\x1b\x1d\x5a\x84\x52\x11\x60\xe7\xfc\xd1\x9a\x71\x3d\xc3\x13\x05\xcc\x8f\x70\x8a\xb7\x91\x01\x89\x02\xe4\xb0\xf8\x70\x99\x8e\xd9\x16\x63\x04\xba\x67\xa9\x02\x8c\xb2\xe2\x1c\xdb\x40\x3c\x55\xd6\xc5\xdc\x13\xe9\x10\x64\x14\xa5\x22\x9b\xf9\x3c\x74\xfa\x23\x92\xa5\x10\x02\x1e\xd3\xda\xdc\x18\x48\x8d\x37\x7f\x21\x08\x5a\xc0\xcd\xd3\x3e\x7a\xf0\x00\xf1\xdf\x03\x50\x0a\x1e\x4d\x7a\xdd\xcd\x8b\x2e\x73\x5c\xb2\xd9\x47\x4f\x51\x07\x97\x33\xba\x7b\x40\x24\xd2\x67\x97\xaf\xa2\x62\xd6\x41\xa1\x9d\xcc\xf4\xb9\x1d\x25\x25\x68\x01\x9f\x7e\xcc\xb3\xf9\xb3\xdf\xa0\xa7\x5d\xde\x25\x2d\x8e\xd0\xb3\x4b\x68\x98\x76\xfa\x20\x8d\x0f\x69\x39\x19\xbe\xcb\x0b\xc9\xc6\x21\x61\xd5\x78\x96\xe9\x38\xc1\xbf\xd1\x00\x8e\x69\x5b\x0d\x5d\xd7\x61\x2a\x3b\x2d\xe6\x47\x1b\xe7\xf3\x6c\x99\xb6\xba\x64\xba\x83\x71\x78\xdb\x66\x24\xa4\x0f\xa5\x02\x8c\x8d\xca\x99\x82\xdf\xb0\xff\xc7\xb2\x41\x6d\x32\x9c\x49\xd0\x01\x8c\x3e\xcb\xee\xbd\x2c\x67\x77\x7d\x40\xf8\xcd\x0f\x07\x10\xf2\xb7\xfa\x70\xc0\x94\x1f\x8c\x8d\x13\xec\xed\xd2\xc2\xe8\xcd\xa2\xa1\x23\x8b\x1b\xf4\x41\xbb\xe3\x66\xfc\x95\xf9\xbf\x40\xba\x23\xef\xc3\x67\x07\x6f\xad\xf8\x63\x9c\xab\x32\xc5\x0c\x7b\x40\xcb\xd5\x33\xd7\x6b\x6b\xac\x77\x03\x66\x19\x25\xdf\xd2\xbc\x2c\x67\x4a\x21\x14\xa0\xae\x1e\xad\xb9\x1b\xf0\x69\x9e\xe2\x32\xac\x50\x7b\x0a\x5f\xa5\x03\xbd\x20\x1f\x49\xc0\x55\x75\x46\xe1\xb3\x28\x31\xfc\xb5\x0f\xac\x58\xd9\x67\x51\xe2\x38\x23\x91\x69\xd7\x6b\x80\x9e\x95\x86\xc2\xfd\xfc\xdd\x64\x30\xbc\xe8\x4d\x86\xc3\x8b\xb6\x1c\x50\x9b\xd3\x28\xe5\x2f\x51\x02\x96\x9b\x8d\x67\x27\x0e\xe8\x9e\x9f\x04\xa3\x72\xf2\xe5\x21\x4a\xb3\xe6\x34\x6e\xf1\x42\x74\xa2\x44\x2a\x76\xc3\xc7\xdd\x68\xfe\xa8\x2e\xf4\x6c\x08\x3d\xd8\x39\xe3\x28\x12\x58\x8b\x16\x69\x5d\x65\x85\x7a\x35\x2c\x4f\xfa\xac\x91\x40\x15\x07\xe7\x2c\x8f\xa6\xf8\xa0\x6c\x73\x76\xe6\xa0\x95\x38\xf2\x41\xc8\x63\x6d\x0d\x96\xd8\xba\x63\x3c\xbb\xcc\xe0\x6c\xb9\x0a\x5a\xbc\x03\xe3\xce\x1d\x1b\xc6\x44\xa1\x2a\x87\x63\x65\xfe\xf6\xf3\xed\x1d\x98\x58\xf5\x4d\xf4\xcc\xd8\x91\x35\x34\x29\x34\xde\x6e\x58\xbe\xde\x06\xce\x12\x57\xf6\xaf\x74\xf1\xa2\xeb\xd5\xe8\x97\x36\x51\x4f\xbb\xb0\x03\x37\x63\x02\xc0\x1c\x4c\x48\x99\xee\x6b\x60\x42\x23\xe5\x5b\x0c\x3a\x58\xab\xa0\xec\xf9\x82\x24\xec\xf8\xd6\x48\xde\x1c\xb4\x86\x

c6\x5d\x08\x81\x87\xcd\x6a\xfa\xb3\x25\xb6\x96\xf4\x68\x17\x73\xba\x55\x27\xb4\xba\x1d\xdc\xba\xe5\x44\xd5\xcd\x8d\x98\xc2\x17\x78\x4c\xe6\x51\x52\x8d\x0a\x25\x07\xb6\x44\x82\x2a\x50\x41\x94\x7f\xdc\x01\x9b\xc2\x53\xc3\x60\xab\xc3\x23\x57\x1c\xc2\x40\xc2\xae\x1d\x74\xf3\x0a\xd2\x2a\xac\x67\x1e\x1f\x3d\x67\xd4\x95\xc6\x24\x4b\x39\x83\xab\x3a\xfe\xfe\x91\x38\xcd\x4d\xf0\xf4\x1e\x8f\x31\x59\xb4\x20\x73\xb7\x4c\x1b\x02\x70\x41\x6f\x4b\x01\xbc\xc6\xd6\x03\x6c\xb9\x8a\x1b\xb9\x98\x67\x70\x36\x60\x1b\x0a\x60\x62\xd1\x1d\x09\x88\x8d\xcb\x9b\x1e\x90\xde\x47\xe7\xed\x97\xb8\x5b\xc0\x8f\x88\x5a\xb8\x36\x9c\x8d\xe2\xc1\x23\x0b\xb9\xd1\xa4\x9b\x7a\xdb\xaa\xab\x37\xef\xa7\x3d\x53\xbe\x35\xe6\x1b\x07\x99\x36\x76\x9e\x4c\xab\x7a\x6c\xe6\xdc\x8d\x8c\x5a\x81\x70\x6e\x17\x55\xd7\x51\x88\x91\xef\xed\xa8\x95\x73\x93\x8e\x52\x1e\x7a\x67\x92\xb4\x19\x4c\xbc\x6e\x4c\x1a\xa4\x7f\x68\x7e\x80\x9b\x50\x8c\x31\xc2\x5b\xad\xe6\x31\x93\xeb\x44\x08\xf0\xa6\x69\x63\xd0\x03\x11\x16\xbc\x62\x0a\xcd\x3a\x7d\x63\xad\xec\xc8\xeb\xd7\xaf\x5b\xf6\x21\xa9\xa4\x20\x59\xd3\x4a\x2d\x7f\xc0\xf9\x02\x37\xb2\x75\x89\x01\x06\x5d\x8f\x00\x07\xa6\xa6\x17\xc5\x72\x34\x27\xe5\xcf\x59\xde\x24\x5d\x28\xc0\x8a\x95\xee\xcb\xaf\xbf\x34\x6e\xd1\x2a\x87\xaa\xdc\xc2\x2a\xda\xb3\x8e\x06\xce\xc5\xb1\x52\x98\x04\x7a\x9a\x54\x10\x18\xa9\xf4\x9c\x6d\x24\xc0\x12\x36\x52\x40\x66\xb6\x0a\xf1\x83\x8b\x5b\xd2\xde\x76\x5d\x08\x25\x82\x1b\x79\x5a\xc1\xaa\x74\x29\xd0\x55\x01\x70\x99\xa3\x2a\xdb\x6a\xd4\xb4\x85\xd5\x18\x89\x4a\x74\xb7\x4f\x33\xcf\x9f\x41\xa6\xda\x97\xb5\x70\x9d\x8c\xd7\xaf\x5f\xbb\xc0\x8c\xc8\xb5\x2a\x25\xfd\x19\x63\xa3\x09\xf0\xcd\x4d\x00\x58\xc8\x32\xa9\x0c\xae\x73\x61\xac\xc8\x85\x52\xa8\x54\x4a\x9a\xc6\x95\x72\x7d\x92\x74\x14\x15\xd8\x0a\x97\x38\xc5\x8c\x1f\xf2\xe5\xc9\x61\x24\xc8\x75\x3f\x58\xa1\x8d\x39\xf1\x04\x64\x34\x5a\xe0\x10\x37\xac\x7f\x16\x15\xb3\x3c\x2a\x6b\xc7\x50\x01\xd3\x6a\x03\x58\xbd\x47\xe2\xfa\xb2\xa6\x43\x7e\x90\x66\x31\x9c\xdf\x97\x9a\xb2\xf7\xea\x3d\x9c\x46\xc5\xbb\x9c\x8c\x6b\x71\x56\x01\x73\x63\x1d\xe9\xea\xbd\xe4\x51\x79\x8a\xba\x5e\x4a\x98\x1b\xb6\x31\xd2\xee\x98\x6a\x9a\xa9\x06\xfb\x4a\x34\x24\x42\x16\xfc\x83\x19\xa3\xd4\xf5\xcd\x06\xd5\x5a\xd4\x59\x88\x71\x2b\x31\x18\xab\x8b\x7e\xed\xce\x6f\x44\x0c\x9b\xfe\x68\x5c\x66\xb9\x90\x72\x84\x69\x00\x18\xda\x06\x88\xc2\x1a\xd6\xb6\x1c\xda\xd7\xd8\x44\x98\x02\x68\x8f\x6f\x48\xe2\x7a\x24\xd2\xa2\xf4\x30\xeb\x81\x9e\xef\x5e\x2f\x40\x50\x98\x9b\x1b\x0c\x70\x39\xeb\xf5\x03\x97\x0c\x5f\x67\x53\x4d\xac\xb5\xfc\x01\x99\xfd\x53\x06\x06\xf5\x8e\xe1\x05\xd2\x7a\xbc\xc0\x60\x9a\x64\xa3\x28\x19\x50\x5c\x0c\x22\x37\x99\x47\xf2\xf2\x35\x49\xc6\xd1\xe2\xed\x4d\x9b\xa5\x85\x9d\x46\x59\x62\x5d\x93\x9a\xb5\x85\x6a\xb0\x66\x0e\xa4\x79\x46\xc5\x34\x34\xf9\x58\x7a\x59\xce\x34\xf3\x6a\xcb\xe2\xa4\x13\x6e\x3d\x09\x3a\x8e\xe5\x0b\xb7\xbc\x56\x26\x27\x9d\x70\xfb\x1b\x48\x60\x44\xd4\x09\xb7\xbf\x63\x9f\x72\xbe\x3b\xe1\x0e\x2b\x42\x46\x51\xda\x09\x77\x76\x02\xd3\x2e\x0e\x3e\x39\x96\x3a\xe1\xee\x2e\x7c\x0b\xfb\x98\x4e\xb8\xcb\xaa\xe7\x1c\xb9\x13\xee\xb2\x6e\x89\x3b\xcc\x4e\xb8\x4b\x1b\x14\xd6\x2d\x9d\x70\x77\xe7\xfa\x34\xd8\xf9\xee\xde\xd0\xee\xde\xd0\xee\xaf\x6d\x68\x57\x65\x65\x77\x6b\x63\xf0\xf6\xf6\x6f\x2d\x8c\xdb\x00\xee\x2d\x2e\xbf\xa6\xed\x38\xa4\x7e\x6d\x53\x8c\x16\xb6\xe2\xc3\xe1\x50\xb9\x5a\xf1\xb9\x6f\xe1\x71\x08\x29\x8f\x87\xea\x70\x39\x43\xd1\x82\x68\x7d\xff\x4a\x07\x89\xaa\xc0\xeb\x52\x50\x31\xa3\xb3\xdf\x54\x28\xc2\x38\xb7\x55\xbe\x4e\x2b\x55\x40\x2b\x08\x6a\xba\xd8\xe4\xec\x6a\x6f\x71\xb9\xe7\x6e\x6a\xe6\xe6\xa5\xef\x2e\xd7\xa7\xc1\xee\xe6\xfd\x6e\x71\xbf\x5b\xfc\xb5\x77\x8b\x3f\xa8\x59\xf6\xdd\x59\x50\xb7\x34\xf0\x56\x36\x8a\xef\x70\x5e\x64\x69\x94\xfc\x6e\x86\x8a\x92\xa3\xfd\x11\xec\x14\x6f\xb5\x37\x36\x74\xe1\xba\x9d\xe1\x5a\x8a\xcf\x95\x35\x5c\x9d\xfa\x56\x01\xba\x1a\xdc\x05\x9f\xd5\x4f\x5e\xa0\x5b\x5c\xbc\x2d\xd3\x24\x1b\x7f\x6e\xd7\x41\x03\xb6\xa6\x8f\x55\x70\x6d\xcc\xc7\xda\x5d\x68\x55\x5e\x6b\xdd\xf1\x3d\xa2\x1c\x52\xf3\x65\xe2\x2a\x97\x5f\xbe\

xcb\xc4\xca\x21\xb5\x9f\x9f\x76\xb3\x53\x3f\x37\xab\xdc\x70\xd9\x73\x63\x75\xde\x27\xd0\xf1\x8
6\x55\x23\x06\xad\xb4\x50\x86\x6b\xd0\xba\xbc\xf4\xf5\xe4\xcb\x83\x06\x75\xa0\xc2\xab\x0e\x5c\
x21\xd6\xb9\x72\x9d\x60\xd7\x4a\xb8\xf3\xa8\x18\x2c\xad\x82\x2d\xe7\x6d\xdd\xcb\x79\xf7\x72\x
de\x5f\x5b\xce\xe3\x42\x5e\x31\xbb\x6b\xad\x40\x0b\x49\x6d\x85\x37\x76\x2d\x5e\x9f\x35\x3d\x6
2\x03\xa0\x0f\xb3\xd9\x57\xd7\x2c\xfc\x81\x1e\x9d\x50\xd6\xf8\x61\xf6\x9b\xe9\xee\x8b\x99\x5f\x
77\xff\x2e\x2b\xca\x6a\xe5\x7d\x8d\xac\x87\xb4\xa7\xe4\x59\xd1\xb4\x4d\x53\x90\x6e\x80\xec\x0
d\xba\x98\xcd\x3e\xd9\x59\x37\xd9\x95\x8d\x51\x78\xe4\x89\x14\x9f\x1f\x42\xd4\xe1\x46\x4b\x12
\x0d\xd2\x95\x27\x68\x77\xfd\x00\x35\x26\xc7\xb3\xa8\x68\xd9\xb6\x06\xe9\x6f\xdb\x0f\x50\x63\x
bc\x91\xe2\xf3\x9f\xf2\x6c\xb9\x68\x1e\x34\x80\x55\x8e\xd8\xce\xad\x19\x6e\x14\xc7\x1f\xb3\x36\
x8d\x2a\x40\x7f\xb3\xde\xfc\xb6\xf6\x1b\x94\xa8\x0c\x7d\x97\xc0\x9b\x4a\xd4\x90\x69\x40\x42\x9
3\x2a\x45\x75\xa3\x56\x57\xf5\x61\xa6\xdf\xc0\x34\xdf\xa0\x18\xe2\x8d\x75\xe5\xb1\xbb\x7d\x2f\
xdc\xdc\x0b\x37\x7f\x6d\xe1\x46\x29\xb1\x46\xbf\xfe\x6a\x29\xb1\x0e\x12\x7c\x81\x9e\xe1\x1c\x4
f\x8b\x5f\xa3\xe2\x57\x82\xbe\x8f\x12\x7c\xf1\xb7\xbc\x9c\x14\x83\xd9\xd2\x14\x6b\x1e\x73\x07\x
5c\xef\xf1\x04\xe7\x38\x1d\xe3\x10\xd1\xf6\x8b\x70\x38\x9c\x92\x72\xb6\x1c\x51\x49\x68\x88\x29
\x39\xe1\xe5\x7c\x38\xcd\x1e\xc9\xdf\xff\x97\xbd\x77\x5f\x6f\xdb\x56\x16\x47\xff\x4e\x9e\x02\xed\
xef\xac\x46\x8a\x69\x5b\xd4\xcd\x97\xc4\xdd\xdb\x91\xed\xd8\x2b\x17\xe7\x67\x3b\x6d\xd7\xf6\x
e7\xe6\xa3\x44\xc8\x66\x23\x91\x5a\x24\x65\xcb\x6d\xbc\xdf\xe7\x3c\xc7\x79\xb1\xf3\x61\x70\x2
1\xae\x14\xe5\x4b\x9a\x76\xd9\x6b\xef\x54\x24\x81\xc1\x00\x18\x0c\x06\x83\xb9\xf4\x47\x49\x7f\
x35\xbb\x0a\xd2\xf1\x6a\x14\xe7\x38\x8d\x83\xd1\x2a\xe9\x12\x9e\xe5\xfc\xbf\x2b\xe7\xc9\xff\x79
\xdb\x6a\x3d\xb0\xce\xab\x50\x64\x1d\x13\x6c\x1e\xb5\x58\xdf\x88\x16\x8b\x9a\xa2\xe0\xfc\x2a\x
49\x3f\x1f\x61\x88\xae\x56\xb6\xa1\xe9\xc5\xcd\x6d\xad\xff\xfb\xef\x9f\x4a\x4a\xdd\xc5\x96\xfc\x3
a\x1e\xec\xc6\x41\x7f\x84\xe7\x61\x29\x95\xb4\x23\x68\x2f\x70\x17\xdc\xae\x82\x49\x45\xdc\x8a\
x92\x0e\xdc\xac\x05\xee\x80\x5b\x98\x5c\xc5\x2c\x70\x5e\x19\x62\xbc\x98\x1d\x2b\xcb\xd7\xea\x
ee\x11\x2e\x1d\xe5\xa4\x02\x5a\xb4\x90\x1d\x29\xe3\xdb\x9d\x51\x4a\x71\x9e\x46\xf8\x72\x9e\x
c7\x22\x2f\x66\x47\xcb\xf2\xf5\x2e\xa4\x95\x93\xdd\x6e\x0e\x51\x91\x32\x0e\x72\xd2\x3e\xdd\x7
9\x88\xce\x71\x05\xf7\x1b\x3b\x2e\xea\x87\x3b\x8c\x09\x0d\x38\x3c\x27\xaa\x97\x1d\x07\xf5\xc3\
x9d\x47\x83\x05\x18\x2f\x47\x86\x16\xb2\xe3\x63\x7c\xe3\x28\xb5\x2b\xa1\x54\xa2\x37\x37\x0e\
x0c\x3a\x5b\x96\xd4\xb2\x05\x3f\x94\x5e\x16\x8c\xa8\x78\xc9\xf9\x80\xa4\xe9\x9d\xa8\xcf\x9c\xf
a\x25\x40\x84\x04\x15\x83\x64\xa9\xb7\x53\xe9\x41\x92\xc5\x1f\x54\xff\x7b\x11\x5d\x3a\x8d\x55\
xc9\x9c\xc0\x77\xf1\x79\x31\xab\x82\x28\x1e\x26\x65\xb0\xe1\xbb\x04\x5b\xee\xad\xe5\xc8\x45\x
84\x25\x5b\x78\x36\xab\xc2\xb8\xf5\x78\xa6\x7a\x3c\x53\xfd\xbd\xcf\x54\xec\x40\xc5\x55\x7e\x5f\
x37\xae\xe8\x6d\x2c\xa6\xb8\xca\x33\x98\x44\x5c\x18\xa7\x29\x69\xf2\x8b\xb2\xab\x65\xaa\xf5\x
2c\x8d\xa3\xc4\x4b\xe7\xd7\x13\x22\x1f\xb0\x98\x49\x52\x8e\xdc\xec\x2a\xca\x07\x17\x35\xf2\x5
d\x0f\x87\x3d\x08\x32\x8c\x9e\x11\x8a\xcf\xf2\x67\x9b\xca\x27\x98\xac\xf4\x3c\x5b\xc9\x2e\xa2\x
a1\x23\xeb\xbc\x9c\xcc\xa6\x61\x16\x60\x2c\x19\xec\xc9\x63\x7c\x45\x23\x26\x50\x6d\xf9\x0b\x0
b\x1a\x13\x1c\x87\x51\x7c\xfe\xe0\x78\x7c\xa0\xed\xc8\xb7\xb4\x36\xa4\x58\xa8\x1b\x13\x1b\x0d
\x9c\x51\x99\x25\xcd\x56\x75\x93\xe2\xea\x60\x8e\x72\x92\x41\xd3\x65\x04\x85\x14\x2a\xe9\x43
\xa7\x71\x14\x67\x79\x30\x1a\x55\x6a\x59\x2b\x6d\xf7\x31\x73\x17\x2a\xc1\xe3\x1c\xe7\x6f\x93\x
f3\x0a\x01\x30\x48\x29\xa7\x6f\x1b\x6d\x51\x2b\x52\xd2\xea\x24\x99\xeb\xff\x4a\x8a\xcc\x69\xaf\
x77\x11\xc4\xe7\xb8\x42\x93\x36\xe1\x83\x82\x90\xaf\xc2\x95\xd1\x53\x04\x21\xd2\x31\xa9\x91\x
64\x34\x92\xe5\x81\x85\xf9\x4d\x76\x71\xb1\x02\xac\xd1\x60\x37\xd9\x45\x05\x76\x73\x6b\x2a\x
e5\x3a\xf4\x45\xe8\xe3\x9e\xe9\x94\x60\xf0\x67\xd0\x29\x69\xf7\x1c\xe7\x2c\xbd\xe4\x43\x53\x2

9\x6b\xed\x9b\xa2\x52\x43\x68\xa5\x7d\xc1\xf9\xc5\x26\xf9\x87\x56\xcc\x2e\x2e\x36\xc9\x3f\x54\
xce\xb5\x45\x1b\x6e\xb7\x1f\xa5\xd7\x47\xe9\xf5\x6f\x2e\xbd\x16\x57\x02\xdc\xd1\xea\x9e\x12\x2
1\x52\x27\xb1\x23\x7c\x4e\xe6\x39\x48\xb7\xfb\x91\x23\xe8\x6e\xb6\xfa\x5a\x2d\x0a\x39\x6c\xb9
\x26\x3f\x1a\x04\x13\x19\x88\x0b\xc6\x41\x6f\xfb\x83\x09\x41\xc2\x84\x79\xa3\x31\x53\x32\xb4\x
85\x9e\x35\x66\x83\x6e\xb8\x11\x36\x07\x61\xbb\xbd\x11\xac\x75\xda\x83\xf6\x46\xbb\xd9\x6d\x
63\x7f\xbd\xb1\x31\xe8\x34\x70\xab\x1d\x76\xdb\x9d\x6e\xb3\xff\xac\xc0\xc5\x06\x26\xf0\x03\xdf\
xf7\xfb\x83\xc6\x5a\x7b\xb0\x31\x18\x06\x6b\xeb\xfe\xb0\x31\x68\xad\xe3\x6e\xab\x1f\x76\xfc\xc1
\x86\xdf\x5f\x0f\x86\x8d\xc6\x33\x37\x6f\xa2\x38\x6e\x4a\x42\x71\xd0\x8f\x36\x2d\x83\xa8\xdc\x9
1\x12\x14\x36\xad\xfd\xa3\xec\x96\x16\x26\x68\x1b\x90\xf5\x71\xb5\xc0\x35\xbb\x4b\xa1\x2a\x1c\
xb3\x7c\x16\xbf\xdf\xf4\xbd\xef\xe7\xcc\xd3\xf7\x9b\x4d\xc2\x6c\x3b\x8f\xcc\xf6\x91\xd9\xfe\xbd\x
99\x6d\xc1\x6b\xb9\x9e\x4c\x63\xb6\x65\xbe\x01\xc3\x34\xf9\x1d\x8f\x83\x78\x25\xc4\x3f\x7e\xad
\x4c\xb5\xfa\x5d\xea\x5d\x32\xd1\x52\x9d\xa3\xf4\x9d\xbe\x50\x82\xed\x6a\x25\x32\xbd\xc4\x6d\x
d2\xca\x2e\x9e\xbc\xb6\x24\x11\x2d\x1f\x8b\x87\x4f\x45\x5b\x35\x83\xe5\x9d\x13\x58\x5a\xba\x5
4\x92\xc2\xd2\xe6\xbf\xad\x8d\xf0\x7f\xd9\xde\xd2\xba\x90\x4c\xf2\x1b\x49\x1e\xe9\xec\xf7\x3d\x
a5\x8f\xfc\x6e\x8b\x12\x8e\xf6\xaa\x48\x99\x7f\x87\xfc\x92\x7f\xd1\x0c\xbc\x96\x21\xff\x36\x73\xf
0\x6a\xc3\x4d\x4d\x21\x0a\xcc\xf2\xc4\x4e\x8a\xaa\xc9\x84\x8b\x18\xd9\x2b\xde\x0b\x27\x35\x56
\x4f\x4e\x09\x75\x84\x28\x89\x78\xec\xed\x22\x59\xe5\x6b\x9c\xd7\xa4\x3b\x23\x1c\x4f\xc7\x38\x
0d\xfa\x23\xbc\x89\xf2\x74\x8a\x4d\x2d\x61\x30\xc6\x59\x69\x22\x4a\x29\x5b\x25\x14\x06\x3d\x2
f\x92\x32\x54\x66\x73\x52\x54\x66\x5a\x8e\xca\xcc\x91\xa4\x52\x2f\xf2\x42\x51\x4b\x88\xe6\xfd\x
33\x25\x49\xae\x3d\x70\x79\xd2\xff\xcd\x83\xf2\x1e\x1d\x32\xd6\x17\x02\x3f\xc8\xae\xe3\xc1\x6b\
xd8\x6f\x88\xc8\x0b\x5d\xa8\x9f\x29\x19\x3f\xb7\x59\x91\x9a\x64\xa6\xab\x55\x53\x26\x09\x40\x
a8\x2c\x03\x2e\xa2\xd1\x12\xe0\xb0\x32\xb8\x08\xd2\xed\xbc\xd6\xa8\xaf\xe4\xc9\xc7\xc9\x04\xa
7\xbd\x20\xc3\xb5\x3a\xff\x0c\x39\x03\x6b\x7e\xdd\xb9\xf1\xf0\x99\x75\xa7\xd5\x2a\x36\xee\x22\x
47\x18\x8f\x66\xc2\x6b\x9c\x93\x0e\x99\x2b\x46\x08\x28\x4a\xfa\x48\xf1\xd6\x96\x40\x52\xd5\xe
7\xf3\xbc\xad\xa2\x0a\xdd\xee\x0b\xd5\xb4\x94\x68\xb2\xac\x83\x7c\xd4\x17\xeb\x65\x61\x16\xe0
\x8e\x04\x82\x0a\x8b\x28\x6b\x87\x68\x52\xcf\x05\x7b\x55\x31\xf9\xa7\x9a\xf0\xd3\x3e\xd8\x66\x
ca\xcf\x1b\x35\x37\xe7\x39\xce\x17\x4c\xcd\x79\x8e\x5d\xdb\xc9\xb7\x9d\x99\xd3\x42\x1c\xd5\x7
3\x73\xea\x16\x76\x9b\xb2\x3c\x6a\x6a\x2a\x4f\xcf\x54\x1d\x27\x4d\x27\x6c\x4d\x80\x5c\x2d\xcd\
xa7\x3c\x65\x0f\x95\xec\x93\x0f\x90\x3b\xdd\x27\x39\x62\x77\x1f\x8f\xd8\x8f\x47\xec\xbf\xf7\x11\
x5b\xd2\x67\x32\x0e\x31\x66\x2c\x5d\x3d\x69\xff\x13\x0f\x87\x29\xbe\x46\x3f\x47\xa3\xc1\x67\x8
c\x5e\xfe\x86\x87\x43\x97\xc7\xfe\x42\xee\xfd\xef\x82\x94\x1c\xe1\x0f\x83\x78\x80\x03\x28\x6b\x
73\xec\xbf\x45\x2c\x00\x56\xe5\x75\x70\x89\x7e\x4e\x92\x10\xbd\x3c\x77\x1e\xf2\xdb\xc5\x21\xff\
x9f\x8c\x9b\x2a\x4e\x60\x8c\xc5\x96\xe5\x6b\xb3\x04\xab\xd1\x53\xac\xd9\xf2\xab\xd1\x84\xc2\x
6a\x6b\xf4\x1d\x35\x57\xa0\xdb\xce\x41\xfe\x2c\x23\x1b\xe3\x24\x89\xb3\xa8\x3f\xa2\x04\x36\x0
9\xb2\x2c\x8a\xcf\xd1\x98\xdd\x5b\x91\xbd\x68\x92\x26\x97\x51\x88\xd3\x4c\xd4\x0a\x46\x59\x62
\x56\x4d\x46\x23\x52\x95\x50\x1b\x37\x34\x47\x71\x12\xd2\xaf\x51\x3c\x48\xc6\x32\x64\x02\x8c\
x85\xfd\xa5\xf7\x5e\x79\x34\xc6\x64\xb1\x45\x19\xf2\x51\x86\x07\x49\x0c\x59\xf8\x8f\xa3\xf8\x7c\
x84\xf3\x24\x86\xe1\x24\xdd\x2b\x39\xe8\x73\x54\x95\xe3\x3e\x7f\x89\xb6\x44\x57\x24\x3d\x03\x
69\x1b\x34\xc0\x37\xd2\x4b\x8e\x8b\xac\x75\x70\x1e\xfe\x88\x84\x72\x91\x26\x71\x32\xcd\x46\xd
7\x34\x6d\xba\x7d\x1f\x26\x9f\x2c\xe7\x11\x14\x06\x79\xe0\x3c\x21\xab\xbd\x55\x54\x1e\x5a\x7e
\x76\x02\x46\x3e\xa9\x7d\xa7\xf4\x5e\xc9\xdb\x93\xc4\x59\x42\xb6\x2e\x42\x14\x35\x4a\x1a\x2b\
x07\xf1\x65\x30\x8a\xc2\x0f\xac\x7c\x4d\x96\x79\xb8\xdf\x1d\x0c\x86\x24\xe1\xab\x7b\x3c\x23\x7

3\x29\x05\x37\x41\x69\x85\xf6\xde\x83\x6e\x32\xbb\x0c\xe9\xfc\xc2\x4e\xe5\x5b\xea\x5c\xa9\xd9\
xd0\xd9\xa1\x08\x3a\xc5\x1b\x89\xb2\x9f\x08\xba\x47\x94\x0a\xb1\x10\xd4\xa4\x6e\xe6\x17\x69\
x72\x85\xd4\xee\xe9\xe5\x95\xee\xb0\x6e\xd2\x4f\x2b\x95\x4e\xfe\xc1\x42\xb3\x0f\xd2\x6c\x29\x0
9\xe8\xe7\x52\x21\xfd\xcc\x27\x06\x00\x6e\x50\x84\xe9\x44\x59\x4e\x1b\x3c\x2b\x94\x24\x1b\x97
\x51\xc7\xfd\x10\x82\x39\xf7\x8b\x26\xf3\x27\x9d\xc2\x69\xaa\x8b\xf8\x96\xde\x58\xb3\xf5\x2b\x7
0\x16\xa1\xb1\xf9\x43\x66\xd4\x96\xdb\x37\x84\x5c\x96\xc8\x4c\x21\x41\x3d\x40\x97\xfb\xd8\x60\
xa3\xc6\xb2\x93\x01\x29\xf0\x8a\x7c\xb7\x28\x99\x68\xbd\xfb\x20\x4c\x68\xe1\x1b\x23\x4c\xc0\x
49\xa6\x4e\xce\x64\x6e\x47\x8a\xd9\x3d\xd0\xa2\x4a\x83\x5c\xcf\x06\xb3\x51\xe3\xad\xdc\x89\xf
4\xb2\x79\xb4\xa7\x74\x48\x10\x1d\x9a\xb3\xfd\xe1\x5c\xec\xab\x44\xda\xe4\x67\x42\x26\xf2\x19
\x14\x97\xf3\xa9\xb2\xab\xe6\x4a\x69\x49\xd4\x55\x77\x7d\xe7\x76\x3f\x6f\xe7\xce\xc9\x91\x8a\x
09\x2e\x3a\xa2\xe4\xdb\x07\xf1\x69\x2e\xc7\xa6\x71\x7b\x6f\x00\xda\x41\x38\x77\xc9\x58\xbe\x0
a\x53\x1b\x8e\x49\x9e\x84\x09\x1a\x8c\x70\x10\x4f\x27\x28\x06\xf8\x64\x80\xc5\xb1\xbd\x6c\xa8\
x24\xec\x2d\x2b\x8f\x22\xe9\x49\x3a\x62\xd1\xb8\x3a\x96\x44\x38\x3a\xa5\xa5\xcf\x88\x90\x44\x
aa\x6f\x22\x0a\x24\x0a\x37\x0d\x40\x9b\x36\x90\x9b\xc5\xcf\x1b\x9e\xee\x71\x75\x55\x1f\x7d\x8
5\x01\x30\x01\x4c\xdd\xcd\x19\x42\x35\xb1\xc2\xe7\x4c\x6e\x32\x11\x42\x29\x11\x41\x99\xc5\x2
d\x9c\x6e\xce\x23\x72\xa4\x8b\x74\xdd\x31\xa9\x63\x99\x73\x63\x6e\x4b\x47\x5e\x80\x50\x89\x1
4\xea\xf2\x0e\x85\x78\x84\x73\x6c\x19\xe4\x17\xd2\xf0\x14\xf8\xb3\xd1\xa9\x31\x8d\xea\x67\x7c\
x9d\xd5\x8a\xba\x75\xae\xe5\x85\x4c\xa8\xe8\x87\x1f\x90\x6b\x0c\x09\x31\xa5\x27\xf4\x7d\x4d\x
29\xf4\x42\x1d\x67\x5d\x00\x2e\x19\xef\x62\xf7\x49\x31\xe1\x05\x44\xfe\xe7\xc3\x3e\xc6\x83\x8b
\x20\x8e\xb2\x31\x3f\x86\x96\x33\x07\x00\x50\x3e\xbc\xb4\x0d\x79\x60\x3f\x63\x3c\x11\x41\x84\x
79\x67\x57\x9f\xff\x96\x5d\x44\x31\x69\x68\x36\x48\xc6\x93\x11\x9e\x45\xf9\xf5\x66\x07\x8e\x64\
xa4\x00\x21\x88\x1a\xd9\x1c\x3e\xe3\x6b\xaa\x29\x10\xa3\x29\x8d\xd7\xea\x2a\x4a\xf1\x38\xb9\
xc4\x28\x18\x8d\xa0\x57\x99\x87\xf0\x6c\x80\x27\x39\x88\xfd\xec\x95\x5c\x3e\xbf\xc0\xd7\x28\xc
6\x74\x44\xfa\x98\xd5\x0f\x49\x8f\xa7\xc1\x68\x74\x8d\xfa\xd7\x30\x64\x64\x78\x58\x38\x60\xa0\
x99\x9f\xc9\x86\x14\xc5\xe7\xb5\xba\xb4\x0f\xd4\xbe\x53\x7a\x87\xbe\x7c\x21\xf8\x16\xf9\x6f\x09
\x00\x42\x6c\x9f\x58\x1a\xdc\x65\x5f\xdf\x20\x24\x0a\xfb\x8c\xaf\xcf\x56\xc4\x4a\xd4\x2d\xa7\x4d
\x8a\x24\xe5\x0d\x2b\xe6\xbf\x30\x79\xc2\x29\x93\xcc\xfb\x80\x1a\x48\xa2\x24\xae\xc2\x13\xa8\x
5d\x63\x19\x4d\x32\xb3\x4d\x53\x05\xea\xa0\x42\xd4\x25\xe0\x2c\x9d\xc9\x70\xae\xf4\x9e\x00\x
96\x54\x91\x1e\x1a\xac\xec\x9e\xec\x7f\xfa\x70\xf8\xf6\xed\xc1\xfb\xd7\x9f\x4e\x0e\xde\xed\x1e\x
7e\x3c\x91\x8f\x47\x55\x66\xc0\x14\xaa\x14\x89\xe9\x41\x8e\x8e\xa6\x4c\x46\xf0\xda\x09\xf2\x00
\x6d\xa1\xd3\xb3\x17\xea\xfb\x03\xf0\x4c\xe6\xaf\xab\x2d\x55\x01\x70\x65\x32\xcd\x2e\x6a\x3a\x
dd\x33\x11\x4f\x29\x7d\x10\x66\xb4\xf0\x67\x7c\x5d\x37\xc6\xa0\x00\xb8\xc0\xe0\x55\x12\x37\x0
5\x64\xd6\x28\x5f\x52\xe3\x60\xa2\x30\xc9\x08\xc8\x16\x18\x0a\x90\x18\x21\x4d\x75\x98\xde\x0
5\x13\x49\x75\x21\xe9\xb5\x55\xa7\x72\x2a\xb8\x02\xd7\xa8\xff\xa1\x8f\xc1\xbb\x60\x72\x0a\xd5\
x22\xd8\xe2\xf9\xc8\x9c\x42\xf1\x33\x3d\x4d\xb3\xe1\x62\x8f\x16\x96\x99\x13\x55\x6a\x7e\x2a\x7
3\xcf\x93\xc3\x9d\xc3\x4d\x4e\x64\x68\x94\x9c\xff\x97\x2e\x55\x27\x0e\xb9\xfa\xae\x92\x74\x05\x
65\x41\x66\x3d\x3a\xb2\x6f\x2b\xe3\x60\x52\x73\x19\x2b\xf0\x3f\xb0\x5f\x0c\x8b\x51\x26\x63\xcf\
x8e\x7a\x51\x28\xfb\xe8\x08\x8a\xf8\x8c\x51\x36\x4d\x41\x4f\xcc\x99\x55\x94\xa1\x2c\x8f\x08\x3
d\x50\x4e\x8e\x43\x14\x0c\xc1\x97\x28\x4d\xa3\xcb\x60\xa4\xed\xb5\x0a\x4c\x32\x20\x10\x21\x8
0\x2e\x8d\x28\x3c\xd3\x51\x2c\xba\xb4\x32\x28\xec\x01\xd4\x3a\xe2\x8b\xd3\xb7\x86\xeb\x4e\xe
4\x4f\x37\x08\x8f\x98\x9e\xd9\x52\x63\x18\x8c\x32\x2c\xdf\xb2\x31\x0f\xa9\xb9\x63\x2a\x32\xd5\x
b2\x36\xd1\x2d\x60\x90\x79\x81\x19\x97\x16\xad\xe3\xf0\xff\xc2\x18\xcf\xef\xa0\x66\x85\x71\xac\

xae\x18\x40\x0a\x85\x21\x25\x95\x76\x14\xaa\xa3\xa4\x2d\x76\xf7\x30\xa9\xb8\xb8\xf5\x0c\x48\x
be\xe4\x74\xa5\x5d\x38\xd2\xe3\x6f\xa8\x37\x5e\x5a\xfa\x05\x33\x7c\x35\x53\x48\x7f\xbf\xd9\x84
\x28\x40\x4c\x19\xfe\xfd\x66\x0b\x3c\x56\xd7\xaa\xdc\x91\xb1\xd8\x69\x38\xcf\xa3\xf8\xdc\xee\x0
4\x0c\x8c\x49\x4b\xb9\xce\xdd\xdb\x5e\x18\x25\x8a\xb0\x9b\xc2\x3e\xc8\x15\xf2\x88\x35\xca\xfa\
x4d\x50\x5e\x7f\xbc\xd6\x7b\xbc\xd6\xfb\x9b\x5f\xeb\xb1\xa8\x8c\xec\xd4\x62\x75\xb2\xbd\x43\x6
0\xed\x92\xe0\x8b\x96\xd8\x8b\x55\x0d\x67\xf9\x92\xf6\xd9\xe1\x60\x3b\x0c\x33\x18\x3a\xb1\xbb
\x05\x31\xa8\xa5\x32\x34\xa5\xe2\x17\xf3\x8f\xf3\x88\xf0\x15\xe5\x38\x25\x58\x26\x97\x6c\x19\xf
1\xdd\xfe\xe9\x53\xf9\x7c\xc0\xce\x67\x4f\x75\x25\x11\xd9\x36\x9f\xb2\x6b\x2b\xa9\x9c\xc4\xab\x
68\x48\x1f\x22\x06\xf1\x7d\x28\x89\x99\xd3\x98\xc2\xd1\x98\xdc\x44\xc6\xde\xa2\x6a\x74\x09\x4
5\x74\xdf\xe6\x3d\xcd\x2c\x9b\x85\xcd\x1e\x87\xff\xa9\xfb\x96\xbe\x3d\xb9\x74\x97\xc2\x42\x90\x
c7\x2c\x02\x94\x7f\xf8\x01\x70\xa7\x8a\xa9\x28\x3e\x07\x6e\x5c\x57\x20\xf2\xeb\x8b\x79\xe9\xc8
\x28\x44\xd9\xa1\xf9\xb6\x9d\x14\xd2\xd0\x28\xc8\xa0\x99\xe3\x9c\x4c\xf6\x77\x5b\x5b\xc6\x40\x
f3\x3f\xe3\xc5\xea\x2a\x4d\xae\xaa\x90\x14\x2c\xb5\x3c\x9d\x12\x99\x2d\xcd\x72\x94\x25\xd4\xc
e\x71\x32\x01\xd6\x0d\x67\xe7\x20\xbe\xce\xc9\x81\xdf\x43\x7d\x3c\x24\x0c\x80\x2e\x71\x7e\x85
\x0a\xa3\x41\x95\x8c\xda\x5f\x34\xac\x7d\x67\xc1\xfa\x87\x1f\x90\x6d\xe4\xeb\x46\x7d\x64\x5e\x
37\x10\x54\x2d\x9e\xd4\xce\xce\x26\x94\x6f\xc6\x78\x96\xa3\xde\x87\x8f\x68\x70\x3d\x18\x61\x4f
\x74\x13\x86\x5d\x6c\x36\xd0\x13\xe8\x32\xb3\x59\x9a\xa4\xc9\x80\xf0\xac\x8c\x8e\x8e\xd1\x8a\
x74\x0c\x16\xcb\xc4\x36\x17\x96\x8e\x30\xd2\xd0\x4b\xdd\x78\xa8\x51\xa5\x7f\x96\x61\xa5\xa4\x
e0\x12\xcd\x24\x63\xb0\xa7\x02\x80\x6e\xc6\x26\xe9\x62\x6b\xa6\x1d\x94\x23\x55\x9f\x6e\x09\x
75\xe3\x15\x42\xf8\x41\xe8\x15\x6c\x82\xbd\x97\x75\x48\x54\x67\x00\x9c\x85\xac\x13\x6e\x27\x7
9\x60\xcd\xcd\xe5\x4c\xb8\x55\x6e\x32\xaf\xc9\x7f\x48\xd6\x35\x1d\x10\x39\x5a\x52\x4e\x2d\x51\
x2e\xbc\xb4\x24\x95\x13\xeb\x55\x3a\xe9\xc3\x87\x20\x0c\x85\x6d\x97\x14\x93\x55\x7c\xd7\xa7\
x47\x3a\x38\x48\x2c\x96\x1b\x6f\xc1\x7b\xc9\x56\x9c\x0a\x74\x62\x24\x64\x4b\xdf\xa2\xdd\x52\x
8b\xc5\x68\x58\xbc\x52\xb5\x52\x05\x0b\x02\xad\x82\x86\x7c\x25\x24\xe4\x59\x74\x4b\xb4\x06\x
81\x09\x95\x73\x4d\x9a\x83\x7a\xc9\x68\x5b\xa5\x5a\x81\x90\xdb\x80\x8d\xc8\xea\x6a\x48\x4f\x
22\xfb\x3e\xe6\x29\x7b\x94\x7d\xff\xee\xb2\x6f\x61\xd2\xc6\x93\xf6\xdd\x97\x8f\xee\x41\x3f\x88\x
55\x69\x37\xea\x07\xc2\xf5\x16\xcf\xa8\xba\xba\xcc\x75\xf7\x78\x1c\xa4\xf9\x2e\x2b\x58\xb8\xdd\
x3a\xaf\xc6\x40\xad\x04\xcd\xf2\xbe\x68\x3a\x6f\xe9\xb5\xb8\x04\x3b\xce\xd3\x28\x3e\xbf\x01\xd
7\x16\xdb\x7b\x22\x2d\xf7\x83\x58\xfe\xf4\x53\x30\x9a\xe2\x1b\x74\x49\xfe\xc3\xae\x43\x08\xe4\
x21\x4e\xf1\x9c\x1b\x52\x4f\x35\x2f\x80\x78\x36\x0c\x27\x55\x2c\xce\x2f\x3c\xc0\x88\x48\xeb\x1
e\x6d\xc9\xdc\xc2\x40\xed\x46\x47\x19\x52\x4e\xf6\x83\xb8\x96\x27\x75\xa6\x2a\x02\x1d\x0e\xf9\
xcc\x55\x3e\x35\x8b\x15\x11\xa9\xb7\x0b\x3a\xef\x67\x11\x55\xdf\x50\x88\xcc\x4f\xf7\x99\xa9\x3f
\x66\x10\x77\xa2\x94\xc8\x62\x36\x87\x18\xde\xa3\x93\x84\x79\xf6\xca\xdd\x81\xea\x0c\x7a\xad\
x6e\x76\x8d\xb7\x27\xe4\x18\xe8\x86\x4d\xd2\x65\xa1\xa9\x99\xa7\x34\xce\x2f\xe4\xbc\xa0\xb5\x
3a\x34\xc2\xb0\x8d\xb3\x3c\xca\xa7\x54\xe0\x32\xcd\xbf\x42\x3c\x49\xb2\x28\x97\xb1\x64\x70\x0
5\x7a\x00\x66\x30\x8a\x70\x9c\xeb\x96\x18\x95\x1b\x36\x4c\x2c\x78\xbe\x51\x73\x04\x17\xc5\xc
8\x1c\x3f\xae\x82\x2f\xbd\x4a\x16\xa4\x37\x9c\xc6\x21\xd8\x44\x0e\x70\x9a\x07\x91\x98\x7e\xc7\
xf2\x11\x13\xbb\xd8\x3a\x7a\xf0\x25\x24\xf0\xba\xc5\x5a\x62\x23\x4f\x66\x53\xcb\xda\x22\xc9\xb
6\xc2\x7b\x3d\x4f\x0a\x89\x96\x80\xde\xa4\x0d\x48\xb4\x39\x9a\xe2\x4d\xfa\x1f\x2e\xe6\x6a\x81\
xf8\x9d\xb3\xc2\x26\xbf\x98\x94\x73\xb2\x0b\x44\x03\xc4\x39\x21\x12\xf9\x87\x6b\xe3\x69\x96\xc
3\x56\x87\xc7\x38\xce\x05\xdd\xf4\xaf\x73\x9c\xb5\x9a\x75\x26\x8c\x7f\x57\xd7\x26\x92\x95\xbb\
xf7\xe9\xcb\x8c\xf9\xe3\xd5\x29\xa5\xa2\x69\x1c\xfd\x7b\x8a\x51\x04\x51\xda\x87\x91\xca\x89\x2

b\xcd\x35\x1f\x9d\x0a\x33\x0c\x4d\xda\xb9\x66\x00\xbb\x8e\xb4\x07\xbd\xd0\x89\x40\xe4\x0d\xe6\xa9\x82\xf3\xa4\xbe\xc2\xc7\x97\x83\xfe\xe3\xae\x44\x60\xc8\xaa\x7c\x14\xad\x41\x10\xcc\xfd\xf0\xfb\xcd\x16\x11\x5d\x79\xf2\xde\x9b\x33\xaf\x53\x29\x5d\x22\xd3\xee\x76\x2a\xe5\xdc\x79\x21\x2b\xe1\x13\x22\x5f\xb0\x74\xd5\x1e\x55\x28\x93\x81\x7d\x42\xd8\x34\x11\xf5\x93\x21\x12\xbd\xd9\xda\x42\xdf\xd3\xd8\x4d\xdf\x43\x99\x27\xab\xab\xa8\x97\x8c\xc7\x49\xfc\xcf\xe3\xa7\x4f\x9e\x18\x9d\x2f\x7e\xb1\x06\x38\x4e\xb5\xef\xc9\x30\xa4\xf8\xfb\xba\x87\xa4\x57\x38\x1e\x2c\xf7\x83\x0c\x77\xdb\xda\x87\x71\xd8\xd1\x8b\x5e\x4e\x3e\x87\x43\xed\xe5\x20\x9a\x5c\xe0\x74\x99\x42\xae\xbf\x78\xfa\xe4\xe6\xe9\x13\x3c\xca\x30\x92\x3a\x43\x15\xe6\xb4\x2f\x7c\x18\xbe\x47\x3f\xfc\xc0\x3e\xac\x04\xe3\x50\xf4\x6d\xfb\xdd\xce\xd3\x27\x4f\xe8\x87\xda\x29\xc7\xd9\x43\x2a\xaa\xf0\x4c\x30\xa4\x1f\x28\x62\xf0\x5b\xc6\xe7\x4c\x8c\xb2\x8c\x18\x6b\x88\x46\xc3\x40\xb5\x7e\x9a\x5c\x65\x38\xad\x3f\x7d\xf2\x44\x8c\x58\x92\xe4\x2b\xbd\xf4\x7a\x92\x27\xff\x3c\xa6\x55\x6f\x58\xd6\xf1\x62\x16\xc5\x77\xf4\xc7\xd3\xa7\x4f\x6a\xea\x71\xec\x09\xa2\x1a\x91\xe3\x8b\x24\xcd\x07\xd3\x3c\xa3\x6f\xc8\xb2\xe9\xa1\x2d\xc4\xeb\xbe\x90\x5e\x7f\x1a\x45\x7d\xf2\x69\x65\x14\xf5\xa5\xf7\xa0\x0c\xeb\x41\xa7\xc8\x57\x52\x6a\x45\x7a\xa7\x40\x08\x46\xe7\x09\x80\x20\x3f\x5e\x3c\x15\x58\xbc\x4d\x92\xcf\xd3\x09\xca\x83\xfe\x08\x4b\x98\x1c\xbf\x3a\xfc\x85\x9d\xf9\xc4\xbb\x83\xf7\x3f\x7d\xb2\xbd\x3f\xfe\xf8\xea\xd3\xbb\x83\x5f\x3e\x35\x5c\x1f\x7c\xd7\x87\xa6\xeb\x43\xcb\xda\xb6\xab\x1d\xf9\xa3\xd1\x96\xfc\xd1\x68\x4f\xfe\xc8\xdb\x14\x43\xd3\x4b\xc6\x13\x72\x50\x1c\x99\x43\x64\x9b\x52\xad\x56\x98\x4c\xfb\x44\xea\x27\xb5\x8a\x02\xc0\x62\x65\x2c\x90\x6c\xa9\x10\x41\xe0\x41\x14\xa1\x97\xa8\xd9\xe9\xbe\x40\xd1\xd2\x92\x02\x5e\xc8\x88\xe8\x25\xf2\x9b\xeb\xc6\x37\xf2\x17\x9e\x46\x67\x68\x8b\xc0\x78\x89\xfc\x17\xea\x77\x7a\x95\x5a\x52\xab\x46\xab\xd5\xd1\xaf\xa8\x31\xf3\xfd\xbe\x5e\xbf\x78\xbc\x79\xaa\xf4\xfa\xe7\x60\xf4\x19\xbd\xde\xab\x35\x7f\x5d\xaf\xab\xbd\x9d\xd1\x60\x8a\xea\xbb\x48\x7b\xb9\xd0\x08\x48\x83\x9c\xf5\x93\x99\xfa\x11\x0c\x0d\x48\x9b\xb3\x08\xfd\x8a\x6a\xb3\xa2\x43\xec\x77\x53\xfa\xdd\x92\x7e\xb7\xeb\x5a\x67\x01\x4a\x2d\x9b\xa1\x1f\x7f\xfc\x11\xad\x43\xc9\x6c\x86\x7e\x40\x8d\xd9\x70\x48\x07\xa8\xdb\xd2\xaa\x90\xd5\x71\x3a\x23\x03\x99\xcd\xb4\x4f\x7c\xf1\x9c\x66\xf0\x7d\xf6\xe2\xa9\xb3\x53\xe3\xe9\x28\x8f\x26\xa3\x68\x00\x5a\x02\xb3\x7b\x33\x42\xc6\xe1\xe9\xec\xec\x85\xe5\x5b\x9b\x7e\x6b\x5a\x3f\xae\xd3\x8f\xed\xb3\x92\xd6\xb3\x69\x1f\x81\x7c\xe3\xa1\x71\x34\x43\x83\x64\x34\x1d\xc7\x99\x42\xfd\x32\x4c\x22\x29\xd4\x42\xe8\xd5\x73\x42\x33\x0d\x9f\x8f\x14\x7b\x6c\xf8\x8d\x86\x3e\xb4\x62\x25\xd3\xc1\xaa\xe5\x30\x31\xed\x3a\xfa\x42\x7e\xd3\xf1\x76\x54\xf1\xe5\x2a\x7e\x57\xaa\xe2\x77\x5d\x75\x9a\x72\x9d\xf5\x3a\x2a\xea\x34\x8d\x59\x17\xdc\x80\xd6\xc9\x4b\x46\x2a\x8a\x2f\xe5\xd1\x22\x8f\x95\x47\x6c\xb6\x2e\x8d\x0f\x23\xcf\x36\x7b\xd5\xe0\x2f\x9a\xca\x90\x96\x8e\xa8\xc2\x1f\x19\x8d\x55\x19\x56\x85\x75\x2a\xf5\xe6\x8c\xad\xc2\x56\x95\x8a\x73\x06\x58\x61\xb9\xac\x62\xd9\x28\xc3\x65\x01\xe8\x81\x71\x6a\x72\xc2\xef\x66\x56\x26\xc8\x18\xc0\xd6\x02\x1c\x10\xaa\x34\xd1\xaf\x28\x3c\x25\xff\x9b\xad\xa3\x5f\xd1\xac\x79\x76\xa6\x2f\x24\x28\x1b\xa1\x5f\xb7\xa0\xe0\x2c\x32\x0a\x28\x4c\x12\x7e\xde\xc0\x99\x56\xec\x2b\x1f\x52\x3c\xa0\x9d\x0b\xd1\xd1\x20\x89\xd9\x06\x53\xec\x4a\x47\xbd\xc3\xf7\x64\x8f\x68\xcc\x1a\x0d\x0f\x35\x66\x0d\x1f\xfe\x6d\xc2\xbf\x6d\xf8\x77\xdd\x03\x5a\x20\xff\x36\xe1\xdf\x36\xfc\xbb\x0e\xff\xfa\x7d\xf2\x6f\xab\x5b\x6c\x66\xcf\x9f\x33\xa4\x9e\xa3\xed\xdd\x63\x1a\xba\x1d\x51\x71\x08\x11\x81\x20\x8d\xf2\x8b\xf1\x0a\x2f\xb3\x5a\xa0\x42\x4a\x6f\x31\xf1\x61\x85\x3e\x48\x12\xc6\x0a\x9e\xe5\x34\x7a\x80\xe8\xf2\xa7\x30\x39\xc2\x19\xce\x37\x91\x63\x8b\x64\x83\x70\xfc\x39\x9a\x30\xcb\xdf\x64\x88\xe2\xa3\x04\x4e\x63\x17\x41\x86\xfa\x18\xc7\xe0\x1d\xc0\xee\xb7\x82\x38\x04\x13\xbe\x30\x0a\x51\x9c\xe4\xcc\x0c\xd3\x24\x05\x9a\xf7\x85\x43\xe2\xe6\xa2\x9f\x3e\xe3\xeb\x0f

\x69\x94\xa4\x47\xd4\x02\x78\x6b\xab\x78\x6f\x25\x1d\x6e\x16\xa6\xcd\xa9\xd9\x01\x55\x7c\xe3\x7f\xdc\xe0\x70\xcb\xde\x7c\xf1\xd6\xc2\x9f\x3f\xe3\xeb\x9f\x93\x14\x8c\x18\x3f\xe3\xeb\x95\x2b\xf2\xdb\x5e\xec\x38\xfa\x1d\xb3\x52\x59\x74\xfe\x8a\x30\x20\xb4\x8a\xda\x65\xcb\x48\xf8\x01\xa4\x30\x40\x26\x58\x3e\x72\x1c\xc7\xe2\x99\x37\xb8\x84\xba\x95\x5a\x20\xfd\xcf\x06\x17\x98\x1c\x3f\x10\x11\xa1\x2d\x7d\xc8\x8e\x92\x2b\x02\xbb\xc6\x9b\x59\x22\xbb\xf4\xf3\xd2\x3e\xc8\x70\xed\xc3\xc2\x1b\x95\xc6\x59\x7a\x77\xaa\x2f\xd5\xc2\x44\x94\xa0\x43\x45\x0f\xfa\xf3\x25\xc3\x90\x3d\x5b\xa4\x10\xc4\xc8\x4e\x94\xa7\x83\x64\x2d\x47\xfe\x24\x54\x4e\xa1\xce\x19\x1d\x59\x98\x71\xf6\xc6\xc2\x6a\xdc\x0c\x0b\x49\xfb\x89\x01\x1c\xa2\xe9\xe8\x43\x29\xa3\xfd\x1d\x43\xfc\x1f\x02\x71\x27\xe6\x6c\x16\x8e\x92\x1c\x11\x92\x74\x17\xca\xe5\x3d\x40\xdd\x02\x4a\x21\x1f\x4f\xfb\x55\x20\x83\xf8\xc4\x61\x9e\x49\x7b\x1b\x7c\x28\x76\x2a\x26\xa3\x9d\x49\xbb\x98\x5c\x62\x5d\x29\x00\x98\x32\xc8\xec\xf5\x1c\x6c\xdf\x45\x33\x60\xdb\x65\xd8\xfe\xba\x05\x4c\xfc\x94\x0d\xf2\x6a\x41\x1d\x5f\x50\x83\xa1\x6e\x99\x6c\x54\x4c\x38\x90\x16\x5b\x77\x3f\xa2\x2e\xe1\x67\xda\x84\xa1\xad\x2d\xd4\x9e\x37\x69\xdf\xdc\xd0\xda\xfb\xec\x18\x71\xd7\x9a\x31\x68\x9d\x0d\xc9\x19\xfa\x95\xc8\x12\xe6\x22\x9a\xcb\xcd\x65\x99\xae\x9c\xcd\x44\xf1\xe5\x1b\x0b\xa7\x31\x5e\xbb\x99\x0d\x29\x5a\xf0\x1b\xf1\x54\xb0\x1c\xfe\xca\xc1\x75\x64\x86\xc5\xf8\xe8\xb2\xa8\x63\x23\x5e\x38\x32\xf2\x66\xfe\x51\x42\x34\x4e\x76\x72\xbf\x9c\xa9\x6d\x05\x37\x0f\xf1\x97\xa8\x0d\x8e\x2c\xf4\xa1\x8c\xf6\xd5\xb9\x38\xe5\x10\x98\xa4\xb9\x60\x47\x4a\x80\xa9\x42\xb7\xba\x86\x08\x29\xaa\xc2\xb5\x63\x29\x9d\xa1\x5f\xdd\x8b\xd3\xf1\xa7\x0a\xdf\xf6\x15\xa8\x23\xd0\x3a\x55\x97\xa2\x7d\x0e\x9c\x92\xac\x27\x4d\x0f\x8e\x07\xe9\xf5\x84\x5a\xc6\xca\x72\xde\x3b\x0f\x25\xc3\x61\x86\x73\x63\x66\xe8\x1a\x09\x93\x9e\xa8\x57\x14\xf6\xcc\xbd\xda\x2b\x4e\x88\xc5\x4f\xbf\xf8\xd9\x2c\x7e\xb6\x3c\x60\x31\xf2\x29\x43\xc1\x35\xc4\x8b\xe2\x4a\xb8\xe6\x55\x30\x41\xcd\x38\x04\xd9\xb3\x9d\x5f\x38\x84\x18\x42\xdf\xef\x4e\x29\x18\x22\xbf\xe8\x43\xaa\x7c\x53\xcb\xb6\x4a\xca\xb6\xac\x47\xa2\x2a\x43\xa8\xd2\xaa\xa7\x12\xa8\xfa\xe8\xab\x8f\x4d\xf5\xb1\xe5\x09\x85\x85\xb1\x79\xaf\xae\xa2\x03\x72\xf2\xfd\x26\xc6\xc8\x3e\xe9\xca\x30\x59\x67\xdd\x43\x77\x23\x37\x1b\xd1\xb0\x03\x41\x65\xc9\xda\x32\xb0\xaf\x31\x8b\x15\x0a\x17\x92\x54\x54\x27\x98\x5a\x74\x5c\x0d\x69\xb0\xce\xe0\xf5\xaf\x0a\xb3\x6d\xd8\x34\x40\x99\xaf\x4f\x87\x56\xcb\x98\x1f\xa8\xd5\x54\x6b\x35\xf5\x5a\x56\x6d\x53\xd6\xd2\xa7\x53\xab\xd5\xb2\xa9\xa1\xde\x68\x67\x07\xfb\xd1\x5f\xde\x02\x6d\x27\x86\x23\xcb\x19\x47\xec\xbf\x74\x54\xb7\x90\xff\x82\xfd\x7c\xc9\x67\x88\xbd\x70\xec\xbb\x30\xc7\xd1\x30\x07\x4a\xf7\x1c\x8a\xb2\xd2\x89\xe3\xa8\xe7\x64\xf2\x24\x75\x4d\x43\x48\x5e\xbf\x4a\x8a\xae\x5a\xe6\x1b\x72\xd7\xaf\x92\x52\xab\x96\x35\x75\xa9\xeb\x57\x49\x7f\x95\xb5\xa4\xd7\xc6\x36\xbc\xb4\x64\xdb\x00\x00\x39\x5f\x45\xce\x77\x20\xd7\x9c\x83\x5c\xab\x14\xb9\xc6\x2d\x91\x6b\xaa\xc8\x35\x1d\xc8\xb5\xe6\x20\xd7\x28\x45\xce\xbf\x25\x72\x2d\x15\xb9\x96\x03\xb9\xc6\x1c\xe4\xfc\x52\xe4\x9a\x73\x91\xb3\x92\xee\xc7\x09\xd8\x10\x65\x79\x90\x63\xb3\x00\xb0\x93\xbc\x61\xe9\x18\xb0\x8c\x5c\xd7\xa3\xc1\x17\x32\x17\x79\xd3\xf6\x85\x0c\x44\xae\x6b\xc7\xad\x4a\x14\xeb\x7a\x9a\xc3\xfb\x60\xf9\xd4\xe8\xc9\x43\x5a\x3b\xfa\xa9\xc5\xb2\x7c\xf4\x63\x8b\xb9\x82\x94\x73\x4b\xb1\x84\xea\xd5\x28\x41\xac\x1f\x8e\x9d\xef\xc6\xce\x5c\x3f\x06\x76\xc6\x12\x52\xb1\x6b\xdc\x06\xbb\xa6\x84\x5d\xd3\x8d\x9d\xb9\x80\x0c\xec\x8c\x35\xa4\x62\xe7\xdf\x06\xbb\x96\x84\x5d\xcb\x8d\x9d\xb9\x82\x0c\xec\x8c\x45\xa4\x62\xd7\x9c\x8f\x9d\x49\xad\x98\x07\xb6\xb6\xcb\x25\x74\x1b\xb6\xac\x23\x5d\xc8\x31\x96\x93\xba\xb9\x5a\x56\x95\x21\xfa\xb4\x5c\xb2\x0f\x3b\x0a\x6f\xa2\x66\xa7\xbb\xda\x6a\x32\x0d\x74\xdd\xa6\x0a\xe6\x12\x8b\x10\x90\x32\xe6\x38\xcc\x54\xc3\xcf\x32\x96\x1a\x0a\x41\xb6\xef\x61\x30\xc0\x42\x47\x2c\x80\xfc\x37\x9e\x05\xe3\x89\x38\x29\x17\x1f\xf8\x9c\x52\x58\x39\x9e\xe5\xd2\xed\xf6\xca\xf6\xee\xf1\

x0a\x3b\x47\xd4\xc6\xdc\x22\xfd\x33\xbe\xf6\xd0\x60\x78\x2e\xa4\xf9\x02\xca\x64\x14\x10\x24\x6
6\x39\xd2\xa1\x30\x09\xbf\x56\xb4\x63\x03\xc4\x74\xda\x3d\x8b\x12\xfb\x13\x8d\x9a\xba\x8f\x47\
x13\x9c\xd6\xb6\x77\xe9\xb5\x3e\xd5\xd9\x3f\x7d\xc2\x6c\x56\xe4\x26\x5f\x3c\x7d\x0a\x11\x70\x
c1\x80\x44\xb1\x2a\xd8\xec\x34\x3d\x6e\x97\xb0\xd9\x01\xdb\x11\xc9\x32\x61\xb3\xd3\xf6\x0a\x9
3\x84\xcd\x0e\xb8\x30\x8e\xc3\xce\xf7\x9b\x5d\xff\xe6\xcc\xeb\x34\xef\x64\x2d\xf2\x35\xcd\x44\x
1e\xcc\x98\xe3\x2b\x9a\x65\xd0\x95\xf0\x1c\x31\x03\x0a\xd2\x3c\x1a\x24\xe3\x49\x12\x43\xc8\x7
5\xf2\x6d\xf5\xe9\x13\x31\xef\xa3\xa8\xbf\xc2\x8a\x7e\xf9\x22\x1b\x00\x08\xa7\xcf\x7b\x36\xee\x0
8\x32\x5c\x58\x75\x04\x19\x96\xbe\xfd\x9c\xa4\x21\xb8\xa5\x8b\x02\xe2\x8d\x0c\x61\x3a\x04\x7b
\x31\xa0\xf5\x6d\x7e\xcb\x53\xc0\xb4\x7e\x56\x30\xc3\xe0\x59\xd5\x23\x0b\x55\x7a\xff\x31\x1f\xa
e\x03\x14\x1c\x0f\x56\xc8\x83\x86\x75\xb7\x2d\xbe\xd2\xc7\x32\x43\x14\xf1\x65\xf7\x72\xf2\x66\x
67\xaf\xb8\x6c\xa2\xcf\xd6\x1b\xac\x7e\x46\xcd\xf3\xc8\xb2\xe2\xb7\x58\x39\x1e\x4f\x46\x41\x6e\
x63\x50\x22\xc8\xf4\x1f\x31\x0b\xc8\xc3\x35\xa8\xe0\x54\x20\x78\x1d\xe8\xfd\xa2\xdf\xf1\x0a\x0f\
x30\xb9\x89\xda\xa8\xe6\x37\xd7\x51\x3f\xca\xb3\x7a\x19\xc0\xe8\xd2\x02\xef\xe0\xa7\xdb\x82\xf
b\xb4\xfb\xbe\xf7\xe9\x97\xbd\xc3\xa3\x77\x9f\xde\x1d\xee\xec\xa2\x6d\x08\x6d\x90\x07\x71\x8e\
x52\x3c\x49\x71\x86\xe3\x3c\x8a\xcf\xb9\x22\x86\x90\xe1\x38\x09\x8b\xbe\x5b\x61\xee\xec\x56\x
82\xc9\xd8\xa9\x01\x53\xba\x14\xd4\x4c\x8e\xc4\xa3\x9d\xa2\x2c\x97\x84\xc5\x6c\x52\x74\x7b\x
e0\xf6\x3d\x4d\xc1\xe0\x41\xe4\xf8\x90\x8b\x28\xc5\xa5\xde\x09\xba\x27\x73\x80\x4e\x2e\x30\x1
9\xf5\x3c\x41\x53\xe6\x26\x40\x58\x00\x22\x85\x01\xb4\x02\x72\xb5\x78\x18\x0c\xcf\x37\x81\x74\
x39\xae\x75\x79\x47\x35\xb0\x85\xed\x22\xa3\xb0\x19\xf9\x45\xb1\x6b\x32\x6c\xe8\x53\x7b\x4c\x
09\x77\x42\x7a\x04\xf9\xcf\xf8\x7a\xc5\x5a\x96\x7b\x86\x0e\x86\xe7\xa8\x76\x08\xad\x04\xa3\x3
a\xd4\x19\xd8\x06\xaf\xe2\x18\xa8\x6d\xf1\x38\xa2\x74\x42\x6f\x08\x89\xf0\xde\x11\x42\x19\x94\
xf5\x89\x9c\x2b\xa2\x81\xfb\xbb\x2a\x25\x98\x05\x90\x22\x2d\xc8\x7b\x3c\xbf\x7a\x5e\xa1\xdb\xf
4\x2e\x1d\xe6\x24\xad\xb1\xcb\x33\x18\x42\x0f\xfd\x81\xa2\xcb\x4d\x14\x5d\x16\xbc\xf1\x46\x31\
x3d\x50\xe6\x5b\x85\xb4\xa9\x84\x85\x62\x92\x83\xae\x01\x90\x13\x87\xd0\xfa\xec\xc6\x59\x5d\x
ab\x16\xd9\x43\x97\xd0\x2a\xd2\x93\x63\x21\x3e\xd2\xd3\xfd\xd2\xd3\x0e\xbe\x2f\x7a\x12\x90\xe
e\x46\x4f\x2a\x9f\xbe\x05\x3d\x1d\xc4\x51\x1e\x05\xa3\xe8\x77\x9c\xa1\x00\xc5\xf8\x6a\x74\xcd\
x30\x0c\xd9\x70\xcc\xa7\x25\xbe\x6b\xcc\x86\x49\x3a\x7e\x97\x84\x18\xed\x52\x5f\x35\x08\xd3\x
5c\x70\xba\x24\x95\xe9\x14\xac\xab\xc1\xcd\x8f\x53\xad\xd8\x64\xec\x64\xf8\xcd\x91\xec\xbd\x9
1\x55\xcd\xfc\x60\xe3\x14\xb7\x24\xb8\x28\x8e\x14\x0b\x1b\x31\x4d\x12\xb9\x58\x54\xd4\xdb\x9
3\x09\xa1\x05\x18\x2d\x9e\x98\x3a\xb3\x5c\x33\x90\x21\xde\x12\x3f\xf9\xa6\x48\x69\xd0\x3c\x15
\xe7\x44\x72\xa6\x86\xf5\x49\x3a\xa6\xd3\x1e\xd8\x74\x37\x94\xbe\x0b\x92\xda\x2a\xc8\xeb\x85\
xad\x24\xb5\xa3\x01\x5b\x19\xeb\x59\x3c\xa2\x84\x4e\x3d\x00\x6c\xfd\x00\xfb\xa2\x5a\xe5\x85\x
03\x36\x3a\x2a\x1f\x86\x58\x0e\x99\x68\x09\xb4\x67\x77\x24\x1f\xb6\x04\x4d\xdc\x94\x19\x4e\xa
b\x18\x51\x51\xa3\xa2\x30\xc8\x03\xd4\x07\xd9\x4b\x2d\xe1\x90\xc7\x00\x34\xcd\x74\xc1\xbd\x9
d\x75\xc0\x1f\x70\x0a\x73\x39\x48\xe2\x41\x8a\x73\xbc\xcc\x86\x63\x94\x9c\x2b\x4c\x59\xba\x97
\x3a\x5a\x6c\xac\x21\x9e\x06\x60\x4e\xdd\x5b\x18\x4f\xc1\xa1\xc4\x52\x70\xb8\xc0\xa6\xf7\x25\x
63\xae\x30\x04\x28\x53\x76\x12\xde\xc0\xdb\x60\x0d\x48\xe0\x2b\xec\x5c\x12\x7f\x12\xb0\x68\xd
0\x2c\x16\x8c\x20\x8a\xcf\xef\x81\x9b\x14\x9d\xdf\xe2\xe4\xc1\xe0\xd7\x9e\x91\x36\x9f\xa9\x64\x
52\xa5\xde\x15\xc7\xdc\x49\x61\xac\xe4\xa6\x16\xe5\x95\x0e\x9d\x83\x7b\xe0\x38\xb4\xcd\x7e\x
00\x5f\xe4\xea\x36\x9a\xa2\xed\xa1\xe0\x32\x88\x46\x41\x7f\x84\xa9\x19\x62\xe6\xde\x16\x3f\xf1
\xce\x54\xa6\xaa\xbd\x28\x66\x1b\x5f\xe9\x3e\xc5\xe0\xaa\xfb\xcc\xfb\x24\x67\xde\xd1\x34\x68\x
1a\x85\x54\xec\x1a\x28\xca\x10\x1e\x0e\xf1\x20\x8f\x2e\xf1\xe8\x1a\x05\x28\xc4\x59\x9e\x4e\xe1

\xd9\x43\x29\x0e\xc2\xe5\x24\x1e\xe0\x4a\xfb\x4c\x55\xea\x05\x34\x1e\x8a\x86\x29\xf0\x87\xa6\x64\x3e\x92\xb5\xea\x44\x2c\xaa\x2c\x4a\xfd\xa2\xe2\x7c\xf2\xe7\x45\xab\xd3\xff\x5e\x31\x17\x53\x28\xa4\x96\x88\x86\xa5\x00\x50\xe9\x6a\x51\x8a\x5a\x2e\x4a\x16\x60\xc8\x10\x0f\x89\xa0\xca\x16\x1c\x0e\x59\xbc\x4c\xce\xa9\xf7\xa4\x09\xb1\x2e\x3e\xb3\xf6\x5c\x65\xb3\xdf\x5c\x5f\x6d\x35\xe5\x4f\x54\x25\x62\xfb\xa2\xc9\x41\x9b\xc8\x57\xbe\xaa\xf2\xef\x26\x6a\x56\x39\x3b\x65\x56\x55\x76\x30\x5f\x91\x8d\x9c\x6b\x93\x9f\x5a\xd8\x48\x9f\x5c\x60\x49\x28\x60\x89\xb6\x02\x74\x01\x5a\x63\x22\x64\x56\x58\x8a\x5c\x84\xdd\x8e\x39\x3e\x10\x60\x80\x2f\x6b\x22\x34\xb1\x75\x6d\xe9\xd0\x57\x38\x2c\x31\x6b\x6f\x53\xe5\xa9\xe9\xc8\x0d\xd9\xd6\xb9\xca\x94\x7a\x9b\x4e\xbf\x29\xf2\x27\x3e\x65\x78\x84\x07\x39\x6d\xf8\x38\x4f\x83\x1c\x9f\x5f\xd7\x5c\xe6\xda\x92\xf6\x19\xc4\xc5\x2d\xf4\x8c\xb2\xd2\x67\x4e\xf3\x30\x36\x1b\x1f\x82\x2c\x23\x6c\xe2\x55\x90\xe1\x50\xf1\x98\x93\xff\xca\x8d\xc3\x18\xa8\x63\x9c\xc2\x81\x8b\xec\x6a\x6e\x48\xe5\x8b\x5c\xcf\xed\xc7\xee\x33\x4a\x6c\xd4\x5d\x48\x31\x72\x92\x19\x9b\x79\xc3\x52\x66\x37\x5a\x04\x01\xb3\xcf\x83\xb8\xb8\xa1\x28\x7a\xc8\x7d\x81\xa3\x8f\x81\xe7\xb0\xf4\x64\x64\xbf\x69\xf4\x5f\xbb\xcf\xb9\x13\xda\xea\x4d\x91\x87\x4a\x6f\x8c\x74\xcc\x2d\x13\xaa\xb3\x6d\x99\x4b\xd6\xea\x4c\xc3\x6b\xbf\x7a\x53\x75\xd8\x59\x9e\xe2\x60\x7c\x2b\x55\x36\xc8\x50\x4c\xf9\x2c\xdb\xe0\xb7\x9a\xcb\xfd\x88\x1a\x6c\xab\x27\x1a\x2a\x9d\x40\x18\x6b\x49\x33\xed\xa3\x5a\xab\xa9\x2a\xa6\x25\x85\xef\x31\xe0\xa7\xa9\x7d\xf5\x97\x25\x1e\x21\x7b\x96\xbd\xd6\xb6\xc3\x72\x11\x71\x12\xa4\x70\xdc\xb2\x09\x88\xe6\xf6\x06\xc7\x9b\xc2\xba\x8a\x0b\x8d\xdf\x7d\xf7\x6c\x38\x9a\x66\x17\xcf\xaa\x6d\x73\x14\x8a\x6b\xa3\x13\xc3\xbc\x89\xfc\xb2\x79\x85\x73\x2d\x64\x35\x9d\xc8\xb7\xa5\xb2\xf2\xfc\xd3\x98\x9e\x7d\x7b\x2b\xec\xc7\x1f\x37\xf3\x29\x44\xf1\xd8\x81\x7a\x06\x95\x48\x6d\x48\xb7\x9b\xec\xa0\x6d\x38\x07\xb3\xf7\xb2\xd2\xbb\x4c\x41\x2f\xab\x28\xc7\x3c\x39\x57\x21\x5f\x2f\xbc\x9b\x6e\xab\x3d\xb2\x2a\x04\xf5\xcc\x32\x85\x82\x1f\xa8\xfa\x2b\xec\x87\x7c\xa6\xf8\x76\x07\x7a\xd8\xde\xab\x9e\xa1\x8a\xe6\x1c\x25\xba\xa4\x5e\x3b\xb7\xd1\x3c\x17\x30\x4a\x75\x85\xa2\x2e\x57\x34\x49\xf5\x6e\xa5\x71\x16\xd3\x59\x1c\x90\xfe\x33\xa7\xb3\xd0\x04\x2f\x38\x9d\x56\xc5\x6f\xc5\xe9\x14\x75\xef\x30\x9d\x65\x0a\xdf\x6a\x57\x07\x5f\x75\x3a\xef\x3c\x5d\x25\x4b\x60\xce\x7c\xe9\x7a\xd3\x92\x49\xa2\x9b\x89\xd0\xf3\x0e\x6c\x62\x1d\xb3\xba\xbe\x44\x5b\x28\xba\x94\x67\xab\x6c\x8b\x60\x3b\x26\x8d\x2b\xdd\xbb\x08\xa2\x18\x52\x9e\xb8\xee\x5a\x5f\x81\xdd\xc0\x27\xde\x79\xb4\xe5\x0e\x3e\xa0\xab\xd8\x94\x1d\x84\xd4\x35\x88\x41\x1a\x9a\xa2\x31\x6d\x97\x10\x77\xa2\x2f\xca\x38\xca\xab\x1e\xdf\x0e\xb4\x93\x90\xd4\x84\x32\x77\xa4\x57\xaf\x7a\x96\xbd\xc7\x04\x4f\x9b\xf8\x20\xc2\x7f\xe6\x5c\x8d\x41\xa9\x34\xc8\x99\x51\xf7\x8a\x5e\xc7\x80\xa1\xd1\x2c\x95\x8e\x84\x56\x84\x09\x4b\x09\x97\x91\x90\xca\x09\x91\xf5\x86\x84\xd9\x65\x11\x20\xec\xe7\xd5\x05\x66\x91\xf7\x29\x7e\x10\xc8\x33\xab\x80\x9c\xb9\x30\xec\x05\xc9\x1f\x4c\x25\x13\x75\xa8\x37\x00\xa4\xc7\x83\x2e\x08\xd7\x06\x5d\x96\x95\x27\x03\x15\x2a\x40\xc3\x4c\x5e\x85\xe2\xb4\x85\xb6\x3a\xc0\x22\xfd\x86\x44\x5e\x48\x0e\xc3\xd9\x42\x88\x15\x9a\x1c\xf1\xca\x61\xce\xfa\xcb\xe1\x11\x9c\x97\x19\xd1\x99\x65\x66\x49\x0a\xfd\x2a\x14\xdd\x1e\x52\xfa\xe5\x15\xcd\xda\x84\x7e\x86\x87\xec\xeb\x52\xd3\x47\xd7\x8a\xd9\x11\x1e\x63\x90\xc2\x61\x77\xa5\x24\xc0\xae\xa2\xe0\xb4\x0f\x0e\xed\xf0\xda\xac\xce\x25\x58\x7c\xc9\xc3\xce\x53\x66\x4a\xf3\xc9\x73\xbc\x85\x29\xa0\xb3\x03\xb2\xe7\xce\xdc\x75\x1b\xe2\x0a\xeb\x56\xec\x53\x8f\xeb\xf6\x71\xdd\xa2\xdb\xaf\xdb\xbb\xac\x0e\xb0\x10\xbe\x88\xb2\x85\xd7\x86\x15\x13\x46\xd1\xc0\x45\x7e\x39\x3c\x72\x72\x00\xd9\x83\xcc\xe0\x00\x77\x65\x3b\x56\xcc\x4e\x8a\xa1\xe9\xe3\x41\x32\x66\x4b\x87\xb0\x85\x28\x99\x66\xd5\x99\x87\x18\xac\xaa\xec\x41\x90\x12\xef\x46\xcd\x89\xfb\x42\x1e\x50\x20\x22\x71\x69\xc9\xe6\xe1\x7f\x91\x24\x19\x46\xe3\x68\x46\x64\x21\x4b\xff\xc0\x13\xd4\x14\xd2\x90\x4

c\x88\x4c\x0a\x73\x91\x5d\x72\x09\xd2\x29\x39\xe9\x64\xd3\x7e\x86\xff\x3d\xc5\x71\x6e\x55\x31\x20\x55\xb4\x93\xb2\x7a\xa8\xa3\xe8\x54\x0d\xca\x28\x69\xb3\x32\x5f\xd5\x4f\x76\x36\x1b\x56\xb6\x18\x49\xc5\x6a\xb3\x46\x4a\x22\x7f\x30\x81\x85\xf5\x78\x74\x86\x7e\xdd\xa2\xf5\x4e\xa3\xd2\xd0\x25\xc5\x6f\x6e\x02\xfd\xaa\xc7\xca\x2b\x01\x4d\x24\xd1\xf6\x43\x10\x86\x64\x02\xe7\x28\x40\x26\x90\xe5\xaa\xb7\x42\xff\x6b\x57\x7f\x7c\x78\xd3\x3b\x46\xff\xa7\xb3\xba\x86\x26\x0c\x68\xc6\x74\x79\x36\x98\x1f\x3e\x0f\xb2\x35\x90\x93\x27\x41\xb8\xc2\x9f\x4a\x64\xe3\x0f\x01\xbf\x7e\x9e\x66\x3c\x74\xbe\x08\x84\xc2\xcc\x95\x21\x6e\xb2\xc0\x63\x21\xfb\x2b\x80\x2c\xdf\x3e\x13\xb4\xac\x95\xec\x7a\x3c\x16\x02\x4a\xba\x8f\x04\x40\x99\x08\x66\x49\x06\x05\xc2\x59\x3e\xf0\xb1\x59\x1c\xbe\xc4\xb8\x92\x5f\xc5\xf5\x9a\xa7\xc5\xcd\x52\x2e\x98\x83\x50\xbf\x5c\xbb\x35\x03\x11\xd5\x68\xac\x93\x2d\x69\xbc\x5c\x31\x43\xa6\x71\x2e\x68\x07\xfc\x8a\x4c\xa8\x31\x23\x58\x03\x28\x7d\xb1\x4c\x53\x4e\x8b\x08\x2b\xff\xd0\x0a\xd8\x9a\xa5\xf7\x42\xbc\x5d\x33\xf4\x02\xcd\xf4\x06\x5f\x09\xbd\x40\x04\x14\x05\x8b\xc2\xd7\xc5\x78\xcf\x1c\x5c\x8c\xf7\xe0\xd6\xa2\xbc\x9d\x8b\x59\x29\x52\x59\x79\xf8\x82\x82\xfd\xa8\x6d\xa2\x08\x2d\xb9\xdc\xf2\x65\xe8\x34\xcc\xbd\xf4\xa6\x44\x7a\xd5\xb0\x43\x5b\x85\xed\x3b\x3f\xfc\xcb\xa0\x3d\x15\x25\x9b\x19\xc2\x76\x18\xda\x07\x01\xe6\x7a\x90\xc4\x83\x20\xe7\x30\x2b\x6b\x60\x3e\xc6\x13\xc1\x50\x60\xc9\x5e\x04\x21\x0d\x64\xc4\x16\xea\xd7\xe1\x32\xd3\x58\xe7\x33\x5f\x85\x23\x40\xb3\x15\xae\xdc\xa1\x9c\xce\x12\x6c\x7c\xe0\x35\xce\x95\xc4\xc5\xd2\x22\x86\x18\xb0\x68\x14\x64\x39\x3c\xcf\x5f\xd3\x85\x78\x7d\x5a\x53\x97\xf3\x32\xf2\xeb\xd4\xc5\xec\x8c\x39\x83\xd9\x3c\x89\xa9\xe0\xe0\xa6\x98\x02\xdc\x96\xbe\x06\xa5\xcd\x94\x6e\x9b\x0b\xea\xf9\xff\x8c\x8b\x20\x9b\x8b\x82\xfd\x66\xc1\x76\xab\x50\x76\x0f\x74\x7f\x46\xff\xef\x92\x10\xdf\x50\xf5\xe0\x89\x38\xad\xd1\x4b\x11\x38\x49\x48\xdd\xe9\xbd\xea\xb9\xa0\xb0\xb9\xba\x11\xf4\x45\x60\xe9\xc2\x86\x09\x11\x48\xde\x41\xe0\xe0\x47\xc0\x06\x40\x32\x9c\xd4\x08\x9c\x60\x0a\x98\x79\xda\xa9\x8e\xb6\x6d\x34\x71\xa3\x78\x23\x2c\x60\x18\x48\x27\x5a\xfd\xd8\x93\xac\x0f\xcb\x6d\x00\x4b\x02\x9c\xa9\xf6\xa1\x16\x3f\x4e\x90\x9b\xc9\x08\x28\x6a\x51\xa4\x2a\x76\xc9\xf7\x31\xd8\x7e\x3a\xf0\x2f\x26\xd6\x3c\x0c\x18\xb6\xa4\x5c\xd2\x56\x8d\x4b\x9c\x27\x06\x02\x15\xb6\x44\xd0\x68\xc0\xa9\x5c\xbb\x9b\xb1\x4b\xfb\xab\xcf\xcb\x9b\x57\xad\x57\xea\xe8\xf9\xea\xc2\x18\x08\x55\x8b\xe3\x2c\xf3\x06\xe3\x09\x0a\x72\x34\xc2\x84\x0b\x26\x31\x5f\x01\x2c\xcb\x07\xb5\x04\x85\xfd\x1a\x18\xae\xc9\xb7\x90\x38\xdf\x8c\xa3\x98\x1a\x89\xb2\x43\xbc\x11\x2e\x51\x7d\x64\x95\xe8\xf4\x49\xf8\x53\x42\x1a\x83\xfd\x31\x3d\xf2\x46\x97\xe8\x87\x1f\xac\xfa\x78\x3d\x50\xc7\x87\x5b\xe9\x32\x0a\x4c\x54\x65\x8a\xf3\x7c\xae\x37\x5b\xf5\x4a\xda\x2d\x92\x16\x22\x89\x30\x94\x66\xaf\x2c\x04\xcd\x9b\xbb\x5f\x42\x5e\x5d\x25\x07\x19\x9a\xee\xcb\x25\x72\x81\xbc\xce\x4c\xbf\x40\x02\x87\xdf\x73\x75\x10\xfc\x2a\x9e\xda\x08\xba\x4e\xc9\xb7\xba\x8c\x7f\xb8\x65\xf5\xb0\x78\x5b\xdb\x03\xc9\x6f\xce\x0c\x50\xf9\xc8\xd6\xde\x3c\xcb\xbf\x3b\x5a\x2a\x80\xe9\x1d\x93\x3d\xec\x66\x28\x68\x90\x8c\x46\x98\xd2\x7f\x32\xe4\xa2\x01\x88\x9a\x18\x72\xe9\x95\x89\x1e\x92\x28\x2a\x39\x79\x93\x6d\x34\x0d\xae\xa4\x57\x56\xbf\x44\xbb\xeb\x07\x75\x40\x17\x42\x4a\x95\xda\xc5\xc5\x23\x64\x78\x60\x5c\x90\xd6\x27\xeb\xd3\x30\xc7\x75\x01\xca\x82\x11\xc5\x1e\x7e\x00\x30\x50\x49\x06\x34\xfc\x28\x4e\xa3\x4b\x2a\xab\x70\x8e\x61\x05\xc8\xaf\x52\x0b\x39\x5f\xb2\x1c\x34\x63\xad\x56\x93\x6b\x6e\xd3\xb3\x72\xf9\x66\x70\x81\xc7\xb7\x83\x6b\x17\x38\x99\xca\x1c\x2c\xa6\x87\x12\x3c\x2b\x08\x9a\x93\xf1\xa6\xc8\xd9\x48\x4f\x31\x54\xc4\xe2\x6f\x75\x31\x6c\x90\xc4\x97\x38\xcd\x15\x19\x96\x66\xbb\xe3\xc6\x94\x60\xf1\x49\xad\xff\xdc\x6e\xab\x1f\x68\x15\xd5\x79\x55\xbc\xac\x68\x0f\x33\xdf\xc5\x4a\x45\x6d\xfe\xb1\x4e\x78\x37\xc9\xf8\x68\x76\xa2\x41\x2c\x92\x58\x4d\x92\x2c\x8b\xfa\x23\xec\x5e\xb1\x96\xa6\x16\x73\x6e\x2a\x06\xca\xb4\x07\xa5\xdf\xf8\x09\xfc\x0f\x03\x0a\x12\xea\x

73\xb2\x82\x37\xa5\xdf\x85\xc3\x93\xb5\xd2\x67\x7c\xbd\xa9\xfa\x45\x59\x8b\x69\x9e\x52\xf6\x4
2\x64\x19\x6f\xc2\xbf\x73\x0a\x8a\x55\xb9\x69\xba\x73\xd9\x6b\x30\x11\x5e\xb7\x4c\xb0\x17\x16
\x72\xbd\x7a\x74\x7e\xd3\x3b\x5e\xb3\x57\x90\x58\x78\xdb\x5e\x42\x2c\x1c\x09\x28\x7d\xb7\x72
\x38\xc1\xf1\xf1\xf1\x5b\xa3\x5a\x75\x67\x32\x79\xfa\xed\x82\xd7\x38\x9a\x1d\xc4\x6a\xb9\xca\xa
6\x47\x74\x15\x67\x8b\x2d\x63\xe4\x5c\x37\x26\x2b\xd1\x7c\x03\x1d\xdc\x84\x1c\xea\xdc\xc0\xb
9\x81\x2d\xf7\xca\x80\x5d\x01\x7e\x47\xc3\x48\x5f\xe3\x25\x70\x20\x09\x58\x46\x33\x80\x41\xf6\
x38\x5c\x78\x51\x16\x18\xc7\x09\x7d\xa3\x31\x40\x96\xb3\x1f\x97\x71\x8f\xaa\x4b\x9a\x22\x2f\xa
e\xe9\xd8\xda\x5e\x42\xcf\x9e\xd9\x7d\x2b\xac\xe5\x57\xf2\x84\xe6\x1b\x72\xb9\x72\xcc\xa9\xe5\
x20\x55\x27\x61\xf2\x8a\x32\x71\x8a\xb1\x71\x59\x55\x15\x25\xd0\x97\x2f\x94\x5c\x8b\x3a\x2b\x
7c\x12\xaf\xf9\xb1\xd7\xd0\xd1\x58\xe5\x24\x4a\x65\xf3\xee\x35\x68\x3b\x70\xb5\x21\x7e\xda\x6f
\x37\x58\xcf\x6d\xc4\x69\x03\xcd\x8a\x8b\x54\xc6\xb0\x7b\xa9\x83\x58\x7e\xdd\x21\x56\x5d\xe0\
x5e\x72\x31\x6f\x66\x79\x90\x8c\x27\x41\x0e\xdb\x4b\xd5\x65\x28\x6f\x0b\xda\x26\x26\x89\x3f\x
55\xf7\x44\xdb\xf2\xbb\x0d\x72\xf7\x65\x38\x98\xd0\xb6\x8f\x39\x79\x3b\x08\x59\xa2\x2e\x17\x6f\
x54\xe8\x5b\x14\xaf\xcc\x7d\xe7\xa8\x65\xe4\x48\x4b\xca\x12\x2c\xbe\xd8\x02\x35\x12\x71\x57\x
ab\x40\xde\xd9\x8e\xb1\xd0\x5f\xf3\x10\x4b\x8a\x3b\x55\x2d\x57\x52\xb4\x1a\x43\x7b\x7f\xda\x9
8\x75\x5a\x5d\xbf\x3b\x58\x83\xc4\x06\xdd\x4e\xb7\xdd\x19\x76\x86\x67\x75\xae\x8a\x07\xd0\xfc
\xa1\xe8\x87\xe3\x1c\x59\x01\x05\xe7\x58\x38\x0e\x5f\xa2\x6e\xc1\xc8\x68\x58\x9b\xc5\xf7\xbc\x
b2\x35\x26\xfb\x2b\x2d\x2a\x3c\xf2\x75\x52\xd0\xe9\xad\x97\x8c\x1a\xb3\x81\x2f\xe8\x5b\xac\xe1
\xfb\x0d\xe0\x60\x0a\xa3\xda\xd2\x9b\x04\x69\x86\x6b\xca\x42\x2d\xb9\x98\x4c\x33\x45\xf1\x53\x
54\xb3\x7a\x25\x90\xe2\x88\xc6\xf0\x9a\xb3\xe8\x28\x61\x18\xc8\x94\xa9\x57\xcb\x20\xf2\xcb\x3
8\xe9\x30\xcc\x92\x42\x18\xe0\x4e\x70\x96\x53\xdb\x86\x60\x64\x59\xa0\x1a\xcc\xd3\xc6\x19\xd
a\xda\x42\xc5\xda\x43\x3f\xfc\xa0\xb7\x7b\xea\xb3\x32\x7c\x4d\xba\x54\x50\xbb\x33\x7a\x81\x61\
xb6\x8c\x54\x0e\x63\x2c\x7e\xad\x45\x66\xca\xd3\xf4\x50\xbb\x5e\x62\x5d\x97\x5c\xb2\x23\x3a\x
5c\x05\x15\x30\xcc\xf2\x06\xfc\x09\x34\xd0\xd0\x6f\xad\x8d\xe2\xca\xad\x8e\xdf\xad\xc6\x28\xac\
x47\x23\xc7\x31\xc8\x93\x4e\x27\xaa\x68\x5e\x7a\x57\xc4\x17\xe1\x55\x1a\x4c\x26\x20\x47\x06\
x39\x6b\x5e\x56\x99\xa0\x80\xec\xf4\x99\xe4\x95\x56\xba\x7a\x15\x57\x1f\xc3\x95\xad\x70\xf8\xb
1\x7d\xaa\xea\x40\x72\xeb\xcb\x1e\x21\xf4\x70\x19\xbf\x4c\xaa\xe7\x3a\x02\xb9\xb7\xac\xb3\xd4\
x21\x34\x0e\x29\xd5\x88\x03\x46\x71\xb1\x63\x39\x38\x95\x85\x88\xd2\xbd\x17\x01\xa1\x4d\x43\
x54\x93\x26\xb6\x34\xa8\x14\xbb\x76\x20\xf3\xc6\xbc\xe9\xee\xe2\xa1\x5a\x28\x9f\x2c\x47\x9d\x
12\xef\x73\xd6\x34\xb5\x41\x61\xbf\x0b\xbf\xf3\xbf\x48\x0c\x17\xfb\x16\xb6\xfd\xe7\x6e\x60\x64\x
59\xda\x35\x2a\xe6\xb2\x12\xfe\x95\xa6\x36\x42\x71\xb5\x74\x9c\xc2\x1e\xae\xc1\x22\x48\x8d\xa
e\x4e\xf8\xaa\x8d\x7b\x62\xb5\x39\xa4\x81\x12\x65\x87\xc5\x39\xd6\xed\xc5\x7a\xbb\x10\x3a\x0
b\x45\xcf\xd9\xb5\xd9\xaf\x4b\xd1\x0d\x92\xc2\xf9\xc4\x16\x00\xcd\xea\xb3\x6a\x88\x25\x85\x67\
x86\x08\x90\xc0\x3a\x7b\x1b\xc9\xa4\x07\xfd\x2b\x60\xc2\x15\xb0\x01\x85\xd9\x1b\x11\x8e\x2b\x
1c\x73\x5d\xfb\x51\xf5\xed\xb4\x6c\xd3\x56\xf6\x57\xb3\x20\x57\x2d\x5a\x3e\x11\xb2\x12\x7d\x5
b\x89\x2e\x2d\x45\x24\x1d\x21\xa3\x17\xb3\x0c\xd5\x0a\x16\x80\xe0\x42\xd4\x2c\x26\xf4\x81\x4
5\x49\xf6\xca\x52\x58\xd2\x05\xea\x16\xd6\x96\xd2\x92\x5e\x90\x90\xde\xd0\x72\x5c\xbb\xa9\x7c
\x6c\x61\xf7\xd0\x99\x98\x38\xa1\xf8\x92\xaf\x65\xd0\x83\x6d\x4f\x32\x01\x88\x1d\x4a\xbb\x68\x
92\x1e\x21\xb5\xf7\x5f\x71\x9f\xd2\x02\xb4\x88\x48\xc7\x5f\x61\x6f\x2a\xa2\x2a\xcf\x67\xd3\xdc\x
7b\xde\xc2\xa6\x39\xd9\xb1\x30\x0a\x92\x47\xfd\xad\x59\xf6\x7d\xa3\xa8\xef\x4b\xf7\xb8\xa5\x38
\x63\x17\x38\x22\x0c\x7c\x85\x5d\x85\x69\x1c\x24\xd5\x82\xbc\x98\x34\xc0\xf2\x4e\xc1\x6e\xbf\x
e1\xfc\x2a\x23\x5f\x70\x13\x5b\x73\x8c\x53\x98\x1b\x86\x3c\x79\xca\x26\xa6\x44\x5d\xa4\xc3\x5

2\xec\x4d\x12\x93\x51\x14\x3e\xd6\x6d\x42\x34\xb1\xb0\x36\xc6\xca\xd6\xf4\xb1\x52\xef\x5f\x40\xc7\x14\x64\xd9\x74\x8c\x43\xf5\x3e\x31\x18\xa5\x38\x08\xaf\xa5\xfd\x4e\x39\x90\x4d\x63\x9a\xb6\xb2\x42\x44\xb3\xc5\xd8\x9e\x9d\x7f\x2d\x74\x68\x22\x8c\x0b\x4c\xd4\xd3\x0c\x2f\xcc\xeb\xdd\xfa\xa2\x69\xbc\x28\xac\x3f\x51\xe2\x36\x48\x9e\xaa\x90\x0e\x39\x15\x20\x41\xfc\x76\x1e\xf0\xc9\xd0\x29\xc9\xab\x87\x55\xb6\xa5\xf2\x66\xb1\x6b\xe4\x45\x38\x27\x84\x0d\xb7\x09\xa1\xec\xc9\x5c\xaa\xfa\xc5\x06\x2a\xd5\x8e\x32\x68\x25\x4a\x51\x43\x33\x61\xbd\x21\x79\x63\x37\x91\x98\x77\x65\xf2\x39\x1c\xc2\x7d\x09\xfd\x6f\xf9\x65\xc9\x3c\x2b\x0c\xf3\xc2\xe4\x0d\x85\x4e\x5a\xa9\x76\x4f\xb2\x43\xc0\xc3\x9d\x3e\x69\x8c\xac\xe5\x83\x9f\xb8\xc2\x60\xc2\xe2\x05\x55\x57\xc7\xf2\x1a\xcc\xf2\x82\x3d\x80\x9c\x42\x9a\x01\xc0\xe5\x5e\x21\x45\xa0\x72\x4c\x6d\x2b\xa2\x98\x59\xf2\x32\x3b\x00\x66\x32\x73\x8e\x63\x30\xe6\x2d\x87\x26\xa2\x94\x3b\x80\xd1\xd0\xd9\xe5\xb0\x4c\x9d\x01\xa8\xb0\x24\x21\x69\x1b\x75\xdb\x60\x72\x0c\x1f\xb8\xfd\xec\xc1\x10\x25\xe3\x88\xc8\x08\x1e\x0a\xe8\xa7\xab\x68\x34\x42\x7d\x2c\x1a\x0c\x51\x1a\xc4\x61\x32\x1e\x5d\xdf\xd3\xe1\x9e\x5a\x4d\xb0\x61\xf2\xd0\xc1\x4f\x1e\x4c\x29\x69\xfc\x2b\x70\x21\x3a\xc9\xa1\xc9\x82\x24\x6a\x5c\xc1\x33\x3c\x98\xe6\xb8\xf6\x8c\x47\xa3\x7a\xe6\xb1\xc4\x1d\x1e\x33\xdf\x72\x88\x45\xf7\x04\xdd\x43\xcf\xc8\x70\x90\xff\x7f\xe6\x3e\x33\x53\x30\x32\x77\xe3\xd4\xec\x71\x12\xf5\x18\x75\x51\xc5\xa6\xdd\xa8\x9f\x4e\x33\x9b\x65\x87\xa2\xfa\x3b\xe7\x55\x92\xa1\x44\xa6\x70\x6a\xdd\xf6\xaa\x91\xd6\xdc\xe2\x56\x47\x97\xb6\xb4\xae\x4d\x69\x85\xc6\x9b\xa5\x89\x07\x0a\x05\xae\x88\x71\x57\xa4\x41\x66\x0b\xe9\xa6\xbe\xc2\x12\x79\x4b\xe3\x01\xf8\x5b\x03\xd6\x12\xda\xcc\xcb\x31\x00\xbb\x69\x43\x4d\x2e\x92\x41\x33\x05\x39\x4f\x26\xcb\xc7\x1c\x3d\x37\xf5\xd9\x4a\x6a\xe8\x22\x85\xb3\xdd\x59\xea\x88\x89\x52\x0b\x1e\xc6\x8b\x23\xb5\x90\xa2\x6f\xa7\xd5\xb6\x69\x06\x14\x15\x77\xc8\xf8\x32\x67\x79\x1a\x4b\xf6\x04\x2c\x87\xf8\x75\x7b\x7d\xb8\x25\x4a\x9c\x50\x88\xdb\xbf\xd9\x34\x5c\x0f\xa8\x1f\x7f\xb3\xb3\x77\x83\xc8\xf6\xc9\x2d\x28\x6d\xbb\x70\x21\xe5\x71\x66\x5b\xbe\xc5\x2d\xa4\x15\xb7\x74\xd8\xed\xfc\xf0\x39\x1c\x6e\x4a\xdb\xb3\x44\x21\x0b\xaa\xc7\x99\x4b\xd5\x22\xfb\xf2\xb7\xa1\x2f\x2f\x95\x0e\xbe\x01\x75\xc4\x5f\x44\x6d\x6e\x59\x7c\x95\x34\xc9\xcf\xf8\x50\xbb\xc2\xca\x3e\x7c\xc3\x1e\xfa\xe3\x81\x35\xd8\xc5\x76\xf4\x95\x14\x0e\xda\xee\x9a\xe4\x2e\xe5\xae\x4d\x76\x21\xe0\x89\xd8\xc2\xc5\x15\x09\x7b\x3a\xbc\x42\xc6\x60\xcf\x74\xdb\x73\x79\x77\x52\x31\x96\xf6\xcd\xe8\xd2\x0a\x6c\xb1\x0a\x06\x2b\xd6\x90\x04\x4e\xc5\xbc\xa2\x2f\x71\x5f\x67\xc8\x01\x20\x8c\xf9\x51\xdb\x97\xf4\xf8\x06\x1a\xef\xa2\x19\x4d\x06\x02\x15\xac\x43\x2a\x9d\xad\xa9\x61\xa6\x02\xdd\xa5\x37\xb1\x9e\xf8\xee\xa0\x0f\xfe\x13\xf8\xf1\x3d\x2b\x88\xbf\x75\xc6\xfc\x2d\xea\x89\x6d\xcc\x70\x51\x45\xf1\x9d\x18\xe3\xbd\xa3\x68\x2a\x8a\xef\x8b\x71\x57\xd4\x13\x7f\x75\xde\xfd\xd5\x95\xc5\x5f\x7f\xab\xf0\x14\xdb\x1e\xc7\x09\xed\xfe\xf6\x8e\x4a\xfa\x70\xf7\xfd\x85\x6d\xeb\x90\xc7\xb7\xe2\xee\x51\xa6\x20\x2f\x54\x79\x22\xd3\xa5\x9c\xd2\x92\xe5\xaf\xbc\x39\xf3\x3a\xad\x6f\x35\x29\xe5\xbd\xe7\xa0\x5c\x34\xf7\xa4\x92\x73\xd2\x40\xcc\x4c\x3f\xa9\xa5\x9d\xe4\x15\x1d\x89\x27\x41\x3f\x5a\x00\x17\x3f\xd5\xe4\x93\xef\x82\xfc\xc2\x43\x96\x14\x94\xc5\xf1\xfa\x6d\x32\x08\x46\x68\x92\x8c\xae\x87\xd1\x08\x25\x43\x44\x37\x2d\x76\x8a\xb7\x1c\x79\x59\x6c\xfb\x2d\xb5\xa0\xd6\xb0\xc2\x98\xc4\xeb\x3d\xf2\xfe\xe6\x85\x19\x3b\x48\xb2\xb5\xec\xff\x66\x30\x35\xb0\x11\x9c\xf6\xc9\x0c\xea\x44\xbc\xb7\x32\x49\x93\x3c\x21\x9f\xd0\x16\x39\x7d\xe8\x05\x58\x3d\xb4\x85\x62\x7c\x45\x10\x28\x87\x10\x4f\x47\x23\xc7\x42\x11\x18\x14\xcb\x44\x8a\x77\x64\x8b\xe4\xc9\xe7\xa4\x5c\xc9\xed\x54\x6c\xbf\x8d\xfa\x69\x90\x5e\xcf\xd3\x91\x4b\xf9\x41\x9d\xa0\x20\x5b\x28\xd3\x7a\x12\xe1\x82\x77\x39\x18\xa1\x28\xbe\xc0\x69\xa4\x04\x70\x55\x22\x3a\xe8\x79\x46\xcd\x08\xa3\xe6\x74\x56\x08\xfb\xc7\x63\x0c\x83\x7b\x9c\xf0\x33\xb8\x08\x72\x8e\x10\x0b\xe5\x41\xc5\x20\xe3\x54\x89\x50\x59\x1c\x40

\x2e\x77\x25\x97\x38\x4d\xa3\x10\x67\xe8\x03\x55\x88\x44\x38\xa3\x0c\x7c\x72\x8d\xa2\x98\x65\x33\x2e\x10\xa8\xd0\x82\x9e\xab\xe1\x64\x51\x00\x86\xcc\xe5\x28\xb7\x48\xd4\x40\x32\x51\xef\xae\x4f\x28\x09\x2b\xd2\x4d\x89\x49\xa2\xec\x2f\x16\xe1\x51\xb8\x89\x9e\x41\xa6\xac\x67\xba\xe1\x88\xbd\x4d\xf2\x37\xc6\xf9\x45\x12\x96\xfa\xc8\x4b\xa5\xf5\x18\xf9\x36\xc7\x33\x84\xcc\x70\x86\x14\x7d\xc5\x20\x9b\xcf\xab\x33\x88\xe1\x24\xb8\x8a\xcd\x2f\x12\x23\x21\xc2\x42\x91\x56\xcf\x65\x4e\xbc\x3d\x3d\x1f\xe3\xd8\x62\x3a\x4c\x76\x94\x72\x2c\x50\xc1\x7c\xd8\xb9\xab\x28\x6f\x4d\xff\x60\x45\x80\x99\x49\x71\xd7\xaf\x48\x38\x96\xa6\x76\x9c\xbe\xe3\x4d\x5e\x04\xd9\xe1\x55\xcc\xc8\xfe\xba\xf6\x8c\xd4\x7c\x56\x17\x3e\x4f\xe4\x11\x36\x41\x5e\x9e\xbc\x98\xdb\x0f\x5a\xab\x74\xba\x2d\xb5\xfe\x9f\x6c\x3a\x21\xa2\x56\x1c\xe5\x2b\x01\x11\x4e\xd9\xd6\x17\xa4\xe7\x53\x32\xba\xd6\xf1\x40\x96\x0c\x0a\x25\xe3\x54\x78\xdc\xa6\xcf\x32\x54\x70\xf4\x88\x2a\x85\xf9\xa4\xd3\x55\x6a\x42\x90\x3b\xa8\xec\x07\x8e\x6d\x07\x71\xc5\xf8\x10\xa7\x38\x1e\x90\x06\x60\x9c\x27\xfa\x7a\x35\x86\x81\xc9\xc5\x36\x80\xce\x7d\x06\xd9\x52\x63\xd8\x98\xea\x2e\xac\x94\x4c\x66\x9a\x54\xe5\x3d\x8d\xe9\x38\xc0\x04\xd2\x55\x6b\x86\x40\xdd\xe6\xf3\x51\x64\xb0\xa9\xd5\xc5\x35\x1c\x11\xa5\x21\xa4\x1c\x00\xa9\xd5\xff\xca\xbc\x92\x47\x2c\x47\x5b\x8c\x6d\xf2\x3b\x8b\xb9\xbc\x88\x96\x2b\xe7\x78\x66\x23\xb0\xe4\x8a\x38\xd9\xe6\xca\xe5\x11\xd4\xa5\x35\xc2\xdf\xa9\xeb\xc4\x49\x35\xbc\xf8\x6d\xc8\xa6\xcc\x5d\xdd\x31\x57\xe8\x90\x31\x33\x96\x24\x00\x48\x0a\x4c\xe8\xc3\x10\x65\xc9\x18\xd3\xd4\x53\xe8\xea\x02\xc7\xe8\x3a\x99\xa6\xc2\xcc\x3e\x20\xe2\x2c\x05\x7e\xcf\xb1\x73\xef\xba\x0b\xea\x8e\xce\x65\x7b\x19\xa2\x0c\x60\x65\xc5\x1c\x19\x31\xf4\xb7\xdc\xee\xe6\xa2\x51\x69\x4e\x7b\xc9\x84\x08\x3b\x93\x42\xee\x61\xf2\xce\x1d\xc4\x29\x09\x18\x68\x98\x14\x99\x6a\x0c\x9a\xc8\x7b\x9e\x52\xb6\x3a\xe9\xfe\x59\x55\x7e\xb9\xe5\xb8\x43\x23\xca\x25\xb6\xe8\x9f\x75\x8d\x8b\x88\x87\xfc\xb2\xed\x7d\x30\x06\xa3\x89\x39\xf5\x10\xdb\xaa\x8b\x62\xfa\x66\x2d\x03\xac\x97\x6e\xb1\x64\x3a\x4f\xe5\xe2\x67\x68\x4b\x6a\x5f\xfd\xb4\x40\xea\x22\xc7\x26\xbb\x8b\xae\x92\xf8\x59\x4e\xe5\x67\xee\xee\x28\x05\x2f\x1c\x25\xc9\x04\x05\xfd\xe4\xd2\xb2\x0d\x96\x77\xf9\x19\x87\xf6\xcc\xdd\x61\xe0\xa2\xa2\x55\xb9\x9f\xe2\x6d\x85\xbc\x5a\x95\x16\x8f\x38\x9c\x40\x4f\xc1\xfe\x65\x91\x75\x63\xdb\xf8\x06\xa3\x24\xc6\x0f\xc0\xf1\x00\x2e\xda\x2a\xf6\x10\x78\x51\x61\x27\x23\xc5\xe6\x6e\x64\x72\x2e\x12\x55\x38\xe2\xfc\xd4\x6a\x4f\x66\x3f\x23\x5b\x6f\xf7\x63\x14\x80\xe7\xad\x16\x8b\xb0\x34\xb2\x90\x11\xe7\xbd\x1c\x84\x2d\x3c\x8d\x30\x7e\x50\xc3\x21\x66\xd1\x79\x1c\x0d\xa3\x41\x10\xe7\x2c\xa0\x64\x44\x7b\x0f\x20\x69\x3b\xb6\x63\xf2\xcf\x92\x07\x31\x3d\x2b\xcb\x6f\xee\x21\x6c\x8c\xd9\xbc\x4e\x16\x8e\x30\xf8\xb2\xe9\xd5\x9c\xb1\x46\x56\xb3\x30\x31\x52\xda\x0d\xc6\xdc\x41\xc3\xf7\x96\xea\x45\xf6\xcf\x56\x36\x76\xc3\x16\xc6\xa1\xfd\xaf\x0e\xe0\xb4\x31\x6b\x34\x1a\x7e\xa3\xd9\x68\x79\xa8\x31\x6b\xb4\x1b\x9d\x46\xb7\xb1\x76\xf6\x60\x80\x3d\xd4\xad\x1c\x7a\x85\x85\xaf\xe3\x33\x62\xac\xd8\x2b\xe6\x10\x0c\xcb\x95\x3f\xd0\xff\x7e\xf9\x02\x31\x7b\x35\x51\x63\x88\x6a\x62\x7a\xbf\xdb\xb2\x28\x0a\xe5\x3f\x80\x2a\x19\x0d\xf1\x9f\x95\x8d\x49\x75\x00\x94\x3c\x46\x38\x3e\xcf\x2f\xa8\xe9\x91\x93\x8b\x54\x8f\x19\x53\x2c\x94\xc5\x22\xc5\xec\xc6\x83\x24\x24\xf4\x8e\xe9\x0f\x9d\xdc\xe1\x75\x79\xec\x4f\x41\x00\x38\x1e\xac\xec\xe3\x99\xbb\xcd\x79\x01\x64\x2a\xad\xf6\x85\x83\xbb\x14\xc4\x5a\x21\xb2\x8b\x25\xae\xc1\xbc\xb0\x2e\x96\x2a\xca\x90\x7c\xcc\x87\xeb\x0b\x45\x73\x61\x53\xe1\x8c\xe5\xc2\xa7\xea\xcb\x17\xb4\x8f\x67\xa5\xe1\x5b\xe6\x10\xd0\x20\xc8\x71\xcc\xf6\x7c\x95\x82\x1c\xcc\xdf\x4d\x48\xd2\x3d\x6c\x31\xe0\x27\x8c\x1b\x4a\x94\x09\x69\x7e\x17\xbd\xd7\xad\x8a\x4b\x15\xda\x10\xd8\xf9\x3c\x7e\x86\x78\xd3\x74\xa7\x34\x83\x92\x3a\x53\xa2\x81\x9d\x17\x0b\x47\x42\x06\xf6\x67\x83\x61\x59\x7c\x15\xf3\x8b\x40\x84\x3a\x28\x48\xcc\x5d\x3a\xca\x8e\x0b\x1e\xa3\xf0\x1c\x07\xf0\x63\x95\x25\x51\xf8\x45\x1d\xa3\x53\xbd\x51\x30

\x9e\x20\x3c\x83\x48\x92\xfd\x48\xef\x1c\xbd\x57\x25\x65\xcc\xdb\x06\x7a\x9f\x3a\xb0\x05\x49\x
51\x10\xff\x87\x23\x50\x3a\xd4\x27\x22\x69\x8c\x61\xab\x45\x41\x8e\x02\x94\x47\x63\x8b\xc4\x6
d\x0b\xc9\x2e\x77\xd7\x9d\x14\x42\x1e\x1c\x52\x14\x6d\x11\xf4\xd8\x2c\x9c\x46\x3c\x2a\x36\xf9\
x4f\xad\xd9\x46\xcb\xa8\x16\x51\x8c\x9f\xa3\xf5\x7a\x5d\x44\xcb\x76\x4a\xf1\x14\x8e\xda\xe3\x2
5\x14\x89\x70\xdb\x5f\xb6\x8a\xa6\x5f\xbe\xe4\x6d\x58\xca\x8b\x46\x2b\x08\xfe\xce\x6d\x49\x1e\
x53\xba\xb8\xee\x34\xa6\xee\x28\xf7\x55\xbb\xbf\x85\xcc\xc1\xae\x92\x31\xd8\xa4\x42\xb1\xd9\x
2e\x6d\xa9\x68\xda\x72\xac\x04\x51\x1c\xf4\xf5\x93\x87\x74\x00\xa8\xca\x4e\x69\x0c\x0e\x22\x0
4\x2a\x82\x61\x94\xdf\x55\x14\x2c\x16\xa7\x58\x5d\x0e\x26\x45\x3e\x57\x0d\xdd\x6b\x61\x4d\xa
6\x1c\x65\x8b\x8b\xe4\x64\x32\x76\x86\x61\x11\xd5\x4e\x05\x0c\x1e\x67\x7e\x0b\x96\x0e\xfd\x0
3\xd2\x6f\x35\x09\xe9\x67\x0a\x5f\xb0\x10\xbc\x22\x4a\x6d\xa1\x77\x41\x7e\xb1\x32\xc0\xd1\xa8
\xa8\xb9\x8a\x16\x88\x48\x64\x3f\xff\x56\xda\x79\x1c\xe6\x48\xc6\xf1\xf7\xb6\x76\x9f\xec\xb8\x2b
\xd3\x82\x71\xde\x55\x69\x61\xde\x39\x57\x06\x0b\x27\x35\x8a\xab\x1c\xfd\xdc\x3c\x39\xaf\x98\x
34\xc2\xcc\xef\x1b\x4e\x93\x3a\x52\x6f\xf1\x29\x90\xc4\x86\x61\x34\x1a\xf1\xb0\xb3\xcc\x4d\x02\
xce\x5b\xf3\x85\x12\x7e\x98\x8b\x6d\x87\x5e\x19\x94\xd3\xc5\xa7\xd2\x2c\x33\x48\x95\x08\xe5\x
be\x8c\xcf\x2a\x1c\xc1\x98\x2b\x88\xef\x3e\x69\xd1\x12\x32\x99\xc4\xf6\x23\x96\xcc\x1e\xcc\x03\
x15\xf9\x9a\xaa\x37\xe4\xe3\x4f\x57\xee\x28\xf3\x9f\xae\xd0\x16\xf9\xd7\x91\x40\x6d\xfc\xe9\x77
\xb2\xcd\xcc\x5a\x41\x88\xbb\xeb\x7d\x3d\xfc\xba\x28\x16\x64\x9f\x91\xcc\x39\x4a\xee\x09\x2a\x
dc\xdd\xd1\x56\x6b\x8d\xd9\x46\xa3\xbb\x81\x9e\x93\x2e\xfc\x0e\x7b\xfa\xde\xde\xde\x5e\x1d\x2
d\xd1\x17\x3f\xfe\x88\x1a\x33\xbf\x01\xdb\x3d\x41\xc0\xb1\xdd\xd3\x2e\xd6\x1a\xb3\x76\xb7\xd3\
xa0\xc0\xae\x74\x60\x57\x55\x81\xc1\xf0\xe2\x6c\x0a\x9e\x3e\x35\x40\xe3\xe5\x4b\x5a\x13\x2d\x
21\x18\xe9\xd2\xfa\xac\xee\xea\x16\xd4\x61\x7f\xe5\x65\x97\xb6\x50\x63\xa5\xe3\x2c\x03\x63\xc
a\x8a\x3e\xa7\xf6\x36\x9c\xda\xea\xe8\x47\xb4\xd2\x41\xff\x85\x7c\xb4\x89\x96\xfd\x2a\x22\x8a\x
c1\x39\x54\x71\xc3\x43\xe9\x20\x18\x5c\x60\x96\x5d\x67\xbe\xc0\x41\x6a\x7e\x22\xf4\x98\xd6\x6
a\xb4\x2a\x39\x2a\x29\x48\x92\xdd\x44\x1a\x0c\xfb\x15\x13\xad\xba\x85\x3e\xa5\x35\x5a\x1e\x0
8\x72\xad\xbf\x66\xe9\xd3\x55\x91\xc3\xa7\x26\xca\x17\xf0\xd1\x17\xd4\xa8\x18\xd6\x3c\xc6\x57\
x92\xb3\x13\xdc\x3a\x32\x05\x48\xcc\xd3\xf7\x3c\xd1\x46\xd2\xee\x7c\xca\x8e\xf6\xf3\x0c\x69\x7
0\x3c\x00\x43\x1a\xfa\x5f\xbb\x21\xcd\x3e\x9e\x99\x9a\x00\x1b\x38\x52\x70\x8b\x02\x5d\xa1\xbf\
xab\xc5\xdf\xd4\xd5\x17\x17\x78\x56\x59\x85\x51\xe1\xe4\xb9\x60\x54\xcd\x4a\xad\xdf\x17\x23\x
bf\xc0\x33\x33\x84\x26\x1b\x3f\xe9\x68\x3f\x3f\x91\x90\x35\x70\xe6\x6d\x8f\xa9\x57\x95\x4f\x9e\
xd9\xa2\xc7\x48\x3a\xeb\x26\xa0\x0b\x3c\xeb\x5d\x04\x69\xe5\x3c\x5b\xd9\xdc\x03\x1d\xe4\x48\
x8b\xe8\x41\xee\xea\x8e\x87\x38\x8e\x1d\x5b\xe3\x00\x96\x00\x69\xd7\x0b\xb5\x8f\xdf\xad\xdb\x
f8\x9d\xad\x2a\x69\xa7\x31\x2c\xaf\xeb\x60\x10\x02\xdc\x6f\x49\x14\xd7\x9e\x3d\xbb\x45\xc4\x4d
\x89\xc2\xe9\x7a\x5b\x44\xd3\xc3\x57\x0a\x25\xdc\xea\x0b\xc6\x21\x3c\xfd\xf9\x52\x13\x5f\x6c\x
d4\x66\x5b\xac\xc7\xea\x91\x32\x69\x95\xc5\x12\xa5\xd0\x3a\x6f\xf9\xd1\x85\x3e\xb2\xa3\xcc\x2
2\xab\xe6\x6a\x91\xd4\x74\x72\xa3\x6c\x0b\x6d\x96\xe4\xc7\xa4\xab\xa5\x05\x9a\x09\xe8\xf4\x4
1\x9c\xb3\xce\xae\x64\xd3\x7e\x96\xa7\xb5\xc8\x43\xcd\xba\x07\x49\xf8\x0a\x95\x05\x59\x51\xeb
\x75\x9b\x03\xee\xc2\x7b\x9e\x32\x4c\xab\xa8\x59\xd5\x7d\xf6\x6d\x90\x47\xb1\x5f\x6d\xd3\x62\x
65\xf9\xbe\x25\x1e\x6f\xb7\x75\xb1\xea\x7f\xde\xee\x55\x15\x81\xfb\x5a\x53\x23\x68\xcf\xbe\x87\
x51\x5c\xfe\xa3\xb6\x31\x3a\x1c\xdf\xf0\x4e\x26\x21\x48\x77\x24\x3a\x75\x2b\xc3\x34\x19\x93\xb
7\xbd\x24\xc4\xb0\x49\x55\xdd\x90\x64\x80\x77\xd8\x93\x14\xba\xbd\xfd\xb6\x24\xc8\x71\xa1\xc5
\xf0\x4d\x6f\x4e\x6c\x15\xd1\xfd\x49\x5e\x6e\xd5\xb7\x28\x51\x6b\xb1\x5d\x4a\x54\x13\x1b\x95\x
78\xf3\xd0\x7b\x95\xd6\xf4\xbc\x5c\xce\x91\xa4\x45\x2f\x7a\xbb\x32\x60\x04\xbd\x9d\xd7\x22\xbe

\x26\xf4\xad\xca\xae\x5b\x5c\x78\xab\xd2\x10\xae\xba\x53\x7d\x3c\xd9\x5b\x5e\xaf\xb6\x51\x7d\x
cc\x87\xeb\x62\x9b\x62\x0f\xb7\xdb\xa4\x68\xa3\x7f\xde\x1e\x55\xb1\xfd\xfb\x5a\x59\xd3\x7c\xb8\
x6e\xdf\xa0\xc8\x28\x3e\xe4\xf6\x94\xa7\xd7\x25\x06\x46\x21\x26\x47\xf4\x8f\x47\x07\x3d\xee\xe
9\x54\xc3\xd9\x20\x98\xe0\x5a\xc9\xc6\x69\xb2\x65\x34\x08\xf2\xc1\x05\xaa\x99\xe9\xa3\x01\x85
\x8b\x34\xb9\x02\xba\x85\x8c\x2b\xb5\x67\xef\x82\xd1\x30\x49\xc7\x38\x64\xd3\x10\x06\x79\x60\
xa6\xa0\x5b\x9c\x81\xcb\x93\x7a\x7b\xfe\xcd\xe6\x6a\x11\x32\xf9\xa6\x99\x37\x50\x18\x65\xdd\x
05\x19\x56\x67\xdc\xac\x8e\xcb\x18\x40\xd9\x1a\xa6\x31\xa3\x1e\x6a\x21\xa0\xd0\x15\x87\x53\x
af\x1c\x80\x46\xa4\xe0\x85\x5c\x98\x38\x64\xd9\xcc\x24\x2f\x74\x67\x26\x5e\xc9\x4e\xf6\x5a\x4a
\x89\x36\x9e\x66\x39\xea\x63\x14\x91\x11\x1d\xe3\x38\xa7\x79\xd6\x02\xb8\x5e\x4f\x71\x2e\x3c\
x16\x2a\xe5\xf6\xd5\xf2\x74\xaa\xca\x7d\x9a\xe3\x90\xba\x56\x15\x09\xe2\x3f\xe3\x49\x8e\xa6\xf
1\x84\x27\x0d\x54\xb3\x83\x4a\x36\x2d\x0d\x0b\xf7\x7d\xc5\xc6\x01\x32\x0d\x6e\x8b\x51\x10\x5
e\x62\xae\xcf\x15\xcd\xe0\x20\xbb\x2b\xb3\xe6\xd1\x46\xfa\x19\x4b\xa2\xcd\x92\x98\xe6\x09\x8a\
xf2\x8c\x7b\xc5\x20\x42\xc1\x77\xbd\x63\xea\x5b\x91\xa7\x09\x71\xdd\x97\x4c\x95\xb2\xee\x32\x
f3\x3e\x04\x56\xca\x36\x9b\x01\xc8\xc0\xc9\x3c\x15\xb5\x9d\x55\x67\x4a\xb4\x7c\xbc\x13\xe4\x0
1\x17\xd6\x1b\x55\x25\xcd\xed\x30\xcc\xa0\x0d\x9e\x17\xdc\x31\xd2\x8c\x16\xaa\x6f\x8a\x22\xc8
\x82\x91\x79\x9c\x19\xbb\x20\xba\xe6\x99\x13\x00\xe5\x97\xd4\xa7\x24\x90\x2c\x28\xa9\x3d\x31
\x70\xbc\x87\x99\xcc\x4f\x14\x9d\xda\x33\x93\xdf\x57\xaa\x37\x7f\x6f\x64\x25\xab\x24\x33\x37\xd
d\xeb\x8b\x74\x74\x72\x40\x51\x69\x80\x58\x30\x51\x15\x94\xec\xe3\x0c\x64\x34\x27\x4e\x24\xa
3\x35\x89\x29\x03\x86\xf3\x23\xa5\x6d\x43\xd7\x5c\xe4\xcb\x4d\x89\x6c\xc0\x0c\xa2\x5d\xda\x52
\x93\xa4\x57\xa5\x60\x9e\xeb\x34\x43\xc1\x65\x10\x8d\x20\x62\x17\xe5\x0b\xc0\xec\xdc\x54\x73\
x22\x39\xab\x44\xf1\x65\xf2\x19\x67\x7a\x92\xe1\x1a\x4b\x0e\xec\xa1\xab\x8b\x68\x70\x61\x65\x
d5\xfd\xeb\x12\x56\x6d\xb6\xca\x17\x4a\x3f\x49\x46\x38\x88\x6f\x50\x98\xec\x8d\xa6\xd9\x05\xfa
\xf9\x02\xe7\x34\x9e\x09\xcf\x45\x0b\xee\x5a\x93\x20\x05\x46\xc1\x5e\x15\x5c\x5b\xb0\xeb\x5b\x
84\x03\x11\x9c\x1e\x46\xfc\xee\xdb\xbc\x00\xb8\x43\x09\xc9\xb5\x66\x78\xaa\x5c\x57\x5c\x8e\x0
5\xc1\xd8\x33\x05\xab\xb1\x56\x69\x51\x65\xf1\xd1\x01\x5f\x50\x67\xc2\x96\x48\x41\xdc\x16\x6d\
x09\x79\xcd\x8d\xd3\x60\x64\x5d\x6a\x15\xf2\x51\x32\x34\x73\xd1\x3d\x2f\x5e\xc8\x0a\x5b\x5a\x
4a\xe6\xb2\xc2\x1c\x7a\x51\xdb\x1e\xd1\xaf\x97\x4c\xe3\x9c\xd3\x97\x85\x99\x10\xa0\x31\x4d\x2
4\x7c\x04\x71\x8b\xb7\x54\xfc\x57\xb5\x26\x5f\x98\xbc\xc8\x35\xe4\x0c\x83\xa3\x64\x1a\x87\x68\
x3a\xa1\x0e\x85\x83\xd1\x34\xc4\x1a\xdd\x9b\xd5\x34\x8c\x0a\x23\x17\xf9\x43\xf5\xd8\xb6\x02\x
8b\x30\xb9\x8a\x65\x3c\x92\x78\x74\x8d\x86\x53\xb1\x28\x2d\x91\xf4\x57\x57\xd1\x08\x67\xd4\x
a9\xd2\x2e\x6b\x01\xdf\x48\xf1\x38\x88\x62\x55\xb8\xaa\xd6\xaf\x71\x30\xab\x29\xfd\x82\x8b\x53
\xb4\x6c\xcb\xcc\xee\xcd\xbf\x52\x15\x73\x4e\x35\x0f\xae\x29\x07\x4a\xe6\x78\x28\xad\x3f\x47\x
12\x01\xba\xe8\x09\x68\xc3\x49\x4e\xe4\xab\xda\xc7\x28\xae\xc9\x4d\x3e\x47\x6d\x4f\xa1\x33\x9
b\xf9\x24\xcf\xe0\x6d\x23\x12\x42\x77\x12\xc0\x72\xb7\x2d\xca\xe7\xa9\x9a\x85\xfd\x7e\x29\x8f\x
80\x78\xbb\x24\xad\x27\xa7\xd1\x04\xc1\x0c\xa7\xe4\x34\x29\x36\x86\xe5\xe2\x80\x00\xce\x90\xf
6\x8a\x8c\xbb\xa8\x7b\x90\xe0\x2a\xb6\x5c\xf5\xae\x39\x46\x4a\x0a\xac\x82\xe1\xc3\x94\x9b\x45
\x15\xee\x2b\xb3\x30\x3d\x19\x96\x3c\xa2\x16\x34\x14\x4e\x86\x96\xb7\xe4\x99\x9e\x4f\x95\x3c\
xb6\x68\x19\xb6\x6e\x85\x93\x8a\xbf\x27\x37\x7d\x57\x63\xb7\xca\x59\x28\x4b\x9d\xbc\xee\x68\x
e5\xe6\xd8\x0d\xff\x24\x93\xb7\x4f\xc6\x86\x58\x60\x62\x9d\xb1\x52\x8b\x37\x95\x87\x89\x93\xa
6\x23\x13\x3d\x3f\x83\x5f\x04\x19\x64\xc8\x75\x9e\xb8\xe7\xa6\x22\x2f\xd8\xb5\xec\x03\x45\x27\
x9d\x41\xa7\x61\xd7\x70\x86\x92\x58\x3a\x0a\xfb\x5d\x54\xeb\xf8\x4d\xb0\x64\xad\x5b\x8e\xc5\x
fb\xb4\x32\x3f\x06\x8b\x47\xfb\x79\xf8\x5e\xa2\xbe\x96\x65\x20\x2b\x0d\x98\x5a\xe6\x6a\x46\x07

\x61\x81\x9c\xe4\xb7\x8d\x6e\x47\x1a\x42\x34\x44\xf2\xbc\x20\x77\x95\x6d\x48\xc4\x1c\x28\xa1\xdb\x8e\xf7\xb7\x9b\x9d\xae\xdd\x49\xac\x2c\xd5\xf5\xad\x23\xac\xf1\xd8\x6a\xd5\xc3\xac\x1d\x63\x11\xde\xc3\xad\x21\x30\xd5\x10\x73\x2c\xb1\x0b\x4d\x0a\x5f\x38\xf7\xaf\x32\x61\xf4\x72\x1f\x2a\x12\x40\x58\x56\xf1\xa8\x25\x1c\x2b\x09\x40\x2b\xcc\xcb\x94\x1a\xf4\xbd\x99\x0d\x87\x65\x63\xe6\x1b\xf2\xd1\x62\x63\xfd\x71\x12\x02\xcb\x90\x07\x9b\xa6\xe5\xaf\x9e\xb1\xcf\x19\x41\x98\x02\xd7\xe3\x08\x57\x76\x21\xa2\xac\x88\xf9\x0f\xcd\x5d\xde\x0b\xcc\xf9\x14\xf0\xaa\x3d\x63\x48\xd9\x74\x29\x6a\xc9\xf9\xaa\x13\x5a\x50\x26\x14\x65\x0c\x1c\xeb\xd1\xa1\x91\x60\x0a\x1b\x15\x82\x85\x3c\xd8\xf8\x12\x21\x9d\xe0\x6b\x03\x25\x9d\x63\x4d\xf1\xf7\xde\x7c\x27\xf6\x58\x92\x9b\x4c\xe0\xe2\x64\x90\xe8\x7d\x02\x28\x07\x39\xcd\x17\xcf\x6a\x16\x31\x43\x51\x94\x21\x3c\x1c\xe2\x41\x1e\x5d\xe2\xd1\x35\x0a\x50\x88\xb3\x3c\x9d\xc2\xb3\x07\x72\xfa\x72\x12\x0f\x70\xa5\x28\xa3\x15\x29\x54\x49\xf4\x00\x28\x15\x01\xb9\xa1\xc4\xe2\x9a\x0b\x32\x08\xf7\xb4\x33\xa0\x2d\x4e\x8e\x22\x99\x90\x43\x2d\xe1\x28\x5d\x46\xe8\x25\xd5\xe6\x53\x3d\x2f\xba\x10\xdd\xef\x59\xc6\xd7\x3c\x10\x95\x83\x41\xf3\xd6\xca\x3c\x01\x7e\x01\xce\x2a\x8d\x10\x67\xb2\x7b\xd2\x3c\x58\x17\x0f\x29\xef\x5a\x3c\x52\xf2\xbb\x8e\xdf\x5c\x6d\x35\xab\x89\xf9\x19\xd3\xf8\x28\xf1\xef\x03\x36\x69\xcf\x44\xe0\xa4\x28\xce\x71\x3a\x94\xac\x85\x91\x73\x55\x70\xfe\xca\xba\xce\xa9\x96\x6e\xb7\x2c\x3e\x62\x80\x2e\xf0\x68\x82\x53\x22\xfe\x54\x58\x04\x7b\x0c\x37\xe6\x1b\xac\xa3\xfc\x15\xee\xf1\xa8\xcc\xa4\x3b\x55\xd0\xae\xae\x7c\xa2\xbd\xda\x87\x2e\xd5\x6c\xc2\x96\x5b\x3f\x27\x57\x55\x8c\x07\x01\xb4\xeb\x7e\xcf\x58\x17\xf6\x00\xb8\x48\x3d\x2f\xb2\x95\x08\x87\x45\x35\x8b\x58\x91\xe1\x52\xa5\xf0\xc5\x8f\x8d\x56\x7a\x22\x2c\x79\xff\xdd\x76\xef\xfe\xe9\x89\x88\xd0\x3c\x28\x05\x69\x81\xd1\xd5\x5f\x82\xa6\xf6\xc7\xc1\xa0\x12\x5d\x8d\x83\xc1\x5d\x68\x4b\x54\xbf\x13\x7d\x7d\xc6\x76\x15\x92\x44\x5f\xbd\x4f\x80\x16\x99\x07\x4a\x64\xb4\x11\x5a\x77\x31\x62\x2b\x3d\xfe\x0a\x4d\xd2\x1c\x1f\x06\x82\x0d\x38\x31\xb0\x1f\x85\x17\x03\xcf\xd4\x02\x21\x7d\xdf\x05\xf9\x05\x0d\xeb\xfb\x84\xbf\x67\xc3\xfc\xa2\x88\xf4\x7b\x73\xe6\x75\xda\xdf\x6a\x78\x5f\x86\x4c\x8d\x87\x23\xae\xdf\x7b\xbc\x5f\x0e\x79\xd1\xb8\xbf\x02\x43\x39\xfe\xaf\x2b\xe8\xaf\xf8\x0e\xc1\x7f\x6d\x01\x74\xcd\x2b\x0a\x1e\x35\xb6\x98\x32\x89\x00\xa4\x68\xb0\xd2\xfb\x92\xf0\x34\x4a\x6d\xc9\x05\xc6\x15\x46\xb6\xdb\xae\x66\xa2\xc5\xca\x72\x23\x2d\xf1\x78\x3b\x33\x2d\x56\xfd\xcf\xb3\xd3\xaa\x8a\xc0\x7d\x71\xca\x3e\xb4\x67\x37\xd5\xa2\xb8\xfc\x0d\x6c\x89\x8d\xf2\xe3\x60\x22\x84\xc3\x71\x30\x59\x3c\xf6\x82\xc5\x45\xdc\x04\xe1\xb2\xca\xa4\x63\x7e\x5b\x83\x65\xb4\xb4\x85\x5a\x6e\x9b\xe5\xeb\x1c\xfb\x16\xa3\x65\xfa\xe7\x32\x5d\xa6\x7f\x4e\x03\x66\x0e\xb8\x59\x00\xae\x45\x68\x09\xf9\x75\x8b\x4d\x34\xff\x52\xc5\x32\x9a\x03\x6e\x69\x80\x9b\x4e\xc0\x4d\x2b\x60\x3b\xe4\x3c\x8d\x26\x23\xb8\x7a\xa9\xd1\x61\x79\xf9\x12\xfc\x26\xbe\xd0\xe7\x26\x79\x5e\x27\x8f\x80\x82\x0d\x8a\x98\x8a\xdf\xe8\x54\xd4\x7e\x43\x2f\x49\xeb\x3f\xfc\x80\x00\x9b\xdf\xd0\x73\xd4\x58\x59\xeb\x48\x33\x54\x7f\x81\x7e\x2b\x09\x77\x21\xcd\x3d\xb5\x05\x1f\x07\x13\xb0\x99\xdd\xce\x6b\x35\x8e\x30\x74\xba\x8b\x9e\xa3\x5a\x0b\x2d\xa3\xdf\xea\xac\xa7\xad\xa1\xd5\xdb\xc9\x88\xcf\x60\x2a\x2e\xc2\x90\xa7\xfb\x36\xa9\x91\x7d\x20\x28\xa1\x2d\x24\xa1\xd3\x35\x9c\x49\x20\xb6\x5e\x51\xdc\x6e\x1c\x7c\x11\x8d\x30\xaa\xc9\xfd\x64\xe1\x02\x5c\xb1\x46\xac\xc3\x22\x37\xb3\x78\x9f\x19\x67\x95\xa1\xde\xc1\x4e\x5e\xe1\xc9\xb7\xb7\xb3\x14\xac\x76\x21\x46\xff\x4d\x9b\x5a\xb2\x1d\x82\xda\xf5\xc8\x5b\x49\x75\x73\x4b\x51\x6b\xc1\xcd\x41\xd4\x13\x86\xf2\xe2\x8d\x30\x94\x9f\xcf\xf7\x8d\x12\x29\xbe\xc4\x69\x86\xdf\x49\x05\x8b\x57\xb6\xb8\x66\xdf\x15\x9f\x9d\xd4\x5d\x0a\xd4\xb6\x05\xf0\x3f\x9d\xff\x10\xf6\x43\x56\x28\xeb\x60\x29\xa7\x51\x1b\x3e\xe5\x0b\x9b\xd9\xe6\xff\x56\x3f\x43\x5b\xe8\xb7\x6a\xb1\x3a\x2d\x2c\xe5\xe0\x3c\x4e\x52\xfc\xd5\xb8\x8a\x04\xf2\x20\x0e\xc1\xcf\xb9\x98\xee\x88\xbc\x39\x1c\xce\xe3\x19\x52\x

3b\x14\xc6\x77\x5b\x5b\x68\xd9\x9f\xc3\x93\x64\x0a\x93\x6b\xdf\x8a\x11\x5b\x45\x82\x54\xa4\xbd\xcc\xf0\xdb\x24\x99\x14\x4b\xc2\xd3\x71\xf0\xa4\x19\x55\x44\x0e\xed\xc6\x33\x98\x6c\xa2\x67\xdb\xaf\x7a\x3b\xbb\x7b\xaf\xf7\x0f\xfe\xf9\xe6\xed\xbb\xf7\x87\x1f\xfe\xef\xd1\xf1\xc9\xc7\x9f\x7e\xfe\xe5\x5f\xff\x13\xf4\x07\x21\x1e\x9e\x5f\x44\xbf\x7d\x1e\x8d\xe3\x64\xf2\xef\x34\xcb\xa7\x97\x57\xb3\xeb\xdf\x1b\x7e\xb3\xd5\xee\x74\xd7\xd6\x37\x96\x56\xb7\x58\x84\x5b\x71\xb4\x13\x8b\x76\x61\x54\x8b\x21\x76\x78\xa5\x14\x96\x1b\x8a\x85\xa9\x4d\x14\xd2\xda\xb1\xb9\xa9\x90\x99\x8e\x1c\xfb\x0d\x73\xec\xca\x88\x90\x24\x2d\x8f\x82\x9a\x64\x07\x16\xb4\x8c\xfc\xfa\x19\x78\xaf\x14\x02\x53\xd3\x24\x2e\x0e\xb4\x59\x05\x68\xfd\x8c\x6f\xf0\xb2\x18\x66\x81\x4a\x05\xa2\x58\x89\xdc\xf3\x85\x08\x33\x80\xfe\x17\xda\xa2\xec\x5b\x13\x97\x07\xef\x41\x6c\x88\x97\x96\x94\x0f\x82\x6c\xc5\x0f\x46\x91\x46\x6c\x49\x6b\x58\x84\x9b\x22\x77\x8f\x7e\xc8\x97\xf6\x88\x17\xce\xcc\x3e\x9d\xc7\xa3\xff\xe3\xd1\x5f\x1c\xfd\x3f\x9e\xec\x2d\xfb\x5d\xf4\x6a\xb7\xb2\x83\x96\xdf\x7d\xb5\x2b\xfb\x68\xf9\x5d\xf5\x09\xbe\xde\xde\x69\x8b\x22\xf3\xe7\x3a\x6e\x55\xc4\xe1\x1e\x9d\xb7\xfc\xae\xd3\x7b\xcb\xef\xfe\x0d\x34\x02\xd5\x0f\xeb\x30\x18\x77\x39\xab\xdb\xfd\xfd\xc1\x32\x2a\x09\xf1\x87\x24\x8a\x73\x97\x93\xb1\xdf\x75\x38\x19\x5b\x0f\xd3\x05\xa6\x6e\x2f\x63\xd1\x64\x55\x57\x63\x09\xe8\x1d\x4e\x50\x3a\x11\xdf\xc9\x59\x0d\x68\x73\xd1\xb5\xf1\x4d\x1f\xa3\xe8\xaa\x12\x2e\x6b\x7c\xf1\x2d\xe4\xb3\x06\x95\x16\xf3\x35\xe6\xb5\x84\x7c\xcb\x5f\x3c\xb4\xa7\xb1\xda\x70\x35\x47\x63\x1f\x64\x1f\x81\xa1\xea\x66\x4c\x44\xa0\x62\xb1\x34\xc9\x62\xd1\x82\xb0\xb9\x29\xdc\x25\xe5\x68\xa3\xf3\xbc\x7a\x28\x0c\x46\x96\x6f\x2b\xec\x61\xd2\x3e\xf5\xf6\xce\xfb\xd4\xdb\x6f\x60\x9f\xaa\x82\xc3\x7d\xef\x53\xd6\xe5\xf4\x76\xf7\x71\x9b\x12\x7f\xf7\xb6\x4d\x65\x57\xc1\x64\x37\x0e\xa3\x20\xae\x2d\xba\x63\xd9\x8e\xe4\xdf\xfe\x96\xf5\xf6\x61\xb6\xac\x2a\xcb\xe4\xdb\xdf\xb2\xde\xee\x6a\x9b\xd6\xe3\x8e\x65\xec\x58\xd2\x8a\x59\x68\xf3\xfa\xaa\xbb\x97\x98\x17\x09\x5b\x02\x48\xe9\x23\x8f\x86\x0f\x5f\xd8\xdd\x09\x5d\xdc\x8d\x06\xf9\x7f\xb8\x58\xa1\x1f\x49\xf7\xd9\x57\xfa\xad\x58\xfe\xf3\xd4\x05\x40\x58\x6e\x6d\x41\xf7\x4e\xda\x02\x96\xa3\xf6\x6b\x2a\x0d\x3c\x24\xbd\xca\x2e\x02\x5f\x7b\x75\x31\x0e\x06\x0f\xa8\x5a\xf0\x10\x6f\x16\x7e\x41\x6b\x7f\x07\x75\x83\x91\x2f\xf6\x16\xaa\x08\xc5\x88\x45\xfa\xf2\x6e\xa7\x03\x35\xc1\xe4\xe6\xdd\x4e\xc7\x26\xe3\x81\x89\xf3\x67\x7c\x4d\xb3\x60\x53\x3b\x58\xd1\x57\x70\xfe\x0d\xe2\x9c\x27\xf1\x4e\xd2\x31\xb5\xd1\xde\xfd\xe9\xc3\x27\xd8\x74\x4f\x92\x37\xb8\x10\x06\xd1\xd5\xd5\xd5\x4a\x32\xc1\x71\x96\x8d\x56\x92\xf4\x7c\x35\x4c\x06\xd9\x2a\x24\xe1\x4e\x56\xb5\x3a\x17\xf9\x78\x64\x51\x84\xec\x5e\x4e\xde\xec\xec\x15\x68\x8b\xe7\x8a\xc1\x10\xe6\xfb\x80\x68\x7b\x9c\xe1\xfd\xc2\x52\x9e\xc3\x1e\x45\x06\x26\x23\x0f\x51\xcc\xdd\x5e\xa4\x70\xcf\x85\xab\x4b\x1b\xd5\xfc\xe6\xba\xe2\xe9\x62\xc0\x77\x18\xa9\xc9\x61\x31\xf4\x04\x29\xef\x76\x3a\xf3\xb0\x8d\x72\x66\x8b\xac\x07\xa9\x96\x3e\xe4\x09\x9a\x50\xab\x53\xd9\x3b\xc7\xb1\xc3\x19\x7e\x31\xda\xee\xc0\x86\x67\x13\xf9\xcd\x75\x30\x21\x55\xbe\xd2\xce\x01\xe6\xda\x97\x02\x1f\xa5\xed\x9b\x5b\xbb\xdd\x38\x88\xf6\xa1\xfd\x70\xb0\xd4\xe8\x3d\x98\x59\x7f\x0e\x87\x86\xf7\x0d\xa5\xf9\x39\x29\x9a\xe6\x57\xfc\xa3\x98\xab\x75\x2d\x9f\xdf\x6d\xc1\x78\xea\x34\x36\x1a\x0d\x1d\xf0\x82\xde\x41\x73\xfd\x7e\xaa\xc9\xbb\x3b\x90\xc2\x9f\xd0\x08\xa1\x0a\x48\x84\x1d\x40\x06\x56\xb2\x68\x6f\x63\xa5\xcf\xeb\xd2\x58\x00\x36\x40\x25\x95\xb3\x60\x94\xa3\x6d\xf8\xcf\xe2\x62\x31\x50\x17\x25\xef\xfb\x20\x2f\x4c\x36\x8f\xcf\xe1\x70\x85\xba\x45\xe0\x1a\xef\x8c\x07\xf8\x95\xe4\xad\x81\xe2\x4a\x7e\x47\xb5\xe6\x42\x02\xaf\x3a\xc5\x16\xf1\x96\xac\x74\xc6\x3d\xcc\xda\xc2\x4b\x8d\x90\x07\x33\x51\x2e\x56\x87\x15\x96\xcb\x2d\x0c\x42\x0b\xd0\x21\x7e\x03\x63\x63\x4b\x89\xb6\xc8\x19\xb9\x00\x26\x7c\x82\xc5\x1b\xe7\x71\x99\xef\x31\xb4\x47\xec\xc9\x52\x4e\x62\xe2\xb4\x68\xf1\xc2\x82\xe5\x6b\xb6\x31\x11\xf0\xea\x47\x66\xcc\xa2\xe1\xca\x0d\x5a\x5e\x72\x7c\xac\x47\x01

\x22\xc6\x81\xe7\x80\xf3\x82\x59\x75\x59\xa2\x65\xe7\x5f\x2b\x23\x39\x18\x43\xe1\x04\xc2\xa0\x70\x62\x93\x8c\x82\x0d\x7a\x55\x9b\x17\xfe\x74\x66\x09\x42\x13\x62\xe0\xcc\xcf\xca\x41\xc9\xa7\x07\x25\x69\xa0\x4b\xd3\xfe\x68\xd8\x0b\x64\x9d\xa3\x60\xc3\xd8\x32\x54\xe6\x3b\x89\xac\x58\xcc\x18\x6b\x1b\xda\x28\x4b\xb5\x24\x1d\x0d\xa7\x3f\x4b\xb4\x0b\x11\x60\x8e\xd7\xab\x6a\x73\x5d\x89\x07\xcb\x7e\xc7\xb7\xe2\xbd\x0b\xf2\xdd\x7b\xf4\xbe\xb5\xf8\x95\x49\xbd\xa9\xce\xcd\xa5\x4a\x8a\x76\x43\x7a\xaf\x72\xf7\xe2\x03\x52\xb8\xba\xd8\xb4\xe9\x7e\xed\xe2\xec\x8b\x55\xf3\x90\x43\x6c\xb8\x0b\x98\x52\xb1\x41\xa8\x90\x0b\x59\xdf\xb5\xe7\x98\x2e\x2c\x6c\xd8\x55\x89\x05\x1c\x57\xca\xf7\xbb\x9b\x17\x25\xc7\x77\x0a\xcd\x7e\x76\xf7\xf8\xe1\x73\xb3\xb3\xee\xf1\x23\xe9\xe6\xda\x1a\x39\xd3\xaf\xfd\xa5\xcf\xf4\x83\x68\x72\x81\xd3\xe5\x07\x36\x11\x80\xd3\xbb\xdc\xd4\x9f\x73\x88\x37\x33\x77\xde\xcb\x69\xbe\x07\x1d\xfb\x40\x38\x4e\x26\x0e\xed\xf2\x4b\xb7\x09\x81\x78\xaf\x65\xc2\x50\x6a\x90\x33\x5c\x90\x43\x25\xfa\x93\x33\x62\x56\x71\x0f\x5e\xe6\x2c\xaa\x02\x2d\xb2\x40\x3a\x0d\x72\xba\xa1\x73\x93\xe3\x59\x4e\x4e\x91\x01\x7b\x46\x13\xda\x27\xe6\x9b\xc5\x53\x6d\x04\x21\x1e\x44\xe3\x60\x34\xba\x66\x69\x40\xc3\xca\x37\x37\xf2\xa8\xdc\xb0\x56\xd8\xc0\x9d\x08\x34\xd4\x66\x17\x4f\xc6\x71\x1b\xfc\x1e\x34\x3d\x47\x31\x25\xd2\xad\x8e\xdc\xf9\xc5\x2e\x76\x94\x9a\x0e\x47\x2d\xb9\xcc\x4a\x31\xbb\x45\x02\x89\x7d\x3c\xbb\x65\x26\x08\xcb\xf0\x4a\xe4\x23\xdf\x37\x2c\x38\x9d\xda\xcd\x43\x14\x4f\xa6\xf9\x5d\xe6\x94\x93\x87\x4a\x74\xb7\xa0\xb3\xfb\x22\x8e\x81\xc6\x28\x2c\xf4\x71\xeb\xa4\x12\x30\x5a\xf6\x10\x36\xc5\xe4\x6c\xa1\xa2\x0d\x5a\xe1\x85\x95\x7a\x7a\x0a\xf5\x70\x8d\x40\x01\x68\x53\x06\x7a\x63\xd7\xcd\xbb\x77\xda\xa2\xbb\xda\x6e\x2b\x6d\x10\x9b\x9d\xa6\xa7\x29\xcf\xd7\x1f\x4d\xed\xfe\xee\xba\x6f\xd7\xee\x68\x44\x32\x2f\xd3\x84\x9b\x87\x14\x70\x00\x16\x1a\x57\x6b\x22\x2a\x52\x62\x4b\x76\x54\xbd\x9f\x84\xf4\xe0\xf2\x3a\x97\xe3\x55\x56\x12\x57\x54\x45\x11\x59\x1d\x9c\x97\xf1\x20\xc5\xf9\x3d\x29\x95\x88\xfc\xbb\x6f\x0f\x1c\x04\xbd\x64\x6c\xc2\xe6\x89\x4c\x1d\x7d\xab\x6a\x0c\x65\xe7\x60\x47\x80\x60\xab\xce\x48\xe8\x8b\xa8\x8f\x82\x78\xd4\x3d\xdc\x4b\xbc\xdd\xee\x33\xbe\x2c\x1c\x98\xe6\x84\x97\xa5\x87\x2a\x29\xba\xac\x3e\x4e\x76\x43\xfc\x12\xc5\x14\xed\xe8\x2b\x29\x2e\x26\xeb\x7a\x59\x64\x4c\xad\x12\xd7\x17\xe8\xb0\xec\x51\x32\xb7\x47\xa3\xe4\x0a\x05\x69\x3f\xca\xd3\x20\xbd\x46\x4c\xbd\xf4\x19\x5f\x5b\xe2\x0e\x7e\x96\x35\x12\x3f\x5a\x1b\x2e\x19\x28\x5d\xdd\x52\x6d\xb4\xe6\x38\x43\x12\x94\x4a\xdc\x20\x21\xfe\x1b\xe8\x36\x92\x14\x45\x71\x8c\x53\x88\x3e\x9b\x4c\x73\x10\x20\xf4\x28\x7c\x10\x33\x91\xea\x18\x29\x19\xb2\x07\xda\x8a\x11\x90\x8e\x6b\xfc\xe4\x1a\x91\xa5\xc6\x22\x24\x90\x48\x5a\xc9\xa4\x4c\x1f\x19\x49\x05\x23\xa9\xa0\xd1\xd8\x2f\x87\x47\x30\x9f\xf4\x1a\x70\x12\x84\x68\x90\xc4\x59\x1e\xc4\x7a\xf3\xd6\x24\x52\xea\x1c\xbb\x15\x6b\x02\xef\xd3\xe8\x0c\xfd\xba\x85\x1a\xb3\xce\x80\xfe\xcf\xe6\x0e\x63\x14\x6e\x75\xe9\xff\xca\x35\x63\x89\xa6\x13\x8b\xb4\x67\x1b\x45\xfe\x09\x71\xc8\x60\x07\x7a\x88\x28\x64\x82\x89\xdf\x4b\x24\xb2\x92\x7c\x65\x36\x66\x6c\x19\x48\xe8\xb4\x8d\x8f\x3b\xf4\xa4\xaa\xbe\xb8\x58\x30\xb7\x8b\x40\x06\xc3\xfc\xcd\xc4\x1f\x7b\xb7\xdd\x63\xd1\xc7\x00\xaf\x08\x96\x58\x69\x24\x94\x05\xa7\xbc\x4a\x20\x32\xa3\xf4\xfd\x07\x23\x93\x49\x82\xb7\x32\x37\xf8\xd8\x43\x45\x0f\x83\xa1\xfe\x4f\x8f\x1e\x36\x47\x4c\x5d\x44\x44\x24\x3c\xb4\xa0\xa1\xb9\x11\xc4\xdc\x35\xe6\x46\x11\x73\x57\x7d\xa0\x48\x62\x77\xe7\x76\x3d\xaa\x9e\x86\xf1\xb6\xec\xc7\x44\xba\xd8\xb7\x07\x47\x2b\x0d\x38\x56\xca\x31\xe5\xb1\xd2\x80\x16\x12\x0a\x97\x34\xf8\x25\x93\x40\xa5\xee\x0c\x39\x36\x0e\x06\xf6\x4b\x22\x71\xf0\x77\x18\xc1\x6d\xfc\xa5\x15\xe6\xb3\x6e\x7b\xd9\xf2\x7a\x14\xf5\x97\x09\x2a\x21\xd8\xb6\x66\xda\x57\x1c\x0f\x96\xc1\xa6\xd1\xf2\x9e\xba\x59\x6a\x1f\xc6\x61\x67\xbe\xf1\x5d\x76\x11\x34\x3b\x3a\x48\xf2\xb2\xa9\x83\xcb\x2e\x82\x8e\xdf\x34\x5f\xb6\xd6\x2d\x25\x5b\xda\xab\x34\x9a\xe0\x71\xe8\x77\x1b\x56\xdb\x3f\xe5\xd5\xa4\xff\x

39\x1c\xea\xed\xe0\xcb\xc9\xe7\x70\x58\x76\xef\xa0\x76\x3d\x09\xf1\xf2\x60\xd8\xb7\xbe\xce\x53\xc7\xeb\xe5\xf3\x51\x10\x8e\x83\xd8\xf6\x39\xb1\x03\xc3\x03\xfd\xf5\x24\x08\x97\x83\x38\x8b\x66\x1b\x4d\x7d\x10\xc8\xa7\x28\x4b\xfc\x86\xdf\xd4\x47\x9c\x7d\xda\x58\xdb\x58\xd3\x67\x88\x7c\xfa\x1d\xa7\x09\x73\xbd\xb6\x7c\x8d\x1d\xdf\xa8\x8e\x6c\xf9\x02\xcf\xb4\x0f\x01\xd6\x89\x8b\xc6\xdd\x08\x8d\xf7\xe9\x40\x9f\xdc\x34\xe8\xf7\xa3\xdc\xfa\x72\x79\x84\xcf\x83\xc1\xf5\x43\xdf\x01\x89\xd5\x03\x4f\xfa\xa2\x81\x97\xc5\x5a\x11\x8f\x6c\x89\xc0\x33\x59\x19\x9a\x59\x28\x5b\x07\xe2\x77\xb3\x2d\x7e\x13\xaa\xe7\xbf\x09\xb1\x8b\xdf\xf4\x57\x41\xda\x85\x7d\x29\xfc\x62\x84\x4c\x31\xa0\xf4\x6b\xdc\x61\x51\x74\x38\xb5\x4a\x4f\x79\xaa\x3e\x09\xda\x2c\xde\x26\x4a\x0d\x42\x89\xb4\x59\x99\x00\xc5\x1b\x41\x77\xf2\x1b\x4a\x6e\xe2\x8d\x4c\x65\xe2\x65\xac\xbe\x92\x68\x0a\x9e\x09\x29\xc1\x8f\x82\x82\xe8\xa8\x0c\xd8\x40\x31\x7a\x91\x7e\x73\x32\x59\x54\x11\xa9\x28\x20\x65\x5e\xbb\xb8\x62\xd2\x1d\x8a\x8d\x75\x69\xb3\xe3\x7b\xe5\xda\x64\x4f\xa5\xab\xcd\x4e\xdb\x53\x08\x6f\xb3\xd3\xf1\x8a\x89\xdf\xec\x74\x3d\x75\xf4\x36\x3b\x6b\xfa\x8d\xb0\x4e\xca\x9b\xdd\x86\xc7\xa8\x75\xb3\x0b\xf8\x08\x4a\xd9\xec\x36\x3d\x99\x56\x36\xbb\x6d\xcf\x46\x2d\x9b\xdd\x96\x27\x53\xc8\x66\xb7\xe3\xc9\xf4\xb3\xd9\x05\xbc\x14\x9a\xd9\xec\xae\x79\x3a\xd5\x6c\x76\xd7\x3d\x9d\x6e\x36\xbb\x1b\x9e\x41\x24\x9b\x6b\x0d\xcf\x42\x4e\x9b\x6b\x80\x3f\x5b\x12\x9b\x6b\x80\x3d\x23\x8d\xcd\xb5\xb6\x67\x10\xc7\xe6\x1a\x20\x4e\xc8\x68\x73\x0d\x70\x2e\xd6\xd9\xe6\x5a\x57\xbe\x40\xf7\x8a\x25\xbb\xb9\xc6\xaf\xd6\xc9\x62\xde\x5c\xdb\xf0\xf8\x52\xdd\x5c\x6f\x78\xc5\x12\xde\x5c\xf7\xbd\x62\x71\x6f\xae\x03\x3a\x05\x05\x6f\xae\x43\xe3\x82\xd1\x6c\xae\xb7\x6f\xce\xbc\x6e\xe3\xf1\xf2\xe0\xcf\xbf\x3c\xe8\x5d\xe0\xc1\x67\xd2\x29\x58\x29\xd4\x0d\x88\xa6\x39\xcb\xa6\x13\x32\x30\x98\xc5\xa7\x96\xfa\x0d\x72\x3c\x0d\x69\x8e\xbe\xdb\x42\xcf\x38\xe4\x67\x16\x8b\x10\xe1\xa4\x71\x8f\xd7\x15\xa5\xe6\xf8\xa2\x9d\x23\x3c\xc4\x29\x86\x83\x5e\x1a\x9d\xc3\x99\x2c\x8a\xa3\xbc\x00\x93\x4d\x27\x38\x05\xd5\xf5\x96\x96\x9e\x43\x82\xb2\x3d\x3d\x1f\xe3\x38\xd7\x0a\xa0\x3c\x41\x17\x41\x1c\x8e\xb0\x32\x6e\x32\xec\xbe\x15\xb2\x62\x53\x03\x55\x4d\x77\x40\x49\xf7\x4d\x63\xc9\x53\x13\xa8\x28\xce\xd7\x25\x0d\xfd\x50\xae\x2f\x14\x13\xea\xec\x98\xc7\xfc\xa2\x06\x55\xc2\x7f\x24\x50\xe1\x85\x8c\x8d\x72\x88\xb0\x22\x96\xd0\xf4\x5f\x00\xe9\x32\xc2\x57\x2e\x14\x9d\xcd\x4b\x08\x1f\x70\x14\xd0\x97\x2f\x6a\x79\x4e\x70\x80\x25\xe8\x8c\x79\xf5\xef\xc8\x9a\x13\xb6\x23\xb0\xe8\xec\xc0\x8d\xaa\x75\xa3\x15\x27\x56\x7e\xd7\x8e\x96\xbb\xa5\xc5\x6a\x1c\xc4\x79\xab\xb9\x68\x13\x8b\xd5\xd8\x1b\x25\xc1\x6d\xaa\x74\xdb\xf0\xbe\x28\x7f\x4b\x52\x5a\xa1\x14\xec\x21\xf9\xd5\x75\x8e\x0f\x21\x39\x90\xf1\xda\x96\x77\x59\xa1\xbf\x7d\xba\xe8\x8a\xb6\xaa\xac\x88\xa2\xf4\x62\x2a\x84\x02\xda\x2b\x81\x1b\xda\xb2\xe3\x6c\xd1\x2c\xec\xce\x58\xf6\xd5\xeb\xdc\x66\xfc\xbc\x90\xbb\xa0\x0d\x95\x45\xf2\x69\x17\xf5\x4f\xa3\xb3\x5b\x25\xcf\x2e\xcc\xb9\xa3\xdf\x31\x55\xd5\x16\x8e\xa3\x6a\x51\xc1\x58\x8b\xd4\x16\x1e\x62\x6e\x84\xb6\x8e\x28\xf3\x6d\xcd\x7a\x46\x46\x93\xbc\x26\xf0\x50\x4c\xa4\x3e\x99\x99\x9b\xed\x06\x93\xc9\xe8\x9a\x35\x1c\xa4\xe7\x53\xc2\xc2\xb3\x32\x7f\x45\xc6\xaf\x57\x26\x69\x92\x27\x04\x47\x99\x73\x97\x19\x4e\xe8\xbb\x8f\x5d\xc1\xd2\xf5\x1f\x65\x9d\x3f\x47\xd6\x81\x80\xd1\x7f\x42\x5c\x22\x6b\x4e\xa5\x0a\x26\x12\xb0\xc5\xd2\x7b\x3c\x94\x17\xba\x75\x52\xe5\x84\x31\x0b\xa9\x24\x55\x5d\x6a\x37\x7f\x36\x49\xcf\xc5\x57\xba\x6d\x3b\x17\x39\x21\x6c\x62\x8b\x0e\xdf\x4a\xd0\xcf\xe8\x8f\x2c\x8a\x59\x30\x56\xc2\x32\x1a\x33\xbf\xc1\xfe\xea\xe8\x8b\x9a\xc6\x97\x2d\xaf\x5a\xdd\x6a\xa1\xfe\x6e\xa7\xa3\x59\x53\xd8\x0c\x40\x74\xaf\x49\xb4\xc5\x46\xd5\x62\x00\xc2\xd3\xde\x94\xde\x8e\x15\x9a\x60\x7b\xae\xe2\x53\x93\x93\x36\x66\xdd\xb5\x76\xa7\xd9\x6a\xf8\x1e\x6a\xcc\xf0\x70\x10\x06\xfd\xf5\x0d\x4b\x5e\xc5\xc6\x6c\x63\xbd\x1f\x84\x83\x21\xf6\x60\x60\x5a\xcd\x4e\x7b\xad\xab\x96\x3b\x73\xde\x88\x69\x69\xf4\xe4\x5e\xbc\x13\x99\xf4\x6c\x7b\xd

7\x55\x30\x41\x18\xdc\xab\xe7\xef\x21\x7e\xd7\xbd\x63\xb8\xaf\xaf\xf9\x6c\x50\x24\x3e\x11\x78\x3c\xbd\x20\x8a\x1c\x11\x78\xdf\x7d\x92\x4a\xbf\x3b\xe5\x0f\x67\x36\x97\x10\xe9\x33\x21\x38\xb3\x00\xf9\xab\xd5\x6a\x12\x4c\xea\x29\x8e\xbe\x20\xf9\x25\xec\x75\xed\xba\xe6\x23\x8e\xbe\x54\x04\xd8\x6c\xd7\x2d\x00\x21\x94\xb1\xe2\x92\x6e\x82\xbb\x9b\x71\xc8\xbe\x72\x43\x61\xbf\xee\x57\x86\xb4\x81\xa4\x31\x45\x4b\xa8\xa1\x8b\x0f\x4a\x69\x5f\x2b\xed\x97\x96\x6e\x6a\xa5\x9b\xa5\xa5\x5b\x5a\xe9\x56\x69\xe9\xb6\x56\xba\x5d\x5a\xba\xa3\x95\xee\x94\x96\xee\x6a\xa5\xbb\xa5\xa5\xd7\xb4\xd2\x6b\xa5\xa5\xd7\xb5\xd2\xeb\xa5\xa5\x37\xb4\xd2\x1b\xe5\xb3\xd3\xd0\x66\x67\xce\x64\xfa\x5a\xf1\xf2\xd9\xf4\x9b\x5a\xf1\xf2\xe9\xf4\x5b\x5a\xf1\xf2\xf9\xf4\xdb\x5a\xf1\xf2\x09\xf5\x3b\x5a\xf1\x8e\xc1\x0d\x56\x57\x09\x43\xfe\x1c\xc5\xe7\xa4\x6a\x14\x8c\xfa\x36\xb1\x39\x20\xdb\xc0\xa9\x75\xa0\xfa\xf0\xc9\x3a\x28\x03\xf8\x64\x1d\x80\x10\x3e\xb5\x6c\xe8\xf4\x8a\x3b\x68\xf5\x1b\x41\x62\x6f\xaf\x16\x78\xa8\xef\xa1\x81\x87\x42\x4f\x5a\xa0\x1e\x42\x6b\x1e\xd9\x42\x1b\x67\x3a\x6f\x08\x69\xbd\xd0\x43\xa2\x6a\x31\x42\x1e\x42\x7e\xd3\x43\x27\xa7\xbe\x51\x6f\x40\xeb\xd1\x96\x68\xd5\x62\xd1\x92\x7a\x6b\xa4\x5e\xd3\xa8\xd7\xa7\xf5\x04\x92\x81\x54\xaf\xe5\x21\xd4\x84\xf6\x5a\x46\xbd\xb2\xfe\xb5\x45\xff\xda\x0b\xf5\xaf\x23\xfa\xd7\x59\xa8\x7f\x5d\xd1\xbf\xee\x42\xfd\x5b\x13\xfd\x5b\x5b\xa8\x7f\xeb\xa2\x7f\xeb\x0b\xf5\x6f\x43\xf4\x6f\x63\xa1\xfe\xf9\x0d\x8f\xf5\xcf\x37\x09\xa6\xac\x83\xbe\xef\xb1\x0e\xfa\x26\xc5\x94\xf5\x90\x60\x49\x7b\xe8\x9b\x24\x53\x4a\xa2\x2d\x8f\x93\xa8\x49\x33\xa5\x7d\x6c\x8b\x3e\x9a\x44\x53\xda\xc7\x8e\xe8\x23\x50\x8d\xd9\xc9\xd7\xaf\x1d\x9d\xf4\x10\xea\xd0\x4e\x9a\x74\x13\xd2\x8a\xd6\x4e\x12\x7a\xdb\xa0\x15\x4d\xc2\x19\xd0\x8a\xf6\x4e\xfa\x1e\x22\x1d\x3d\x39\xf5\x4d\xca\xe9\xd3\x8a\xd6\x4e\x12\x8e\xd1\x6c\x40\x45\x93\x74\xca\xfa\xd8\x11\x7d\x6c\xda\x79\x8d\xab\x8f\x84\xe6\x68\x1f\x9b\x76\x66\xe3\xec\x63\x87\xf7\xb1\x69\xe7\x36\xae\x3e\xb6\x45\x1f\x9b\x76\x76\xe3\xea\xe3\x46\xd1\x47\x3b\xbf\x71\xf6\xb1\x2d\xfa\x68\x67\x38\xae\x3e\x12\xc6\xc8\xfa\x68\xe7\x38\xae\x3e\xae\x17\x7d\xb4\xb3\x1c\x27\xad\xb6\x3c\xde\x47\x3b\xcf\x71\xf5\xb1\x29\x68\xb5\x69\x67\x3a\xae\x3e\xae\x89\x3e\xb6\xec\x4c\xc7\xd5\x47\xb2\xfc\x69\x1f\x5b\xbe\x7d\x41\xee\xef\xbb\x89\xb5\x0d\xb8\xb6\xec\x5c\x67\x7f\xdf\xde\x49\x32\xac\x64\x6d\x9d\x9c\xb6\xec\x5c\x67\x7f\xbf\x64\x41\x76\xa1\xa2\x9d\xeb\xec\xef\x3b\x3a\xd9\xf6\x50\xb3\x05\x15\x4d\xd2\x29\xeb\xa3\x5f\xf4\xd1\xce\x74\x5c\x7d\x6c\x17\x7d\xb4\x33\x1d\x57\x1f\x61\x22\x69\x1f\xed\x4c\xc7\xd9\xc7\x86\xe8\xa3\x9d\xe9\x38\xfb\xd8\xf2\x58\x1f\xdb\x76\xa6\xe3\xea\x63\x43\xf4\xb1\x6d\x67\x3a\xae\x3e\xb6\x44\x1f\xdb\x76\xa6\xe3\xea\x23\x61\xe5\xb4\x8f\x6d\x3b\xd3\x71\xf5\x71\x43\xcc\x63\xdb\xce\x74\x5c\x7d\x24\xcb\x83\xf5\xd1\xce\x74\x9c\xb4\xda\xe1\xb4\xda\xb6\x33\x1d\x57\x1f\x9b\x45\x1f\xd7\xec\x0b\xf2\xe0\xc0\x2d\xa8\x76\x69\x27\xed\x5c\xe7\xe0\xc0\xde\x49\xa0\x39\xe0\x01\x6d\x3b\xd7\x39\x38\x28\x11\x03\x3a\x20\x02\xda\xb9\xce\xc1\x81\xbd\x93\x84\x77\x34\x61\x58\x3b\x76\x51\xc7\xd5\x47\x32\x1f\xb4\x8f\x1d\x3b\xd3\x71\xf5\xb1\x25\xfa\xd8\xb1\x33\x1d\x67\x1f\x1b\xa2\x8f\x76\xa6\xe3\xea\xa3\x5f\xf4\xd1\xce\x74\x5c\x7d\x5c\x17\xf3\xd8\xb1\x33\x1d\x57\x1f\x81\xe6\x68\x1f\xed\x4c\xc7\xd5\x47\x10\xc9\x69\x1f\xed\x4c\xc7\xd9\xc7\x96\xc7\xfb\x68\x67\x3a\xae\x3e\xb6\x45\x1f\xbb\x76\xa6\xe3\xec\xa3\xcf\xfb\xd8\xb5\x33\x1d\x57\x1f\x9b\xa2\x8f\x5d\x3b\xd3\x71\xf5\x71\x43\xcc\x63\xb7\x65\x2e\x48\xb8\x46\xc9\x71\x3a\xc6\x61\x14\xe4\xcc\xa9\x0c\xdc\x15\xd4\x72\xe4\x88\x8b\xb6\x50\x0d\xfe\xbb\x84\x02\x5d\xc3\x4a\xcb\xf8\xac\x8c\x4f\xca\xf4\xed\x65\x9a\xac\x4c\x93\x94\x19\xd8\xcb\xb4\x58\x99\x16\x29\x13\x1a\xda\x5c\x4d\x55\xb9\x67\xb1\xd4\x5d\x30\xa0\x2d\x64\x4a\x17\xd9\x74\x83\x3c\xb0\x1d\xcc\x83\x3c\x10\xa1\x7c\x82\x3c\x70\x2b\xc7\xe2\x57\x51\x9e\x9d\x24\x79\x30\x12\x30\xe3\x9d\x20\x0f\xa8\x07\xc9\x73\xb4\x6e\x81\x0e\x75\xde\xe2\x61\xce\xa1\x0b\x8f\x13\x28\x6f\x74\xc6\x99\xf2\x4a\xa0\x79\x5a\x80\xfc\

xf1\xc7\x1f\x51\x07\x2e\xde\x1a\xb3\xf5\x46\x71\xdf\x56\x94\xf8\x07\x6a\x35\x0d\xe2\x50\xfb\xb2\x8f\xb6\x10\xa8\xdd\x87\xa3\x24\x49\x6b\x52\x27\x57\x15\xdd\xbb\xab\x73\x50\xf6\x2d\xda\x92\x9e\xf4\x85\x23\x50\xaf\xd5\x6a\x05\x6e\x4b\xa8\xdb\xa6\xf9\xd2\x36\x20\x98\x68\xbb\x4e\x15\x36\x76\xfd\x2c\xaf\xca\x70\x2e\x94\xb3\xf2\xdb\xea\xda\x59\x13\x1c\x53\xcd\xea\xe0\xe6\xe9\x66\x0d\x2e\xb1\x48\x67\xdb\x55\x3a\xfb\xd6\xda\xd9\xb7\xb7\xed\xec\x5b\x6b\x67\xdf\x56\xed\xac\xd9\x5b\xd9\x89\xaa\x26\xba\xcf\x83\x4d\x41\x4e\x3d\xbb\xff\x20\x18\xbc\x53\x37\x06\xf0\x51\xb4\x79\x52\x95\xe6\x95\x9f\xe3\x0d\xa9\xe8\xbc\x2d\xe4\xbb\xcf\x0c\xe3\x9d\xde\x6f\x0b\xdd\x7b\x38\xae\xb8\x50\xd9\xf5\xbf\xc0\x04\xae\x30\xf6\x4f\xed\x77\x17\xfb\xec\x96\xac\x56\xdb\x57\xae\x25\xf6\x17\xbe\x8f\xa0\xb4\xb0\xaf\xdc\x45\xec\x3b\x2f\x21\xe6\xdf\x38\x1c\xb1\xdc\xc0\x30\x87\x2c\x02\x4f\x08\x63\xaa\x16\xad\x90\xac\x1c\xdc\x10\x4a\x59\x3d\x28\x58\xc1\x29\x53\xdc\xd0\xc1\x63\x71\xfd\x6f\x6c\xbc\xf0\xf9\x93\x41\x0b\x2e\xef\x4a\x1e\x41\x83\x7c\xb5\x7b\x38\xd0\x5f\x02\x49\x4d\xf5\x35\xf3\x50\xe6\x21\xf5\x0a\x0d\xf8\x24\xda\x42\x01\x5a\x42\xb5\x5a\x1f\xfd\x40\x37\xc7\xda\xff\x92\x9f\x61\x9d\xb0\x81\x19\x5a\x42\xb9\xd4\x9e\x08\x58\x1c\x93\x69\xca\xe8\x4a\xa5\x71\xca\x5b\x4d\xb4\x8c\xb2\x3a\x54\xeb\x6b\x46\x6f\x02\x2b\xed\xfc\x5f\x0d\x2b\xd8\x8e\x6b\x03\xf4\x03\xfa\xdf\x87\xc1\x4a\x3b\x04\xcd\xc5\xaa\x8f\x7e\x45\x03\xf4\x2b\x41\xec\xfe\x91\xd1\x04\xc0\xb9\xc8\x10\x44\x6a\x7d\xf4\xe5\x9e\x07\x47\xbe\xad\x3e\x76\xa5\x49\x9f\x9b\x78\xbf\x4a\x90\x35\xee\x27\xa6\xb9\x28\xc2\x6a\x30\xc1\x38\x9c\xc5\x1c\xa5\x6f\x1b\xd6\x8c\xad\x4b\x61\xe4\xf2\x6e\xa7\x63\xf1\xfd\x2a\x2f\x6f\x3a\x7c\x15\xf1\xc5\x94\xcb\x7c\x35\x23\xff\xbb\x9d\x8e\xd5\x64\xc0\x39\x09\x73\x72\xd5\xdf\xd7\x14\xdc\x2a\xb4\xc3\xfc\x89\x93\xbd\xfc\xee\x63\xe2\xa8\x53\x99\x98\x88\xfd\x71\x30\x20\x93\xa1\x64\x86\x37\xe7\x83\x15\x33\xe7\xa4\xc8\x66\x4f\xe7\xa5\x34\x03\x3b\x8b\x6c\xed\xb0\x80\x6a\xfe\xa5\x5d\xcc\xfe\xfe\x31\xd9\xe8\x62\x7b\xce\xe2\x0c\xa1\x3d\x8c\xc3\x7e\x30\xf8\xcc\xe2\x6a\x8e\x93\x10\x96\x14\xa1\x19\x31\xdf\xf0\xb2\xb7\xf7\x8a\x88\x40\x16\xf1\x00\xcc\x9c\xe0\xab\x62\x2d\x07\x16\x2e\xb4\x95\x77\x04\x00\x33\xe6\x11\xab\xbe\xb7\xf7\x6a\x65\x37\xa6\xb1\xca\xc1\x80\x6a\xef\x95\xc5\xe0\x67\xe2\x30\x97\x61\x66\x86\x25\x26\x33\x6e\xd1\x94\x85\xa0\xe2\x02\x09\x7d\xb4\xdd\x33\x4b\xa1\x3c\x68\x21\x39\x94\x87\x5a\x9e\xc7\x28\x7f\x83\xaf\xb3\x3c\xc5\xc1\x78\x3b\x0e\x59\xef\x2c\xd6\x91\x09\x33\x8b\x15\xe0\x3c\xd6\x80\x4d\xc8\x3e\xc2\x63\x0c\x41\xc6\xc1\x18\x93\xce\x13\x8b\x95\x09\xfe\xf3\x31\x9e\xe5\xf4\xb5\x5d\x7c\xc7\x97\xaf\x58\xcc\x54\x68\x7d\x25\x1b\x45\x03\x5c\xe3\x28\x88\x9b\x7a\x81\x8b\xcd\x7e\x52\x99\xb5\x1d\xfc\x77\x99\xb5\x3b\x8c\x2e\x18\x0e\x5f\x44\xd9\xc2\x63\xfb\xd5\xe8\xe6\xa4\xe8\x50\x1f\x0f\x92\x31\xf3\xba\x27\x04\x11\x25\xd3\xac\x1a\xc9\x88\x2e\x56\x12\xc7\x4b\x7a\x53\x9b\xdb\x05\xcd\x37\xc2\x3c\xb0\xc1\x79\xef\xb2\x08\xd6\x72\xf9\x42\x35\x1a\x97\xc3\x31\xd3\xe6\x8b\xcf\x90\xd9\xf5\xd2\x7a\xa4\x11\xa5\xd1\x16\x8a\x2e\xd9\x14\x36\x1c\x2b\x31\xb9\xc4\xe8\xe0\x27\x38\x7f\x66\xd3\x7e\x86\xff\x3d\xc5\x71\x5e\x72\x7a\x06\x7c\x85\x03\xc3\x5c\x03\x68\x1d\x1f\x6d\x42\xcc\x49\x20\x7f\x8c\xca\x31\x1d\x68\x28\x58\x13\x40\x3c\xa4\x76\x65\x75\x15\xb1\x19\x29\xde\x59\xb3\xe5\x96\x47\x8d\xa1\xa6\xe7\x85\x85\x20\x44\x82\x11\x8d\xc2\x39\xda\xa0\x17\x86\x05\x17\x27\xf6\x5e\x95\x19\x5c\xf3\x4d\x67\x91\x38\x75\xdd\xd6\xa3\xf0\xf1\xad\x0b\x1f\xe8\xbf\x27\x29\xce\x70\x7a\x89\xa9\x18\x92\x4c\x89\x28\x2f\x89\x1f\xa0\xc6\x08\xf2\xa8\x3f\x62\x1c\x18\xed\xa4\xe8\x55\x1a\x05\x31\x7a\x4d\xdd\x33\xd1\x30\x1a\x61\x1c\x0f\x56\x06\x00\x82\x87\x7c\x86\x08\xd8\x1a\xfd\x9c\x1c\x41\x91\x7f\x06\x31\xda\x4f\xa7\xfd\x6b\xf4\xdb\x05\xf9\xcf\xca\x15\xee\xff\xf7\xf9\x38\x88\x46\x2b\x83\x64\x6c\x97\x77\x4e\x8e\x78\x73\x25\x62\x8f\x5c\xa8\xb2\xf4\xf3\xa4\xc8\xf7\x12\x0f\xc8\x41\x81\xa6\x4c\x7a\xfa\xe4\x09\x19\x74\x20\x3d\x91\x0e\x09\x94\x44\x54\x29\x54\x87\x59\xa7\xbf\xfe\x40\xab\xab\xc9\x25\x4e\x87

\xa3\xe4\x8a\xd4\x81\x8d\xcf\xe7\xe9\x40\x49\x3d\xbf\x5b\xff\x81\x94\x7d\x21\x3e\x37\xe5\xcf\xe
b\xfa\xd7\x16\xdb\xc3\x58\x63\x80\x27\xa0\x42\xc0\x8a\x76\x57\x57\x11\x6f\x16\xf5\x7d\x52\x04\
x50\x86\xa6\x1b\x2f\x44\x95\x66\x51\x45\x94\x79\x02\x08\xd0\x42\xb4\x54\x4b\x2d\xc5\x8a\x3d\
x01\x54\x58\xb9\x1b\xf8\x97\x10\xa4\x5c\x62\x69\xa9\xdf\x92\xbe\xc3\x3f\xbc\x0c\x2d\xb2\xb4\xd
4\x6f\xbe\x78\xea\x2e\xb0\xb4\xd4\xf7\xd9\x77\xf2\x2f\x74\x9c\x37\x0a\x0f\x4b\x5b\xd0\xf3\x97\x
2f\x59\x3e\x48\xf9\x75\x93\xaa\x00\x95\xb7\x0c\x21\xb3\x25\x51\xad\x31\x6b\xf8\x4c\xeb\x57\x1
4\x65\x5c\x8f\x14\x22\x2f\x6f\x74\xea\x60\xcb\xa3\x36\xa0\xff\x55\x69\x84\xbd\xa4\x37\x48\x9c\x
94\x8a\x97\x75\x46\x30\xd2\x14\xac\xae\x22\xb2\x4b\xc0\x4d\x0c\x8a\xa4\x85\x44\x17\x8f\xb1\xd
2\x9e\x65\x08\xe0\x65\x28\x89\x47\xd7\x74\x39\xee\xfc\x7c\x78\xb4\x83\x7e\x43\x2f\xd1\x3a\xc0\
xe4\x0d\xfa\x36\x2c\xe8\x5d\x9c\xda\x59\xf6\x8d\xf7\x97\xaf\x25\xe5\x2c\x20\xd6\xd5\x8a\xe3\xf5
\x9f\x28\x73\x2e\x2a\x72\x1a\xc5\x35\x19\xc6\x6c\x95\xf1\x44\xd1\x2c\x1f\x30\x03\xf5\x32\x89\x0
7\xb9\xa5\x1e\x10\x1a\xec\x8d\x94\xcb\x40\xe8\x16\x72\x10\x9a\x2f\x0b\x71\xe9\x80\x10\xb6\x4
9\xf3\x94\x15\x3d\xd1\x45\x23\xf6\x59\xc2\x55\x55\x3d\x2f\x22\x14\x21\x87\x60\x84\x6e\x27\x1c\
xa1\x05\x05\x24\xa4\xca\x73\xe6\xa1\xab\xa0\x7b\xf9\xec\x25\x96\xc6\x0b\x4d\xb2\x12\xc5\x25\x
01\xcb\x29\x62\x49\x85\x17\x90\xb4\xda\x8f\x92\xd6\xb7\x2e\x69\x39\xe4\x2b\x87\x7a\xe7\xe4\x
a8\x5c\xce\x59\x54\xbd\x63\x61\xe9\x3a\x2f\x7f\x64\xe2\x7f\x3f\x26\x5e\x7a\x9a\x7d\x00\x96\x7d
\x10\x0f\x52\x0c\x91\x1b\x18\x70\x0d\x24\x93\x43\x8a\xc9\x5d\x46\xd4\x98\xc6\xf1\x05\x6e\xcb\x
bf\xa0\xc6\x5f\x6a\x73\xa8\xba\x2b\xcc\x3f\x6f\x93\x32\x0b\xec\x02\x9d\xc7\x5d\xe0\x2f\xb1\x0b\
xec\x8e\xf0\x20\x4f\x93\x38\x1a\xa0\x5e\x12\xe2\x7e\x92\xcc\x57\xf8\xef\xf6\xca\x14\xfe\xf4\xeb\
x42\x3b\xc2\x6e\x4f\x55\xf8\x93\xe7\xfb\xda\x01\x64\xd6\xae\x32\x10\xb5\x5e\x99\x16\x93\xe0\x
a3\x2c\xa4\x87\xc2\x2f\xc4\xb7\xc2\x8f\xa7\x5e\xea\xcd\xd7\x9b\x41\x99\x05\xd6\xf1\x5f\x3b\x39\
xf2\x7f\xce\x3a\x3e\x9c\xe6\x93\x69\x5e\xfd\xd2\xee\xb0\xf4\xd2\xee\x70\xf1\x4b\x3b\x5d\xaa\x3b
\xd4\x2e\xf1\x0e\xff\xdc\xeb\xa0\x07\x97\xea\x4c\xdd\xbc\x78\x73\xbf\x92\x5d\x49\x43\xdf\x8a\x7
4\xf7\x77\x3a\x61\x1f\x6a\xd7\x9a\x2e\x21\xea\xb0\xc2\xa5\xc5\xe1\x82\x97\x16\x8f\x59\xec\xfe\x
1a\xcc\x77\xfb\xfd\xf1\x01\xfa\x65\x65\xa3\xd9\xe2\x06\xe2\x28\xcb\xc9\xf2\x3e\xbf\x36\xb8\xef\x
24\x08\x57\xb6\xe3\x2c\xfa\x85\x94\x16\xb9\xe0\x26\x41\x28\xb3\xbf\x30\xc8\x03\xe9\x22\xd4\x7
5\x01\x9a\xa9\x37\xa0\xa4\xd6\x71\x61\xf0\xab\x18\x00\xbf\x50\x8b\xf6\xf5\xb4\x22\x7d\x57\x42\
x11\x20\x8a\x69\x9c\x8b\x9e\x69\xc1\xac\xc0\x16\xef\x03\xfd\x66\x00\xa3\x2f\x96\x55\xcc\xfe\xa
1\x7d\x37\x5a\xa3\x31\x6d\x46\x41\x46\x23\x67\xa1\x49\x92\x45\xaa\x07\x3e\x69\x94\x7c\x27\xf
5\x3f\x24\xbc\xb3\xa2\x85\x25\x0d\xa3\x65\xe4\x6b\x8d\x7c\x08\xc2\xe2\x19\x06\x4a\x64\x1b\x51
\x5f\x53\x56\x22\xb7\x55\x84\xd4\x52\x1b\x29\x42\x6a\xc9\xa5\x6d\xc1\xb5\x54\xcb\xec\x25\x0d\
x10\xb7\x43\xe4\x16\xb8\xd3\xd8\x42\x1c\x3a\x45\xbc\xc6\xb9\x94\x70\x5e\x99\x2a\xaa\xc0\x17\
xa3\x59\x3e\x73\x52\x9f\x6b\x2a\x9a\xcb\xe4\xf8\xcb\xfa\x5e\x5c\x04\x49\x28\xb0\x7d\xc5\xf0\x9
0\xd0\xc0\x38\x7a\xfb\xf4\xc9\x8d\x95\x6f\xf2\xe5\x32\xdb\x68\xb6\x16\xe2\x9d\x77\x4b\x4c\xf6\x
c8\x3b\xbf\x16\xef\x3c\x38\x3e\x44\x10\x12\xb7\x1a\xeb\x3c\x60\x01\x74\xef\xca\x3a\xff\x74\x76\
x58\x2c\x89\x39\xfc\xd0\xc2\xaa\x68\x3a\x00\x7b\x04\xba\x95\x34\x88\xc3\x64\x5c\x33\x38\x60\x
bd\xbe\xa2\x49\x4a\xe5\x70\x58\xea\xb0\x53\x83\xcb\x35\xdb\x67\x1e\x01\xf7\xc8\xa8\x74\x46\xc
5\x89\x73\x21\x46\xf5\xd7\xce\xbc\xf0\x1f\xc5\xa8\x56\x0f\x76\x7b\x68\x63\x6d\x63\x6d\xd9\x47\x
8c\x36\xd0\x3b\x9c\x5f\x24\x21\x6a\xba\xb8\x15\x84\xf6\xbe\x2d\xb7\xda\x0e\x43\xea\x3f\xa8\x2
e\x88\x0a\x5c\x80\xaf\x5e\x52\x9b\xfe\xf1\x45\xab\x34\xf0\x3f\x38\x4d\x20\x77\x58\x7e\x81\x51\x
8a\x33\x89\x2f\x2a\x1d\x21\xe5\x58\x8f\xc9\xb3\x81\xf7\xad\x78\x01\x5b\x88\xbf\x33\x1c\xd4\xd5
\xe8\x6c\x1e\x40\x53\x78\xf6\x85\x9d\xc4\x18\x8d\x93\x14\x53\xe1\x71\x79\x19\xfa\xe6\x1a\x45\x

be\xde\x97\x97\x2b\x2e\x70\x98\xcf\x45\x16\xf8\xda\xdd\xa2\x9c\x3f\x2e\xf0\xaf\x76\x8a\x43\x71\x92\x4c\xaa\x89\x21\xef\x39\x39\x3a\x57\xb6\x20\x76\xf7\x9a\x28\x8a\x94\xd1\x9c\x68\x6a\x21\xa2\xbb\x5b\xb8\xd9\x47\xa2\xfb\x5a\x44\xf7\x3f\x12\xf3\x2b\x27\x39\x89\x07\xfe\x89\xc2\x6f\xe5\x83\xb3\x7c\xbe\x35\x04\xe0\x5a\xad\x5c\x04\xae\xa3\x2f\x5f\xf4\x57\xb7\xda\x62\xec\x3d\x9e\x1f\x57\x60\x75\x15\x7d\x24\xf0\xd5\x7a\x91\x11\x29\x00\x34\x0b\xa2\xcc\xd5\x45\x34\xc2\xa8\xf6\x5d\xad\xf0\xb5\x2e\x62\x70\x83\xc7\xa1\x11\x73\x5b\x98\x70\x1a\x8a\xcc\x48\x6c\x49\x48\x55\x51\xea\x8e\xdd\x10\x8f\xb7\xca\xee\x25\x51\xd0\x42\xbc\xe4\xaf\xed\xb8\x65\xc9\xd1\x45\x93\x64\x3d\x2c\x5f\x29\x32\x21\x41\x6b\x7f\x7e\x9e\x8f\x87\x4d\x12\x5e\x2d\x26\xb6\x11\xf3\x5a\x7c\x39\xde\xdf\xf6\x8b\x58\xcf\xe4\x49\xfa\x68\x26\x02\xb7\x39\x88\x7e\x08\xb2\x8c\x2c\xe4\x65\x82\x5a\x88\xde\xe0\x6b\xb4\x83\xd3\xe8\x92\xe6\x84\xdc\xe3\x83\xd2\x2c\x8f\x39\xfd\xe1\xd5\x9b\x9d\xbd\x66\xd1\x9a\x78\xae\x98\x78\xbc\x97\xc4\xc3\xe8\x7c\xca\x32\x51\x26\x90\x15\x32\x2b\xcb\x2f\x99\x26\x13\x9c\xe6\xd7\xe8\x0f\x7a\x2c\x06\x6f\x52\x60\xbe\x27\x17\x34\xc7\x71\x46\x1e\xa2\x98\xa5\x0b\xc8\x13\xe1\x4b\xb3\x82\x76\xf0\x30\x98\x8e\xf2\x4d\xd4\x46\x35\xbf\xb9\x0e\x89\x94\xeb\x2e\xf8\x8e\x84\xe6\x38\xe5\x89\xcc\x0b\x70\x64\xfc\xe7\xa1\x19\xe5\x2c\x79\x66\x06\xa0\x8a\x43\xbd\xf4\x21\x4f\xd0\x04\xa7\xc3\x24\x1d\x4b\xc0\x15\xc8\x52\xfa\xc7\xc1\xf0\x7c\xd3\x35\xca\x88\x5e\x7c\x1d\x43\xcc\x19\xbf\xb9\xbe\xda\x6a\x6a\x21\xb8\x69\x57\x28\xea\xda\xa7\x02\x21\xa5\xf1\x9b\x7a\x59\x42\xd2\xb2\x04\xf2\x64\x56\xc2\x82\xb4\xf8\x7a\x9b\x9f\x45\xf4\x10\xf8\xdc\x0d\xe9\xaa\x9c\x31\x94\x8c\xdf\xc0\x46\x37\xdc\xdf\x6c\x98\xa4\x70\x8a\x29\x1a\xbd\x87\xc4\xa0\x9f\xc3\xa1\x91\x34\x9e\x52\x3b\x3f\x3d\x2a\x66\x58\x8b\x54\xfc\xa3\x98\xac\x75\x9a\x7e\xf2\xce\x60\x3c\x75\x1a\x1b\x8d\x86\x0e\xb8\x24\x7b\xfd\x60\x78\x6e\x37\xbc\x20\x13\xb1\x25\x7e\x72\xc2\x23\xc5\x5d\xc1\x30\xcc\xf5\x0e\xd7\x15\xd4\x83\xae\x2a\x0b\xba\x4d\xbe\xd9\x09\x83\x0d\xd4\xc2\x1f\x56\x2a\x56\xce\x82\x51\x8e\xb6\xe1\x3f\x8b\x27\xa2\xe5\x6e\x34\x92\x5f\xfb\x5d\xc8\x8e\x26\x52\x0f\x87\x2b\x2c\x2a\x49\x8d\x77\xc6\x03\xfc\x9c\x93\xca\x8a\xcb\xf3\xaa\xd5\x5c\x28\xb7\x8b\x3a\xf5\x56\x03\xc2\x28\x77\x24\x85\x65\x5e\xf6\xe0\xbb\xcf\x68\x95\x90\x0f\xe5\x41\x9e\x98\x1d\xbb\x59\xa2\x3b\x41\x39\xc8\xa6\x74\xb0\x69\xba\x79\x43\x9f\x63\x0b\xf5\x04\x72\xf2\x41\x1c\xe2\x99\xad\xc6\x69\x63\xc6\x14\x40\x96\x68\x9d\x73\x42\x74\x09\x54\x84\xb0\x2c\xde\x38\xf3\xd7\x17\xd8\xf0\x4a\xc5\x1b\x67\x25\xbe\xe5\x6d\x91\x59\x59\x61\x4f\x36\x23\x8c\x62\x6b\xa1\x45\x8b\x17\x73\x8c\x2c\xd4\x8f\x4c\x50\xd7\x3a\xc8\xe3\x22\xbd\xe4\xf8\x58\x8d\x0b\x44\x27\x59\x9e\x63\x9e\x2c\x1b\x28\xb0\x48\xe3\x5b\xf4\x5a\x9f\x33\xc4\x32\x7a\x17\xa9\x81\xcd\xef\xf3\xb3\x31\x00\x7c\x65\x88\xad\xa3\x6b\x16\x17\x59\x8c\x8a\x57\xac\xe3\x0e\x44\x0e\xc4\x18\xdb\x41\x47\x72\x34\x3b\x06\xd6\x82\x85\x62\xcb\xe1\x53\x5b\x0e\x69\xfa\x9c\xc6\x1c\x08\xf8\xb9\xd2\x04\x8c\x9e\x18\x69\xf9\xa3\x6d\xac\xab\x8c\x37\x9a\x17\x0a\xca\xd6\x59\x3e\xfa\xf2\x3b\x7b\xc0\x2a\xa9\x89\x5f\x0e\x8f\xd4\xee\x80\xeb\x94\xc5\xe3\xda\x18\xb7\xdf\xa8\x0d\xcc\x6f\xdc\x06\x46\x9a\xcd\x17\xe8\xb7\x92\xd1\x23\x7f\x45\x8d\xd3\xdf\xc0\x1c\xc6\xe8\xc8\xe9\x6f\xba\x59\x0c\xff\xbb\x31\x5f\xeb\x01\xa7\xc8\x9f\xc4\x1c\x98\x6e\x1a\x1a\xb5\x4d\x89\xc6\x24\x4e\x1b\x67\x4b\x4b\xe5\x26\x45\x12\x70\xe9\xe8\xcb\xf9\x86\x25\x88\x19\xdb\xcb\x8a\x7a\x65\x06\x94\xf2\x31\xe2\x4e\x1b\x7a\x95\x60\x33\xa5\x1b\xf9\x82\x9b\xf8\x7d\x89\x96\x51\x66\x4b\xb7\x3f\x3f\x7a\x8d\x45\x34\xb8\x87\x20\x36\x54\x44\x10\x92\x21\x15\x0a\x5d\x62\xc2\x62\xd5\x3c\xe4\x90\x4d\xef\x02\xa6\x54\x36\x2d\x82\xec\x88\xa3\xa4\x4b\x80\xf1\x90\x2e\xa8\xb2\x61\x57\xc5\x62\x52\x68\x8e\xf0\x74\x53\x66\x8b\x46\xa1\xd9\x03\xf5\xe8\x29\x74\x79\x4e\xd8\x9b\x33\x6f\xed\xaf\xed\x43\xbf\x40\x5a\xf7\xf9\xc9\xd1\x1f\x56\x77\xe4\x4c\xaf\xed\xca\x7a\xfd\x77\xd0\x2e\x1d\x83\x71\x66\x8f\x1b\xef\x52\x25\x92\xfc\xb2\x4c\x8f\x24\xf0\x38\xc2\xd3\x2c\xe8

\x8f\x30\x0b\x07\x26\xa1\x73\x8c\xe4\x54\x8b\x14\x8a\xfe\xe6\x35\x52\x33\xac\x49\xdb\xc2\x11\x64\x53\x46\xcc\xd0\x96\xd9\x18\x9b\x9a\x24\x51\x1e\x62\xac\x44\x19\x0a\x10\x4d\xc0\x8c\x2e\x71\x9a\x41\xd4\xb2\x8b\x20\x47\x31\x3e\x1f\xe1\x41\x8e\x43\xc2\x86\x07\x2c\xa5\x6a\xce\x14\x3e\x79\x82\x46\x51\x9e\x8f\xf0\x32\x0d\x70\xb9\xa2\x02\xc5\x69\x9a\xa4\x28\x4c\x70\x16\x3f\xcb\x51\x30\x1c\xe2\x01\xad\x4b\x91\x7a\x96\xa1\x0c\x0f\xa6\x69\x94\x5f\x7b\xa2\x62\x7f\x9a\xa3\x28\x87\x4a\xbc\x46\x94\x67\x22\xa0\x42\x34\x8a\x72\xe6\xc4\x4d\xf3\xba\x46\x84\x3f\x8f\x71\x4c\xf7\x83\xcc\xa6\x28\xa3\x03\xf2\x96\x76\x4e\xa8\xcb\xb4\xb7\xf2\xfc\xdd\x36\x69\x5b\xf9\x21\xe5\x8d\x6c\x06\xed\x3c\x60\x14\xd6\xdb\x70\x6a\xb8\x2c\x3b\x2d\x44\xec\x84\x46\x76\x2f\xec\x3c\xa7\xfd\x22\xda\x25\xbf\x2c\x89\xe3\xde\x9c\x36\xce\x3c\x54\x7b\x73\xda\x3a\x63\xc1\x02\xd0\x17\xf2\xc8\xae\x02\xfc\x6e\xdd\x92\x44\xee\xcd\xa9\x4f\x2b\x35\xd4\x4a\xad\xf2\x4a\x4d\x5a\xc9\x57\x2b\x35\xca\x2b\xb5\x68\xa5\xa6\x5a\xc9\x17\x95\xd4\x3a\xb6\xec\x48\xc6\x90\x71\x2f\x43\xd7\xa0\xf5\xc4\xa0\xf5\xec\x83\x66\xe2\x23\x0d\x17\xeb\x13\xbd\x30\x19\x0e\x79\xda\x41\x8a\x34\x0d\xb2\xda\x68\x90\x2f\xb6\xfe\x9a\x13\xd1\x52\x21\xfb\x56\xc8\xcd\x4a\x90\x1b\xce\x81\x97\x60\x68\x90\x5b\x95\x20\xfb\xae\xd9\xf1\x24\x18\x1a\xe4\x86\x06\x79\xfe\x44\xf6\x82\x34\xbd\x46\x7d\x3d\x9d\x2a\x9d\xaa\x3e\x8d\x7f\x61\x6a\x32\x72\x3a\xf9\x84\xf5\x64\xd7\x59\x8e\xc7\x68\x98\x4c\x53\x94\x47\x63\x7d\xee\x17\x0c\xca\x1b\xe3\x59\x7e\x4c\x56\x9f\x3b\x7e\xac\x25\xe2\xed\xbb\x24\x8c\x86\xd7\x94\x13\x52\x3a\xac\x80\xc5\xba\x1b\x8b\xde\x29\x75\x1c\xf8\xe5\x14\x52\x5e\x42\xb4\x15\x23\x53\x9c\x2d\x49\xee\x4f\x28\xc3\xf9\x74\xa2\x7e\x28\xf1\xe8\x98\x7f\xd8\x3f\xf8\x89\xba\x76\x94\x9d\xf0\x0f\x7e\xfa\xd4\x40\x5b\xe8\xe0\x27\x33\x35\x9a\x54\xc4\xa7\x45\x7c\x6b\x34\x63\x79\x49\xc3\x54\x66\xd3\xfe\x25\x26\xa2\x82\xeb\xe8\xdf\xa0\xc1\x8f\xa1\x6d\x1a\xfd\xf8\x0b\xa2\x4f\xae\xe8\xc7\x72\x71\x16\xe6\x58\x94\x2f\xae\x43\xed\x61\x8e\x45\xb3\x4d\xd1\xac\xaf\x34\xeb\xcf\x6b\xd6\x57\x9b\xf5\x17\x6b\x16\xc2\xe8\x44\x0d\xbe\x04\x09\x90\xa8\xa9\xae\x40\x57\xd5\x16\x54\x6d\xf2\xc5\x0c\x55\x1b\xea\x32\x75\xcc\x08\x23\xeb\x32\xd6\x8a\x80\x5a\x1b\xf4\x5c\xaf\xc7\xf6\xa7\x1f\x7d\xfa\xd1\xb7\x7e\x6c\xd2\x8f\x4d\xeb\xc7\x16\xfd\xd8\xb2\x7e\x6c\x97\xb5\xd9\x29\x6b\xb3\x5b\xd6\xe6\x9a\x68\xb3\x44\x23\x55\x89\xf3\xa0\xc5\xb9\x0f\xaa\xc6\x81\x90\xa9\xa4\x90\xfd\x88\xee\x25\xb9\xab\x53\x79\x2d\x49\x1f\x95\x38\xb3\x5a\xc4\xde\x3b\xf7\xf6\x0e\x83\x5b\x78\x99\x01\x17\x52\x4b\x1f\xd3\x50\x43\xbf\x00\x11\xa2\xda\x2f\x64\xee\xf9\x2a\x81\x67\xb1\xf7\xbe\xd0\x2b\xfa\xb4\x62\x93\x55\x5c\xd3\x2a\x76\x9c\x15\x9b\xb4\x62\x9b\x55\xf4\xb5\x8a\x6b\xce\x8a\x2d\x5a\xb1\x7b\x26\x50\x53\x2a\xfa\x45\xc5\x3b\xed\x62\x65\x51\xea\x29\x22\x3c\x76\xfc\x31\x4b\xc9\xce\x82\xc7\xc3\xe3\x6d\xa2\xc7\x73\x38\x8c\xc1\x09\x38\xb6\xf8\xf1\x56\x7c\xad\x4e\x78\x48\xca\xd1\x2b\xbc\xe9\x8e\xcb\xbd\xe8\x64\xea\x17\x76\x3c\xc5\xcd\x6d\xf1\x31\xba\xa4\x5f\xba\xed\xd5\x56\x53\x57\xcb\x89\x65\x22\x08\xb6\x56\xd1\x15\x4a\x59\x1f\xca\x17\x49\x04\xd5\x0c\x7e\x8e\x83\x4b\x8c\x92\x51\xe8\x64\xb5\x0b\xc8\x0f\xbd\x4f\x74\x72\x7b\x7a\xbc\x43\xa5\xc5\x5e\x30\x1a\x4c\x47\x64\x85\xc5\xf8\xca\xd9\x6c\x8f\x25\x82\xe9\xd1\x44\x30\x8d\x59\x3b\x6c\xc1\xff\xa1\x25\x2e\xa1\xe9\xf9\x5a\x7a\x2c\x2f\x4c\x8f\xe6\x85\x69\xcc\x58\x8d\x16\xc4\x94\xef\x71\x01\xb5\x51\x47\x2f\x51\xad\xf7\x49\x7a\xfe\x2f\xe4\xa3\x4d\xd4\xa8\x9b\x10\x9b\x0c\x62\x93\x42\x64\x00\xdb\x0c\xa2\xaf\x41\xf4\x2b\x40\x6c\x31\x88\x2d\xa3\x5b\x35\xda\x8e\x02\xb1\x59\x01\x62\x9b\x41\x6c\x5b\x7b\xdd\xd2\x20\xb6\x2a\x40\xec\x30\x88\x1d\x6b\xaf\xdb\x1a\xc4\x76\x05\x88\x5d\x06\xb1\x6b\xed\x75\x47\x83\xd8\xa9\x00\x71\x8d\x41\x5c\xb3\xf6\xba\xab\x41\xec\xce\x85\x58\x88\xfd\x14\xa8\x52\x7d\x4d\xaf\xae\x7b\xc7\x08\x9a\x26\xbb\xcf\xf9\xf2\x1d\x16\x11\x29\x75\x3e\x03\x5e\x1d\x91\xae\xf5\x2c\x49\x38\x78\xba\xfc\x74\x3a\xc8\xd1\x45\x74\x7e\x81\x82\x38\x44\xa3\xe4\x0a\x05\xe9\xf9\x14\xc2\xbf\x80\x9b\xf3\x

bf\xa7\x41\x6a\x24\xee\x81\x06\x02\xb4\x45\x5a\xe1\x52\x9c\x45\x79\x70\xde\xa7\x45\xe8\x2e\x61\x3d\x3e\xf1\x3e\x2b\x18\xa4\x38\x9b\x8e\x72\x94\x0c\xcb\x9a\xbf\xa0\x5b\x40\xed\x3c\x40\xcf\xd1\x79\x40\x5d\x57\xfc\xb5\x3a\x5a\x42\xf4\x55\x9f\xbd\xea\xc0\xab\x3e\xbc\xb2\x21\x39\xa2\x80\xa4\xae\xd0\x23\xe1\x73\x74\x3e\x83\x19\xae\x03\x41\xf0\x02\x42\xec\x94\x0a\xd8\x12\xc1\x90\x0e\xfd\x72\x78\x84\x20\x9c\xa4\xfc\xf1\x35\xe5\x70\xe7\x17\xe8\x57\x74\x3e\xaa\xca\xe4\xec\x4a\x95\x5f\x18\x8b\x7b\x4d\x59\x5c\xad\xf6\xba\xd8\xbe\xc9\x4e\xf6\x5a\x12\x0b\xea\xac\x40\x57\x2d\xd0\x2d\x0a\xe8\xf4\xfc\x0b\xe3\x86\xaf\x29\x37\xac\xd1\x66\x8a\xfd\xf6\x35\xe7\x7f\xb0\xdf\x2e\x21\xd2\x9a\x09\xa3\xc9\x60\x34\x39\x0c\x5f\x45\xc0\x37\x30\x6c\xa8\x05\x1a\x65\x18\xb6\x18\xf4\x16\x87\xde\x54\x31\x6c\x6a\x18\xfa\x16\x0c\xdb\x0c\x46\x9b\xc3\x68\xa9\x08\xb4\x0c\x0c\x9b\x6a\x81\x66\x19\x86\x1d\x06\xbd\xc3\xa1\xb7\x55\x0c\xdb\x1a\x86\x2d\x0b\x86\x5d\x06\xa3\xcb\x61\x74\x54\x04\x3a\x06\x86\x6d\xb5\x40\xbb\x0c\xc3\x35\x06\x7d\xed\x4c\x21\x11\x81\x61\x57\xc3\xb0\xa3\x60\x58\x29\xf1\x47\xc6\x93\x4e\x08\x5d\x6b\x85\xb4\x13\xf3\xae\xbb\x28\xac\x1c\xcf\x72\xf9\xde\x49\xd6\xa4\xf2\x50\x0a\x4a\x1a\x07\x7a\x5b\x64\xde\x5f\x4d\x46\x01\xc1\x66\x96\x23\x27\x38\x16\x67\xa6\x56\xb4\x6c\x83\x28\x2e\xae\xca\x94\xba\x6a\xf2\x0e\xb9\x64\xbd\xec\x0e\x4a\x2e\x58\xd9\x18\xd9\x53\xef\x46\x36\x3b\x6d\xaf\xb8\x14\xd9\xec\x74\x3d\x76\x57\xb2\xd9\xf5\x6f\xce\xbc\xb5\xbf\x76\x24\xc2\xc7\xfb\xaa\xc7\xfb\xaa\x07\xbb\xaf\xd2\x96\x78\x71\x9f\xa3\xdf\xe4\xfc\xb5\xee\x70\xee\x2b\x2b\xdc\x1b\x71\x34\x7f\xa3\x1e\xcd\xdf\xdc\xf6\x68\xfe\x46\x3d\x9a\xbf\x29\x3b\x9a\xcf\x53\x30\x3f\xde\x54\x3d\xde\x54\x3d\xde\x54\x29\x5f\x1e\x6f\xaa\x1e\x6f\xaa\x1e\x6f\xaa\x8a\x66\x1f\x6f\xaa\xf4\x8f\x8f\x37\x55\x8e\xc7\xc7\x9b\xaa\xc7\x9b\xaa\xc7\x9b\x2a\xf8\x7b\xbc\xa9\xaa\xa6\xc4\x7d\xbc\xa9\x7a\xbc\xa9\x7a\xbc\xa9\x92\xfe\x1e\x6f\xaa\x1e\x6f\xaa\x1e\x6f\xaa\x1e\x6f\xaa\xfe\x93\x6f\xaa\xee\xed\x8e\xea\x76\xb7\x53\x55\xee\xa5\x2a\xdc\x48\x3d\xd4\x5d\xd4\x5f\x3b\x1f\xca\xe3\x5d\xd4\xdf\xff\x2e\x4a\xbe\x3b\xea\xb5\xe7\x3a\x3a\xc9\x37\x47\xbd\xb6\x74\x6d\x04\x0f\x0f\x7f\x67\x44\xbd\x34\xc5\xad\x91\x3d\xa8\x00\xf7\xd0\x2e\xbb\x56\x02\x37\x4e\xd9\xa3\x58\x8a\x99\x6e\xea\x2b\xe2\x28\x47\x59\x3f\x99\x99\x70\x8e\x05\x3a\xc7\xf2\x35\x1d\xff\xb3\x49\x93\xcd\x4e\xd7\x7d\x28\x67\x87\xee\x68\xbe\x1a\xf7\x0d\xbe\xb6\xe9\x71\xd5\x16\x3d\xee\x3f\x3e\xb7\x61\x36\x28\x64\x08\x78\x54\x89\x08\xfd\x43\x1e\x27\x87\xea\x90\x55\x22\x5b\x1b\x1f\xfb\x53\x05\x90\x19\x09\x4d\xf9\x6c\x04\x45\xb3\x9d\xfd\x49\x2f\x6a\xbf\xa1\x25\x3a\x3e\x4b\xbc\xd1\x3a\xfa\x07\xf4\xca\x11\x4b\xe1\x2a\x98\xd8\x71\x86\x7d\xc3\xd4\x10\x48\x13\x70\x6c\x77\x8c\x27\xaf\xc9\x8c\xcf\x9f\x9e\x9e\x55\xc5\xcf\xb2\x6a\x08\xa2\xf9\x8d\x65\x99\x15\x80\xee\xac\x96\xe3\x9a\x10\xd0\x82\x18\xf9\xd7\xc9\xf4\xd8\x55\x86\x4a\xcb\xc2\xc9\xb9\xd9\xe9\x3a\x14\x22\x0d\xa7\x32\xc4\xda\x68\x55\xc5\x88\xb4\x9e\x34\xc5\x48\x31\x68\x91\xf6\xe5\xb7\x62\x38\xe7\x66\x80\x07\xe5\xa0\x5a\xfd\xb3\x8c\xa7\x36\x1f\x62\x35\xc5\x74\x19\xc5\x54\xa5\x16\x5b\x16\x51\x04\x1a\x74\x9a\x30\x8e\x51\xa5\xf2\x5d\x21\x61\x07\xe1\x5a\x89\xb6\x84\x60\xdd\xc4\x5a\x10\xaa\xfa\x5e\xed\xec\x17\x52\xb7\xc6\xd6\x14\xa9\xc2\xf0\x3a\x2b\xf2\x1a\xc4\x7a\x1e\x03\xed\xf8\xf4\x11\xe2\xa0\x58\x6e\xb4\x0a\x52\x8f\x8c\xb3\x3b\x19\x0b\x65\xae\x98\x58\xa6\x60\xf7\xad\xca\xbd\xbd\xf6\x7d\x08\xbd\xbd\xf6\xc2\x12\xaf\xb9\xc7\x6a\xe2\x6e\xaf\x6d\x8d\x6d\x01\x37\x34\x11\x0e\x6f\xb1\xc3\xef\xa4\xc9\x44\xd9\xe5\xd9\x0b\x18\x84\xaf\x10\x15\x2f\x24\xcd\xa9\x81\xe6\x34\x3d\x3f\x99\x78\x52\x4a\x84\x9a\x43\xfe\x46\x53\x06\xab\xc7\x9a\x23\xa8\x4b\x51\xbf\xb4\x55\x4c\x40\x6d\xaa\x20\xd4\x88\x71\x95\x84\x18\xd2\x06\x2f\x58\x7e\x87\x41\xc6\xb3\x64\x03\x17\x86\x2f\x04\x2f\xb2\x8b\xff\x08\x9b\xf9\xf2\xb2\x75\x0f\x5f\x80\xdd\xa3\x39\x09\x90\xbe\xa1\xd5\x46\x86\xe8\x7e\x56\x1c\x40\x5a\x7c\xd5\x31\x9a\x2f\x5f\x79\xa4\x50\xf9\x49\xb3\xd7\x7e\xa8\x63\xe6\xdd\xd2\xf5\x7d\xcd\xf3\xe5\x83\x9d\x02\

xbf\x6e\x10\x67\xc2\xaa\x70\x86\xd3\x4b\xfc\xf4\x49\x6d\x50\x47\xcd\x86\xdf\x44\xfd\x6b\xd4\xfb\
xff\xfe\xdf\x30\x8d\x06\xe8\x1d\xce\xe2\x68\xb4\x82\xb6\x47\x23\x94\x46\xe7\x17\x79\x86\x58\xf
9\x70\xe5\xe9\xd3\x27\x47\x38\x8c\xb2\x3c\x8d\xfa\x53\x80\x1f\xc4\x21\x04\xe5\x89\x62\x94\x25\
xd3\x74\x80\xe1\x4d\x3f\x8a\x83\xf4\x9a\xb0\x83\x71\xe6\xb1\x28\x0d\x29\xfc\x37\x99\xe6\x68\x
0c\x3c\x7d\x00\x9c\xd5\x43\x41\x8a\xd1\x04\xa7\xe3\x28\xcf\x71\x88\x26\x69\x72\x19\x85\x38\xa
4\x41\x27\xc8\x3a\x1d\x26\xa3\x51\x72\x15\xc5\xe7\x68\x90\xc4\x61\x44\xd7\x30\xa9\x34\xc6\xf9
\x26\x5b\xf1\xcb\x48\x45\x2b\x03\xc5\x30\xc5\x67\x90\x84\x18\x8d\xa7\x59\x4e\x36\xea\x20\x8a\
x01\x68\xd0\x4f\x2e\xc9\xa7\xc9\x35\x74\x11\xc5\x49\x1e\x0d\xb0\x47\xe3\x0a\x8d\xa2\x0c\x34\x
cb\x72\x7b\x71\xa8\x21\x13\x46\xd9\x60\x14\x44\x63\x9c\xae\xb8\x70\x88\x62\x79\x20\x38\x0e\x
93\x34\x09\xa7\x03\x7c\xef\x68\x20\xd6\xb5\x30\x19\x4c\x45\x1c\x0c\x52\x63\x35\x49\x59\x8c\x8
c\x71\x90\xe3\x34\x0a\x46\x59\x31\xcc\x30\x37\x50\x4d\x42\x9d\xcc\xf3\xc9\xfe\xc1\x31\x3a\x3e\
xdc\x3b\xf9\x79\xfb\x68\x17\x1d\x1c\xa3\x0f\x47\x87\x3f\x1d\xec\xec\xee\xa0\x57\xff\x42\x27\xfb\
xbb\xa8\x77\xf8\xe1\x5f\x47\x07\xaf\xf7\x4f\xd0\xfe\xe1\xdb\x9d\xdd\xa3\x63\xb4\xfd\x7e\x07\xf5\
x0e\xdf\x9f\x1c\x1d\xbc\xfa\x78\x72\x78\x74\x8c\xbe\xdf\x3e\x46\x07\xc7\xdf\xc3\x87\xed\xf7\xff\x
42\xbb\xbf\x7c\x38\xda\x3d\x3e\x46\x87\x47\xe8\xe0\xdd\x87\xb7\x07\xbb\x3b\xe8\xe7\xed\xa3\x
a3\xed\xf7\x27\x07\xbb\xc7\x1e\x3a\x78\xdf\x7b\xfb\x71\xe7\xe0\xfd\x6b\x0f\xbd\xfa\x78\x82\xde\
x1f\x9e\xa0\xb7\x07\xef\x0e\x4e\x76\x77\xd0\xc9\xa1\x07\x8d\x9a\xd5\xd0\xe1\x1e\x7a\xb7\x7b\x
d4\xdb\xdf\x7e\x7f\xb2\xfd\xea\xe0\xed\xc1\xc9\xbf\xa0\xbd\xbd\x83\x93\xf7\xa4\xad\xbd\xc3\x23\
xb4\x8d\x3e\x6c\x1f\x9d\x1c\xf4\x3e\xbe\xdd\x3e\x42\x1f\x3e\x1e\x7d\x38\x3c\xde\x45\xa4\x5b\x3
b\x07\xc7\xbd\xb7\xdb\x07\xef\x76\x77\x56\xd0\xc1\x7b\xf4\xfe\x10\xed\xfe\xb4\xfb\xfe\x04\x1d\xe
f\x6f\xbf\x7d\x6b\xed\x25\xc1\x5d\xe9\xe3\xab\x5d\xf4\xf6\x60\xfb\xd5\xdb\x5d\xda\xd2\xfb\x7f\xa
1\x9d\x83\xa3\xdd\xde\x09\xe9\x4e\xf1\xab\x77\xb0\xb3\xfb\xfe\x64\xfb\xad\x87\x8e\x3f\xec\xf6\x
0e\xc8\x8f\xdd\x5f\x76\xdf\x7d\x78\xbb\x7d\xf4\x2f\x8f\xc1\x3c\xde\xfd\xbf\x1f\x77\xdf\x9f\x1c\x6c\
xbf\x45\x3b\xdb\xef\xb6\x5f\xef\x1e\xa3\xda\x9c\x21\xf9\x70\x74\xd8\xfb\x78\xb4\xfb\x8e\xe0\x7c\
xb8\x87\x8e\x3f\xbe\x3a\x3e\x39\x38\xf9\x78\xb2\x8b\x5e\x1f\x1e\xee\xc0\x40\x1f\xef\x1e\xfd\x74
\xd0\xdb\x3d\x7e\x81\xde\x1e\x1e\xc3\x68\x7d\x3c\xde\xf5\xd0\xce\xf6\xc9\x36\x34\xfc\xe1\xe8\x
70\xef\xe0\xe4\xf8\x05\xf9\xfd\xea\xe3\xf1\x01\x0c\xda\xc1\xfb\x93\xdd\xa3\xa3\x8f\x1f\x4e\x0e\x0
e\xdf\xd7\xd1\xfe\xe1\xcf\xbb\x3f\xed\x1e\xa1\xde\xf6\xc7\xe3\xdd\x1d\x18\xdd\xc3\xf7\xd0\xd5\x9
3\xfd\xdd\xc3\xa3\x7f\x11\xa0\x64\x0c\x60\xf0\x3d\xf4\xf3\xfe\xee\xc9\xfe\xee\x11\x19\x50\x18\xa
9\x6d\x32\x04\xc7\x27\x47\x07\xbd\x13\xb9\xd8\xe1\x11\x3a\x39\x3c\x3a\x91\xfa\x88\xde\xef\xbe
\x7e\x7b\xf0\x7a\xf7\x7d\x6f\x97\x7c\x3d\x24\x50\x7e\x3e\x38\xde\xad\xa3\xed\xa3\x83\x63\x52\x
e0\x80\x36\xfb\xf3\xf6\xbf\xd0\xe1\x47\xe8\x32\x99\xa3\x8f\xc7\xbb\xf4\xa7\x44\xb1\x1e\xcc\x24\x
3a\xd8\x43\xdb\x3b\x3f\x1d\x10\xb4\x59\xe1\x0f\x87\xc7\xc7\x07\x8c\x4e\x60\xc8\x7a\xfb\x6c\xb8
\x57\x9e\x3e\x79\xbe\xaa\xea\xbc\xde\x05\xf9\xc5\xfd\xea\xbd\xaa\x45\x9d\xa6\x81\x8f\x45\x11\xf
a\x58\xc9\x3a\x1b\x2e\xec\x82\x38\xcf\x50\x1e\xf4\xb9\xc4\x42\xaa\x7c\xfa\x7d\x64\x0d\xb6\x59\
xc8\x51\x0d\x0f\x21\xdf\x43\xa8\xe9\x21\xd4\xf2\x10\x6a\x7b\x08\x75\x3c\x84\xba\x1e\x42\x6b\x1
e\x42\xeb\x1e\x42\x1b\x1e\xf2\x1b\x1e\xf2\x7d\x0f\xf9\x4d\x0f\xf9\x2d\x0f\xf9\x6d\x0f\xf9\x1d\xc9\
xc2\x72\x8d\xd6\x25\xdf\x08\x3c\x52\x9e\xc0\xf0\x3b\x14\x2e\xa9\x07\x6d\x6d\x30\xf8\x4d\x06\xc
3\x87\x36\x0a\x38\x2d\xd6\x56\x9b\xe1\xb2\xc1\x60\xac\x4b\x78\xae\x31\x58\x5d\x86\x8b\x4f\x6
1\xfa\x72\xac\x65\x9f\xd5\xe5\xb8\x34\x28\x0c\xc0\x83\xe3\xd9\xa2\xb0\x08\x7c\x5f\xee\xb7\x0c\
xa7\xcd\xea\x76\x18\xee\x6b\x0c\x46\x53\xc2\xd3\x67\xb0\xd6\x19\x2e\xac\xdf\x7e\xeb\xac\xfe\x
42\x9e\x8b\x74\xce\x5c\x70\x3c\xd6\xa4\xb1\x6a\x32\x98\x1c\xe7\xae\x3a\x1e\xd0\xb7\x96\xd6\xf

7\x2e\xab\xd3\x2a\x60\x41\xdd\x4e\x81\x33\x87\xc1\xc7\x03\xda\xf2\xb5\xbe\x43\xa1\x8e\xd4\xc1\x35\x86\x60\xb7\x18\x5c\x01\xa4\x29\x0d\x34\x45\xb6\x00\xb4\xce\xea\x48\x83\x05\x13\xd3\x29\x06\x57\xc0\x68\x49\x03\x4d\x91\x95\x10\x6a\xb2\x91\x6d\x48\xc0\xf8\x68\xac\x89\xd9\x13\x14\x8a\xd8\xe8\x50\x64\xd5\xd9\xc8\xe6\xad\x0c\x8a\x22\x1b\x2b\x40\x4f\x6e\x89\xd3\x56\x4b\x1a\xcf\x6e\xf1\x4d\xa1\xe9\x35\x0f\x3e\xc1\x50\x71\x7a\xdd\x28\x68\x8f\xd3\x94\xdf\x91\x86\x75\x8d\x95\x55\xe6\xc3\x2f\x88\x40\xcc\xc5\x06\x2b\xc8\x89\x67\x5d\x2a\xc3\x11\x5f\x83\xdf\xf2\x59\x4a\xac\xe5\x76\x51\x95\xb7\x2f\xd6\xbc\xbc\x26\xd6\x15\x90\x05\x28\xbe\x3e\x3b\x05\xed\x8b\x7e\x36\x0b\x14\xc4\x38\x31\x92\xa1\x70\x91\x36\x25\xf3\x16\x08\x43\x4c\x19\xfc\x4e\x81\x00\xf4\x73\xad\x58\x88\xd0\x60\x9b\x21\xd2\xd5\x90\x6e\xa9\x83\x2f\x3a\xed\x17\x70\xc4\xd8\x89\x05\x0d\xdf\x15\x38\x82\x81\xf8\xd2\x20\x75\x8b\x76\xc5\xc2\x63\x0b\xd8\x6f\x59\xe6\x43\x74\x40\x43\x9c\x03\x12\x0b\xae\x29\xfd\xb7\x23\x56\xb1\x3a\x40\x1d\x4b\xb9\xb6\x3a\x33\x62\x26\x8b\x4e\x21\xdf\x47\x67\x4a\x96\xec\x4f\x17\x64\x85\x58\xe6\x03\x89\x50\xcd\x0d\x0f\x35\x66\x9d\xed\xf5\xe6\xda\xc6\xc6\x06\xf9\xdd\xdd\xdd\xd9\xd8\x7d\xb5\xed\x93\xdf\xeb\x7b\xfe\xab\x57\xbd\x9d\x1e\xf9\xbd\xbd\xd1\x69\xed\xed\xb4\x77\xd5\xf9\xbe\x48\x9d\x0d\x74\x1a\xdb\xcd\xf5\x57\xbb\x5d\x68\xa0\xd7\xde\xd9\xf1\x9b\x6d\x68\x60\x67\xad\xd1\xda\xdd\x6b\x91\xdf\x6b\xdb\xdd\x9d\xb5\xee\x2e\x34\xcc\x11\x3a\xb3\xea\x03\x8e\x0e\x3e\xec\xbe\xdb\xf1\xbb\x0d\x08\xbf\x3f\x47\x87\x24\xca\x16\x5a\x24\xe9\x15\xdd\x95\x6f\x7b\x57\x44\x95\x89\x80\x84\x23\x08\x76\x77\xad\xdd\x69\xb6\x1a\x30\x82\xbb\x7b\xbd\x9d\xed\x57\xeb\xd0\xc1\x8d\xf5\x57\xdb\x3b\xbd\xbd\x5d\xf2\xdb\x6f\xb4\x9a\x9d\xf6\x1a\x0c\x4e\xaf\xb5\xd3\xdc\xf5\xf7\x1a\x67\x4e\xd5\x78\x55\xa5\xbc\x55\xb1\x5b\xd9\x4b\xc9\x2f\xb9\xa9\x99\x6f\x8e\x4f\xb1\x00\xdd\x6b\x61\x16\xe9\xb8\xbe\x79\xf7\x49\x2a\xcd\x2f\x0f\x3e\x99\x86\x4c\xa8\xec\x4e\x45\xaa\x87\xb6\x50\xcd\x2c\x80\xa8\x01\xa8\xd4\x58\x61\xf8\x20\xbd\x5c\xcc\xa8\xd4\x00\xc8\xec\x4a\x35\x80\xa6\x75\xa9\x09\xae\x44\x35\x86\xe6\xd9\x3a\xef\x23\x71\xff\x40\x48\xd1\x79\xe5\x08\x0c\xe0\xd3\xc5\xc8\x5d\x20\x85\x02\xa9\xb3\x00\x88\x9f\x9f\x7e\x77\x43\x00\x99\xe8\xd3\xef\x6e\x08\xb0\x4d\x7f\xca\xdc\x10\x60\xd3\xf8\x94\xa5\xf6\x88\xd6\xab\xab\x64\x95\x7d\x26\x87\xe6\xcb\x20\x8d\x88\x74\x6c\xb9\xa4\x0d\x46\x1e\xea\x8f\x3c\x34\x18\x79\x28\x1c\x79\x08\x8f\x2c\x0d\x05\xa9\x87\xfa\xa9\x87\x06\xa9\x87\xc2\xd4\x43\x38\xd5\x1b\x0b\x08\x2a\x01\x41\x78\xdf\x74\x19\xe9\xa7\x10\x74\x1c\x3e\xfa\xfa\xc7\x01\xf9\x38\xa0\x1f\x9b\xfa\xc7\x90\x7c\x0c\xe9\xc7\x96\xfe\x11\x0e\x0c\x98\x7e\x6c\xeb\x1f\x45\x9a\xea\x40\xcd\x4b\xcd\xbb\xa4\xdf\x0a\x5a\x4d\x09\xe1\xbf\x4b\x5b\xc8\xb7\xae\xed\x9c\x2c\x9f\x60\x84\x96\x8a\x35\xb5\xf4\xfb\xe8\x34\x3a\x3b\xab\x7f\xb1\x39\x31\x80\xd7\xce\x4b\xbf\x5b\xff\xe3\xe9\x13\x95\x35\x92\x36\xd0\xd0\xaf\xf5\x47\xde\x60\xe4\x85\xa3\x3a\x5a\x42\x17\x23\xbb\xef\xcd\x0d\x12\x0a\xb9\xe8\x65\xab\x49\x55\x6d\x16\x68\x4d\x1d\x9a\x31\xf2\x06\xb4\xf6\xba\x13\x5a\x4b\x87\x66\x4c\x95\x01\xad\xdb\x76\x42\x6b\xeb\xd0\x8c\xb9\x95\xa0\xfd\xb1\xba\xca\x20\xae\x37\x9c\x10\x3b\x3a\x44\x83\x20\x90\x3d\x4c\x3a\x99\xc4\xdc\x3a\x5d\xe4\x0b\x4a\x93\x7c\x54\xcb\xbd\x8c\x4c\xab\xcd\x69\x03\x68\x20\x5f\xc2\x23\xfb\x94\xc3\x8a\x30\x96\x14\xf9\x03\xba\x0d\x6d\x5f\x80\xdc\xa1\x5d\xb2\x26\x7d\xab\x1b\x10\xac\x97\xbe\xad\x36\x2c\x33\xe3\x26\x51\xa0\x1a\xa4\x68\x49\xa2\xd6\xf4\xf6\xd4\xda\xa9\xf5\x53\x6f\x90\x7a\x61\x0a\x23\x9e\xde\x8d\x5a\xdb\x3a\xb4\xbb\x52\xab\x0a\xed\x4e\xd4\xda\xd4\xa1\xdd\x99\x5a\x7d\x1d\xe2\x3d\x53\x6b\x0a\xb7\xd6\x25\xe4\x9a\x3a\xc8\x15\x38\x6a\x6a\x23\x57\x60\xc4\xb6\x2f\xc0\xa2\x29\xb9\xa6\x4e\x72\x85\x0d\xc0\x56\x1b\xb6\x06\xd3\x42\x43\x67\xe5\x07\x72\x3a\x06\x90\x21\xc1\xea\x57\x93\x30\xc9\x3f\x5b\xa8\xb6\x4f\x4d\x73\x07\x84\x33\x87\x96\x9e\xee\x33\x13\xde\x7d\x6a\x7e\x1b\x92\x72\xb6\x11\xd9\x67\x66\xba\xfb\xd4\x90\x16\x93\x72\x81\xb5\x5c\x8b\x95\x03\x63\x59\xd

8\x11\xfa\xd6\x72\x6d\x56\x0e\x0c\x93\xfb\xa4\xdc\xc0\x5a\x0e\x0c\x98\x95\x61\xd1\xc5\xda\x3d\x96\x5a\xe3\x0e\xe6\x59\x61\x90\x07\x42\x18\x22\x0f\x96\x8d\x7f\x7e\x1a\x46\x5e\x32\x7e\x15\xe5\xd9\x49\x92\x03\xc7\xa3\x30\xe3\x9d\x20\x0f\xa8\xd5\xd6\x73\xb4\x6e\x81\x0e\x75\xde\xe2\x61\x6e\x24\x6d\x84\xf2\x46\x67\xb6\xc3\xd0\xcc\x42\x8c\x58\xbe\x45\x6a\xcc\x54\x80\x24\xd2\x64\xe7\x0c\x7d\xd9\xa2\x89\x85\x0b\x1b\x09\x51\xe2\x1f\xa8\xd5\xd4\xa9\xb5\x80\x54\xab\xd5\x8a\xa2\x4b\x88\xf0\x07\x02\x72\xa3\x4e\x40\xb5\xc9\xba\xf5\xdb\x0e\x01\x9a\x57\xa5\xc3\x51\x08\xcf\xd2\xcb\xea\xc2\xb3\x01\x8c\x09\xce\x1a\xb0\x79\x82\xb3\xad\xa3\x72\x9e\x8e\x22\x1f\x26\xcf\xb1\x03\xc6\x31\x96\xb4\x1d\xab\xab\x70\x12\x44\x90\xdd\x85\x3a\x64\x59\x0d\xa7\x26\xf4\xe4\x65\x66\x73\x29\x27\x4b\x58\xdd\xb2\x8c\x6e\x21\x9c\x7d\xb4\x85\x64\xf1\xfd\x6e\xe7\xb7\x4e\xa5\xe3\x9b\xfd\x44\xb6\x0f\x47\xb1\x7d\x8b\x33\x09\x2a\x3b\x83\xed\x0b\x77\xbd\x7d\xe5\x78\xb5\xbf\xf0\xb9\x8a\x52\xc8\xbe\x72\xa6\xda\x77\x1e\xa6\xe6\x9b\xc2\x1d\xd1\x9b\x70\x3a\xb9\x2c\x83\x45\x08\x83\xad\x16\x65\x37\xe6\xda\x04\x29\x6c\x6a\x30\x4a\xe2\x72\x06\x05\xa6\x04\xa4\x54\xa1\x5d\x80\x47\xb7\x19\x04\xfd\xfc\xc9\x20\x12\x5a\xcf\xa4\x35\x86\x26\x7c\x55\xec\xa2\xe0\xe7\x0d\xbd\xfd\x47\xb2\x45\xdc\xd0\xaf\xcd\x3c\x74\xed\xa1\xdf\x6d\x69\x3e\x6a\xb5\x19\x78\x76\x5e\xc3\xbf\xbf\x17\xd9\xda\x6f\x0c\x38\xcd\x72\x38\xb5\x59\xfd\x87\xda\x75\x9d\xba\x93\xff\x2f\x79\xf8\xbd\x5e\xaf\xbf\x70\x41\x6b\xcd\x85\x46\x00\xfd\x2f\x81\x58\xa0\xe6\x80\xd5\x9e\x0f\xeb\x07\x80\x00\xb8\x5d\xd7\x7f\xa8\xfd\x2f\x20\xe7\x86\xd8\xa9\x32\x66\x64\xd0\xbe\x14\xa0\x1c\xb0\x40\x94\x98\x79\xb1\x15\xd2\xec\xe5\xcb\x18\xb0\x9a\xfd\xf8\xe3\x8f\xb5\x56\x73\x39\x96\x91\xa2\x3f\x4a\xad\x61\xb8\x31\x0c\xcd\x03\x57\xcd\x18\xc6\x99\xed\x87\xd9\xb7\x80\xcd\x13\xff\x9d\x27\x94\x33\x99\x60\x1c\xf9\x79\x1c\xa5\x6f\x9b\x98\x87\xad\x8c\xc2\x92\x85\x2b\xf0\x6a\xcf\x18\x8a\xcf\x2c\x56\x38\xee\x5a\x57\x1c\x5b\x9b\xb9\x8d\xa9\x1c\xd4\x4c\x6d\x78\x81\x6a\xa6\x4a\x7c\x72\xf6\xdf\x6d\xf7\xbe\xc2\xd4\x94\x54\xff\x8c\xaf\xa1\x6a\x86\x07\x29\xce\x1d\xb9\x93\x1c\x13\x0a\x29\x07\xef\x71\x42\x69\x22\x43\x31\x35\xfb\xe3\x60\x50\x4c\x8f\x6c\x62\x65\x99\x21\xa5\xb0\x39\x4b\xe3\x60\x60\x99\xa9\x27\x37\xf4\x1e\xd8\x61\x1a\xc5\x4b\xda\xb3\x13\xdd\x9c\x79\x6b\x6b\x8f\x26\x4e\x7f\x07\x97\x95\x87\xbe\xba\xd7\x02\xab\x49\x0d\x3b\x43\xa6\x1d\xef\x6f\x2f\xfb\x15\x6e\x32\xcc\x5c\xd5\xf7\x79\x7f\xb1\x05\xde\xa7\xc5\x15\x46\x14\x47\x79\xcd\x12\x80\x4a\xbd\xd2\xc0\xc3\x41\x18\xf4\xd7\x37\x2c\xb1\x99\x1a\xb3\x8d\xf5\x7e\x10\x0e\x86\x58\xb9\xe3\xb0\x15\x1c\xb4\xc2\x26\xf6\x87\x0d\xf5\xdb\xdd\xaf\x40\x5c\x12\xba\x5d\xf8\x36\x35\xe8\x06\x80\x2a\xba\x67\xbb\xba\x98\x7c\xea\xdb\x95\xc5\x20\x30\xda\x55\xc5\x70\x5c\xb5\x2b\x8a\xc9\x27\x2c\xd4\xc4\x06\xa6\x4e\x3d\xb1\x53\x27\xec\x38\x2d\x80\xde\x07\x51\x0f\x53\x47\x2c\x98\x9f\xa9\xe0\xaf\x86\xc0\x50\x7d\x4f\xf9\x1f\x57\x28\xd9\x01\x71\x3f\x87\x9f\x4f\x23\xb4\x8c\x5a\x67\xe8\x57\xf6\x73\xbd\xf8\xe9\xb7\xa5\xdf\x5d\x57\xee\x48\x86\x52\x2d\x06\xe7\x58\x7a\xb6\x84\xe3\x43\xcb\xb7\x87\xa9\xb1\x9f\x84\x40\xb5\x54\x0b\x08\x90\x0e\x00\x09\xe8\x49\x66\x0d\x1c\x64\x31\x5a\x82\x86\x5c\x8a\x46\xf4\x12\x35\x1b\xce\x51\x03\xb5\x59\xad\xd6\x47\x3f\xa0\x01\x95\x73\xc9\xcf\x10\x20\x37\x66\x9d\x80\xde\xc2\xce\x51\xf1\xa1\x97\xa8\x3d\xaf\x89\x3e\xfa\x15\x0d\xd0\xaf\x28\xa4\x90\xbb\x38\xdc\xc0\xfd\xc0\x16\x74\x48\x83\xdc\x5d\x00\x79\x8a\x3b\xf9\x35\x60\xbd\x58\x46\x8d\xd9\x5a\x03\xb7\xdb\xad\x66\xdb\xdd\xd6\xea\x73\xd1\xdc\x7a\xa3\x8e\x9e\xaf\x56\xee\x0b\x81\xdf\xea\x6c\x84\x2d\xdc\xd4\xb5\x3c\xc8\x31\xa5\x64\xbd\x84\x36\x75\x1f\xda\x42\x03\x9b\x8a\x0f\x41\x93\x2f\x5f\xa2\x56\x83\xf5\x12\xa6\xdf\x9a\x5b\x14\x6d\x21\x1b\x1e\x41\x35\x6f\xad\x4a\xca\x40\xa6\x44\xe3\xca\xb6\x40\xf7\xf0\x46\x8a\x22\x10\x14\x86\x46\xe4\x13\xa4\x28\x01\x41\x59\x38\xb0\x97\x69\xc9\x8a\xc2\xd0\x5e\xa6\x2d\x2b\x09\xb1\x5e\xe6\x51\xc1\xf7\xad\x2a\xf8\x88\x2c\xbc\x32\x1c\x25\x49\x2a\xeb\xdc\x56\x61\xa3\x66\x7f\x77\

x6a\x04\x62\x21\x14\x90\xe7\xe8\xe9\x0c\x35\xdd\x03\x69\xe8\x16\xd4\x03\x59\xd5\x75\x7f\x45\x6d\xd0\xa3\x0a\xc1\x50\x06\x10\xf1\x79\x21\xed\x01\x54\x28\x53\x1c\xa8\x02\xb9\xaa\x33\x20\xdf\x1e\xd5\x05\xf7\xaa\x2e\x80\xf9\xa8\xa0\x29\xb0\x4f\x4b\xa1\x24\x60\x53\xe3\x76\x9b\x22\x05\xdc\x6a\x81\xf5\xbf\x74\x80\x8d\xec\x22\x68\x76\xba\x0f\x1d\x1b\x83\xb5\xf2\x9f\xa3\x3e\x30\xd4\x03\xf2\x19\xbe\xd9\xe9\x2a\xa7\x78\xc9\x0b\x5b\xd7\x0a\x34\x9b\xed\x6a\x7a\x01\x52\x50\x81\x09\xcf\x14\xf8\x57\xd5\x0d\x0c\xfc\x46\x67\x03\x87\xeb\xe4\xc8\xdf\xea\xae\x0d\xc2\x4e\x63\x0d\x7e\x37\xd6\x1a\x61\xe8\xc3\xef\xe1\x5a\x03\x77\x36\x5a\x76\x9d\xc1\x70\x38\x68\x34\xfa\x2d\x50\x2e\x74\xd7\x3b\xeb\x7e\xc7\xa7\xbf\xdb\xc3\x8d\xf5\x61\x00\x00\xfa\x78\x18\xb4\x87\x41\x7b\x01\x75\x41\x25\xc9\x53\x62\xfb\x6c\xe8\xa4\x9a\x25\x5e\xb4\xc0\x51\x85\x38\xb3\xbc\x65\x0a\x2f\x8e\x8b\xa5\xc7\x2d\x7a\xce\x8e\xdb\x6c\xb6\x17\xdd\xa4\x49\x95\x39\xdb\xb4\xb2\x3a\x8c\x8d\xba\xd9\xb4\x3b\xb1\x3f\x6e\xd5\x77\xd8\xaa\xc9\xac\x54\xdb\xac\xad\x93\xa3\x6c\xd7\x74\x82\x4a\x37\xec\x66\x53\x77\x75\x96\xfc\x9a\xd9\x76\xb4\xb9\xb6\x41\x36\xf0\x8d\x47\xbd\xfe\x9f\xb3\x31\xff\xf5\xdc\xf2\x0e\x68\x12\x87\xe8\x77\xe1\x95\x8b\xd2\x64\x1a\x87\x68\xa0\xfa\xeb\x49\x3d\xd8\xd7\x53\xa7\xbc\x51\xaf\x01\xb8\xa2\x16\x17\x30\xe8\x17\x9b\x04\x83\xe4\x2b\xe5\x28\xfb\x90\x46\x63\x5c\x8b\xad\xdb\x58\xf6\xef\x34\x7f\xcf\xcf\xf9\xe4\xa1\x16\xeb\xe7\x4c\xa1\x08\xa6\xd3\x89\xb6\x50\xf3\x05\xff\xfd\x72\x8b\x42\xe0\x2f\x4a\x74\xc3\xdf\xfd\xff\xec\xbd\xff\x7a\xdb\x36\xb6\x28\xfa\xf7\xf4\x29\x56\xf6\x37\x8d\xa5\x98\x96\x09\x92\x92\x28\x27\x4e\x4e\x9a\x38\x6d\xce\x38\x4e\x4e\xe2\x4e\xbb\x8f\xeb\x74\xf3\x07\x64\xb1\x91\x48\x85\xa4\x6d\x79\x26\x99\xef\xbe\xc6\x7d\x8c\xfb\x0a\xf7\x51\xee\x93\xdc\x0f\x0b\x20\x09\x92\x00\x29\x3b\xe9\xec\x3d\xfb\x54\xfd\xea\x48\x24\xb0\xb0\xb0\x7e\x61\x61\x01\x58\x18\xc4\xf0\xad\x28\x36\xd4\xc6\x0b\x85\x8e\xce\xbd\x65\x46\xfb\x77\x05\x36\xe3\x63\xc5\x7c\x3c\xbd\xac\xcf\x70\x15\x64\xb9\xa0\xf9\x8b\xd4\xc3\xef\xde\xf2\xbb\x28\xcf\x14\x04\x2a\x97\xf0\x63\xd8\x83\x41\x8c\x99\x3d\x87\xf0\xa0\x16\xfc\x68\x46\xb2\xa4\xb6\x8a\x28\xb5\x9c\x99\x1d\x9f\x21\x43\x1a\xf9\x7b\xae\x17\xd1\x92\xc2\x40\xbc\x7b\x04\x62\x4b\x66\x93\x8a\x15\x37\xb5\x84\x2e\x41\xb8\x5a\x2a\xff\x70\xc6\x0b\x61\xda\xd1\x16\x21\x50\x16\xd6\xc9\xf5\x20\x36\x80\xc0\x3e\x58\xc3\x2d\x32\xb6\x03\xde\x84\x72\x1b\xb0\xf6\x50\x99\x3c\x9b\x83\xd8\xdd\xed\x09\x85\xc6\xb5\x12\x85\x87\x34\xa8\x60\xde\x7d\x8d\x8d\x39\xde\xdb\x79\xd3\x6d\x0f\xfd\x77\x5f\x69\xfb\x61\x94\x2d\xa3\x80\x0e\xcc\xe1\x1f\xab\x5e\x5b\xaf\x7a\xb5\x5e\xcd\xf1\xd5\x58\xf5\xea\x02\x5f\xb5\x16\x8c\xd0\x67\xc1\x57\xd3\x2f\x5e\x46\x9b\x74\xe4\xba\xff\x67\x2f\xa3\x5d\x78\xab\x95\x67\x6e\xca\xc5\x34\xd2\x22\x4a\xbb\x34\x6e\x34\x1e\x14\x35\x1f\x3d\x02\x8b\x2f\x7a\x15\x4f\x1e\x3f\x7e\x0c\xd3\xe1\x10\xe0\xbd\x1a\x52\xfd\x53\x83\x44\x9c\x16\x24\xe2\x0e\x87\xdb\x41\xaa\xd7\xb3\x95\xe6\xa5\xd6\x13\x52\xf5\x5b\xb9\x49\xbe\x5e\x58\xea\x36\xe1\xc8\x4a\xdd\x26\x9b\x22\xdf\xf4\x96\xc8\xd6\x21\xd9\x6d\x48\xb3\x5b\x76\xbb\xa8\xa7\xbe\x93\x00\x2a\xc1\x11\x4c\xdc\x15\x3d\xc7\x24\xbf\xa2\x87\xbb\x9d\x0b\xa6\xba\xd5\xcf\x00\x4f\x35\x0e\x28\xdc\x87\x39\x6e\x76\xfb\x07\xfb\x7a\xa1\xbb\xc2\x65\xe5\x61\x86\x39\x0f\xee\x83\x8f\xc5\x3d\xbe\x3a\xf8\x1e\xc4\x3a\xa1\x0a\x7f\x74\x56\xa2\x0b\x86\x78\xb9\xd4\x2a\x16\xdb\xc4\x5a\x2b\xdf\xfa\xc7\xdf\x90\x99\xf4\x86\xd8\xb5\x57\xb5\x4a\xea\xb1\xad\x6c\x0c\xef\xa9\x19\x50\x94\x71\x9e\x39\x99\x62\xbd\x89\x80\xc8\xdf\x10\xe9\x0d\x21\xf2\xab\x29\xdf\xd9\xca\x5f\x59\x63\xf5\x88\x87\x0b\xc8\xac\xa5\x05\xec\x16\xcd\xee\x32\xa2\xee\xf2\x8b\xde\xb4\x8b\xc7\x58\xd1\x82\xc3\x82\x30\xbb\x8c\xb4\xaa\x16\x98\xe1\xba\x50\x00\x60\xb6\xae\x99\xa7\x9d\x7d\x98\x79\x54\xb9\x5f\x98\x3b\x13\x6f\x4b\x20\xaa\x65\x3e\xe8\x59\x22\x6d\x66\x5b\x87\x9e\xe5\xd0\x41\xce\x08\x91\x5b\xaa\xb6\xfe\x4f\x59\x1a\xe5\x65\xc6\xa2\x0c\xa6\x0c\x9f\xab\xcb\x4c\x44\x19\x4c\x09\x7e\xa1\x2e\x33\x15\x65\x50\xe7\x17\x7f\x2c\xc3\xfe\xb1\x0c\xfb\

xc7\x32\x6c\xdb\xdb\xfc\x63\x19\xf6\xbf\x64\x8c\x77\x3c\xb9\x75\x8c\x77\x3c\xe9\x8d\xf1\xca\x73\xb6\x76\x8c\x77\x3c\xf9\x23\xc6\xfb\xd5\x63\xbc\xe3\xc9\xb6\x31\x5e\x15\x73\xea\x31\x5e\x64\x50\xf7\xa6\xed\x72\xed\x4c\xbd\x34\xeb\x9a\xff\xd2\x4b\xb3\x9b\x89\xf3\x4f\xb9\xb8\xa0\x6c\xe7\x8f\x28\x70\x3d\x0a\xbc\x99\xe0\x9a\xea\x68\x33\x71\xa4\xe7\x3f\x4f\x1c\x91\xa5\x1b\x4b\x8c\xa4\x3c\xd1\xb7\xca\xe9\x26\xf5\xef\xed\x0f\xaf\x7f\x7d\xfd\xe2\xc5\xbb\xa3\xd3\x77\xcd\x68\xf1\x9b\x97\xbf\xbe\x3c\x79\x7e\xf4\xf3\x51\xfb\x56\xee\xb7\xaf\x7f\x3c\x79\xfe\xeb\xb3\xd7\x27\xef\x4e\x9f\x9e\x94\x35\xa5\xe6\x78\x58\xf9\xd9\x76\x61\x65\xa9\x46\xba\x48\x8a\xa4\x2d\x8d\x98\x74\xd1\x34\x9b\x5d\x13\x03\x6e\x74\xa9\xca\x73\x1e\x12\xc9\xe1\x11\x58\xce\x43\xc8\x15\x21\x11\xa9\xcf\x67\x1b\xd8\x85\x31\x3c\x80\x1b\x7e\x7a\x30\x2f\x0e\x69\xe2\x37\x6b\x88\x91\x4a\xf8\x16\x26\x2d\x5f\x04\xdd\x40\x7a\xfd\x33\x1c\xc2\x0d\x7c\x0b\x63\x95\x97\x48\xaf\xff\x9d\x41\xb5\xe0\x01\xb0\x76\x6c\xd6\xce\x50\x51\x78\xc3\xc3\x72\x3f\x37\x1e\xdf\xf0\xc7\xff\xae\x09\x05\x4b\x64\x5b\x47\x10\xe1\x75\x02\x0a\xa2\x95\x94\xd9\x70\xca\x6c\xf8\x01\xcd\x8d\x82\x30\x65\x51\x4e\x5d\xb8\xe1\x45\x6f\x34\x61\xa5\x4a\x40\xea\x64\xbc\xc1\x0b\x7e\xda\xbd\x66\x74\x6d\x76\xfd\x73\x6f\xdf\x1a\xab\x1c\x75\x69\x38\x7e\xf1\xee\x2d\xc3\x75\x63\x12\x95\x30\xc8\xf7\x4e\x68\xe2\x63\xac\x18\x36\x51\x08\xeb\xab\xec\xba\x21\x5b\xca\x62\xc7\x45\x31\x0d\x09\xc5\xcd\x13\xbf\xc1\x23\x98\x3e\x84\xdf\x3a\x22\x73\xd8\x07\x3c\x9a\xaa\xce\x8a\x52\x34\xef\x47\xf9\x9b\x24\xc3\x3c\xae\x4c\xaa\xf0\xb2\xdc\xdf\x86\xb0\x07\xaa\xdd\xd4\x05\x70\xb9\xd2\x23\x10\xf9\x22\x54\x85\xd9\xa7\xd5\xc1\xf7\x87\x80\xcd\x48\x50\x34\x6d\xd5\x77\x54\xcb\xad\x3e\x3e\xc4\x66\xf5\x9b\xab\x5b\x2d\xbf\x92\x5a\xae\x81\xda\x53\xcc\x7b\x4a\x04\xb6\x0b\x2d\x49\x82\x15\xd3\x4d\x8e\x02\xd4\xc3\x16\x57\xbf\x13\x7d\x7f\x1f\xde\xa4\xd1\x2a\xca\xa3\x2b\x0a\xeb\x64\x79\x13\x27\xab\xc8\x5b\x42\x72\x45\x53\xf8\xfe\xc5\xc0\x1a\x1e\xc0\xe6\xbd\x0b\xbb\xb0\x79\x3f\xc1\xbf\x63\xfc\xeb\x30\x33\xa3\x06\x29\x24\x9a\x37\xcf\xcf\x0f\xbc\x07\x73\x33\xed\xd8\x32\xaf\x41\x4e\x40\x38\x54\xca\x47\xcf\xa2\x57\xc3\xc0\xf3\x18\x9f\x18\x7e\x8a\x04\x63\x4d\x9e\x19\x2d\xf9\x19\xde\x76\x35\x25\x43\xfd\xc9\xe9\x6a\x9d\xa4\x5e\x7a\x53\xbb\x89\x8e\xa9\xc0\xa9\x3c\x10\x69\x57\x29\x95\xb7\xce\xa8\xb5\xff\x54\xd9\xb3\x3e\xbc\x1b\x6b\x3b\xf6\x76\x2b\x3b\x76\x6d\x5d\xc7\xee\x5a\xd5\xf9\xfa\x57\x09\x24\x97\xf9\xfa\x32\x3f\xc6\xa9\x75\xad\x2c\xa0\x93\x1e\xd2\x2c\x4a\x69\x28\x5d\x34\xe0\x47\x79\x56\x24\x84\xe6\x95\x6b\xb3\x85\xa2\xf2\xeb\x78\x59\xb0\x49\xca\xc1\xed\xa5\xf4\x00\x2c\xcb\x31\xc0\x1a\x4f\x0c\xb0\x5d\xc7\x80\x31\xb1\x9a\x95\xc5\x9d\x05\x07\xec\x9d\xfc\xaa\x79\x69\x41\x31\x69\xd6\xde\x5b\x20\xf7\xae\x01\xed\x0e\xf7\x17\x60\xa4\x16\x6f\x42\x2c\xe6\xde\xc5\xaf\xb3\x73\x8d\xb5\xdf\x42\xd4\xd8\x07\xe1\x70\x91\x8b\xe9\x75\x29\x76\xb8\x08\xd7\x97\x4a\x00\x31\x29\x6f\xeb\xc5\x11\x60\x62\x9a\xb0\x07\x6c\xa0\x2d\x6f\x4a\x90\x29\xc1\xbc\x17\xdb\xfa\xbd\x56\xf4\x14\x81\x39\x05\xd1\x94\xc1\xb3\xa2\x13\xc7\x5e\x8c\xb1\x9f\x46\xd7\xf6\xc1\x52\xc5\xd0\xfc\x2c\x49\xfd\x7e\xfa\x37\xc0\x7f\x49\x26\xc1\x57\x56\x04\xf5\x45\x31\x46\x6b\x6d\xd8\xfc\x95\x85\x77\xd0\x37\x8b\x33\x5b\xdf\x95\xcc\x42\x7b\x05\x35\x6b\xbe\x33\x9f\xa0\x55\x4b\x24\x68\xdd\x25\x83\xa0\x55\x4b\x1d\x68\xdd\x3d\x67\xa0\x40\x98\xf4\x61\x4c\xea\x28\x93\x3b\xe1\x4c\xea\x48\x93\xdb\x60\xad\xe4\x03\x17\xae\x32\x34\x12\xc5\x79\xc2\xa5\x59\xcd\xe9\xa5\x87\xc1\xbc\x42\x9d\x15\xa4\x60\x25\x46\x78\xdf\xec\xfb\x43\xa4\x8b\xae\xcc\x32\xb9\x06\x51\xa6\x7f\x35\xe2\x2d\x1b\x60\x33\x8d\x0e\x70\x47\x19\xf5\x80\x7f\xe5\x4e\x2f\x7e\xd7\xab\xc0\xe9\x82\xe6\x5e\xfb\xcd\x2d\x66\x0d\x12\xb0\x57\x11\x9b\x82\x2c\x2f\x57\x31\x76\x4e\xa1\x56\x05\x05\x0b\x37\xdb\x80\xca\x93\x56\x16\xbe\xe5\x9c\x44\x6e\xa3\xc6\xa5\x6a\x86\xa2\x69\x88\x7d\x0a\xd7\xb3\xe4\x5e\x57\xd9\x63\xa9\xec\x32\xb9\xd6\xfa\xa5\x5a\x6a\x9d\x2a\xfd\x1c\x55\x4f\x4e\x19\x17\x4e\xcf\x36\x3a\xdc\x4f\x37\x5c\xd6\x0e\xb1\x07\xfa\x42\x28

\x6c\x87\x88\xfa\x76\xbb\x6f\xee\x26\x06\x1d\x66\xb5\xea\x91\x83\x5d\x1a\x30\xbe\x38\x38\x3d\xec\x5a\x2c\x3f\xdd\x90\xaa\x38\xd9\xa6\x38\x97\xaf\xd3\x0d\xe9\xe2\xa3\x28\x7b\x5c\x96\x45\x3e\x76\x8a\x77\x76\x99\xa2\x46\xf1\xeb\x44\x98\xa8\xf7\x4b\xf9\xe9\xc6\x11\xb6\x00\x06\x03\x81\x5b\x79\x34\x58\xb4\x2f\xce\x07\xeb\xa6\x37\x08\xed\xb8\x84\xc6\xad\x06\x87\x76\xdc\x80\xf6\xaa\x1f\xda\x3f\x55\xa9\x6a\xa6\xb0\x43\x3e\xa1\x69\x12\x35\x62\x0a\xb7\x9a\xed\xbd\x5d\x24\xf0\x26\xea\x90\x6c\xd6\x64\x71\xe7\x23\x79\x28\xfd\xe4\xae\x5c\xf9\xfb\x8b\x45\xbe\x46\xb9\x12\x6c\x97\x18\xb3\x42\x5c\x82\xfa\x0c\x52\x51\xfa\xb8\x2a\xad\x37\x49\x38\x58\x2c\x92\xd7\xdc\x4b\x39\xac\xc5\xc3\x64\xbc\xb4\x9d\x7d\x9b\xa0\xa3\xd7\x61\xe2\xd9\x04\xba\x6a\xa2\x37\xf0\x20\xe9\xca\xa0\xe8\xf4\xa3\x47\x15\x92\x28\xda\x45\xff\xf0\x2a\x4d\xdb\x82\x3d\xe9\xbd\x4e\xd0\xa1\xae\x3a\x25\x0c\x25\xf0\x57\xb7\x04\x5e\x8f\x79\x54\xdd\xdd\x2a\xe2\xd1\xec\xb2\xc0\x4a\x02\x83\xd1\x8e\x36\x72\x13\xe7\xce\x3d\x7f\xd5\xd3\xc6\xf1\x2d\xdb\xe8\x1a\xdb\x52\x2f\xce\xd6\x49\xd6\x29\x25\x68\x7e\xdf\x44\xc7\x5c\x31\x4e\xcf\xa4\x80\x62\x25\x87\xda\x31\x8f\x57\xdc\x66\xe0\x13\x25\xfb\x46\x3f\xad\xfd\x58\x47\xe0\xe5\x38\x04\xa2\xbd\x54\xfb\x84\xa7\x26\xf6\x41\x99\xb4\xb5\x9c\x1c\x99\xa5\x01\x50\x96\x3b\x35\x8b\xee\xf0\xd2\x3a\x95\x3f\x35\x8b\xce\x88\x72\x9a\x71\x6b\x7f\x1f\x9e\x2d\xba\x8c\xdf\xf6\xc3\xfa\x1d\x87\x8c\x7e\xd3\x08\x92\xf9\x2a\xec\x70\x39\xae\xf4\x08\xf7\xed\x4c\x6a\x51\xeb\xb4\x14\xb8\xed\xab\x6c\x48\x59\x69\x20\x39\x21\xc3\x6d\x06\x40\x0e\xc0\x6a\x00\xb0\x5a\x00\x3a\xa9\xc8\x7c\x8f\x34\xb9\xee\x20\xe2\x52\xd2\x86\xd3\x4a\x35\xde\xc3\xe0\x1f\x02\x7d\xfe\xe0\x7e\x81\x0c\xfe\xec\xb2\x1f\x4b\x49\x6b\x4e\x2b\x15\x92\x21\xe2\x83\x0a\xe2\x32\xb9\xfe\xf2\x00\xed\xcb\x44\x35\x23\x69\xf1\x5b\xab\x69\xb5\x30\x24\x1b\xdf\x1a\xc1\x4c\x7c\xdf\x3b\x69\xab\x41\xd1\x29\x62\xcd\x5f\xa9\xd7\x60\x2a\xd9\xb1\xd8\xf1\x5f\x6b\x5b\x94\x22\x48\xf3\xd5\x77\x45\xb5\xca\x97\x11\x1f\x56\xaf\x1d\x06\x7a\x80\xc1\xab\x76\x1c\xe8\xae\x7b\xa9\xc8\x5d\xb6\x52\xe1\x26\xa9\x80\x46\xcb\xfa\x7e\x27\x32\x84\xfd\x3a\xfe\x43\x78\xd0\x7c\x80\x8d\xe3\x02\x4d\xb9\x9b\xeb\xbf\xc8\x26\xa8\x2f\x8e\xe1\xc9\x61\xc6\x02\x79\x65\x0c\x12\xf6\x95\xac\x97\x8b\x14\x51\xc0\x36\xcc\x7d\xe5\x66\xba\x77\x1f\x2f\x29\xfd\x1b\x6d\x03\x5d\x78\xd9\xa2\x10\xee\xad\xee\xa2\x6f\x61\xf1\x25\xc1\xc2\xfe\x98\xd0\xf6\x2e\xbd\xce\x9d\xbf\x7d\x0c\xb1\x6a\x4f\x1f\x95\x93\x5c\x43\x11\x98\x93\x1d\xce\x5b\xc5\xe6\x24\x50\x22\x3c\x27\x83\xba\x6b\x5c\xb1\x22\x45\x77\x27\x8e\x5b\x9d\x38\xbe\x6b\x27\x8e\x5b\x9d\x38\xbe\x5d\x27\xd4\xac\xe2\xa2\x2b\x94\x2c\x4f\x20\xa5\x79\x1a\xd1\x2b\xaa\xd8\x80\x08\xe2\x70\x37\xb7\x07\xeb\xcb\x6c\x51\xa0\xa1\x22\x91\xa2\xe4\xab\x76\xc9\x2f\x4f\x4f\xac\x38\x3d\x54\x36\x6d\xb4\x55\x58\x7b\x9e\xe8\x2b\xed\x9a\xd4\xdb\x2f\xb1\x85\x52\x61\xce\xca\xc3\x4e\x5b\x58\x88\x2d\x17\x73\x8a\xaf\xd5\xfe\xcc\x4e\xb2\xff\xb1\x5d\xf3\x8e\xdb\x35\xed\xdb\x6e\xd6\xb4\xfb\xb6\x6a\xda\x1d\x1b\x35\xed\x3f\xb6\x69\x7e\xed\x6d\x9a\xf6\x96\x9b\x34\x15\x6c\xa9\x6d\xd1\xb4\xb7\xd9\xa0\x69\xeb\x8f\xe1\x97\x1b\x0f\x0f\x5c\xe7\xf3\xb9\xe1\x92\xff\x26\xdb\x35\x9b\x09\x76\xc6\xc4\xfa\xa7\xed\xe1\x2c\xd2\xed\xb0\x36\xff\xb5\xd2\xed\xdc\x69\xb7\xa5\x78\x5d\xed\xf6\x2c\xca\xdc\x2a\x21\xcf\x98\x58\xb5\x6d\x21\x63\x62\x69\xb7\x99\xb8\x5b\x26\xe4\x61\x05\x6b\x5b\x4d\x5c\x91\xd5\x62\x4c\xac\xaf\x76\x84\x58\xee\xbe\x36\x27\x4f\x6b\x93\x83\xb9\x09\x7c\xdf\x9f\x85\xe3\xd0\x90\x12\xf6\x0c\x0d\x55\xc9\x89\x35\xf3\xac\x99\xe5\xc9\xe9\x7c\x86\x8a\xbc\x3d\x8a\xaa\x33\x32\x9e\x99\x64\xec\xc9\xd9\x7f\xd4\x8d\x90\xb1\x35\xa7\x01\xcf\x19\x54\xe4\x06\xda\xb2\x91\xc9\xd4\xb6\xad\xc9\x84\xa7\x15\x12\x99\x83\xd4\x8d\xb8\xd4\x77\x1c\xcf\x9d\xca\x79\x85\xb6\x6c\x24\xf4\xcd\xc0\xa2\x66\x28\xa7\x21\x52\x37\xe2\x4c\xfd\xb1\xe3\x92\x50\x4e\x52\xd4\x70\x4d\xbf\x76\x96\x22\x26\x4f\x77\xcc\x52\x44\x26\x7f\xa4\x29\xfa\x4a\x3e\x91\x7b\xeb\x34\x45\xac\x4a\x9f\x5f\x24\xdb\x8c\xb6\x67\xe4\xfe\x91\xa6\xe8\xeb\xfb\x46\xee\xb6\x69\x8a\x

94\xcc\xa9\xfb\x47\x6e\x6f\x9a\x22\xdb\xed\x4e\x53\xc4\x86\xf1\x03\xd7\x52\x79\x4b\xd6\x7f\x13\x6f\xe9\xbf\xf5\xe1\x96\xaf\x7b\xb0\xe5\x77\x3a\xb2\x72\x77\x27\x8a\xbf\x2a\xbb\x2b\x00\xfd\x5a\xec\xe0\x55\xdc\x75\x53\xdf\xe4\x3b\xf2\xd6\xeb\xe5\xcd\x40\x3c\x34\xc0\x4b\x2f\x2e\x57\x34\xce\xb3\xe6\x9d\x3c\xf2\xf1\x99\x0a\x1f\x4c\xa5\x54\x35\xd1\x68\xde\xdc\x38\x96\xeb\x59\xf3\x19\xfa\x15\xe1\xd4\x72\x3d\x6a\x59\x43\xa3\x5d\x6e\x4a\xec\xa9\xe3\xcc\x30\xcd\xa0\x65\xd3\xf9\x64\x1c\x84\xb2\x6b\xd0\xaa\xe0\x8f\x03\x73\xee\x07\x73\xbc\x00\x21\x70\x42\xdb\xb7\xe6\x2a\xc0\x74\xe6\x8f\x43\xdf\x1b\xe3\xed\xd9\xc4\x9d\x85\xbe\x1f\x74\x02\xb6\x67\xe3\x49\x60\x8d\x7d\x74\x67\x6c\xc7\xf5\xc7\xb6\xab\x02\x3c\x9e\xcd\x09\x21\x73\xc4\xd8\x9f\x98\xe3\xd0\x24\xb3\x4e\xc0\x33\xcb\x9e\xbb\x96\x87\x57\x6e\x7b\x73\x32\x73\xe6\x33\x5f\x05\xd8\xf3\x49\x30\xa6\x21\x62\x1c\x7a\x93\xd0\x25\xc4\xed\x04\x1c\xba\xe6\xd4\xf3\x38\x8d\x3d\xdb\xb4\x4d\xcb\x51\xd2\x98\x58\xae\x3d\xf6\xf9\x9d\x11\xce\x78\x6a\x4e\xe6\x3e\xed\x04\x6c\x39\x36\x71\xc7\x3e\xde\x1d\xe1\x50\xea\xf8\x96\x1b\x28\x49\x31\x36\x83\x69\x18\xe0\x05\xe2\xe1\x78\x3e\xf7\x1d\x6a\x75\x02\x9e\x5a\x3e\x1d\x87\x53\x24\xc5\xdc\x9a\xfa\xee\x6c\xa2\x64\x9e\x6b\x86\xd4\x27\xfc\xf2\x0a\xdb\x27\x93\xd9\xc4\x27\xdd\x34\xf6\xc3\xc0\x9c\xf0\x0c\x95\xd6\x38\x98\x12\xcb\x1e\xab\x00\x07\x64\xe6\xcf\x09\x47\x20\x98\x4f\x66\xd6\x64\xe6\x74\x02\xa6\xce\xcc\x9f\xcc\x02\xa4\xdd\x8c\xce\x89\xe3\x85\x4a\x1a\xd3\xb9\x4f\x9d\xa9\x8b\xd7\x88\xdb\xae\x33\xb7\xc6\xd4\xee\x04\x6c\xce\x03\x32\x0b\x03\xac\xe0\xfa\x6e\x10\x8e\x7d\x25\xc6\x96\x63\x06\x1e\x09\x02\xbc\xa4\x7d\xea\x05\xb3\x60\x32\xee\x66\x5e\x48\x67\x56\x30\x41\x05\x19\xcf\x2c\xdf\xb4\xa6\x4a\xc0\x8e\x37\x75\x5c\xc7\xc3\x39\xc2\x84\x7a\x13\xea\xb8\xdd\x18\x8f\x03\xdf\xf4\x66\x21\x62\xe2\x87\x0e\x99\xfb\xa1\xa3\x54\xe9\xc9\x7c\xe6\xba\x21\x02\x76\x6d\x42\xc6\xb6\xdf\x8d\xf1\xcc\xb5\xe9\x98\x8c\x2d\x54\x69\x3a\x99\x84\x73\x4f\xad\x20\xae\x4d\x82\xc9\x04\x3d\x7c\x2b\xf4\x1d\xdb\x22\x66\xb7\xad\x30\x4d\xdb\x9a\x06\x2e\xbf\xf3\x7d\xee\x5b\xc4\x56\x8a\x9b\x3f\x1f\xcf\xa6\xf3\x40\xe4\x37\xa5\x73\x93\xd2\x6e\xa9\x08\x26\xd4\x34\xfd\x39\x0a\xbe\x1d\x7a\xae\x3b\x0f\x94\x52\x11\x8e\xbd\xe9\x8c\x38\x08\x78\x66\x9b\x9e\x37\xb5\xba\x49\x61\x4e\x02\x6f\x62\x8f\xf9\xf5\x2e\xa6\x69\xbb\x96\x5a\x41\x88\x63\xcd\xac\x19\x9f\x7b\x99\x9e\x49\x27\x74\xda\x4d\x0a\x6b\xea\x4f\x4d\xcf\x45\xe3\xe2\x4c\x42\xcb\x9a\xcf\x95\x2a\x6d\x51\xc2\xc8\x84\x24\x1b\x07\xd6\x24\x98\x59\x93\x4e\xc0\x4e\x68\x05\x93\x70\x8e\x52\x31\xf6\x02\xc7\xf2\x68\xa8\xb4\x15\xb6\xed\x9a\x21\x41\x92\xcd\xc2\xd9\xd8\xb7\xc3\x79\x27\xe0\xc9\xd8\xf4\xa6\xf6\xd8\xe1\x0a\xe2\xcd\x27\x76\x48\xd5\xe2\x36\xf1\x4c\xcf\x47\xbb\x6d\x07\xd3\xa9\x6f\x79\xdd\x66\xd3\x25\x81\x15\xcc\x2c\x6e\xdd\xa6\x34\xf4\x28\x9d\xa8\x00\xcf\xac\xa9\x65\x05\x9c\x64\xc4\x71\x2d\x7b\x6c\xfb\x9d\x80\x3d\xcb\x9f\x53\xd7\xe3\x76\x36\x98\x13\xd3\x9e\x28\x15\xc4\x73\x89\x37\x99\x38\x88\xb1\x1f\x38\x96\x6d\x9a\xdd\xd6\x2d\xb0\x1c\xdf\xf5\xa7\x26\xda\x59\x73\xee\xce\xa6\x33\xa2\xb4\x6e\xd3\x49\x30\x26\x1e\xd2\xd8\x9c\x8c\x1d\x9f\xda\xdd\x52\x11\x92\x99\x45\x5d\x32\x43\xc0\x13\x3a\x1f\x5b\x44\x39\xe6\x85\x93\xd9\xcc\x9c\x58\xc8\x8b\xf1\x78\x32\xf6\x66\x3d\x9a\x37\x77\x4c\x6a\x8f\x39\xed\xc6\xd3\x29\xb1\x4c\xcb\x53\xca\xb1\x39\xf1\x3c\x93\xf7\xcc\xb6\x7c\x3f\x24\x7e\x37\xf3\xc8\xcc\x73\x02\x42\xd0\x6c\xfa\x6e\x68\x85\x66\xa0\xc4\x98\x50\x7b\x3a\x09\x4c\x2e\xc7\xc4\x21\x9e\x3f\xee\xb6\x6e\xd6\xd4\x71\xa7\x53\x07\xe5\x38\x9c\xbb\x94\xfa\xb3\x99\x0a\xb0\xed\xf8\xa6\x1f\xf8\xd8\x33\x4a\x66\xbe\xe3\xf6\x88\x9b\x3d\x23\x81\x19\xf8\xc8\x94\x60\x1c\xcc\xc6\xde\xc4\x56\xda\x63\x1a\xba\x9e\xe7\xa0\xd9\xa4\xb6\x43\x5c\x2f\xe8\x16\xb7\xb1\x3f\x0b\x02\xcf\x99\xf3\x91\x61\x62\x53\x7b\xaa\x04\x3c\x71\x2d\x3a\x99\x73\x63\x15\x4e\x7c\xcb\x77\xbd\x6e\x52\x4c\x1d\x77\xee\x5a\x14\x15\x64\x1c\xd2\xb9\x6f\xa9\x6d\xc5\xd4\xf5\xc6\x13\x9b\x8f\x34\x8e\x4d\xa6\xd6\x7c\xd2\x2d\x15\xae\x13\xb8\x53\x97\x70\x4f\x88\xcc\x4d\xcf\x9f\

x2a\xcd\xa6\x1b\x04\x53\xd3\xe2\xcc\x23\xde\xc4\xb1\x67\xb4\xdb\x77\x9b\x99\x3e\x9d\xcf\xe7\x1e\xf7\x22\x27\x36\xa1\x96\x52\x2a\x3c\x67\x6c\x4e\x02\x8a\x9a\x17\x52\xd7\xf2\x43\xda\xed\xbb\xf9\x74\x3e\xf3\xec\x39\x1f\x19\xac\x60\x32\x9d\x11\xb5\x5f\x31\x99\x92\xa9\x3b\xe7\x43\x98\x3d\xb5\xc6\xb6\xd5\xcd\xbc\xc0\xb3\xa6\x36\x0d\x90\xc6\xd4\xb3\x26\x13\x32\x53\xd2\x38\x24\xee\xc4\x77\xf9\xd0\x64\x31\x41\xb2\xea\x41\xc0\xb6\x23\xe2\x85\xde\x34\x0c\x51\x41\x82\x90\x9a\xd4\x27\x4a\xb3\x39\x1f\x4f\x43\x67\x3e\x9d\x8b\x41\x97\x86\x64\xda\x2d\xc7\xe6\x64\x6e\x4e\xa6\xdc\x5f\x98\x5a\x64\x3a\x99\xfb\x4a\x95\x36\xbd\x89\x3d\x0d\x03\x54\x10\xcf\x0a\xdc\x99\xeb\x75\x8f\x20\x84\xd8\xf3\x99\x6b\x3a\x22\x70\x37\x33\x43\x4f\x89\x31\xf1\xa7\xc4\xf4\x6d\x6e\x8f\x6d\x12\x38\x53\xd2\x4d\x63\xcb\x0d\xfd\xe9\x74\x3e\xe6\x52\x61\x3a\xd3\xd0\x55\xda\x63\xdb\x0a\x3c\xcf\x9f\xa2\x54\x38\x66\x30\xb5\x9c\x59\xb7\x82\xd8\xc1\x8c\xfa\xd4\x44\x52\x90\x71\x30\xf3\xa9\xaf\x64\x9e\x63\x93\x70\x32\x0d\xb0\x67\xb3\x80\x98\x66\xe8\x74\xcb\xb1\x13\x04\xe3\xd0\xe1\x8e\x77\xe0\xdb\xd4\xb1\x7c\xe5\xd0\xc4\xdc\x15\x6b\x36\x43\x63\x35\x0f\x26\xe3\x29\x65\xe6\xb5\xcb\x56\xcc\x03\x7f\x32\xf7\xf8\x20\xe9\x85\x93\xb9\x47\x95\x18\x4f\x02\xc7\x21\x33\x17\x01\x3b\x9e\x33\x1d\xbb\x64\x2a\x82\xa8\xe7\x1d\xc7\x56\xab\x79\xe1\x4f\x77\x3d\xa1\xaa\xbb\x06\xed\xa7\xda\x09\xd5\x5f\xef\x76\x42\x75\x4c\xac\xed\x96\x0e\x14\xcb\x11\x5f\x3f\xfb\xe8\x5d\x97\x0e\x26\x9e\x39\xa3\x45\xc0\xdd\xf6\x83\x60\x66\x6a\x96\x0e\x7c\x7f\x32\xf5\x28\x1f\x7e\x5d\x27\xf0\xbc\x69\xdd\x75\xe9\x68\xc4\x0e\x26\x74\x6e\x4f\xd1\x92\xcd\xe9\xcc\x99\xbb\xcc\x92\xa9\x4a\x7a\x63\x67\x3e\x1f\xdb\xa8\x05\xe3\x39\x09\xed\xc9\x7c\xdb\xa8\xfe\x98\x98\x74\x6c\x71\xe3\xe3\x85\x74\xe2\x5a\xa1\x66\xe9\x60\xe6\x9b\xe3\x89\xcb\x05\xd2\xf2\x6d\x3a\x09\xc8\x7c\xcb\x46\xc8\xdc\xb5\xc3\x19\x97\xf9\xb9\xef\x10\x3f\x9c\x68\x7a\x32\xf6\xa9\x19\x84\xdc\x0d\x22\xf6\x94\x5a\x64\x3a\xbb\xcd\xd2\xc1\xd7\x3e\x47\xba\x4d\x6a\x58\x2c\x67\xea\x33\xbf\xfe\x40\xf4\xa9\x5f\x7f\xb0\xf4\xb9\x5f\x7f\xb0\xf5\xc9\x5f\x7f\x70\xf4\xd9\x5f\x7f\x18\xeb\xd3\xbf\xfe\x30\xd1\xe7\x7f\xfd\x61\xaa\x49\x00\xcb\x3b\x88\xe9\x61\x95\xfb\xc0\xf9\xfb\x25\x7f\xdf\x3e\xec\xc1\x69\x80\xd5\x95\x47\xa0\xf8\xfb\x25\x7f\xaf\xa9\x6e\x61\x75\x4b\x5b\xdd\x5a\xf2\xf7\x9a\xea\x36\x56\xb7\xb5\xd5\xed\x25\x7f\xaf\xa9\xee\x60\x75\x47\x5b\xdd\x59\xf2\xf7\x9a\xea\x63\xac\x3e\xd6\x56\x1f\x2f\xf9\x7b\x4d\xf5\x09\x56\x9f\x68\xab\x4f\x96\xfc\xbd\xa6\xfa\x14\xab\x4f\xb5\xd5\xa7\x4b\xfe\x5e\xb1\xad\x6f\xcb\xa4\xc7\x5c\x32\x54\xc0\x3d\x2e\x14\xcd\x8c\x7b\xb8\xe5\x96\x0b\x84\xaa\x96\xcf\x65\x41\x55\x2b\xe0\x72\xa0\xaa\x15\x70\x11\x50\xd5\x0a\x39\xfb\x55\xb5\x42\xce\x79\x55\x2d\xca\xb9\xae\xaa\x45\x39\xc3\x55\xb5\xe6\x9c\xd9\xaa\x5a\x73\xce\x67\x55\xad\x0b\xce\x63\x55\xad\x0b\xce\x5e\x55\xad\x05\x67\xad\xaa\xd6\x82\x73\x75\xa9\xca\x3b\xd8\x75\x74\x77\xcb\xeb\x50\xb5\xf9\xb4\x8b\xf6\x7f\x8a\x78\xee\x61\xdd\x71\xf3\x23\x1c\xc1\x8b\xe5\xb3\x76\x91\x2d\x12\x45\xf3\x66\x18\x09\x7e\x8a\x8a\xd3\x06\x72\xd6\x68\x78\x00\xd6\x39\x96\x54\xe7\x72\xad\x60\x2c\x39\x0c\x71\xbe\xa0\x09\x03\x4f\xcd\xdf\x29\x03\xf5\xfe\x3e\x7c\x8f\xd9\x88\xf5\x8d\x17\x29\x9d\x6f\x95\xa1\x7a\xb3\x28\xf3\x1c\x6f\xfa\xce\xe2\x89\x62\x4b\xa9\x46\xf7\x79\x3c\x5e\x6a\x51\xcb\x82\xbd\xe0\xc9\x7f\xe5\xe4\xd5\x4b\x4c\x51\x5c\xa4\x03\xae\x95\x73\x5b\xe5\x70\xd3\xeb\x7b\xa8\x17\x9b\x76\x9d\x30\xe5\x25\x97\x35\x2c\x96\x6d\x2c\x16\x2a\x2c\x96\x6d\x2c\x16\x32\x16\xf5\x72\xd3\x76\x39\x4d\x26\x63\x99\xa5\x9a\x9c\x39\x57\x52\xee\xed\xdb\x24\xdf\xae\x38\x4a\xb6\xe3\x28\xa9\x38\x4a\xb6\xe2\x28\x59\xd4\x12\x7c\x2f\x8a\x2c\xdc\x52\x62\xee\xa5\xc8\xd5\x2d\x11\x89\x08\x0a\xd7\x8b\xe1\x3e\xe6\x99\xc4\xd2\x02\xde\xa4\x97\xa5\x64\x59\x43\x63\xa9\x40\x63\xa1\x42\x63\xd9\x42\x63\x51\x43\xa3\x0e\x70\xd2\x82\x67\x4d\x3a\x79\x7a\xab\xdc\xe1\x5d\xa6\x64\x5a\xb1\x7d\xda\xc5\xf6\x9f\xa2\x29\xb7\x5c\xca\x81\xb9\x51\x72\x29\x4a\x76\x9c\x09\xe7\x25\xc9\x44\x32\x24\xda\x5b\xa1\x8b\xb2\x1c\x01\xa2\xf4\x2c

\x9a\x65\x97\x45\xd9\x5e\x1c\x2a\x4b\xb3\x64\x44\x8b\xa6\xcd\x91\xab\x5e\xbc\x32\x65\x0b\x5e\x7c\x81\x39\xdb\x18\x1c\xc6\x49\x73\x08\x8f\x0a\xed\x2c\x9f\x3c\x01\x02\x07\xd0\xda\x36\xdd\xc6\x83\xfd\x2d\x38\xd8\x8f\x06\xfb\xbb\x5b\x6a\x8b\x06\x0b\x72\x57\x2c\x90\x8a\x5b\xe2\xc0\xb9\xd3\xc6\x80\x73\xa2\xd5\xbe\x1a\x68\x35\x2a\xfe\x14\xe9\xd8\x5b\x8d\x7a\x3f\x45\x2a\xe4\xf4\x39\xf1\x45\x52\xfc\x05\xdc\x87\xf9\x42\xa4\xc5\x67\x3f\xd4\xe7\xf8\x78\x1d\xae\xfb\x74\xc9\xea\x2c\x45\x1d\xf6\xe3\x62\xd9\x91\x4c\x7f\x81\xd9\xf4\x19\x68\x9f\xb7\x83\xdf\x03\xfe\xdd\x17\xdf\xf5\xd5\x97\x58\x9d\xb5\xe2\xf3\x26\xf1\x7b\xc0\xbf\xfb\xe2\x7b\x77\x4a\xfe\x05\xcf\xc9\x2f\x0c\x0e\x1f\x57\xbc\x25\x4f\x2f\x3d\xe4\xc9\x0f\xbc\x45\x91\xb1\x5f\xbc\xac\xe5\xec\x5f\x48\xb7\x48\x78\xc5\xa8\xd3\x99\x99\x1f\x67\x53\x83\x12\x90\x68\x73\x51\x6f\x73\x59\x6b\x73\x51\x6f\x73\x29\xb7\xb9\xd8\xa6\x4d\xc2\xfb\x49\xc5\xd0\xc0\xcf\x9b\x50\x3e\x28\xb8\x45\xda\xff\x45\x71\x69\x85\xf4\xd2\xa9\x5e\xb2\x36\xed\xe2\x1d\x4f\xc3\xdd\xdd\x26\xef\xa7\x28\x5c\xb4\xb9\xa8\xb7\xb9\xac\xb5\xb9\xa8\xb7\xb9\x94\xdb\x5c\x54\x6d\x2a\xbd\xce\xfe\x7b\x08\xd4\xb8\xfe\x05\xb3\x2f\xfd\x45\x7f\x98\xea\x2f\xa8\xbc\x7f\x89\xba\x8e\x51\xfd\x05\x8d\xc1\x5f\x22\x9d\x09\xbd\xc2\x8b\x12\x58\x99\xc5\xb2\x44\x51\xa5\x94\xbc\x20\x6b\x70\x51\xf5\x85\x9b\x8b\x9c\xc8\xe6\x62\xb1\x8d\xad\xaa\x9a\x65\x7f\x19\x45\xba\xdb\xcc\xb1\xa9\x60\xa1\x6a\x30\xb8\x53\x8b\x7f\x51\x9a\x9e\x66\x8b\x7f\x89\x54\x2d\xfe\x25\xba\x4b\x8b\x6a\x63\xd7\x6c\xf1\x27\x65\x8b\x3f\xa9\x5a\x54\x4b\x5b\xf3\xf2\x0a\x4d\x93\x18\xbc\x28\xd4\x1e\x0b\x6a\xb1\xc3\x38\x48\x61\x95\x76\xb9\x79\x44\x14\x2d\x19\xc5\x02\xd6\x76\x68\xfe\xb8\x0e\xbd\x9c\xc2\x75\xf7\x4c\x9f\x7d\x70\xbe\xa9\x94\x6f\x9c\x6e\x5e\xa8\xd0\xc6\x01\x68\xae\xaa\x83\x13\xdb\xb9\xaa\x0e\xce\xa1\xa9\xaa\x0e\x4e\xa1\xa9\xaa\x0e\x4e\xc9\x07\xe1\x12\xaf\xef\x58\xea\xee\xef\xc0\x39\xfd\x20\x5c\x60\x29\x4e\x3a\x2a\x53\x2e\x6c\x11\x4d\x7b\x13\x08\x83\x14\xa8\x70\xc4\x90\x42\xa0\xc2\x11\xa3\x17\xbe\xaa\x0e\x06\x2f\x7c\x55\x1d\x8c\x93\x78\xaa\x3a\x18\x26\x69\xdd\x66\xc0\x3e\x18\x76\x19\x70\x51\xcf\x2d\x2d\x31\x30\x70\x33\xe0\x74\x60\x92\xb5\x5b\x8d\x38\x9c\x1a\x79\xdb\xd9\xf9\xaa\x97\x95\x48\x31\x43\xf4\x0c\x7e\x40\xf9\xf7\x5a\xde\xc0\x0f\x65\x32\x8a\xc1\x0f\x28\xf7\x1e\x47\xf6\x07\x53\xc6\xd6\x6b\x23\xdb\x84\x23\x45\x19\x79\x83\x48\x22\xbf\xdd\x20\xa9\x1a\x44\xf2\xf8\xa2\xc1\x9a\x25\xf0\xfb\x1b\x94\xe2\x92\xbc\x41\x0b\x4d\x6c\xbb\x41\xab\x6a\xd0\x5a\x14\xe3\xd2\x00\xcb\x4b\xe6\xb5\xbf\x41\x29\x92\xc9\x1b\xb4\x59\x83\x61\xbb\x41\xbb\x6a\xd0\x66\x6d\x85\xa2\x41\xbb\x47\x1d\x9a\x70\xa4\xd8\x27\x6f\xd0\x61\x0d\xd2\x76\x83\x4e\xd5\xa0\xc3\xda\xa2\xa2\x41\x47\x6e\x90\xf6\x37\x28\x45\x4b\x79\x83\x63\xd6\xe0\xbc\xdd\xe0\xb8\x6a\x70\xcc\xda\x9a\x8b\x06\xc7\x72\x83\xf3\xfe\x06\xa5\xf8\x2a\x6f\x70\x82\x93\x8a\x76\x83\x93\xaa\x41\xf4\xde\x2f\x44\x83\x93\xda\x24\xa2\xbf\x41\x29\x22\xcb\x1b\x9c\xb2\x06\x17\xed\x06\xa7\x55\x83\x38\x6d\x12\x63\x32\x2b\xdf\xe5\x04\x7c\xf1\xd9\x8b\x3f\x2e\xc5\xf9\x7a\x97\xe2\x10\xe6\xdc\x8b\x9b\xcd\x18\x30\xcc\xc3\x62\x9b\x5f\xfb\x5a\x1c\x75\x33\xe4\xbf\xe4\xc5\x38\xcf\x92\xf8\x8a\xa6\x3c\xcb\x2f\xe4\x09\xd8\xd6\x9e\x1f\xe5\xcc\x41\x09\xc1\xc3\xfd\xd9\x3e\x9d\x27\x29\x15\xdb\xa9\x5b\x5c\x93\xce\x9a\x48\x6b\x77\x79\xf2\xb3\x6d\x7d\x8d\x8b\x78\xfe\x55\xaf\xe0\x91\xf1\x2c\xf3\x83\x1c\x00\x31\x2d\x67\xdf\x16\x79\x8a\xff\x38\xdd\xa4\x3d\xaa\x34\x26\xd6\x6d\x4f\x37\xb1\x2a\x3d\xa7\x9b\x6a\xdb\x1a\x5a\xa7\x9b\xc6\xc4\xfa\xe3\x74\xd3\xd7\x3e\xdd\xc4\xb8\xb2\xdd\xe9\x26\x25\x73\x6a\xa7\x9b\x38\x83\x3a\x4f\x37\xf1\x73\xb4\x5b\x9e\xfe\xb6\xff\xa5\xcf\x33\xd1\x38\xd8\xf3\xbd\x8c\x4e\x9c\xc6\x8b\x55\x38\x6e\x16\xbd\x5a\x7f\x08\xe7\x8d\x87\x41\xb4\x5e\xd0\xf4\x9f\x72\x24\x4a\x42\x15\x7f\x33\x0c\xf9\x0b\x8e\x18\x7e\x97\xf1\xf9\xef\x70\x74\xea\xa7\xad\xee\x04\xc2\xcd\x33\xcf\xb0\xeb\x65\x39\xe9\x59\xff\x51\xa8\xfd\x7d\x78\x43\xd3\x15\x8e\xa2\xcf\x16\x49\x14\x50\x20\xcd\x6b\x53\x58\xf5\x37\xcf\x48\xfd\xec\xd2\x78\x6a\x80\x33\x33\xc0\x21\x0

6\xd8\xb6\x01\xd6\xd8\x00\x32\x35\x60\x66\x00\x10\x69\xab\xd1\xd8\x35\x60\x6c\x1a\xe0\x58\x0
6\xd8\x8e\x01\xd6\xc4\x00\xe2\x1a\x40\x4c\x03\x2c\xb9\xdc\xcc\x80\x31\x31\xc0\xb1\x0d\xb0\xc7
\x06\x58\x53\x03\xc8\xcc\x00\xc2\xe0\x4b\xe5\x26\xa6\x01\x63\xcb\x00\xc7\x31\xc0\x9e\x18\x30\
xb1\x0d\x18\x8f\x0d\x70\xa6\x06\xd8\x33\xa9\xa0\x4d\x0c\xb0\x6c\x03\xc8\xd8\x80\xa9\x01\x30\x
b1\x0c\x18\x3b\x06\x38\x78\xb5\x80\x5c\x90\x61\x62\x19\x40\x1c\x03\x26\xac\x20\x31\x60\x6c\x
1b\xe0\x8c\x0d\xb0\xa7\x52\x41\x6b\x66\x80\x45\x0c\x20\xac\x49\x03\xc0\x72\x0d\xb0\x4c\x03\x
08\x43\x87\x17\x3b\xef\xa0\xab\xa5\xa6\xab\x55\xa7\x2b\xc3\x82\xd1\x91\xf5\xdb\x62\xdf\x0d\x8
0\xb1\x8c\xad\x68\x98\x75\x8b\x61\x8b\x08\x99\x32\x96\xb6\x20\x1c\xc3\x8a\x15\x98\x18\x20\x7
7\x97\x4c\x38\x3d\x18\x81\x11\x7b\xbb\xce\x08\xc6\x50\x46\x60\x46\x3f\x7b\xca\x09\x3b\x1e\x37
\xe8\xe5\x98\x82\x5b\x63\xce\x7d\x47\x6e\x81\xb1\x86\x89\x86\xcd\x58\x3a\xe1\x6c\x1f\xcb\x3c\
x64\x2c\x60\xf2\xc0\xe4\x82\xf1\x90\x11\xb6\xf0\x6a\x6a\x37\x42\x5d\xae\x2e\x97\x1e\x5e\x93\xc
2\x9c\xca\x6c\x11\xcd\x5b\x37\x3c\xa1\x16\xbc\x3c\xfd\xf5\xdd\x0f\x2f\x5f\xf0\x3b\xa5\x18\xc5\x2
c\x03\xb0\xf3\x8c\x42\x2e\x93\x48\xc1\x26\xa4\xae\x90\x54\x22\xd8\x69\x09\xe9\x45\x82\xb8\x72
\xfb\xef\xbe\x7b\xfd\x33\xcd\xc0\x8b\x43\x91\x1b\x7d\x8d\x2c\xe5\xf7\x69\x28\xf0\x60\xe5\x7f\x7d
\x53\xe7\x67\xc3\xa5\x34\x37\xe6\x01\x4e\x46\x5c\xcb\x34\x8d\xe6\xbb\x62\xae\xc0\x8b\x28\x0a\
x58\xb5\x02\xae\x69\x5a\xad\x22\xb6\x54\xa4\xfd\xd6\x91\xdf\x2a\x1a\x18\xd7\x1b\xb0\x14\x0d\x
4c\xea\x48\xaa\x8a\x4c\x1b\xfd\x50\x34\xe4\xd6\x10\x69\x83\x98\x35\x5b\x69\x83\xf0\xe4\x22\xa
a\x02\x7e\x93\x5a\xed\x22\x41\xa3\x99\x56\x81\xb0\xd9\x95\x76\x11\x2a\x15\x69\xb7\x30\xaf\x6
3\xd9\xae\xee\x76\xd5\x26\x6e\x2f\x3f\x2c\xb7\xa7\x01\xdb\xed\x91\x2a\xa7\xd9\x88\x42\x2e\xdc
\x6e\xb9\x99\xb8\xbd\x82\x39\x75\xbb\x04\xd3\x75\x7b\xf9\x3d\x73\x7b\xf8\xed\x35\x91\x50\x88\
x44\xb3\x99\x36\x26\x81\xdb\xcb\xf1\xd0\xed\x91\x1a\xea\x76\x4b\xf7\xbc\xd9\x86\x82\xf3\x5a\x7
6\x09\x2b\x41\xd4\x84\xb4\xa4\xb7\x1a\x66\xda\xb5\x22\xca\xd6\x9d\x3a\x14\x55\x1f\xc7\x72\x1
1\xa5\x4c\xc8\x78\x2a\xde\x4f\xeb\x68\x74\xe8\x06\xe9\x10\xff\x59\x13\x53\xad\xa1\x20\x1d\x1c\
xf5\xeb\x9d\x51\x48\x45\xad\x33\x5a\x3b\x41\x3a\xe4\x97\x36\x8a\xe8\x4c\x05\x51\x9b\x02\xb7\
x97\x14\xc4\xed\x25\x85\xe5\xf6\xb2\xde\x76\xbb\xd9\xe6\x34\x40\xe8\x6c\x45\x17\xb9\x27\x6e\x
97\x08\x4f\xdd\x1e\x66\xb8\x6e\x0f\x25\x67\x6e\xaf\x68\x79\x6e\x37\x43\xfd\x26\xbd\x15\x83\x47
\xb3\x95\x76\x91\xd0\xed\x62\x29\x75\x7b\x54\x68\xde\xe4\xa8\x7c\x47\x95\xd1\xe7\x65\x38\xa6
\xe9\x3a\x26\xd1\x5a\x10\x51\x46\xeb\x66\x94\x0c\xd4\x59\x90\xa2\x11\x53\xd5\x88\x53\x6f\x44\
x59\x66\x5c\x87\xa3\x44\x66\x52\x87\xa3\x2c\x33\xad\xca\x28\x5a\x91\x8d\xad\xb2\xfa\xac\xd9\x
84\x02\x88\xd7\xec\x8e\xde\xe1\x10\x0d\x29\x80\x04\x35\xc2\x2a\x0a\x84\x55\x01\xad\x01\xe1\x
28\x28\x2a\xcf\x9b\x5c\xd1\xfa\x5d\x9d\xc4\x24\x6e\x4f\x2f\x2c\xb7\x8b\xda\x76\xb3\x09\x95\x6c\
xb8\x0d\xbe\xab\x64\xc3\xed\x27\xf8\xc4\xed\x11\xd4\xa9\xdb\x2f\xa8\xae\xdb\xc3\x94\x99\xdb\x
c1\x14\xcf\xed\xd6\x25\xbf\x89\x81\xde\x90\x74\xaa\x4a\xe8\xf6\x08\x31\x6d\xd2\x54\x6f\x4f\xb4\
x12\x24\x4f\x40\x14\x6f\xc9\x16\x6a\x4f\xac\x2d\x94\x89\xd8\x5b\x28\x3e\x71\xb6\x90\x67\x32\xe
e\x54\x7d\x32\xe9\x53\x49\x32\xed\x31\x86\xb2\x0b\xae\x86\x30\xeb\x33\x97\xc4\xeb\xd3\x7b\xe
2\x6f\x61\x2d\x49\xd0\x67\xc8\x48\xb8\x85\xb1\x24\x74\x0b\x53\x46\xe6\x4d\x0e\x29\xc5\xa5\xcf\
x54\x10\xd2\xa7\xa1\xc4\xda\x42\x41\x88\xdd\xa3\x65\xc4\xd9\xc6\xb0\x8d\xb7\x30\x3b\x64\xd2\
x69\xdd\xc8\x74\x0b\xb3\x44\xdc\x2d\x74\x91\xcc\xb6\xd0\x7a\xe2\x6d\x61\x4d\x89\xdf\x67\xc1\x
48\xd0\x65\xc2\x48\xd8\x67\x16\xe8\x16\x66\x94\xcc\x1b\x16\xea\x36\xae\x0a\x31\x1d\x8d\x31\x
52\xa3\x6c\xd5\xa8\x42\xb4\x2e\x0a\x87\xad\x82\xee\x48\xef\x4d\xc5\xfb\x71\x83\x39\xed\x12\x9
3\x1a\xd1\x54\x6d\x4c\x6b\x25\xfa\x87\x63\xbd\x6f\x52\xb5\xa2\xf3\x4c\x8a\x9e\xea\xbc\x92\x0a\
x8b\x36\x9e\x41\x83\x9a\xed\x12\x61\x8d\x5a\x3a\xd7\x04\x21\x68\xdc\x12\x51\x57\x4d\x81\xae\

xee\x11\xb7\x0f\x7d\xcb\xd5\x0b\x8a\xed\xf6\x09\x8a\xe3\xf6\x31\x7a\xec\x76\x77\x7e\xe2\x76\x8b\xd2\x54\x7a\xdf\x7e\xeb\xba\x7a\xd2\xcd\xdc\x2e\xd2\x79\x6e\x9f\x78\xf9\x6e\xb7\x12\x04\x6e\xb7\xe8\x84\x6e\x9f\x60\x50\xb7\x4f\x09\xe6\x6e\x9f\x88\xd7\xdc\x0a\x8d\x10\x90\x1e\x75\x25\x56\x8f\x84\x12\xbb\xd7\x64\x10\xa7\x53\x52\xc9\xb8\x57\xe1\xc9\xa4\xd7\x6a\x90\x69\x97\x25\x76\x7b\x35\x91\xcc\x7a\x4d\x06\xf1\x3a\xb4\x91\xf8\x3d\xe6\x82\x04\xbd\x56\x8b\xc8\xe6\x40\xd1\x04\xed\xb1\xbd\x64\xde\x6b\x92\x84\x6b\xd1\xd9\x4d\xd2\xa9\x57\xc4\xea\x37\x2d\x76\x87\xe5\x20\x4e\x8f\x5a\x93\x71\xaf\x6d\x21\x93\x4e\x05\x26\xd3\x5e\xdb\x46\xdc\x1e\xe3\x43\x66\xbd\x1a\x48\xbc\x1e\x33\x40\xfc\x5e\x1b\x48\x82\x5e\x53\x40\xc2\x5e\x7b\x44\x68\x87\xb1\x23\xf3\xba\x35\xba\x8d\xff\xe0\x9a\xbc\x49\xb5\x6d\x29\xbc\x4f\x62\x3a\x1a\x57\xa2\x40\x5a\xf1\xde\xae\x20\x38\x6a\x41\x74\xf4\x42\x34\xae\x53\x44\xed\x43\x94\xce\xb1\xaa\xf9\xa9\x59\x73\xff\xf4\xe3\x67\xb1\xa2\xa2\xf6\x20\x2a\xde\xaa\xfd\x07\xfe\x5e\xed\x3b\x54\xe4\xd3\xad\xa0\x54\xe4\x51\xc0\x08\x25\x2d\xd5\x78\x0e\x85\x78\xab\x7d\x87\x8a\xc1\x9a\xfe\x77\xf2\x97\xb8\xfa\xee\x59\x6e\x1f\xf2\xb6\xdb\x47\x00\xc7\xed\x66\xf1\xd8\xed\xeb\xc2\xc4\xd5\xca\xcf\xd4\xed\x13\x3e\xd7\xed\xa2\xdf\xac\xde\xb8\xce\x89\xe8\x90\x0e\xdf\xed\xe2\x5e\xe0\xf6\x49\x5f\xe8\x76\xcb\x2f\x75\xbb\xd5\x6f\xee\xf6\x69\x08\x31\x7b\x54\x84\x90\x1e\x2d\x24\x56\xaf\x1a\x12\xbb\x6b\xa4\xe8\x94\x70\x32\xee\x55\x11\x32\x31\xfb\xf8\x44\xa6\xbd\x96\x8c\xb8\xbd\xda\x42\x66\xbd\xe6\x82\x78\xbd\x06\x8f\xf8\x3d\x36\x93\x04\xbd\x76\x83\x84\x3d\x66\x89\xd0\x0e\xbb\x44\xe6\x9d\x66\x83\x7b\x0f\xdd\x7d\x20\xbd\x7a\x49\x2c\xbd\x62\x12\xbb\x47\xed\x89\xd3\x23\xf8\x64\xdc\xab\x3b\x64\xd2\x6f\xdd\xa6\x1d\xe6\x8d\xb8\xfd\xca\x33\xeb\xb4\x1f\xc4\xeb\xb5\x7f\xc4\xef\x35\xa2\x24\xe8\x34\x22\x24\xec\xb5\x52\x84\xf6\x98\x29\x32\xaf\xdb\x91\xdb\x39\x0f\x4a\x9b\x52\xe0\xab\x5b\x21\x29\xb1\x51\xba\x0c\x07\xd2\x76\x0d\xa5\xc7\x20\x0a\x60\x3c\x45\xe9\x37\x94\x3e\x9f\xe2\xfd\xa4\x00\xa0\x2b\x30\xad\x10\x54\xbc\x95\x79\xae\x73\x19\x2a\xfc\x34\x3e\x43\xd5\x43\x45\x0b\x7e\x85\xa0\x1a\x85\xa0\x56\x40\x35\x70\x68\x75\x8f\xca\xcc\x51\x80\x9e\xd7\x88\xa3\x8e\x39\x74\xd5\x27\x6e\x0f\x71\x2d\xd7\xd4\x09\x8e\xed\x76\x0b\x8e\xe3\x76\x09\xce\xd8\xed\x91\x8b\x89\xdb\x43\xb5\xa9\xdb\x23\x7a\xae\xdb\xc3\xda\x99\xab\xa3\xbb\xe7\xf6\xf0\xd4\x77\xbb\xa5\x36\x70\x7b\xa4\x26\x74\x7b\x38\x47\xdd\x6e\xc1\x9d\xbb\x5d\x62\x4f\xcc\x4e\xb5\x25\xc4\xd4\xf2\x95\x58\x7d\x3a\x4d\xec\x3e\x9d\x24\x4e\x8f\x56\x93\x71\x9f\x52\x90\x49\x9f\xe5\x20\xd3\x1e\xdd\x2e\xc7\x3d\x2d\x1b\xc9\xac\x4f\x81\x88\xd7\x63\x1f\x89\xdf\x67\x41\x48\xd0\x69\xa1\x48\xd8\x67\x61\x08\xd5\x0f\xce\xf3\x1e\x0b\x81\xfe\x41\x37\xaf\x48\x8f\xa4\x11\xab\x47\xd3\x89\xdd\xa7\xcc\xc4\xe9\x53\x56\x32\xee\x33\x55\x13\xbd\x29\x22\xd3\x3e\x63\x41\xdc\x6e\x75\x99\xf5\x29\x3c\xf1\xb4\xc6\x82\xf8\x7d\xba\x4c\x82\x1e\x73\x41\xc2\x4e\x63\x49\x68\x9f\x29\x23\xf3\x86\xc1\xb9\x8d\x57\x20\xd0\x76\x55\x56\xa4\x80\xa9\xf2\x0b\x78\x5d\x4b\xdd\x67\xbb\x7a\x6f\xa9\x60\x3b\x15\x45\x94\xf0\xc7\x72\x7f\x54\x5e\x41\xf9\xb6\x0d\x7b\x5a\x13\x68\xed\xa8\xa8\xf4\x06\x24\xa4\xda\x80\xbd\xa2\x59\x25\xca\xbe\x10\x50\x95\x07\x20\xd1\xaa\xfd\x3e\x94\xc0\xb6\xdf\xd2\xb2\xaf\xed\x77\xf3\x1a\x95\x55\x3d\xed\x64\x12\x71\xbb\x99\x64\xb9\x9a\x1e\xd9\x6e\x17\x77\x1c\xb7\xab\x3f\x63\xb7\x5b\xea\x26\x6e\xb7\x64\x4c\x5d\x3d\x3d\x5c\xb7\x4b\x2e\x66\xae\x5e\x9e\x3d\xb7\x9b\xf5\xbe\xdb\xcd\xc3\xc0\xd5\xc8\x54\xe8\x76\xb3\x88\xba\x5d\x32\x35\x77\xbb\x45\x99\x98\x3d\x7a\x44\x48\x8f\xf0\x11\xab\x47\x53\x89\xdd\x21\x80\xc4\xe9\xd4\x53\x32\xee\x51\x45\x32\x31\x7b\x6c\xd0\xb4\x53\xe7\x4a\x0f\x56\x83\xfb\x4c\x6b\xb5\x3d\x9d\xb6\x12\xbf\xc7\xb4\x91\xa0\xc3\x2e\x92\xb0\xc7\x86\x10\xda\xa3\xb3\x64\xde\x69\xdc\xd8\x88\xae\x41\x9c\x74\x8a\x12\xb1\x3a\x95\x96\xd8\x3d\x7a\x49\x9c\x1e\xc5\x24\xe3\x0e\xcd\x24\x93\x1e\x5b\x43\xa6\xbd\xc6\xaa\x47\x93\xc8\xac\x47\x47\x89\xd7\x61\x00\x88\xdf\x69\xb5\x48\xd0\x69\x5a\x48\x

a8\xd3\x7f\x42\xfb\x54\x78\x5e\x37\x3d\xb7\x1f\xba\x15\x32\x52\xa0\xea\x98\x44\x31\x74\x0b\x5
7\x43\x31\x68\x0b\xa0\xaa\x6a\x4e\xe9\xe4\xa8\xde\x8e\x35\xdd\x9f\x70\x90\x8a\x31\xba\x72\x9
9\xda\x6f\x5d\xa9\x03\xaa\x61\xba\xec\x7b\xbb\xaa\x27\x09\x79\xfb\xad\x2f\x75\x42\x35\x55\x97\
xfc\x38\xc5\x30\xcd\xe9\xd6\x86\x4a\x2b\xba\xa9\x26\xe9\x92\xe7\xdb\xee\x69\x17\x19\x88\xab\x
26\xaa\xe5\x76\xf1\xd7\x76\xbb\xfa\xe8\xb8\x1d\x82\x33\x76\xbb\x88\x37\x71\xbb\x7a\x32\x75\x7
5\xe4\x71\xdd\x0e\xb1\x9a\xb9\x5d\xac\xf6\xdc\x2e\x8e\xf8\x6e\x87\x20\x04\xae\x4e\xcc\x43\xb7\
x4b\x92\xa9\xab\x96\xd8\xb9\xdb\xc1\x64\x62\x76\x72\x99\x90\x4e\x75\xb5\x3a\xf5\x95\xd8\x9d\
xba\x42\x9c\x2e\x75\x20\xe3\x4e\x55\x22\x93\x4e\x85\x20\xd3\x2e\x8b\x20\xc6\x1b\xe5\xab\x59\
xa7\xb5\x20\x5e\x97\xc6\x10\x5f\x63\x34\x48\xa0\x33\xb2\x61\xa7\xe6\x12\xda\x69\x14\xc8\x5c\x
6b\x11\x89\xd9\xc9\x75\xd2\xa9\x88\xc4\xea\xd6\x6e\x5b\x23\x69\xc4\xe9\x54\x34\x32\xee\x52\x
61\x32\xd1\xea\x21\x99\x76\x5a\x06\xe2\x76\x6a\x3f\x99\x75\xea\x22\xf1\x34\xc6\x8a\xf8\x9d\xe
a\x46\x82\x2e\xeb\x40\x42\xad\x16\x13\xda\x69\x39\xc8\x5c\x32\x0e\xb7\x19\x53\x5d\x36\xc0\x5
b\x0a\x80\x25\x71\xda\xf6\xf8\xa0\x5a\xdc\x68\x9b\x63\x5e\xaf\x6d\x88\x05\x3c\xc5\xab\x31\x87\
x67\x29\xf1\x98\x94\x2f\x55\x46\x58\x60\xa2\x1e\x67\x5c\x53\x8d\xff\xac\xec\xb7\xca\x04\x73\x3
c\x55\xaf\xfc\x12\xa8\x02\xcf\xe0\x80\x1f\xf6\x68\x9b\x5f\xb5\x9c\xd0\x92\x88\x8a\x3a\x73\x81\x8
4\xe2\x55\xb1\xa8\xa4\xed\x39\x7f\x4d\xba\x68\x2a\xca\x58\x5d\xfc\x17\x65\xec\x2e\x5e\x8b\xe7\
x4e\x17\xb1\x45\x99\xb1\x9e\xac\xa2\xc4\xa4\xb7\xcf\x53\x8d\x68\x89\xd7\x6e\x17\x45\x45\x99\x
99\x8e\x4b\xe2\xbd\xa7\x97\x52\x51\xc2\xef\x92\x47\x51\x26\x50\xb3\x5c\xbc\x0d\xbb\xc4\x48\x9
4\xa1\x5d\x22\x2a\xca\xcc\xf5\x1a\x5a\x78\xc4\x4a\xc5\x26\x5d\x3d\x20\x96\x86\xc8\xc4\xd6\x49
\x1c\x71\xba\x90\x25\xe3\x2e\xb6\x90\x49\x17\x31\xc8\xb4\xa3\x8b\x3a\xfb\x3b\xd3\xb3\x90\x78\
x5d\x92\x4a\xfc\x4e\x7b\x18\x74\x69\x14\x09\xf5\xf2\x4d\xa8\x4e\xe8\xc8\xbc\x5f\xbb\xaa\xc9\x8
d\xb6\x04\xe9\xb6\x05\xc4\xea\x17\x38\x62\xf7\x69\x1f\x71\x3a\xb5\x8f\x8c\xfb\x8d\x40\xc1\xec\x
ce\xee\x4e\xfb\x8d\x12\x71\xfb\x8d\x1b\x99\xf5\x5b\x83\x42\x1c\xba\xb4\x8c\x0b\x85\xf6\x6d\xd0\
x67\xd6\xb8\x60\x74\xe0\x49\xfb\x2c\x4e\x21\x24\xd8\x8a\x34\xb2\xf3\xaf\x72\x5e\x83\x57\x5e\xf
6\x21\x83\x7c\xe1\xe5\x90\xd1\x25\x0d\x72\xcc\x47\xf4\xee\xbb\xd7\x3f\x43\x14\xaf\x8b\x6b\x22\
xca\x8c\x06\xaf\x9e\xbe\x6b\x5c\x5c\x5c\x1d\x4c\x34\xa0\xda\xf8\x8f\x17\x28\x8a\x1f\xf8\x5d\xfc\
x30\xe4\x8a\xa6\x78\xca\x0b\xf0\x1f\xc5\x77\xf6\xc3\x90\xfa\xd3\xc4\x5c\xca\xaa\xf4\xfc\xe8\x1d\
x4f\x8c\x05\x3c\xf1\x4b\xf7\x1d\x55\xac\x74\x79\x41\x15\xff\x21\x65\x49\xb9\xeb\x15\x55\xdd\xa9
\xf5\x3e\xd0\x9b\x32\x05\xd8\x07\x7a\xa3\x48\x7d\xf7\x81\xde\x14\x79\xf5\x3e\xd0\x1b\x75\x5a\x
3d\xd6\x06\x67\xd1\x78\x02\x7e\x94\x67\xe0\x05\x41\x92\x86\x51\x7c\x01\x79\x02\x6f\x9e\x11\x
25\xdc\xef\x22\x4c\x05\x74\xd6\xcc\x81\xac\xba\x3b\x64\x3c\xd1\xdf\x1d\x52\x81\x7b\x93\x30\x8
0\x6f\x9e\x91\xb3\xe8\x1c\xf6\x80\x28\x72\x94\x8a\x76\x79\x7a\xfe\x41\xd1\xbb\xb3\xaa\xbe\x48\
xc7\xc7\xfe\x19\xd8\x04\xf6\x24\xd0\x98\x87\x6f\x08\xf7\x5b\x80\x15\x09\x4b\x9f\x66\x19\x5d\xf9\
x4b\x0a\x64\x02\xd9\xa5\xff\x81\xde\x28\xc8\x9f\x5d\xfa\x7f\xa1\x37\x59\xc9\x82\xea\xb7\x9e\x28
\xf1\x3b\x2c\xc4\x49\x53\xfc\x78\x04\x64\x52\xfe\xd2\x5f\xb1\xf2\x0c\x33\x4e\x09\x7c\xd4\x84\xcc
\x0a\xe8\x02\x97\x33\x01\xf4\x5c\x20\xa5\x84\xdb\x7d\x75\x8b\x1f\xe5\xef\x30\x2b\xca\xa1\x94\x
04\xa5\x84\xab\x03\xc9\x05\xca\x71\x95\x02\x65\xb5\xeb\xa8\xa4\xc6\x72\xf4\x52\x53\x6f\x67\x9
e\x26\x2b\x34\x30\x4b\x3a\xcf\xc1\x72\x51\x33\x58\xcb\xea\x8a\x9c\x38\x67\x83\x08\xf6\xf9\xdd\
x10\x26\x26\x70\x2c\x84\x6b\x30\x78\xf3\xcc\x12\x32\x38\x84\xdd\x92\x02\x43\xf8\x16\x2c\xf7\x1
c\x73\x3c\xa2\x6c\x45\xf0\x2d\xde\x71\xb1\x35\x7a\x69\x74\xb1\xd8\x1e\x3f\x07\xd3\x77\x56\x48\
x0e\x6b\x58\x5a\x2e\xbe\xe6\xb8\xc2\x2e\x58\x8e\x06\xe1\xa1\x02\xe3\x56\xb3\xaa\xcc\xfe\xac\x
03\x51\x1c\x50\xa0\x5e\xb0\x10\x62\x07\x51\x06\xde\x7a\xbd\x8c\x68\xc8\x78\xe9\xc5\x40\x37\x

6b\x2f\x0e\x69\x58\xe4\x65\x44\xf3\x6e\x28\xa1\x31\x12\x08\x30\x81\x17\x83\x4f\xc1\x4f\x93\x0f\x34\x86\x28\xce\x13\x70\x79\x52\xe0\x0c\xb2\xc0\x5b\x72\xf0\x1c\x64\xa6\x86\x76\xbd\x88\x82\x05\x78\xcb\x65\x72\x9d\x21\x68\x06\x37\x4f\x18\xd8\xcb\x8c\x86\x70\x1d\xe5\x8b\xe4\x32\xe7\x08\x66\x51\x12\xb7\xa1\x08\x42\x63\x7a\xcd\x41\xf5\xe3\xd1\x23\x71\xad\x4c\xf5\x88\x19\x14\x9b\xa8\x28\x57\x93\x5c\xc2\x25\x77\xda\x2d\xb8\x02\x2c\x1a\xb1\xea\x3b\xda\xac\x41\xc4\x99\xf8\x00\x18\xf7\x6d\x35\xab\x74\xfd\x98\xca\xfd\x98\x9e\x8b\xc4\x9e\x9f\xe4\x47\x78\x29\x40\xeb\xaa\x1d\x85\x05\x7c\xc6\x13\x5f\x42\x14\x5f\xd1\x34\xa3\x7a\x2b\x18\xc5\x57\xef\x1a\x86\xb0\xf6\x68\xab\x01\x82\x74\x0c\x10\x15\x34\x99\x62\xd9\x19\x19\x33\x81\x6e\x42\xff\x5c\x0b\x38\x54\x3f\x68\x1c\xa4\x37\xeb\xfc\x16\x57\x01\x8a\x8c\xb5\xc9\xb3\xb2\x5e\x55\xd8\xa8\x9b\x7c\x6d\x0a\xdd\x90\xfe\x1e\xad\x56\x14\xe9\xca\xdd\xfb\xac\xbb\x65\xa3\x20\xa4\xca\xe9\xf8\x9e\xe6\xb2\x9f\x56\x47\x6e\x89\x40\xa5\xab\xb1\x9a\x3c\xe0\xc5\xd2\x66\x31\xbc\x39\x4b\xe1\x7d\xbc\x8c\xa3\x3c\xf2\x96\x72\xea\xab\x7a\x19\xba\x09\x16\x5e\x7c\x41\x8f\xdf\x56\x69\x51\x79\xe6\x31\x73\x63\xce\xf9\x7f\x4d\x91\x56\xd7\xe1\xf7\x53\xe3\x8c\x75\x3e\xd7\xd6\x79\x7b\x2c\xd7\xb1\xb0\x1d\x5b\x7c\xb6\xab\xe3\x72\xdc\xcc\xf9\x9c\xfd\xbf\x25\x6e\x58\x67\x2c\x3e\xca\xcc\xb4\x5d\x57\xb5\xf1\xf4\x61\xa8\x51\xfc\x2b\xd7\x2a\xfc\xde\x7f\x6d\x9b\x62\x24\x52\xfa\x13\x08\x4e\x77\xed\x45\x29\x18\xb2\x9c\x68\xca\xa6\xf5\xb2\xa9\x28\xab\x44\xf2\x05\x8d\xb2\x9c\x2e\x4b\x29\x56\x43\x9c\x63\xe7\xb7\x73\x2d\xdc\x6e\x03\x3d\x67\x03\x2d\x4f\xb5\x76\x16\x9d\x9f\x0d\x06\x02\xdb\xf7\x95\xb9\x66\x8e\x64\x39\x75\xc1\xdf\x98\x56\x5b\x45\x1a\x85\xc1\x6e\x28\x52\xaa\xa3\x54\x43\x93\x96\x05\x1a\xf3\x7e\x03\xfe\x63\x1c\x26\x90\x5d\x7b\x6b\xee\x7e\x2c\xbd\x2c\xe7\xc2\xd0\x36\xe1\x79\x37\xcb\x1a\xc8\xd6\x19\xd6\xa5\xf8\xb9\x42\x86\x31\xa3\xf8\x6d\x55\xbd\xa5\x1a\x5f\x4d\x05\xef\xa2\xea\x77\x31\x29\x3d\xa6\x4b\x31\x23\xcb\x21\xb9\xcc\x5b\x16\xb8\x34\xb9\xdd\x2c\xab\x99\x5c\x3d\xcf\x6a\x43\xc6\x07\x7a\xc3\x53\x40\x4f\x9c\x7d\xdb\x92\xdf\x44\x57\x9a\x17\x52\xde\xe8\x89\x32\x6b\xf4\x3e\xbc\x63\x12\x28\x26\x01\x69\x92\x65\x95\x9b\x8e\x39\x0f\xd1\x21\xc6\x69\x29\xaf\x51\x0e\x54\x15\xe1\x06\xc5\x78\xb5\xf2\xb2\x0f\x35\x95\x2d\x64\x77\x30\xa8\x89\x28\x53\xc4\x62\x74\x7d\x5f\xeb\x3a\x53\x5a\x06\x45\x22\x41\x4d\x64\xdf\xa3\xcc\xfe\x49\x29\xf8\xec\x1d\xf3\xa8\x38\x64\x51\xaa\xd0\xbb\x16\xda\x6f\x8f\xb7\x47\x3b\xd5\xa3\xbd\xec\x46\x7b\xd9\x81\x76\xba\x05\xda\x9d\x49\xa4\xb3\x22\x8b\x34\x0f\x7f\x6c\x97\x47\xba\x2f\x09\x33\x87\x95\xd3\x4d\x2e\xa7\x62\x7e\x7e\xf4\x6e\x24\x1c\xb4\x5a\x2e\x66\x03\x82\xf9\x85\x22\xb9\xf6\x7a\xe9\x31\x24\x36\x39\x34\xa1\x08\x87\x6b\x50\xb5\xa3\x02\x54\x66\x76\x6e\x07\x6a\xea\x49\xb7\x9f\x1f\xbd\x53\x66\xdc\x3e\x4d\xa3\xf5\x92\xee\xdd\x2e\x44\xc4\x2b\xd5\x02\x45\xf2\xa3\x7f\x9d\x70\x91\x08\x44\x30\xb4\x23\xcc\x50\x1a\x34\xaf\x07\x12\x5e\x2c\xcd\x08\x1c\xb2\x72\x23\x4e\xd5\x23\xce\xe3\x24\x1d\x54\xf7\xac\x8b\x8b\xe3\x8b\xa6\x47\xd9\x32\x0a\xe8\xc0\x34\xc0\x1a\xb6\xee\xc2\x28\xc1\x5a\x77\x04\x6b\x19\xe0\x74\x80\xb5\xef\x08\xd6\x31\x60\x32\xd4\x5f\xa4\x71\xe7\xb9\x07\xcd\xc8\x48\xae\x2c\xd5\xd0\x52\x66\x24\xcf\x39\xb6\xa8\x60\x6f\xd1\xc2\xd7\x99\xd3\xb0\xb6\x6e\x89\x9c\x75\xdb\xee\x93\x2d\x5a\x50\x8f\x7a\x64\x66\x7d\xb5\x61\xef\xbf\x88\x59\x2d\xad\xcb\x57\x30\xae\x15\xac\x5b\x9a\x58\x9d\x89\xab\x1b\xda\xb2\x54\x67\xfe\xfc\xb2\x54\x23\x85\xbe\x94\x98\xfd\x60\x6c\x19\x8d\xac\xfa\x52\x72\xf7\x83\xb1\x63\x54\x59\xdd\x0f\xc6\x13\x43\x24\x7b\x3f\x98\x90\xcf\xe7\x86\xeb\x7c\x51\xc2\xfd\x7f\x66\xa6\xfd\xdf\x2d\x1f\xfe\x7f\x4e\x66\x7b\xbc\xa9\x20\x8a\x69\xf8\x75\x53\xdc\x7f\xe7\x65\xb4\xca\x5a\xef\x65\x54\x7a\xf7\xb3\x6d\x75\x66\xc0\x6f\xeb\xf2\x66\xe2\x40\xec\xad\x68\xb6\x96\xb5\x74\x5f\x46\x83\x15\x61\x68\xf0\x7f\xff\xfe\x59\x05\xe6\x29\x4c\x9c\xf2\x0a\x1b\x15\x98\x9f\x27\x0e\xc3\x03\x91\xda\x4c\x9c\x91\xf8\xc1\xf0\x57\x78\x06\x15\x68\x0e\x5e\x84\x53\xa2\xbf\xd1\x0c\x3c\x88\xe9\xf5\xf2

\x06\xb8\xae\x85\xaa\x86\x65\x83\x02\xb5\xdb\x3c\xe2\xcb\x95\x4f\xd3\xcf\x80\xb7\x4a\xe1\xad\x2a\xec\x8b\x6d\xa1\x3b\x3f\xea\xac\xb2\x4c\xae\xb1\x06\xfb\x57\x55\xa1\x5e\xb9\x6e\xdd\xda\x05\x0a\xba\x6c\x2a\xba\x14\x16\xa1\x20\x4f\x31\x30\xf3\xd5\x3f\xd3\x32\x6d\x9c\x95\x39\xe6\xd8\x9c\x98\xf5\x78\x67\x41\x69\x34\xf1\x71\x54\xf3\xa8\x58\x0f\x0d\x86\xb5\x7a\x0c\x13\xf7\x6b\x29\x6e\xf5\xc4\xd7\xac\xb7\x87\x50\xbf\x7d\x5b\x9e\x99\x37\x39\xf5\x5d\x94\x5f\x47\x19\x85\x93\xd7\xa7\x19\x42\xe8\x63\x4c\x71\x51\x8a\x10\x90\xcf\xf0\x94\xf1\x97\xd1\x65\x0f\x09\x23\x46\x12\x6f\x9e\xd3\x14\x62\x7a\xe1\xe5\x51\x7c\xf1\x15\x08\x8f\xa0\x28\x23\xbc\x60\xc1\x28\x4e\xf2\x81\x96\xaa\xfb\xfb\x10\x27\xbd\x9e\x2a\xde\xc9\xc2\x09\xfa\x8f\x92\xba\x0f\x95\xc5\x38\x61\xff\x51\x10\x59\xe1\x92\x0a\xca\x08\xc2\x14\xd2\x50\xb1\xf3\x61\x0d\xbb\x9a\x07\xa0\xe3\xca\xd3\x93\xe7\x12\x57\x70\x39\x01\xc7\xed\xb5\x97\xe1\xf2\xc2\x56\x3a\x54\x72\x0a\x61\x30\x95\x28\x99\x95\x27\xac\x89\x02\xee\x57\x66\xfe\xd3\x93\xe7\x5f\x87\xf5\x7c\x6d\xa7\x62\xbc\x17\x87\x03\x2f\x4e\xf2\x05\x4d\x05\x22\x5d\x62\xe0\xc5\xa1\x2c\x06\xac\x87\x3d\xa2\x50\xe9\xd9\x7d\x4e\x90\x3e\xa9\x28\x35\x4f\x94\xff\xa7\xc9\xc7\xeb\xb7\xbf\xb7\x78\xbc\x7e\xfb\x3b\x49\xc7\xeb\xb7\x5f\x47\x38\x92\xb4\x26\x1b\x49\x7a\x0b\xd1\x48\xd2\x3b\x4b\xc6\xa7\x5b\x4a\xc6\xa7\x7f\xb2\x64\xfc\xfc\xfb\x8b\xc6\xcf\xbf\x9b\x6c\xfc\xfc\xb5\x84\x63\xd3\x90\x8e\xcd\xad\xc4\x63\xf3\x05\xf2\xf1\xfe\x96\xf2\xf1\xfe\x9f\x24\x1f\xb8\x28\x2f\x4b\x46\xcc\x23\xa3\x62\x42\xb8\xa4\xf3\x7c\x7b\xaf\x2c\x46\x99\xe0\xbf\x20\x99\x97\x90\xf0\x0a\x9b\xaf\x25\x0c\x08\xec\xeb\x88\x03\x82\xaa\x09\x04\x3e\x39\x1e\x58\xe3\x2e\x39\xe0\x85\x64\x51\x88\x55\x72\xc0\xa6\x40\x31\x3c\x02\xdb\xd2\xad\x74\x49\x92\x32\xa8\x44\xe5\xd1\x23\x88\x71\x89\xbc\x14\x06\xbe\x75\xc8\x82\x3d\x88\x95\x97\xd5\xab\x45\x88\xc1\x69\xcb\xda\x67\x28\x26\x4f\xdd\x08\xc9\x60\x06\x31\xec\x29\x6e\x0c\x6d\x35\xdd\x5c\xea\x62\xcd\xfd\x67\x4a\x2f\x86\xf2\xff\x8f\x13\xdf\xb7\x03\xfd\xe4\xa2\x90\xde\xb7\x5f\x49\x7a\x39\xdf\xeb\x92\x2a\x09\x6f\x21\xcf\x5b\x08\x6f\xcb\x62\x22\xa8\x3b\xc8\xaf\xa4\x05\x25\x9c\x7e\x01\x16\xcd\xff\xd3\x25\xf8\x6d\x92\x7b\x39\xfd\xbd\x0d\x70\x8a\xad\x7c\x2d\x11\x46\x68\x5f\x47\x84\x39\x62\xb2\x08\xa7\x49\xaf\xfd\x65\x45\x7a\xe5\x57\xf4\x08\xe5\x40\x58\xf5\x78\xc8\xdc\xc1\xea\xc9\xdb\xc1\xc4\x69\x89\xe5\x97\x32\xec\x2b\xd9\x9c\x7f\x2d\x8e\xf5\x98\x1c\x56\xe2\xf6\x0c\x7b\xdb\x62\xd8\xf1\x5d\x18\xf6\x34\x0c\x7f\x6f\xcf\xd7\x0b\xc3\xdf\xc9\xf3\xe5\x57\x7e\x7f\x8d\x39\x73\xd8\x98\x33\x87\xb7\x9a\x33\x87\x5b\xcf\x99\x9b\x23\xc2\x6e\xe9\xc8\xe2\x86\x51\xb5\xf3\x1b\x78\x69\x7a\xc3\xaa\x15\x63\x08\xbf\x18\xbe\x36\xac\x54\xd7\xc3\xab\x61\xb4\x1d\xa9\xdd\xca\xe7\x86\x5d\xde\x86\xc0\xe1\x4b\x2d\x3a\xff\xa5\x5e\x5d\x79\x1a\x8b\x2b\xc0\x93\xb9\x1c\xdb\xcc\x54\x37\x1c\xa7\xc9\x9a\xa6\xf9\x0d\xfc\x5d\x5c\x31\x8c\x05\x51\xbc\x4a\x10\xad\xb0\xa2\x10\x90\x6c\xa4\x82\x53\x98\x95\xf2\x4e\xf4\xba\x75\xc9\xa2\x8b\x38\x9a\x47\x81\x17\xe7\xe0\xe3\xfb\x28\x96\x74\x03\x1b\xed\x88\xfe\x56\x71\xe9\x02\x99\xe2\xc9\x57\x88\x03\xb7\x31\xd0\xab\x63\x8d\x5c\x83\xd7\x6b\x26\x96\xde\x72\x58\xa3\x7d\x2f\xe1\x40\x69\x90\x4b\xca\x49\x60\xb7\x22\x22\xad\xb3\xf9\x0b\x74\xf5\x5a\x26\x75\xb3\x17\xb5\x35\xdf\xba\xce\x7e\x21\xb0\xb3\x56\x7d\xf6\xb9\x6d\x58\xdb\xb8\x2d\x14\xe2\x92\x19\xf1\x88\x8f\x67\x6a\x02\x12\x12\x4a\xe6\xc3\x16\x90\xf3\xff\x83\xba\x6a\x00\x31\xb7\x5e\x1e\x40\xa1\x33\x4a\xb1\x6d\x99\xe5\x6b\xb1\x79\x02\xcd\x62\xf1\x83\xff\xfb\xe9\x93\xe2\x00\x06\xf3\xfb\x4b\x1d\xb8\x77\x08\xed\x55\x30\xf9\xc3\xc7\xe6\xa2\xf8\x61\x89\x46\x73\x2f\xa0\xd6\x69\x6f\x02\xe0\x3a\xb4\xa4\xf1\x45\xbe\x80\x07\xe0\x6e\xb9\x95\xba\x69\x68\x9e\x25\xf1\x15\x4d\x8b\xa9\xa1\x64\x86\x85\x7d\x60\x83\x76\x71\x3a\x60\x2b\xc3\x53\x8c\xda\x25\x77\x6b\x2b\x73\x9f\xe1\xb4\x6e\x44\x77\x32\x08\xbd\xdc\x03\x2f\xbb\x65\x3b\x5b\x47\xb2\xea\x2b\x85\x1b\xc9\x40\x8f\xf2\xe4\x67\xdb\xd2\x2f\x85\xe0\xeb\x2f\xd8\xb3\x23\xda\xaa\x0b\x95\x62\xe7\x4e\x51\xee\x98\x33\xb3\x44\xb2\x60\xaf\x6a\

x17\x0f\x67\x9b\x02\x16\xef\xee\xd6\x9b\xf7\xeb\x6d\x77\x9f\xf4\xaa\x96\xf0\x8a\x5a\x67\xad\x2d\xfc\xec\x53\xe0\x30\x5a\x5f\x66\x8b\x41\xe1\x48\x31\x1f\x41\x35\xaf\x54\x97\x6e\xf8\x12\xa0\xd8\x27\x5b\xb8\x22\x12\x83\x0b\x0b\x52\xc0\x34\xea\x6a\xa3\xdd\x48\xd2\xd2\x0a\x04\xc3\x44\x32\x48\xd6\x38\x48\x6a\xc6\x7e\xe8\x75\x5b\x4b\xb1\xa7\x10\x2c\x93\xb8\x6b\xa6\xb2\xad\x48\x23\x9c\xa6\x2c\xe3\x43\xbd\x2c\xe3\xeb\x4e\x59\x96\x21\xa3\x97\xc2\xd1\x2d\x77\xbe\xaa\x76\xba\x3e\xc3\xf2\xff\x86\x82\xfd\x6f\x9c\x32\x6d\xa0\x85\x2d\xe5\xf0\xda\x66\xb6\xd8\x35\xa6\x6f\x00\xcf\x30\x15\x0b\xeb\xdc\x39\xd1\x34\x53\xaa\xd0\x75\x4d\x7f\x7a\xd5\xe0\x7a\x1b\x1d\xb8\x16\x22\x5f\x80\x3f\x8b\xce\x55\x64\xd7\x8b\x2a\x16\xae\xad\x2f\x97\xee\xb1\x76\xdf\x4c\x63\xb7\x8c\xd8\x1a\xf3\xf9\xdc\x70\xc7\xdb\xec\x77\xd9\x7f\x70\x0f\x16\x79\xbe\xce\x0e\xf6\xf7\x57\xf9\x22\x1b\xf9\x74\xff\x32\x9f\xbb\xbf\x65\x70\x65\x8d\xc8\xc8\x02\xff\x06\xfe\xc7\xca\xcb\x17\x91\x97\x31\x89\xa9\x36\xc8\xe0\xae\x10\xbe\xd9\x63\x7f\x1f\x9e\xd3\x9c\x1f\x87\xa3\x94\x91\x3b\xf2\xfc\x25\xcd\xe0\x3f\x44\x4b\xff\xf1\xcd\x9f\x70\x1b\x7f\x4a\xe9\x51\xb9\xff\xa5\xb5\x93\x06\x76\x38\xf3\x76\xe0\xfe\xfd\xe2\xf1\x43\x3d\x78\xf8\x0f\xde\x1d\x09\xf8\x2b\x7c\x50\xc1\x5e\x89\xdf\x75\xd0\xe2\xe9\xfd\xfb\x8a\xfd\x39\x87\x35\x24\xcb\xc2\x9d\x68\x5c\xe0\xce\x99\xff\x30\xf8\x6e\xfc\x93\x24\xa4\xa3\xdf\x32\x48\x52\xf8\x8e\x6f\xa5\x89\xe6\x11\x0d\x21\x48\x42\x6a\x20\x14\x2f\x0e\xe1\x32\xa3\x10\xe5\x6c\x5c\xfb\x0f\x46\x47\xa9\x0f\x62\x1f\x4e\xd9\x87\x0b\xf1\xbb\xde\x07\xfe\xf4\x21\xdf\x93\x54\x55\x1b\x95\xa5\x0f\x65\x60\x9f\x3e\x49\xbf\x46\xd7\x51\x1c\xb2\xd9\x65\xad\x0c\xdf\x3a\xc4\x70\x01\xf9\x31\x6e\xf6\xf9\xe6\x4f\xfb\x0f\xf6\xbe\xda\xe7\xc1\xfe\x37\xbc\xb7\x59\x9e\x46\xf1\xc5\x8b\x34\x59\x3d\x5b\x78\xe9\xb3\x24\x64\x9c\x7b\x87\x0f\x47\x73\xe9\xa9\x20\xfe\xa9\xf7\x81\xc6\x9c\xc6\x4d\x91\x5d\x5f\xc6\x37\x8c\xbe\xdf\xfc\xa9\xb4\x60\x97\x41\x66\x85\x94\x3d\x1c\xf0\x76\x78\x07\x71\x69\x13\x37\xdf\x17\x43\x20\x3e\x0a\x92\xcb\x38\xa7\xa9\x88\x5c\xe2\xa3\x65\x61\x2b\x78\xf5\xca\x58\xe0\x5b\x3c\xcf\x58\xfc\xa0\x9b\x3c\xf5\xd8\x8f\xeb\x45\xb4\xa4\x30\x28\xa0\x3d\x12\x40\x78\xd3\x7f\xc2\x3a\x15\xc0\x40\x74\xef\x69\x5e\x54\xd8\xdd\x65\xaa\xfe\x27\xe4\x29\x2f\xfc\xf8\x10\xcc\xcd\x73\xd7\x34\x19\xcf\xf9\xa3\x47\xf8\xe8\xbb\x17\x2f\xd8\x23\x4d\x4b\x8c\x5c\x38\x5d\xcf\x2e\xd3\x34\xb9\xf0\x72\x6a\xa0\xd4\xe5\x0b\x9a\x52\x3c\xe7\x09\x31\xdd\xe4\xc0\x50\xf0\x82\x9c\xa6\x58\x09\xbb\xb1\x0d\x7e\x88\xe0\x80\x17\xbf\x0f\xe6\xe6\xc5\x33\xd3\x1c\x32\x09\x35\x37\xcf\xf1\xeb\xdf\x99\x71\x5e\x26\xd7\x55\xfb\x58\xed\x4f\x9c\xf2\x7c\x28\x1f\x88\x2e\x32\x00\xf6\x8b\x17\x43\x3c\x9a\x69\x0e\x61\x17\x24\xc8\xf8\x62\xb7\xc8\x38\x24\x5a\xaf\xbc\x60\xd1\xd5\xcb\x78\xe5\xe5\xc1\x82\x86\x55\x7b\x0f\x21\x89\x97\x37\xe0\xad\xd7\x14\xfb\x1d\x65\xa8\x80\x70\x19\x47\xb9\xc1\x26\x9a\x81\x97\x51\x9c\x6d\x32\x42\x94\x90\xca\x32\x8c\x48\x79\xb1\x2f\xaa\x84\xca\x86\x7a\x4f\xfa\xb9\xf6\xa2\xb4\xdd\x33\xec\x97\xc0\xf5\x4f\x82\x74\x7b\x7b\x02\xf7\x6f\x9a\x1d\xd0\xd4\x64\x05\xd9\xff\xc2\xde\xf3\x52\x85\x36\xde\x45\x19\x68\x8c\xca\x80\xa3\x70\xa5\x0b\xa5\x94\x73\xbf\xa5\x2e\xe4\x51\x1c\xd2\x0d\x1c\xc2\x1e\x51\x8a\x7d\xa9\x47\x3b\x3b\x92\xf0\xef\xee\xf2\x6a\x1a\xe1\xc7\x76\xce\xb0\xc8\x79\x53\xd8\x99\x28\xbd\x60\x1c\xe7\x94\xe1\x4f\xf7\x0e\x0b\xf6\x3f\x94\xe8\x05\xbb\x87\x0a\xfb\x51\x00\x7a\xfc\x18\x88\x59\x08\x10\x7c\x12\x3a\x24\x58\x52\x60\xc2\x85\x15\x3e\x41\x4d\x0e\x4b\xe2\x6f\xd1\x10\x02\xd4\x31\xa9\x24\x7e\xb0\xa0\xc1\x87\x77\x81\xb7\xf4\xd2\xbf\xb2\x5a\x03\xc6\x87\x37\x49\x14\xf3\xdd\xd4\x48\x80\xf2\x51\x5d\xe3\xab\xc7\x5c\xeb\x2b\xe2\xe4\x8b\x34\xb9\x86\xa3\x34\x4d\xd2\x01\xf6\x6a\xe7\x98\xb9\x42\x95\x68\xfe\xb8\xbb\x03\xbb\x15\x80\x51\x9e\x70\xcb\x3a\x20\x93\xe1\x28\x4f\x7e\x5c\xaf\x69\xfa\xcc\xcb\xe8\x60\x08\xbb\x1c\x00\x13\xf9\x38\xc9\x99\x80\x23\xb2\x9c\x2e\x3b\xec\x65\xd1\xd1\xcf\xbf\xc3\x48\x50\xd1\x09\xbd\x6a\xe6\x89\x57\xe4\x30\xf8\x32\x9b\x18\x9c\x38\x95\x15\xdc\x18\xc8\x04\x7c\x5c\xd4\xe1\x1c\xc5\x50\xe5\x

c6\x35\x87\x4d\xbe\x70\x85\x78\x56\x54\x54\xb1\x45\x02\x7b\x5f\x08\xe7\x8b\x17\xae\xb0\x75\x
c2\xcc\x91\x3d\xff\x26\xa7\x90\xd1\x8f\x97\x34\x0e\xd0\xd0\xe9\x11\xad\xda\x28\x44\x07\x07\xc2
\x9b\x95\x9f\x2c\x4b\x45\xd2\xb5\xec\x9a\xf5\x96\xad\x76\xcb\x25\xa4\x7e\x22\x4d\x38\x81\x88\x
20\xd0\x33\xb3\x44\xa9\xdc\x78\xac\x40\x02\xcd\xb0\x8c\x84\xdd\x46\xa2\x43\xe0\x1f\xde\x12\x4
9\x62\x71\x2c\x4d\x81\xe5\x91\x59\x03\xb1\x7b\xa8\x91\x9a\xc9\x16\x9d\x39\x32\x5b\x9d\x71\xb
e\x88\xa2\xc4\x15\xc8\x4e\x39\xb2\x2f\xb6\x44\x96\x58\xb7\xed\x54\x55\x52\x85\x55\xbd\xa3\x7
5\x0d\x28\x65\x13\x21\x34\x55\x82\xb9\xfe\x62\x9c\x68\x3a\x4d\x25\x50\xe6\xba\xb7\x9d\xab\x9
6\xd7\x54\x95\xef\x1d\x54\xca\xa2\xc5\x03\xc6\x04\x6e\xad\xb6\x1c\x5c\xaa\x1e\xcb\x0d\xcb\xa3
\x8c\x04\x72\xf7\xb0\x43\xf5\x1b\x16\xbd\xaa\xf6\x7b\x39\xc2\x25\xed\x53\xea\x85\xcf\x92\x38\x8
f\xe2\x4b\x3c\x3c\x8b\xdc\xaf\x4c\x11\xc3\xe4\x25\xf6\xfd\xf1\x21\xa2\xf5\x8c\x39\x16\x8a\xd1\x6
0\xe7\x65\x7c\xe5\x2d\xa3\x10\x0b\x71\x6a\xef\x88\x6e\x95\xf4\xae\xb7\x02\x1c\x20\x06\x0a\xce\
xca\x76\xce\x85\x9a\xb0\xaa\xe5\xc3\xdd\x5d\xe6\x8c\x17\x16\xaa\x01\xe6\x3e\x37\x23\xdc\x11\
x64\x56\xf2\xef\x92\x31\x54\x96\xb6\x5f\x94\x88\xed\xef\xc3\xcb\x39\x5c\x53\x60\xfe\xda\xe5\x1a
\x98\xa7\x6a\x40\x94\xff\x7f\xff\xd7\xff\x5d\x0c\x4b\x32\x08\xc4\xf8\x1b\x4d\xcf\x5b\x05\x77\x5a\x
c6\x9f\x4b\xef\x3b\xd4\x82\x41\x25\xe5\xac\x30\x91\xc5\xd0\x92\x7f\xd8\xf2\x0f\x47\x21\xbe\x6d\
x5e\x7d\x01\xab\xea\x90\x0e\xdb\x5c\x17\x94\x9d\x7b\x4b\x3c\xfc\x50\xd2\xf1\x2d\xf5\x42\x98\x4
7\x69\x96\x17\x54\xc2\x6e\xdd\x9e\xcd\xed\xd1\x0d\x06\x71\xd2\x26\x6f\x36\x2c\x64\x82\x37\x74
\x5f\xf0\x5f\x58\x56\x09\xd7\x92\xbe\x05\xae\xed\x31\xac\x01\xe7\xa8\x10\xa8\x67\x05\x28\x64\x
0b\x1c\x6a\x14\xe6\x61\xd3\x1e\xc8\xc0\x08\x9f\x66\x60\xce\x9d\x92\xbb\x2a\x07\xac\x94\xde\x4
a\x7c\x25\x1b\x55\x77\xe0\x6f\x21\x82\x85\x5b\xcf\xfb\x6e\x37\x69\xbb\xf2\x6e\x20\x8a\x83\xe5\x
25\x4e\x42\xd8\xe4\x42\x9e\xd2\xa8\xa8\xfc\xa2\xa0\xce\xd1\x2d\xa8\x83\xa2\x7c\x37\x02\x9a\x6
2\x9e\x66\xe1\xde\x24\xde\x96\x4c\x50\x5b\x47\x50\x13\x9d\x17\x4e\xb0\x3e\xff\xe0\xf7\xa4\x79\
x7b\x84\x6f\x52\xd4\x15\x14\x7d\xf1\x75\x29\x8a\x26\xe3\x8e\x44\x9f\x22\xd1\xcd\x4d\x93\xec\xe
6\xc6\x7c\x36\x84\x4f\x48\x91\x01\xc7\x81\x3f\x2d\xf9\xe1\x68\xf9\x81\x33\x2a\xc5\x1c\x83\x98\x
f2\x14\x4c\xcd\x89\x82\x9e\x4a\x2e\xfc\x78\xfa\x62\xcf\x85\x10\x23\x65\x34\x2c\x2d\x6f\x61\x36\
xc5\x09\xac\xf2\x37\x1a\x34\xe9\x37\xda\x9f\x87\x0d\x9f\x44\xf8\x1a\xd5\x68\xcc\xf1\x2b\xe1\xd5
\x5d\x12\xa9\x58\x61\xd5\xb0\x15\xd9\x00\x4a\x4e\x89\x64\x63\xab\xe8\x4f\xcd\xdd\xa9\xe2\x44\
xf9\x6a\x2d\x79\x23\x83\x7c\xb5\x86\xc3\xc6\x58\x32\x84\x7b\x87\x87\xdc\x28\x37\xbd\x13\xb1\x
88\x91\xaf\xd6\x4d\x3f\x43\x9a\xa0\x57\xa5\x87\xbf\x67\xf0\x8d\x91\x15\x0e\x11\xc1\x9d\x2b\x9a
\x66\x51\x12\xef\x1c\xc0\x0e\x06\x7d\x77\x0c\xf6\x94\xe3\xb3\x73\x20\x79\x85\xf8\x9c\x77\x57\x
3c\xe7\x3f\xbe\xf9\xd3\x67\x11\xa4\x7b\x97\xac\x28\x3c\x7d\xf5\x1c\xfc\xcb\x68\x19\x42\xb2\xce\
xa3\x55\xf4\x37\x9a\x66\x06\x2c\xa3\x0f\x14\xd2\xd1\x6f\x99\xc1\xa7\xc4\x18\x69\xcf\xd6\x34\x8
8\xe6\x51\xc0\x94\x37\x8c\x90\xe1\x6b\x2f\xcf\x69\x1a\x67\x08\x0f\x2b\xe5\x0b\x0a\xf3\x64\xb9\x
4c\xae\xa3\xf8\xe2\x80\xc7\x3c\x99\xf8\x35\xce\x45\xc2\x4e\x21\x34\x3b\x3c\xb8\x5b\x2b\x30\xf2
\x56\x61\x23\x8a\x5a\x1e\x91\x64\xef\xbe\xf9\x13\x67\x97\x38\x34\x59\x86\xb9\xeb\x03\x18\xeb\
x33\xf2\x0e\x99\x53\xcd\x2e\x1a\x51\xe3\x7b\xd2\xef\x51\x9c\x84\xf4\xf4\x66\x4d\x2b\x67\xae\x8
a\x55\x8b\x89\x47\x14\xcb\x71\xe3\xb7\x51\x7c\x91\xfc\xcf\x77\x70\x65\x8e\xdc\x91\x89\xd3\xf3\
xaa\x86\x74\x96\xb4\x44\x46\x98\xc6\x02\x92\x97\x5e\x2f\xbc\x65\x03\xd2\x74\x64\xee\xf1\x40\x
4c\x5a\xec\x8d\xe2\xa7\x18\xc5\xb3\x85\x97\xbd\xbe\x8e\xdf\x14\x5b\x60\x0e\x45\xa1\x51\xfd\x3
9\x16\x2f\x97\x48\x30\x6b\x1c\x27\x4a\x61\x31\xea\xc5\xf9\xfa\x10\x7b\x8f\x07\x89\x87\x8c\x36\x
32\xad\xce\x3e\xf0\x04\x86\xac\x04\x7e\xaf\x05\xbf\x1a\xfd\x7a\xbb\x88\xe2\x84\xf5\xca\x83\x6b\
xea\x83\x38\xa8\x2a\xa2\xd6\x23\x21\xd0\x82\x26\x9f\xbf\x11\x47\x54\x71\xd9\xe4\xb3\xf1\xf7\xc

f\xe7\x86\x3b\xd9\x66\x49\xa4\x75\x62\xf7\xe7\x57\xc7\x3f\xe4\xf9\xfa\x2d\x1b\x32\xb2\xbc\x84\xf6\x6f\x7e\x74\xc1\x37\xb3\x8c\x7e\xcb\xfe\x6d\xdb\xc5\x16\xb9\x12\x5c\x59\x23\x73\x34\x2d\x03\x78\x17\x51\xbe\xb8\xf4\x47\x41\xb2\xda\x7f\x15\x7d\xa0\xaf\x82\xe5\xbe\x5c\x7c\xff\xf8\xe5\xb3\xa3\x93\x67\x47\xc0\x74\x58\x3e\xa8\x7c\x51\x06\xf0\x01\x00\x76\x2e\x33\x8a\xd3\xc2\x20\xdf\x79\xf8\x0d\x3e\xda\x7f\xf0\x0d\x5f\x51\x52\xb4\x2e\xde\x3c\x85\xff\xe9\x5d\x79\xef\x82\x34\x5a\xe7\xb0\x8c\xfc\xd4\x4b\x6f\x50\x41\xbd\xd4\x8f\x72\xf6\x6b\x6f\x9d\xd2\x20\x62\x76\x02\x3c\xcc\x83\x41\xf3\x28\x18\x89\xea\x5b\x76\x41\x94\x7e\x96\xac\x6f\x78\x7a\x98\x41\x30\x04\xcb\x24\x63\x78\x15\x05\x0b\x8f\x2e\xe1\x55\xb0\xf4\x2e\x2f\x16\xcb\x28\x86\x47\xaf\xdc\x60\xe1\xba\xcb\xff\x71\xb1\xf2\xa2\x25\x83\xf9\x58\xd4\x7f\xf5\xf2\x14\x8e\x36\x6b\x2f\x87\xe3\x28\xc0\x61\x5c\xac\x67\xf2\xee\xe2\xe9\xe0\xe8\xe2\x04\x5b\x35\x80\x67\x7a\x30\x60\xed\xa5\x19\x3d\xb9\x5c\xd1\x34\x0a\x8c\x6f\x8a\x45\xb6\x28\x13\x8f\xe0\x10\xf6\xdf\xef\x3d\x19\xfc\x12\xee\x0e\x7e\x19\xfd\x12\x3e\x18\x3e\xf9\xc4\xfe\xdd\x1d\x0e\xe8\xd9\xee\xde\xf9\x13\xf6\xf5\xc9\x9f\xf7\xa3\xaa\xee\xca\xcb\x17\x01\x8d\x96\x70\x08\xaf\xbc\x7c\x31\x62\xdf\xeb\x6f\xe7\xcb\x24\x49\x8b\xd7\xf8\xa3\x7a\x1f\x27\xf9\x77\x09\x0f\xfb\x88\x09\x8e\x9f\x24\x4b\xea\xc5\x4c\xc4\xfd\x28\x66\x1c\x08\xa3\x8b\x28\xdf\xa9\xea\x60\x96\xa7\x28\xbe\x78\xc5\x17\x4b\x76\x8a\xdf\xb0\x62\x26\xb9\x2a\x97\x27\xc9\x2b\x2f\xbe\x79\xce\xaa\x33\x19\xde\x11\xdb\xae\x98\x45\x64\x9a\x0e\xab\x24\x65\x76\xd5\x8b\x81\x8c\x6b\x3b\xb1\xb0\xc5\x4c\x02\xf5\xf4\xf8\xcd\x0f\x4f\xbf\x3b\x3a\x65\x50\x4c\x62\xd9\xce\x78\x32\x75\x67\x9e\x1f\x84\x74\x7e\xb1\x88\x7e\xfb\xb0\x5c\xc5\xc9\xfa\x63\x9a\xe5\x97\x57\xd7\x9b\x9b\xbf\x3d\xfd\xee\xd9\xf3\xa3\x17\xdf\xff\xf0\xf2\x7f\xfe\xe5\xf8\xd5\xc9\xeb\x37\xff\xeb\xed\xbb\xd3\x1f\xff\xfa\xd3\xcf\xff\xfe\xbf\xff\xfc\xab\x04\xf6\xbb\xa7\xef\x8e\xe0\x10\x08\x25\x4e\xf5\xf0\xf8\xf5\xf7\xbf\x16\x2f\xa4\xc7\xaf\x9e\xfe\xfc\xeb\xbb\xa7\x2f\x8e\x7e\x7d\x79\x72\x7a\xf4\xfd\xd1\x5b\x0c\xde\x92\xb9\xfc\x31\xca\xc5\x54\x36\xa9\x78\x3f\xb6\x61\x0f\xc8\x37\xd2\x33\x06\xe3\xe5\xc9\xa9\x6d\x61\xe5\x69\xab\x1a\xd4\x01\x60\x4a\xcd\x0a\xc0\x9b\xd7\x3f\xbd\xfb\xf5\xf4\xe8\x84\x39\x03\xc4\x00\x62\xb2\xff\xd9\x1f\x6a\xb3\x3f\x0e\xfb\x33\x66\x7f\x26\xec\xcf\x94\xfd\x71\xd9\x9f\x19\xfb\x83\xa5\x29\x21\xf8\xd7\xc2\xbf\xf6\x79\xd5\xbb\x77\xff\xeb\xed\x69\xd9\x6b\x3a\x35\xbe\xa9\xd0\x7e\x50\x7e\x85\x07\xfc\xd0\x75\xb4\x8a\x72\x48\xf8\x3e\x39\x1e\xb7\x4e\xe6\xf0\xfc\xe8\xd9\xcb\x57\x4f\x8f\x7f\x7d\x73\xfc\xf4\xd9\xd1\x3b\x03\x4e\x5f\xff\x7a\xf4\xf3\x9b\x5f\x4f\x8e\xbe\x2f\xbf\xbf\x79\xfd\xce\x80\x57\x2f\x4f\xd8\x0f\x03\x89\x81\x5f\xbc\x38\x94\x9b\xc8\x71\xc7\xe4\xc5\xe5\x8a\xc6\x62\x27\x76\xc2\x0c\x75\x4c\xe3\x3c\xf2\x96\x06\xe4\xc9\x8b\x68\x43\x43\xfc\x92\xa4\x2b\x2f\x17\xcb\x4a\xc9\x9b\xc2\x36\x18\xe0\xd3\x9b\x24\x0e\x8b\x14\x9e\xb1\x0c\x9e\x6e\x02\x8a\x9b\x01\xf9\xea\x4a\x9a\x5c\xc7\x30\x88\xe6\x70\xf4\xf6\xed\xeb\xb7\xef\xf0\x61\x7a\x49\x87\x23\xa9\xce\xbe\x2c\x04\x8c\x40\x47\xb3\x87\x0a\x9e\x29\x78\x68\x32\xfc\x4b\xae\x7f\x53\xb7\x84\xc5\x1e\x0e\x9e\xa9\x8a\x0f\xed\x5e\x65\x34\x98\x37\x92\xe5\xe9\x65\x90\x27\xa9\xc0\x46\x60\x52\x5a\x5d\xb1\xe9\x77\x10\x24\xf1\x3c\xba\x78\xed\xff\x56\xd8\x5f\x10\x16\x28\x8c\xae\x24\x56\x0a\xa4\xa2\x10\xf2\xd4\x0b\x3e\xf0\xc5\x25\x36\x38\xd2\xb4\x84\x69\x40\x96\x00\xd3\xd7\xd8\x5b\xd1\x22\x8d\xaa\x98\x0f\x86\x6c\x00\xa4\xcc\xd3\x06\x91\xe8\x23\x1b\xd5\x80\x47\x98\x20\xd0\xa8\x3d\x7b\x03\x87\x55\x9f\x46\xeb\x34\xc9\x13\x66\x00\xea\x85\x5e\x9f\x30\xc1\x8b\xe9\x75\x55\x74\x40\x86\xc6\x37\x0d\xdc\x1f\xf4\x7d\xe0\xe8\xf9\xcb\xd3\xa7\xdf\x1d\x1f\xc1\xf3\xa3\x17\x4f\x7f\x3c\x3e\x7d\x07\xbd\x75\x4a\x93\xad\x92\x78\x28\xa5\x3e\xa4\x73\xef\x72\x99\x17\x49\x64\x7d\xba\x4c\xae\x61\x75\x99\xe5\x9c\x40\x39\xbd\xa0\x69\x86\x3b\xc7\xc5\x0e\x52\x24\x5a\x16\x5d\x51\x48\xbd\xf8\x82\x66\x90\xe1\x66\xfb\x91\x0a\xb8\x00\xca\xc8\xed\x2d\x33\xcc\x31\xcb\xf3\x7c\x85\xe0\xe5\x90\x5e\xc6\x7b\x79\xb4\xa2\x70\x99

\x31\x33\x5b\x51\x93\xf3\xbd\x09\x71\xbf\xc5\x71\xd6\xc4\xca\xdb\x44\xab\xcb\x95\xb4\xf7\x35\x
a4\x41\xb4\xf2\x96\xb0\x5e\x7a\x01\xcd\x70\xa0\x65\x0e\x93\xc7\xb3\xc8\x44\xf1\x55\xb2\xbc\x6
2\xed\x85\xd1\x15\xaa\x55\xbd\x9d\xba\xbe\xc3\x21\x58\xa6\xca\x90\xd5\xd4\x60\x1b\x3a\xd7\x4
6\x13\x91\x69\x77\x41\xe3\xea\xb9\x38\x9b\xe1\xf9\xc9\x15\x6d\xf4\x81\x5b\x02\x2c\x8e\xa4\x6a
\xc2\xef\xb3\x24\x6d\x43\x82\xaa\x89\xe9\x2e\x07\x35\x01\x18\xb6\xd8\xf8\xe3\x9b\xf2\x87\x09\x
4f\xaf\xbd\x1b\xbe\x30\xfa\x37\x9a\x26\xad\xb2\xcf\x5f\xff\x74\x22\x7e\x10\x38\x4d\xae\xbd\x34\x
cc\xd4\x25\x9f\x1d\xbd\x3c\x16\x3f\xac\xb2\xe4\xee\xcb\x78\x1e\xc5\x51\x7e\xd3\x2a\xfe\xe2\xf8\
xf5\xeb\xb7\xfc\x87\x5d\x16\xdf\xd3\x16\xff\xe1\xe9\xf1\x8b\x5f\x39\xe2\x4e\x59\x3c\xa6\x5e\x4a\
xb3\x1c\x62\x1a\x5d\x2c\xfc\xe4\x32\x1d\xc1\xcb\x39\x30\x7f\x32\x8c\xb2\xdc\x8b\x73\x03\x2e\xd
7\x6a\x50\xbc\x5f\xe3\xdb\x80\x0a\x93\xeb\x58\x0d\xec\xe8\xaf\x47\x27\x00\x93\xdb\x00\xcb\x45\
x51\x7a\x45\x63\xa9\x9c\x12\x3c\xa7\xec\xf4\x2e\xe0\xf5\xf4\x47\xc8\x9c\x09\xee\x5d\x20\x6b\x5
9\xb5\x5f\xfb\xfd\xf6\xf5\x8f\x27\xcf\x5f\x9e\x7c\xff\xeb\xab\xd7\xcf\x99\xe1\x74\xd4\x9a\x57\xe9\x
9e\xdb\x32\x09\x47\x3f\xbf\x79\x7d\x72\x74\x72\xfa\xf2\xe9\xf1\xaf\x4f\x4f\xe1\x00\xce\xaa\x01\x
1b\xe4\x11\xfb\x5c\x69\x4d\xa8\xd0\x23\x31\xfe\x0b\xd5\xf1\x69\x4c\xbd\x7c\x21\xc6\xdd\x62\xb5
\x57\x8c\x6d\x59\x59\x29\xf2\x96\xcc\xd9\x44\x43\x33\x6a\x42\x3f\xa9\x9c\xc4\x03\xd8\x9b\xd6\x
5e\x4b\x28\x1e\xc2\xde\x54\xd7\xeb\xaa\xdf\x7b\x6d\xa3\xd3\x85\x3f\xb7\x29\x5f\x0d\x7b\x8b\xa
8\xb0\x7f\xf3\x1a\xad\x25\xe9\xc7\x5e\x85\xfc\xdb\xa7\x27\xdf\x1f\x31\x76\x35\xfd\x28\x35\x9f\x5
6\x51\x8c\x56\xbf\xde\x5f\xa3\xc1\xa9\xcb\x38\xa4\xe9\x9c\x0d\x68\x79\x82\x76\x08\x92\x20\xb8\
x4c\xb3\x1e\xe6\xd8\x96\x03\x30\x18\x53\xf6\x65\x58\x2b\x2a\x90\xc3\x15\x36\xda\xc1\x27\x06\x
73\x8f\x74\x72\xaa\x18\xb7\x9a\x3d\x90\x79\x95\x5c\x55\xe8\x17\x1a\xb4\x55\x17\xc0\x36\x5d\x
80\x01\x19\x4d\x67\xd3\xc9\xcc\x26\xb6\xe3\x4e\x2c\x9b\x8c\xa7\x74\xd7\x36\xdd\x61\xb3\xee\x
8b\x24\x2d\xc8\x0d\x8f\xb9\x8f\x4d\x47\x17\xa3\x86\xe3\xb2\xc3\xbc\x6d\xf1\xd9\x19\x8e\xd6\xcb
\xcb\x6c\x40\x86\xb0\xf2\x6e\xd8\xb8\x9e\x2d\x93\xeb\x3a\x52\x05\x44\xee\x79\xeb\x69\x85\x0b\
xfd\x3a\xb9\xf8\x69\x41\x99\x2b\x28\xb9\x8f\x18\x12\xcd\xc0\x4b\x29\xb3\x87\xa9\xf0\x75\xeb\x4
d\x0b\xa7\xf7\x10\x7d\xde\x9e\xa6\x59\x11\x36\x2b\xc4\xa8\x62\xab\xfd\x67\xe8\xae\x30\xec\xa2
\x38\xff\xab\xb7\x8c\x42\x2f\x4f\xd2\x93\x44\x60\x51\x73\xb0\x11\x42\xc3\x73\xcc\xb0\xce\x4b\x0
c\x90\xca\x10\x7e\x8a\xf2\x05\x87\x61\x14\x4d\xa9\x5f\xef\xab\xda\xd5\x92\x29\x4f\x70\xf7\x23\x
9f\x96\x5f\xa4\xde\x7a\x11\x31\x47\xf8\x66\x2f\xa3\xc1\x65\x8a\x2e\x5b\x98\x94\xce\xd2\x05\x8d
\x85\x5f\x64\xb0\x9e\x78\x57\x5e\xb4\xf4\xfc\x65\xa3\x0f\xcf\xde\xfe\xfb\x9b\xd3\xd7\x20\xe2\xae\
xdd\x42\xdf\x45\x4c\xa5\x5b\x84\x81\x9c\xa4\xe9\x14\x05\xde\x32\xb8\x5c\xe2\x71\x50\x74\x8a\x
b0\xd4\x65\x76\x00\x1e\xfb\x0a\xed\x91\x95\x81\xfa\x78\x99\xe4\x11\xd3\xa4\xc1\x47\x38\x04\x0
f\xf6\x21\x1e\x32\xb6\x14\xc0\x98\xd3\x29\x5f\xa8\x81\xb3\x84\x24\x4d\x69\xb6\x4e\xb8\x0b\x56\
xf3\xd1\x94\x6d\xa4\x74\xe5\x45\xcc\xaa\xc0\x20\x6d\x01\xcf\x0e\x20\xc5\x96\xf7\x20\x86\x07\xf
0\xb1\x09\x41\xef\x5a\x99\x0d\xe0\x51\x06\xeb\x24\x8b\x72\xe6\x65\x47\x73\xc4\x94\x79\xab\x2
1\x8d\x43\xdc\x49\x84\x29\xaa\xae\xa8\xc1\x63\x78\xd2\x93\x0e\x97\x8c\x34\xda\x58\x78\x7c\xa
2\x94\xb1\x09\x51\x16\x5d\xc4\x20\x1e\x14\x2d\xb5\x60\xc9\x1f\x3c\x4b\x22\x33\x0f\x37\x01\xae\
x56\xb8\x33\xf0\x43\xcc\x66\x9f\x5e\x06\x3b\x79\x7a\x19\x07\x48\x9d\xc2\xdb\xde\xc1\x11\x29\x
ca\xba\x80\x33\x1f\xe2\xca\x5b\x32\x5e\xe6\x09\x0c\x3c\xf8\x16\x59\x19\x4b\x71\xb4\x0e\x17\xd
1\xbe\x4d\x47\xb3\x24\x85\xc1\x9b\x9b\x7c\x91\xc4\xf0\x6d\xdb\xfb\xad\x3c\xb6\x89\xa2\xcb\xab\
xf5\x92\x8a\xe9\xfc\x82\xc2\xcb\xa3\xa3\x23\x98\x8e\x1d\xa9\xe9\x62\xf6\xd9\x82\x7b\xf4\xe3\xb
3\xe3\x97\xcf\xd9\xf7\x19\x1c\x5d\x06\xcb\x28\x8c\xbc\xb8\x9a\x91\xc0\x47\x7e\x4e\x29\x1e\xc4\
x43\x78\x00\x18\xdc\x1a\x30\x89\xf6\xfc\x6c\x10\x0f\xdb\x78\xd6\x59\xd3\x10\x25\x6f\x79\xed\xd

d\x54\x12\xd5\x27\x96\xac\x7e\x9b\x6f\x06\xc7\xa2\xf6\xa4\x8d\x39\x32\xb7\x4d\x88\x66\x13\x8c\x7c\xdc\x84\x97\x32\x83\xca\xcf\xe6\x6a\x92\xbe\x57\xf3\xb6\x56\x87\x9f\x2e\xf3\x45\x72\x79\xc1\x8f\x49\x63\xb4\xa0\xae\xbd\xf5\x49\x27\x03\x6e\xb0\xa2\x37\x38\x5c\xc5\x49\x0e\x17\x4c\xb7\x2e\x33\x3a\xbf\x5c\x42\x4a\xb3\xcb\x65\x9e\x75\x3b\xa7\xaf\x5e\x3f\xff\xf1\xf8\x75\xe1\x9a\x6e\xe1\xe6\xcc\xb6\x9c\xaf\xb6\x23\x84\xfc\xd4\x0b\x15\x78\x15\xbf\xf2\xe4\x4d\x72\x4d\x53\x1d\x51\xf8\x26\x8f\x37\xaf\x7f\xfa\xf5\xcd\xdb\xa3\x67\x2f\xdf\xbd\x7c\x7d\xc2\x98\x6f\x1a\x62\x67\xf2\x75\xb4\x5c\x22\x2d\x62\x0c\x74\xd1\x50\xd1\x70\x1d\x62\x1d\xd6\x21\x8f\xca\x75\xf5\x58\xe7\xeb\xcc\xf9\x2c\xb4\x58\x8f\xe2\x8b\xc8\xc8\x70\xff\x06\xbb\xa6\x08\xa4\x8c\xca\xc9\xeb\x8a\xe6\x8b\xa4\x61\x8d\x5e\xbc\x7e\xfb\xea\xe9\x29\xae\xb1\x35\x91\x11\x53\xe7\x77\x74\xed\xa5\x6c\xcc\x3c\x80\x9d\xd1\x8e\xd1\x2a\x76\x91\x26\x97\x6b\xb9\x90\xa1\x2d\x84\x19\x6b\xed\xf6\xdb\x8c\x06\x49\x1c\x7a\xe9\xcd\xf7\x55\x31\xb3\x5d\x6c\x9e\x7a\x68\x04\xbe\x6f\xb6\xf8\xcb\xe6\xa9\xb9\x63\x94\x94\x8a\x93\x78\xcf\x4f\xa9\xf7\x81\xc9\x30\xe6\xca\xec\x01\xc5\x1b\xac\x15\xfa\xfc\xf0\x1b\x39\xc4\xf9\xbb\x7d\xe4\xd8\x12\xf3\x92\x5e\x9f\xbc\x3b\x7d\xfb\xe3\xb3\xd3\xd7\x6f\xbf\xe9\x0a\xb1\x2a\x83\x80\x68\x33\xf8\x4a\x0e\x33\x01\x6d\x93\xa9\x0e\x28\x32\xe7\xb4\x48\xa3\xcd\xf7\x8e\x57\xd0\xc5\xf2\x9a\x04\x42\x86\x16\x17\x27\xa9\x3f\xf1\x0d\x76\x9f\xca\x8a\x98\xe8\x40\x2c\x59\xa0\x3f\x5e\xc3\xe2\xcc\x3f\xaf\xce\x60\xb3\xee\xf8\x5e\x86\x4d\xc7\x23\x78\xc9\x23\x65\x06\x58\x4c\x11\x26\x4e\x15\x26\x53\x47\x5c\xcb\x58\x67\xe5\x5f\x43\x6c\x80\x0f\xc3\x86\x48\xe3\x82\xb9\x01\xd4\x80\xc8\x60\xb8\x19\xb0\xa4\xb1\x01\x59\x9e\xb6\x25\x6d\x23\x0e\x77\x3d\x6c\x4f\x8b\x63\x3c\x4e\x23\x53\xfd\x32\xf3\x2e\x68\x79\xdb\x53\x4c\x1b\xde\x7b\x34\x87\x01\xdc\x1b\xc0\xa6\x24\x72\x32\x97\x28\x3c\x44\x44\x5b\x18\xec\xef\xc3\x4e\xd5\xa3\x61\xad\x41\xe6\x88\xca\xed\x1d\xc0\xdf\xe3\xcf\x3b\x2d\x10\xac\x61\xee\x53\x0f\x21\xf5\xa2\x8c\x0e\xc0\x9a\x18\xb0\xd3\x05\x6a\xc7\x80\x18\x86\x0f\x5b\xb0\x84\xb0\xd4\xe7\x31\x82\xce\xf5\xd2\x9f\x5b\x24\xdb\xa9\xd7\x1a\x72\x6e\xe3\x62\x52\x5c\x04\x46\x0f\xe0\xef\x7e\xa3\x0b\xda\x9a\x0c\xd5\x64\xce\x83\xa6\x8a\x7a\x48\x6f\x1f\x0e\x0f\x21\xbe\x5c\xe2\xa9\xa2\x7b\xd5\xe4\x61\x00\x3e\xde\xb2\x31\x71\x0c\x88\x42\x03\x76\x18\xc4\x9d\x0e\x26\x3c\xbf\x5c\x2f\x99\xa1\x6d\x0e\xf6\x45\x43\xb1\x8e\xab\x6d\x7b\xca\x3e\x9b\x51\x06\x87\x10\x8f\xb2\x36\x89\xf9\x6b\x8a\xaf\xa9\xee\x75\x00\x87\xd8\x28\x2b\x14\xc0\x10\x9e\x30\x58\x78\xf4\x70\x08\x07\x10\xab\xab\xf1\xa8\xbb\xfa\x1d\x67\x6c\xfb\xdd\xe7\x36\x35\xb0\xbf\x03\xa6\x3a\xd5\xb9\xaf\x18\x37\x2b\x70\x55\x66\x64\xbc\x7f\x1f\x3d\x76\x13\x77\x27\xf5\x50\x81\xb0\x89\x05\x3c\x02\x13\x9e\x88\x3e\xed\xc5\x06\x9b\xed\xb3\xae\x90\x87\x6d\x04\x40\x4c\xb2\xbd\x2c\x87\xb5\x97\x2f\xd0\xcb\x29\x22\xeb\x6d\xfe\x40\xc5\xa3\xc3\xc3\x43\xf8\xc7\x3f\x62\x2d\x42\x50\x1c\xa9\x04\x3c\x73\x61\xe0\xb1\xca\xf8\x21\x44\xf0\x98\x8d\xd6\xec\xcb\xfe\x21\x2e\xa8\xd1\xdd\x5d\x95\x7e\x54\x3d\x63\x00\x34\xec\x83\x92\x85\x67\xf1\xb9\xbe\x48\x17\xbb\xa0\x83\x65\xa0\x66\x1b\xfb\x64\x39\x9b\x4f\xc5\xb0\x8b\x1b\x95\x5b\x95\x8a\x4d\x6b\x4a\x6c\xd0\x7c\x95\x0b\xd0\xa3\x9c\x66\xf9\xa0\x06\x10\xb5\x47\x18\x08\x79\xf9\x7a\x00\x1b\x6e\x5e\x51\x64\x34\x44\xe3\xa2\x90\xe5\xa9\x7c\x92\x0b\xf7\x0b\x1e\x82\x33\x46\xc9\xe0\x6d\xb1\x22\x5c\xd2\xc9\x50\x96\x92\x36\x05\xb6\xea\x9a\x0f\x87\xe0\xf3\xfc\x30\xdd\xc4\x52\x19\x85\xa3\x38\xc3\x18\x00\xef\x33\x8f\x10\x46\x19\xf7\x9b\xf9\xc5\x81\x8d\xc5\x0f\x8f\x2f\xfc\x08\xff\x9a\xd9\x1c\x85\xbc\xee\xef\xc3\xd3\xe5\x32\xb9\x56\x06\x12\x9b\x17\x0b\x72\x53\x48\xcc\x72\x25\x54\x63\x9f\xd0\x10\x92\x0e\x55\x6c\x2d\xab\xe9\x4c\xda\x13\x88\xe1\x00\x89\xa3\x61\xa5\x20\x07\x52\x01\x79\xdf\x20\xc2\x2e\x6a\xc7\x2e\xf3\xf8\xeb\xf1\x69\x05\x3c\x85\x1c\x33\xf2\x5c\x25\x51\x08\xeb\x24\x17\xc4\xc1\x1b\x0b\xd6\x29\x15\x24\x4a\xe6\x55\x70\x8f\x79\x38\x27\xde\x09\xa3\x3c\x52\xca\x71\x76\xc5\x22\x9a\x9a\xf0\xf1\x4d\x31\x91\x88\xe

2\x1a\x03\x98\xcb\xcd\x5d\xfe\xb9\x17\x2d\x21\xbc\xa4\x45\xc8\xe3\xec\x88\x9e\x9f\xed\xee\x9d\x6b\x28\xbf\xad\xa5\xbc\xc7\x2c\xe5\xa7\x4f\x0a\x92\xde\x1b\x20\x6b\xde\xd2\x8b\xa3\xcd\x7a\x00\x3b\xef\xf7\x9e\xec\xc0\x2e\x0c\x80\x99\x90\x9d\x33\xf6\xbd\xd8\xf7\x20\x54\x83\x99\x2e\xe6\xf9\xec\xc2\xce\xf9\x2e\x6b\x65\x57\xc9\xa9\x9d\xc1\x93\x83\x5f\x7e\x19\xe1\xd9\x2b\x56\x76\xf8\xe4\xcf\x3b\x86\x0f\x8f\xc0\x9e\xc2\x13\xd8\x89\x76\xe0\x40\xe8\x35\x57\xf7\x2c\x4f\x87\x5a\x09\xea\xd3\x7c\x85\x8f\xa0\xe1\x30\x66\x21\xbc\x5c\xdd\x66\xd4\xe8\xb6\x0d\x7a\x83\x26\x62\x8d\xf7\xef\x63\xdd\x94\xe2\xf2\xe0\x00\xf6\xdf\x9b\xbf\x8c\xcc\x07\x9f\x7e\x19\xed\x1b\x48\x81\x22\x39\xc9\x63\x20\x63\xb5\x8f\x50\x7c\x54\xee\xca\x2d\xf7\xb3\x68\xfc\xb8\x92\xd0\xdc\x87\x8b\x70\x3d\x52\xda\x39\xa3\x71\xd9\x34\x44\x16\xb8\xbe\x49\xe9\x15\x8d\x73\x58\x7a\x39\x9b\x44\x94\x5b\x1d\x45\x7f\x13\x3c\x0b\x71\x45\x71\x26\x21\x76\x46\x29\x41\x71\x19\x17\xbb\xfb\x5b\xcd\x77\x0d\x2c\xbf\x83\xe5\xd7\x98\x72\xd1\x93\xef\x3c\x46\x3e\x14\x4b\x36\x92\xfb\x06\xa2\xd0\xeb\xbe\x3e\x2f\x56\x90\x93\x28\xce\x9f\xb4\x5d\x4d\xee\x33\x30\x24\xf1\x6c\xc4\xeb\xf9\x60\x67\xb4\xc3\xd4\xe5\x31\xc7\xb5\xea\x44\x29\x66\x6c\x1e\x8d\xe2\xa5\x98\x5f\x34\x4c\x8f\xb2\xc1\xa8\xa0\x0a\xf5\xd2\x60\x31\x80\x7d\xba\x1f\x01\x6f\xd2\xd4\x7b\xb2\x34\xa7\xe9\x2a\x8a\xab\xb5\x2d\x8d\xe1\xa2\xa8\x5d\x43\xec\x56\xd4\x26\x33\x85\xdd\x43\xd8\xad\x98\x02\x11\xb3\xe9\x2a\x09\x94\xd8\x77\xe9\xf3\x19\xe2\x80\x3b\x57\x4d\xa2\x43\xb9\x45\xb5\x6c\x5d\xdd\x09\x31\x3b\x6c\x63\x5e\xf0\xa0\xd8\x31\x5d\x03\xaf\xe0\x69\x41\x8b\x25\xf5\x30\xd4\xf5\x37\x9a\x26\x8d\xa1\x81\x3b\x84\x22\xc1\x46\x43\x4e\x23\x21\xa7\xfc\x1e\x47\x15\x23\xab\x26\xf2\xd4\x8b\x96\x9d\x6d\x2c\x69\x5c\x47\xbf\xd9\xdc\xde\xde\x92\xc6\x55\x93\x0d\xf2\x35\xd5\x84\x4d\x70\x19\x48\xc1\x96\x96\x04\xa1\x39\x6f\x2b\x65\x0b\x0b\xb5\x18\x45\x19\x5e\x79\x2c\xac\x42\xe1\xd8\x5c\xd1\x54\x6d\xd4\x58\x8b\x92\x1d\x54\x8e\xbe\xbf\x93\xe1\xe4\xce\xff\xe5\x8a\x99\xf9\xca\xe0\xb3\x6e\x0a\x73\xae\xb7\xa8\xcc\x32\x3c\xe0\x76\x55\x29\x6a\x14\xef\xb2\xde\x53\x0f\x32\xfb\xfb\xf0\x5a\x2c\x38\x3e\xd1\xa9\xd8\xe3\x72\x55\x4f\x3b\xaa\xa0\xb8\x2b\x57\xfb\xc5\x87\xcf\x25\xc4\x9c\xf1\x72\xb9\x54\x33\xec\xc7\x62\xed\xb6\x8d\x4b\x53\xef\x8a\x55\xd9\x2e\x94\xfe\x77\x7b\x17\x4a\x0d\x9d\x33\x81\x90\x09\x8a\x29\x4e\xcf\x80\xd0\x31\x77\x12\xc0\xcf\xf5\xd3\xc2\xd3\xd4\x8b\x33\xf4\xd6\x98\xc3\xa7\x2d\x46\x8b\xa4\x09\x85\x03\xad\xb7\x86\xa2\x46\xc4\x6a\x5c\x63\xe8\x37\x4f\x00\x55\x0c\x55\x2e\x4f\xe0\x82\xe6\x7c\xdf\x3f\x9e\x63\xa3\x7c\x11\xa3\x08\x33\x07\x09\x9d\xcf\xa3\x00\x57\xd2\x44\xc2\x25\x55\x0b\x11\x4e\xe9\x29\xd7\x56\xf8\xb6\xdc\x5e\xaa\x99\xcb\xcb\x36\x32\x62\xa6\xb8\x2a\xaf\xaf\x10\xf1\xb3\xa8\x9d\xf3\x5f\x56\x30\x1a\x32\x42\xf3\x03\x20\x35\x1b\xcf\xad\xb6\x52\x1d\x8a\x4f\x65\xca\xf6\x24\xa4\x8a\xa6\x1f\x76\xb6\x0d\x9c\xc3\x8a\x86\x23\xa3\xde\x4b\x8e\x84\x0e\x86\xc6\xdf\x01\x85\x91\x8c\x3a\xc0\x30\x96\x94\x0d\xee\x69\xc7\x95\xb2\xd5\x2e\xa9\xe6\xf0\xf6\x0e\x91\x08\xb7\x71\xd2\x38\x3d\x1f\x42\xb4\xb7\x87\xe3\x01\xa3\xc2\x8e\xb9\xa3\x9f\x48\xcb\xe4\x53\x7b\xdc\x4a\xa4\x55\x36\x43\xad\xe4\x3d\x0a\xde\xe8\x47\x33\x8c\xf1\x59\x1b\xe9\x86\x37\x6f\x5f\xbf\x39\x7a\x7b\xfa\xf2\xe8\x9d\x54\xa6\x5a\xca\x10\xbb\x58\xe1\xb0\xd8\xcf\xfa\x50\x55\x0a\xe7\x94\xbf\xfe\xf8\xa6\xde\x6a\xf3\x3d\x2e\xe3\x1e\xca\xee\x63\xb3\x04\xee\xfb\x3a\x04\x4b\x5f\x82\xaf\x90\x1e\x82\xad\x2f\x52\x6c\x9c\x3b\x04\xa7\xa7\x90\x40\x68\xdc\x53\x0c\x97\x4e\x0f\x61\xd2\x53\x4c\x20\x3f\xed\x29\x56\xf4\xc0\x55\x95\x13\x4b\xaa\x87\x30\x7b\xa8\x5d\x83\x78\x86\x1b\x4b\x2f\x53\x0a\x51\x3c\x4f\xf1\x9c\x62\xbe\xbc\xd9\xc3\x3d\xa9\xcc\xd5\x11\x87\x33\xf6\xae\xa3\x90\x42\x46\xf3\x3c\x8a\x2f\x32\xdd\x32\xc2\xd3\x20\xa0\x6b\x8c\x06\x8b\x23\x3f\xb8\x94\x51\x46\x36\x60\x19\x65\xb9\x21\xfc\x8c\x18\x77\x45\xac\xbc\xf8\xa6\x30\xb3\xe5\x99\x2b\x10\x99\x71\x23\x9a\x41\x92\xca\x0d\x60\xd6\x57\xe6\x8f\x65\x90\xd2\x6c\x4d\x83\x3c\xba\xa2\xcb\x9b\x03\x0d\x3e\xad\xbd\xab\xd5\x62\x45\xb5\x3c\x51

\xac\xd3\x55\xeb\x13\x75\x10\xf5\x10\x87\x1e\x84\xab\x03\xd0\xd8\x80\x57\x2e\xb5\xf0\x7f\xce\x
ce\x65\x48\x7b\x0c\x91\x26\x3e\x0d\x22\x6c\xf5\x39\x13\xe1\xd4\x12\xa2\x89\xf0\x46\x8d\xfe\x8e
\xce\x1b\x9d\xc5\x4d\x67\xd5\x47\x81\xeb\x49\x12\xef\xe1\xce\xb1\xe8\xf7\x47\x7a\x8f\x14\x58\x
17\x6f\x48\x17\xf6\xc2\x37\xac\xb0\x17\x47\x5c\x3e\x95\x2c\x13\xdb\x9f\xc4\xbe\x1d\xc2\x44\xd0
\xac\xc3\x10\x7b\x7b\xbe\x08\x86\xbc\x30\x0e\x35\x91\x29\x3f\x7c\x3d\x5c\x27\x32\xf5\x75\x66\x
3d\x80\x0e\xb1\x15\xeb\xc1\x55\x3f\xb8\x4a\xca\x20\xde\xd1\xee\x75\xe6\x16\xff\x9a\x0b\xc9\x05
\x68\x3e\x25\xfc\xdc\x2a\x5f\x5f\x51\xae\x50\xe9\x2e\x1f\xfd\x8d\xd6\xc5\xa2\xe8\x7d\xab\x7c\x7
b\xb9\xb9\xbb\xbc\x7a\xdd\x59\x8f\x4f\x6b\x71\x59\x8b\x8f\x5c\x75\x20\xed\xf3\xf7\x32\x36\xc9\x
e1\xa1\xe4\x6a\x2f\xbb\xe0\x8d\x30\x92\x92\xb5\xf3\x52\xbe\x9e\x86\x61\x1c\xb1\xfb\x03\xcf\x39
\xb3\xa9\xc3\x50\xd3\xde\xd1\xe8\xa2\xb6\x28\xdb\x3c\x35\x30\xb0\x4c\x03\x1c\xdc\x9b\x55\xdb\
x46\xd4\x59\xe7\xef\x4d\xd3\x79\x80\xdb\xfe\xeb\xc6\xf0\x00\x1c\xf8\xac\x43\xeb\xe5\x45\xcc\x2
6\x7b\x55\xef\xf6\x25\xeb\x9d\x51\xdc\xc8\x84\x0b\x7b\x49\x5a\xa5\x2b\xae\x75\xe4\xad\x58\xd8
\x2e\x87\x93\xea\x16\x80\x8a\x64\xc1\x65\x9a\x96\x9b\x45\x33\xf5\xd2\x72\xb3\x77\x70\xd8\xc8\
x3d\x2a\x4b\x1b\x26\x1e\x33\x60\xdd\x5e\x4c\x8e\xda\x67\x5e\xd8\x27\xc5\x93\xad\xed\xe7\x1e\
xe6\x1f\x13\xc7\x9b\xda\xaf\x13\xf6\xfa\xcc\x3c\x6f\xbf\x61\x13\xe5\x43\x48\xd8\x1c\x57\xc4\x9c\
x13\xf9\x28\xb1\xc2\x6b\x7c\x52\xef\x0f\x9f\x2b\x24\x8d\xc3\xb4\x83\xb5\xb4\xbe\x33\x80\x2b\xd
6\xc6\xd9\xfa\x1c\x86\x70\x4f\x4c\x3b\x1b\x9e\x25\xff\x1c\xa8\x60\x7b\x55\x3c\x35\x6a\x02\xf5\x
ce\xa2\xdd\xdd\x3a\x58\x45\x54\xa5\x2e\x5d\xa5\x81\xeb\x18\x96\x5b\x9b\x7b\x76\x84\xac\x0e\x
9b\xd0\x5a\x4b\xd2\x57\x8a\x25\x69\x5d\xe5\xc6\xaa\x74\xb3\x2a\x76\x7f\xe1\x65\x03\x58\xc3\x
21\xec\xd4\x2b\x8b\xf5\x00\x79\x99\xfa\xca\x60\x73\xae\x57\x4f\x7f\xc6\xe5\xea\x35\xa8\xe3\xef\
xad\x33\x36\x57\xed\x25\xad\x3a\x73\x52\xc6\xba\xc3\x46\xc5\x36\x9d\xeb\x0a\xab\x21\xb3\xbb\
x15\x91\xeb\xa0\x6e\x47\xe3\x7a\xdd\xdb\x91\xb8\x56\x57\x4b\x61\xb7\x93\xbe\xcd\x73\x14\xdb\
x92\xb7\x56\x4f\x11\xe4\xad\xbb\x76\x2d\x6f\xa9\x59\xbe\xd7\xcf\xd3\xf9\x6d\xf8\x5e\xa5\x14\xe
7\x1d\x0c\x6b\x60\x77\x3b\x8e\x35\x2a\xdf\x8e\x65\xf5\xca\xda\x6d\x19\x3c\xa4\x91\x61\x8a\x8c\
xc1\x95\x7e\x65\x4a\x94\x93\x58\xce\xec\x26\x92\x08\x79\x2f\x18\xdf\x94\x8b\x33\x72\xbe\x95\xf
2\x15\x9f\xda\xa1\x13\xd6\x84\x7a\x59\xb9\x51\x9c\x9f\xf2\x60\x8d\xe9\x8b\xb7\xed\xaa\x1c\xb1\
x6b\x08\x33\xef\xd6\x36\x48\xd7\x4f\xc9\x0c\xea\x18\x31\x63\xcc\x57\xd7\xf6\xae\xe0\x00\xae\x0
0\x13\xb3\x6d\x11\x4f\x50\xe9\xc1\x99\xe6\x94\x2f\x34\xe3\x78\xe5\xc1\x95\xf6\xcc\xe1\x2e\x13\
x07\x0e\x51\x37\x2f\x10\x5a\xa1\x9c\x1b\xf4\x6a\x07\x47\xf3\x96\x66\x0c\xeb\x04\x5e\xcc\xaa\xf
9\x14\x17\x02\xb6\xab\x73\x4b\x93\xc7\xea\xfc\xbe\x6a\xb3\x47\xba\xf5\x86\x6c\xaf\x37\xd5\x19\
xa0\x7e\xa5\xa9\x8e\xc0\xfc\xa7\x68\x0c\xd6\xbe\xe2\x8a\x20\x9f\x5d\x1a\x48\x88\x6d\xaf\x38\xe
c\x53\x22\xd5\xdc\xa1\xc7\x10\xd9\x85\x9d\x86\xb4\xec\x18\x0c\xe0\x1d\x75\xb0\x79\x12\x4c\xa
5\x7d\x62\x0e\xdc\x9a\xb6\x2a\x26\xad\x5d\x43\x07\x87\xd2\x99\x28\x62\x8b\x61\x00\xa1\x74\xc
b\xf1\x15\x2e\x86\xdd\xbb\x77\x05\x9f\x3e\x89\x1f\xa4\xfa\xaa\xdf\xef\xd2\xb5\xb9\xaa\x76\xc6\x
68\x50\x9d\x7d\x62\xad\x0c\xe1\x89\xe6\x5c\x11\x1c\x28\x4f\x34\x69\x17\x3f\x64\xa6\x6b\x76\x5
4\xd4\x44\x41\x64\xe0\xb8\xbb\x00\xf0\xc6\xda\x2c\x17\x21\x8b\x2f\x64\xb9\x80\x72\x5b\x96\xd7
\x60\xf0\x53\x56\x70\x19\x97\x07\xa7\x0e\xe0\xef\xfc\x61\xb7\xa4\xf0\xc6\x7f\x1f\x49\x29\x4f\x6b\
xdd\xbb\xc7\x80\xdc\xbf\x5f\xa0\x59\xcd\xaf\xc4\x03\x39\x5f\x93\x46\xdd\x05\x26\xf7\xef\xc3\x3d\
x01\xb7\x5a\x93\x94\x75\x7f\xa7\x4d\x8a\x9d\x22\x39\x8c\x92\xfd\xff\x49\x32\xc5\x3b\xd1\x96\x2
9\x39\x84\xa5\x99\x35\xcc\xb6\x9a\x35\xc8\x80\x6e\x37\xd8\xca\x35\x6f\x37\x7c\x4a\x35\xb5\xf3\
x85\x59\xe7\x58\x51\x3f\xda\xb2\xed\x6c\x41\xaa\xd5\xa6\x68\x3d\xa6\xf7\x85\x13\xde\x3a\xb0\x
db\xd1\xb5\x5e\xf7\x76\x94\xad\xd5\xbd\xf3\x6c\xb7\x79\x8c\x66\x5b\xfa\xd6\xea\xb5\x29\x2c\xc2

\x6a\x45\xa4\x53\x4f\x01\x51\x50\x90\x8d\x17\xef\xef\x3b\xaf\xd5\x6d\xa3\x84\x3d\xb9\xaa\x9b\x12\x8d\x2a\x97\x47\x76\xae\xbe\x9a\x3d\xd8\xa9\x77\xea\x0b\xbc\x0d\x8e\x5c\x83\xc8\xc5\xce\x50\x69\x31\x50\xbf\xb4\xf4\x56\x3e\x97\x52\x6d\x3f\xbd\x5e\x24\x19\xad\xf6\xda\xe6\xd2\x09\x30\xb1\x10\x54\x06\xcd\x74\xeb\x4c\x55\xd2\x20\xed\xb1\x95\x9e\x38\xe0\xca\xdb\x34\x83\x80\x45\xe7\x56\xde\xe6\x75\xfa\x2a\x8a\x07\x52\xf0\x0e\xde\x8c\x96\x39\x0c\x1f\x7e\xa5\xee\x8a\xa3\xfa\xff\xc4\xee\x46\xf1\x2d\xbb\x7b\x71\xd7\xee\x46\xf9\x02\xbc\xe2\x54\x35\xef\x36\xfd\x78\xe9\x2d\x99\x75\x4b\x52\xb8\xc0\x73\x4b\x29\xdf\x03\x64\xe2\xee\xde\x25\xcd\x32\xb1\x27\xc8\xa8\x75\x3b\x16\x3b\xa4\xc3\xb5\xc1\xaa\x36\x62\x64\xd1\x1c\xc2\x35\x23\x69\xb2\x8a\xf2\x9c\x86\x46\x33\xcd\xcd\x00\x77\x41\x66\xb8\x71\xa9\xd8\xba\x25\xc3\xc7\x5d\x5c\x18\x72\x5f\xa7\x49\x78\x19\xd0\x70\xa8\xe3\xc0\x59\xb8\x96\x0e\x3a\x3d\xaf\xb5\x33\xda\xd2\x84\xd7\x00\xee\x70\x02\x0d\x86\x4d\xa4\x5b\xa6\x3c\x5c\xcb\x56\xa9\xa3\x66\xc3\x90\xeb\xeb\x6d\xe9\xa4\x29\x05\x49\xf0\xf5\x10\x06\xdd\x01\xf4\x75\x72\x6d\xfd\x3a\xb6\x31\xfb\x99\x65\xca\x1f\x45\xbc\xa0\x90\xa4\xb1\x0d\x7e\x94\x97\xd3\xf9\xd8\x10\x9b\x75\x4c\x78\x74\x88\x5b\x85\x67\xa6\x39\x25\xb3\x99\x35\x76\xa6\x8e\x39\x9b\x59\xad\x01\xf2\x19\x6e\x7f\x8d\xe6\x3c\x03\x5e\xd9\x63\xc1\x5f\x79\xfb\x99\x6d\xf1\x4b\x9b\x39\xc5\xc2\x64\x15\xd3\xac\x9d\x42\xe2\xe5\x1c\xa2\x1c\xc2\x04\x13\x0e\x65\xd9\xe5\x0a\x13\x88\x2c\xa9\x97\xe5\x02\xdb\xba\x00\x19\xfc\x60\xc0\x75\x94\xd1\x56\x79\xdb\xc4\xf2\xad\x36\xcc\x8d\x23\x48\xc3\x84\xd9\x7a\x6f\x9b\x06\xa6\x84\xae\x9e\x58\x78\x81\x25\xcf\x40\x27\x9e\x60\xbe\xb8\x51\x8b\xec\xbc\x2b\x63\xdb\x8f\x72\x31\xf9\xa9\x13\xe2\x41\xc1\x18\x71\x1b\x03\x82\x6c\x8c\x0e\xcd\xb5\x84\xd2\x4e\x88\xfc\x82\x03\xd0\xc0\x44\x83\xd1\x00\xd6\x5c\x3c\x28\x16\x0a\x5a\x78\x49\x54\xf8\x04\x26\x7f\xe2\x8a\x2b\x13\x54\x7b\xdf\x15\x00\xdc\xaa\x7a\x7b\xc9\xa1\xc8\x5e\x5e\x62\x13\xae\x55\x23\x2a\x5e\xba\x8c\x5b\x8a\xa9\x01\x1f\x0c\xb8\x6a\xaf\xce\x80\x7e\x25\x88\x7d\xf8\xee\x36\xf5\x3b\x86\x6e\xeb\x98\xc6\xeb\x93\x23\xd5\x6e\xac\x10\x73\xfd\xae\xb5\xc7\xdf\x98\x51\x2c\x5c\x2d\xe2\xe0\xb4\xb6\xb5\x62\x17\xae\xd5\x01\x96\x0f\x70\x58\x66\x93\x64\x90\x60\x5f\xda\x96\xa5\xc0\x85\x79\x21\x7c\xa2\xd0\xb5\xa1\x50\xdc\x3f\x96\x41\x76\xb9\x5e\x27\x29\xa6\xaa\xe0\x66\x65\x74\x41\xf3\xb7\xc8\xaa\xbf\x6a\x4e\x6e\x14\xad\x48\x53\x33\x75\xd5\xee\xbd\xfb\x1e\x1c\x6a\xea\xf1\x13\x18\x3f\x46\x71\x6e\x5b\x3c\x70\x06\x1f\xe0\xc1\x21\x58\x5b\x6d\x86\xe3\xbb\xdf\x3e\x3c\xec\x6e\x5d\xd0\x41\x18\x86\x83\xbe\x72\x42\x09\xd6\xc9\xf5\xc0\x32\xc0\xb6\x86\x4c\xa7\x99\x28\xcb\x8f\x2d\x32\x1c\x56\xb7\xe9\x58\xc3\x3e\xa0\x84\x7d\xf8\x5f\xed\x17\xae\x27\x85\x41\x2e\xbf\xdc\x11\x5f\xbc\xfe\x88\xdc\x0a\xc9\x6d\x3e\x1a\xfc\xfb\x40\x8b\xe1\xa6\x30\x93\x6a\x49\x2b\x3e\x62\xb5\xf2\x1c\xcd\x07\xaf\xb7\x0b\x03\xef\x0c\x37\xc2\x9f\x17\x3d\xeb\x90\x0e\x28\x46\x30\xe6\x6c\x33\xb3\x92\x79\xab\xf5\x12\xd3\x46\xf7\xe1\xc9\xc6\xb4\x2b\xc5\x98\xd6\x57\xf1\x4d\x9a\xf8\x9e\x1f\x2d\xa3\xfc\x86\x8d\x64\x39\x5c\xc1\xe3\x43\x98\x51\x32\x36\x5a\x39\x44\xda\xb5\x6b\x4d\xc1\x7e\xab\x75\xf8\xc7\x21\x98\x23\xd3\x34\x5d\x03\xa2\x11\x1d\x01\x81\x28\x06\x62\x8d\xbb\x09\x2f\xa2\x22\x02\x91\xde\x1d\xa2\x20\xce\xed\x6d\xab\xaa\x56\xe7\x5e\xd1\xe2\xe3\xf1\xdb\x30\xfd\x33\xb3\xe3\x4c\xa6\x54\x98\x33\x99\x55\x20\x3d\x15\xf4\x5b\x2c\x9b\x1f\x89\xb3\x87\xe0\xce\xea\x9f\xad\x6b\x0f\xae\xe0\x5b\xcc\x55\x3b\x64\xbf\x6e\x09\xa4\xd8\x3d\x5a\xc0\xd8\x86\x74\xb8\x33\xd7\xea\x21\x82\xf6\xad\xfe\x0d\x1b\x27\x3f\xc0\x3e\x03\xad\x1b\x35\x8a\x44\xe4\xed\x41\x83\x0f\xee\x78\x57\xae\x5a\x8d\xe5\x15\x81\xd6\xb8\x21\xd5\xee\x3d\xef\xe5\x5f\xce\xe7\xcd\x94\x2c\xd2\x47\x1a\x55\x24\xa8\x62\x00\x99\x7e\xed\xe1\x43\x5c\x3f\x67\xd6\x9c\x42\xc7\x35\xea\x2f\xc4\xe3\x5e\xa3\x2d\x55\x12\xee\xa5\x25\x41\x12\x66\xd2\xf9\x1a\xa3\xca\x5d\x0d\xf6\x9d\x0c\xe1\x15\x06\xf6\x07\x5c\xe7\xef\x83\x4d\x80\xfb\x80\x75\xd2\xe1\xf5\x8e\x95\xaa\x37\x0a\x68\x4f\x55\xd6\x3

f\x05\x00\xab\x01\xa0\x06\xdd\xae\xbd\xbc\x25\x64\xe7\x1c\x6f\x39\x99\xd4\x20\x8e\xf1\xa1\x8b\xcf\xf8\x93\x89\xee\xcc\x44\xf1\xb9\x83\x15\x56\x88\xf5\x74\x38\x0a\x92\xf5\xcd\x80\xf9\xc1\xad\x03\x5e\xcd\xcf\xed\x2d\xe3\x7f\x8e\x6d\x9b\xfe\x7e\xb6\x4d\x03\x7a\xeb\x00\x1e\x54\x41\x3c\xe2\xdc\x7e\xe1\x40\x8d\xa4\xfa\xd8\xf5\x8f\x19\x95\x27\x6f\x07\xc5\xd2\x48\x91\xa9\x0f\x92\xb4\x68\x29\xca\x64\x0c\x30\x16\xd3\x99\xd6\x0f\x84\xfc\xdd\x8b\xb4\x36\x4e\x61\x0b\x75\xf4\x60\xea\x5d\x9f\xcb\x0e\xba\x4e\x79\xcc\xc5\x9a\xad\x10\xfb\x80\x6f\x40\x3b\x2c\xc5\xe4\x0b\xa8\xc6\xa6\x4a\xc1\xd9\xde\x5e\xa4\x70\x11\xc2\x35\x7c\xdb\x79\x6c\xa7\xba\x1b\xbf\x3a\x3b\x28\x0e\xd3\x89\xf4\x9b\x59\x3d\x01\x5f\xd8\x4c\xb7\x5b\xf4\xee\x03\x1b\xd9\xc2\xb5\x96\x68\x57\x3c\x3a\x8e\x59\xe3\xcf\xa4\x13\x2f\xe1\x5a\xe3\xda\x04\xdc\x59\x92\x66\xf5\x4c\x96\xaf\xd0\x8c\xaa\xc2\xd1\xca\xce\xbd\xa5\xab\xe4\x4a\x3a\x17\x49\x8b\xe4\x6f\x22\x1d\x7b\x4a\x55\xb9\x8e\x25\x51\xe0\x58\x1c\xe2\xc1\xcc\x60\xb4\x4e\xd6\x83\xa1\x01\xd1\xde\x9e\x72\x4c\x15\xc7\x59\x34\x27\xf1\xa2\xf2\xb8\xa9\xb2\xbb\xb8\x7e\xde\x7f\x9c\x4d\xe7\xa5\x88\x9e\x16\x87\x4c\x35\x1d\xe5\xa9\x65\xc3\xdf\x2e\xb3\xbc\xca\x62\x5a\x32\x78\xa9\x39\x37\x56\xe5\x2b\xd9\x23\x48\x13\x53\xa2\x09\x5e\x2e\xca\xa8\x42\xe5\xd3\x58\x3a\x97\x03\x05\xee\x32\xce\x45\x32\x3f\x39\x73\x5a\xeb\x64\x5b\x80\x02\xa7\x3e\x3f\x8b\xe9\xb0\x47\xa3\x2e\x7c\x23\x9e\xf2\xed\x0a\xf5\xc3\x3c\x7f\xc8\x07\x1e\x62\xb2\x2f\x22\xc5\x4a\x84\x37\x38\xeb\x10\x15\x74\xca\xe5\x1c\xbd\xbd\xb4\x2a\x59\x2d\x1d\x23\xab\x51\x06\xf6\x54\x47\x9d\x15\xe2\xcb\xcc\x8b\xee\x9c\x22\xbe\x63\x22\x13\x68\x13\x28\xb1\x22\x8d\x15\x2b\x69\x51\x64\x38\x18\x3e\xac\x9f\x92\x7a\xf3\xf6\xe5\x5f\x9f\x9e\x1e\xc1\x8b\x1f\x4f\x9e\x9d\xbe\x7c\x7d\xf2\xae\x71\x88\x4a\x98\x09\xaf\x4c\xb6\xc5\x43\xfc\x8c\x51\xbe\x97\xd1\x97\x98\x37\x44\xf7\xf6\xf5\xa5\x74\xe8\xb1\xba\xe1\xb6\x75\x5a\x5e\x94\x35\x04\x48\x83\xe7\x7e\x54\xc5\x6f\xc3\x22\x00\x96\x1a\xb0\x31\x60\x13\x18\x70\xa3\xde\x10\x2d\x9f\x97\x87\x9d\xd1\x0e\x0c\xdb\xe5\x30\x96\xd5\xb8\x36\xa2\x4d\xd8\x55\xcf\x7e\x4b\x9e\x06\x85\x13\x03\x33\x5c\xc8\x47\xf2\xf3\xe4\x38\xb9\x2e\xee\x18\x6e\xc7\x98\x4f\x92\x78\x2f\x52\x1d\x39\x17\x02\xf5\x58\xb7\x81\xe0\x43\xcf\xb2\x23\x14\x67\x73\x8b\x14\x82\xe5\xbd\x35\x6d\x01\x6e\x2e\x7b\x6a\x53\xd6\xe8\x72\x0c\x34\x4b\xdf\xb4\x82\x89\x9c\x3c\x8a\xa2\x1b\x38\x84\x1b\x8c\xdd\x48\xa7\x20\xf1\xec\xb3\xa2\x70\x13\xd1\x0f\xdd\x63\x1b\x43\xda\xc3\x65\x96\x6a\xe5\x02\x73\x2c\xc6\x90\xd2\x2c\xe7\x31\x77\x5a\x9e\xa6\x80\xb5\x97\xe6\xe0\xdf\xf0\xb4\xaa\x22\xa1\xad\x0a\xbe\x48\xf6\xe8\xdf\xe0\x35\x15\x78\xcc\x17\x1d\xa5\x90\xeb\xc2\x9a\x31\xbc\x4d\xe5\x1b\xd4\xdc\x3c\xf9\x8e\x0b\xfb\xa0\x48\xfe\xcf\xef\x5e\xe5\x87\x79\x4f\x8b\x58\x15\x1e\x85\x1c\x1a\x78\x12\x72\x28\xd2\x49\xf0\x7a\x4a\x7a\xa3\xc1\xb8\x19\x05\x5b\x66\x27\x28\x07\xff\x05\x2d\x4e\xc2\x33\x3a\xa9\xe4\x70\xd3\x40\xba\xd4\x57\xa6\xa6\x1a\x94\x28\x7a\xa0\x87\xb0\x09\xd4\x07\xfd\x15\x23\xb4\x36\x73\xc1\x43\xd8\x30\x37\xe7\xc3\x39\xf0\x91\x67\x23\x86\xe3\x66\x9b\x3c\x8b\xd4\x06\x07\xa9\x72\xdf\xff\x8e\xd9\xcc\xaf\xd4\x33\x24\xef\xed\x51\x65\xe2\x88\x76\x49\x71\x34\x5e\x61\x88\xcb\x53\xe6\x2a\xd1\x41\xc3\x86\x59\x80\x69\x28\xce\xd1\x60\x62\xe3\x20\x2f\x93\xa2\xaa\x8e\xc5\x66\x22\xc9\xac\x5a\x7f\xc2\xe8\x0a\x13\xe2\xdc\x18\x18\x7d\x4f\x57\x9d\xc2\xb2\xe1\x67\xfa\x55\x43\x08\xbe\x48\x55\x19\x39\x30\x0b\x40\xa7\x58\x85\x98\xb4\x60\x97\xd9\xd4\xdd\x56\xd6\x02\x91\x5d\xb4\xcc\xfb\x8a\x2e\x80\x88\xe4\x95\x2e\x41\x71\x16\x89\xdf\x89\x95\xcc\xe5\x37\x0b\x2f\x2f\xd2\x97\x17\x49\xb0\x9a\x57\x40\x44\xc8\x8f\xb3\xb0\xe1\x46\x31\x41\x2c\xa8\xb1\xdf\x8c\x25\xb1\x1e\xa7\xf0\xe9\x13\x84\x28\x12\x9f\x3e\x21\x04\x3e\x2b\xbf\xa7\xcc\x79\x80\x35\x56\xf0\x08\x1c\x4c\x20\x13\x15\xc5\x58\xdd\x94\xef\x3b\x19\xe0\x90\xc1\x93\x2e\x89\xaf\x03\xe4\x21\xdf\xbc\x69\xc3\x01\x8f\xee\xeb\xe6\x0d\x07\xcc\xee\xc3\x07\x56\x3b\x62\x95\x99\x8f\x5d\xc0\x74\x78\x3b\x25\xe0\x09\x6b\x10\x71\xde\x63\x38\xdf\xc7\x9d\x67\xfa\x29\x65\

x1b\x1b\x17\x0e\x60\xaa\x58\x6b\x40\x45\x61\x54\xc1\xad\x6c\xa5\x62\xa9\x4f\x73\x93\xf7\x7b\x
e1\x5a\xb1\x8f\x0f\xca\x91\x23\x85\x27\x0d\x7b\xb7\x43\x76\x0c\xd8\xc3\xa9\xc3\x01\x57\x54\xb
9\x96\x5e\xef\x0a\x9b\xc2\xe4\x5e\xb3\x24\x94\x76\xad\x06\xbd\x2d\x64\xf0\x72\x8d\x12\xb5\xa2
\x5e\x2c\xce\x6e\xd1\xab\x28\xb9\xcc\x84\xc8\x61\x2e\xea\xa4\x2e\x70\xe8\x47\x67\x09\xa8\x06\
x51\x28\x4d\xd6\xde\x9e\x90\xb7\x87\xb0\xbb\x8b\xd6\x2b\x3c\x87\xc7\x50\x3e\xec\x9a\x60\xa2\
x00\xf3\x21\xb8\x7b\x5a\x79\x2f\xec\x8d\xa3\xec\xee\x76\xa4\x09\xe4\x8d\x8d\x2e\x63\xee\xc8\x
93\xce\x9c\x07\xdb\x3d\x55\xcf\xc5\xb6\x4b\x53\x03\x25\xf5\xea\x03\x07\x4a\x1e\x33\xff\x0f\x75\
x53\xaf\xa3\xd1\xc5\x08\xce\x1c\x03\xf0\x66\xb1\xf1\x39\xf8\x34\x48\x56\x34\x03\xc7\x9f\xeb\xda
\xe0\xcb\xa4\x42\x34\x77\x76\x70\xea\xcf\xbc\x89\x22\x0b\x42\x99\xf4\x2c\x58\x78\xe9\xd3\x7c\x
c0\xd8\x22\xce\x8e\x69\xb3\x10\xd5\xa5\x1b\x87\x49\xda\x9f\xfb\xe9\xb4\xba\xff\x0a\x53\xc0\x79\
x61\xc8\x33\xa0\x47\x17\x0d\x19\x2b\x6e\xac\xcf\x53\x6d\xc6\x83\x37\x34\xc5\xf4\x24\x65\x8a\x
6f\x71\x0f\x94\xc8\xe6\x4c\x43\x94\xc1\x11\x3c\x63\x0d\x86\xc2\xd3\x41\x99\x96\x5c\xf2\xaa\x55
\xf6\xb2\xb5\xa1\xa2\xd9\x81\xa7\x7c\x37\x41\x5c\x9c\xa1\xd8\x20\xbc\x0f\x8d\xa1\xbc\x00\xb1\x
ba\x5c\xe6\xd1\x7a\x79\x83\xc3\xd5\x07\x3e\x4c\x29\xe5\x98\xf9\xfa\x2b\x03\x72\xba\x5a\x1b\xb
0\x59\x26\x06\x6c\x16\x91\x66\x61\xdb\x4b\xd3\x1b\xfd\xba\x37\x0e\x0b\x42\x9c\xd4\x25\x3e\x2c
\x13\x74\x58\xbe\xad\xee\xa0\xd3\x14\x5c\x14\xb1\xb5\xea\xb2\xba\x4f\x4a\x55\xe5\x62\xb6\xc1\
xa6\x45\xf6\x54\x91\x67\x43\x9b\x7b\x0e\x91\xd8\x9c\x45\xe7\x32\x1e\x9a\x4c\x1c\x88\x07\x96\x
6d\xa1\xa2\x2a\xcf\x26\x30\x0c\xf7\x07\xd8\xca\x2e\xd6\x7f\xc0\xba\xad\x2e\xce\x88\xce\x6a\x2c
\x93\xb2\xc6\x00\x06\xb0\x92\x11\xc3\x78\x8c\xfc\x6b\x97\xf3\x41\x13\xd4\x11\x2c\x1a\x70\xd8\x
fb\x9c\xef\xfc\xb0\xc3\x2e\x82\x6e\xf4\x03\x9f\x0b\x94\x17\x8a\x79\x33\x52\x81\x47\x8a\x10\xe2\
xb7\x08\x71\xeb\x1c\x80\x88\xcf\x10\x36\xa5\xf9\xe3\x0f\x14\x8c\x14\x5a\xb7\xe9\xd4\x63\x69\x6
6\xbb\x5a\x7b\x29\x1d\x88\xad\x1a\xde\xb1\x01\xfe\xb1\x56\xbe\x23\x03\x82\xd5\x5a\x33\x86\x8
1\x77\xcc\x5c\x0b\x4d\x75\x24\x2a\x72\xc9\x3b\x66\x43\xcb\x31\x3c\x01\x02\x07\xb0\xa7\xca\x5
a\xd7\x19\x43\xaa\x0c\x22\x87\x67\xf2\x38\xa8\x77\x2c\xf2\x8e\x75\x85\xd0\x39\x9e\x8c\x0d\x0c\
x53\xf6\x6f\xdf\x90\x24\x70\x66\x45\x1f\xf3\x1a\x7a\xbc\xe5\x0f\x26\x5b\xff\x1a\x23\x54\xf3\x49\x
71\xb3\x37\xe3\x43\xbd\xa4\x9a\xc3\xd9\xa5\x9f\xb3\xe9\x62\x8d\xc5\x5d\x46\x2c\xd2\x0c\xe6\xf
b\xfb\xf0\x4e\xc0\x02\x9f\x5f\xd4\xe6\xe9\xe3\x91\xde\x71\x97\xed\xf0\xce\xbc\xe3\x73\xd8\x53\x
26\xd3\x03\x61\x02\x79\x99\x47\xe0\xe3\xbf\x9c\xea\x1a\x73\xc1\x8b\x1e\x02\xd3\x3e\xec\xdb\x6
e\xd1\x02\xaf\x7d\xcb\x28\x6c\x47\x02\x3c\xb9\x8b\xf7\x3c\xe6\x61\xde\xbf\x2f\x1f\xe1\x26\x0f\xc
1\x2b\x22\x8d\xfd\x63\xe9\xe6\xa0\xbc\x10\xc5\x80\x9b\x83\xe2\xd2\x10\xe5\x30\x5a\x0d\x6a\x8
a\x89\x93\x96\x99\x01\x1b\x8d\x78\x7e\xf7\x55\x92\x52\x03\x62\x03\xb7\xd8\xf1\xbf\xc7\x06\x7c\
x34\xe0\x63\x60\x40\x4a\x57\xf8\xe7\x18\xff\x9a\x06\x6c\x22\x03\x36\xc7\x06\xdc\x04\x9a\x51\xe
a\x86\xbd\xfc\x9b\xfa\x5d\xc6\x87\x11\xe6\xb9\xdf\x8c\xb2\x52\x63\xd4\x85\x8b\xd9\x9d\xa6\x99\
x80\xc7\x09\x34\xae\x54\x84\x59\x90\x4e\xbc\x13\x43\xba\xc4\x2a\x05\x53\x13\xda\xbe\xb7\x09\
xa4\xc9\x01\xfb\x76\x13\x88\x7f\xb4\xb3\x05\xd0\xe5\x7a\xd7\x99\x99\x6a\x23\xe6\x89\x77\xc2\x
da\xa5\x12\x96\x49\x0a\x7e\x92\x2f\xea\xd8\xea\xf6\xd6\xdc\x63\x44\x44\xf4\xf8\xbf\xcc\xad\x83\
x27\x8c\x26\x7c\x0e\x85\xe1\x6d\xe0\xb8\x1f\x60\x5f\x86\xf0\x04\x5b\x3d\xe8\x5c\xa5\x17\xe8\xf
d\xbf\xff\x8f\xc9\xd0\xdb\xb0\xe9\x3d\xfb\x9e\xa4\x70\xc3\xbf\x17\xd8\x21\xbe\x69\x51\xba\xca\x2
4\x9c\x15\x05\xb5\xa8\x23\x5f\x65\x2c\xcd\x82\xdc\x4f\x20\xc3\x24\xbf\x07\x90\xc1\x3e\xa8\x97\
xdd\xb7\x4b\x8e\xfb\xb1\x15\xb4\xcb\x14\x15\x3f\x32\x09\xfa\x58\xe6\xca\x6b\xbe\x16\x91\x02\xd
8\x83\x1b\x55\x8e\xf9\x8c\xef\x1d\xdc\xe5\xd9\xe8\x74\x43\xdf\x3d\xad\x12\xb2\x0f\xbe\x3c\x04\x
bd\x7b\xc4\xde\xfa\x51\xfe\x82\x2f\x1f\x31\x6c\xe4\x6d\x84\xcc\x8a\x95\x2f\x6f\x1a\x2f\xd5\x00\x

31\xfa\x22\x97\x53\x7a\x5a\x3a\x23\x88\xb1\xc2\x72\x25\x41\x04\x31\x12\x5c\xd9\xc0\xfd\xe5\x9
4\x3b\xe9\xb5\x7c\x21\x90\xcc\x41\x9d\xc9\xf2\xb4\x9e\xf7\xaf\x5c\x46\x29\xb9\x96\xf1\xc1\x44\x
f2\xe8\xb1\xcd\x85\xd7\x13\x6e\x83\x66\x42\xd2\x1b\xb1\x08\x06\x7c\xfa\x83\x0a\x6e\xc2\xb0\xc
c\x47\xaa\x64\x1d\xaf\xf4\xb8\x51\x07\x86\x40\xf7\xf6\x74\xec\xce\x3a\x57\xc8\x3e\x8a\xb5\x75\x
dd\x0c\x15\xb7\x4e\xf3\xeb\xe3\x6e\x9b\x0e\xf2\xb8\x1e\xa3\x54\x95\xb9\x61\x65\x6e\xba\xcb\x4
4\xfa\x03\x9f\x99\xd8\xbf\xa4\x0b\x46\x9c\x24\xe9\xca\x5b\x46\x19\x65\xfa\xcd\x26\x4e\x37\x01\x
64\x09\x2c\xa2\x8b\x05\xcd\x72\x48\xd2\x90\xa6\x22\x1e\x91\xcc\xd9\xcb\x28\x83\xc7\x3c\x9e\x
05\xfb\x60\x8d\xd4\x80\xe3\xfa\x02\xaa\x28\x3d\x10\xa6\x8d\xa7\x81\xec\x58\x00\x3b\x49\x72\x8
8\x69\x40\xb3\xcc\x4b\x6f\x0c\xf0\x2f\x31\x32\xb7\xf0\xe2\x70\x49\x21\x09\xf9\x2c\xb2\x48\x58\x
79\x53\x58\xa4\xb2\x1d\x0b\x75\x4c\xb3\xc3\x71\x7f\xbf\xb8\x31\xe1\x31\x8f\x2c\xc5\xbb\xbb\x98
\x40\x9e\x59\x37\x0e\xeb\x91\x0c\xa8\xe3\xb4\x34\x07\xd1\xe5\x71\xe2\x68\x57\x4d\x39\x6f\x02\
x1c\xb0\xb9\x71\xd1\x3b\x93\x9b\x7a\xad\xcd\x76\xb5\xb6\x11\x14\xd8\x52\xe8\x34\xf9\x1a\x37\x
4c\xd0\x6e\x8e\xd5\x75\x52\xba\xe2\x80\xab\x7c\x9a\x37\xc7\xfa\x44\xfa\x2b\x86\x46\x4a\x57\xfa
\x4c\xbc\x20\xe6\xf7\x61\x28\x36\x04\xe4\x09\xac\xbc\x0f\xf2\x85\x65\x5e\x06\xcb\x24\xbe\x60\xf
f\x2a\x7d\xad\xe2\x53\xb8\x78\xd8\xec\x23\xd6\x05\xf6\xf5\x8c\xfd\xe4\xfb\x21\xb4\x87\xd8\x6f\xf
e\xc6\xa9\x5a\xcc\xa1\x35\x85\xca\x79\x9c\xa9\x2b\x12\x98\x20\x86\xf6\x8e\xe3\xb3\x37\xc1\x19
\x39\x97\xf5\x0b\x86\xac\xe6\xee\xae\xba\x8e\x56\x55\xd6\x22\xf5\x79\x9e\x46\xde\x52\xa8\x6f\x
2c\x22\x80\x86\x74\x39\x34\x6f\xc7\xd6\x6a\x4a\xb5\x8b\x90\x8f\x79\x87\x60\x73\x35\x31\xb9\xc
a\x0c\x45\xbf\x08\xd3\x69\xba\x47\xc6\x1a\x36\x86\x49\x87\x92\xc4\x3d\xd1\x46\x5c\x2e\xc2\x69
\x6d\x79\x0b\x1f\xbf\xdc\x49\x48\x81\xde\x65\xe1\x93\xbd\x72\x4e\x7c\x53\x38\xc6\x37\xdc\x2d\x
3e\xee\xdc\xa0\xc8\x0f\xb9\x14\x2d\x3e\xda\xa6\x3d\xbe\xdf\x72\xb5\xd6\xe7\xf7\x6e\xf6\xab\xb8
\x11\x52\xe6\x94\x01\xb1\xc6\xac\x16\x1f\xe6\xd2\x73\xed\xe9\xdd\xd1\xcb\xc5\x0a\x67\xf2\xbc\x
c7\x52\x65\xb3\x9a\x5e\x0d\x10\x18\x29\x87\xd8\x5e\xcc\x71\x0d\x69\x91\x5c\xf3\x3c\x95\x79\xb
4\xa2\xf5\x7b\x12\x2f\x12\x8a\xeb\x79\x49\xcd\xb5\xd8\x82\x86\xd0\x1e\x3e\x10\xd5\x7d\x14\xb5
\x2d\x30\x83\xa7\xcb\x8b\x24\x8d\xf2\xc5\xaa\x77\xab\x39\x90\x91\x38\xa0\x94\xf3\x45\x2c\x44\x
fd\x41\x4d\x6d\x06\x71\xff\xa6\x7d\x6b\xc4\xe8\x5c\x40\x7a\x5c\x75\xf3\xa0\x7c\xb8\x57\xc2\x37\
x20\xde\xdb\xeb\x05\x69\x8f\x24\x33\xb7\x77\x58\xc0\xe9\xad\xe7\xd4\x50\xb9\xf6\x32\x59\x72\x
c1\xcb\xc1\xea\xa7\x0a\xc0\x78\x54\x68\x0d\x3a\xe4\x12\x80\x38\x2c\xfa\xb1\x05\x98\x09\xde\xf
2\x5d\xd5\x7e\x5c\xd4\x3d\xa8\x77\xae\xa2\xcc\xee\xee\x16\x1b\x39\xab\x71\xb7\xb3\x2c\x14\x9
2\x2a\x3c\xde\xc7\xe2\x6e\xad\x78\x79\xc3\x4d\x20\xfe\x8c\x32\x9d\xfd\x6b\xb6\x1b\x17\x86\x79\
x88\x02\x8a\xb5\x35\xe9\xd1\xdb\x58\xf4\x88\x59\x3f\x06\x0c\xc0\x5d\x9c\x09\xb9\x3e\x1b\x72\x
d9\xbf\xbd\x3e\x42\xf1\xd9\x76\x98\x6e\x74\xb6\x30\xd7\x45\xa7\xb7\x34\xd7\x12\x84\x97\x4a\x7
5\xda\xaa\xea\x69\x6d\xcc\xcb\x13\xee\xcc\x6e\x55\x15\x8d\x1a\x29\xeb\x80\xe7\x27\x97\x39\x8
c\xbf\x2d\x6f\xd5\x8c\x56\xd4\x10\x77\xd2\xe2\x79\x54\x7e\x79\x2d\xfa\xa6\x57\xed\x8d\x1a\x8a\
x16\xf0\x16\x6a\x9f\xd2\x58\xce\xf7\x7f\x0b\x24\xaf\x17\xd1\x92\xb2\x41\xa6\x18\xd3\x78\xf0\x07
\x47\x35\x11\x01\x12\x66\x5e\x8c\xcf\xfd\x1b\x9d\xd9\x27\x56\xce\x8d\x34\x7d\x28\xa3\x86\x85\x
24\xe3\x74\x4f\xb0\xab\xbf\x0b\xec\x53\x05\x31\x39\xfe\x37\xcc\x2b\xe3\x12\xfa\x84\xb9\x5c\x07\
x28\xdd\xa2\x43\xdb\x4a\x38\xdc\x55\xca\xa1\x74\x15\x7a\x82\xc0\xd0\xb9\xed\x19\x6e\xbb\xeb\
x1b\xc5\x0d\xc3\x34\x04\x03\xf1\xec\xe7\x5e\xcf\xc1\x2c\x28\xf5\x23\xae\xdd\xd8\x8a\x02\x89\x9
b\x3a\x98\x38\x16\xc6\xfb\x26\x28\x74\x0f\xbc\x0b\x2f\x8a\xc1\xa7\xcb\xe4\x5a\x1d\x99\x6b\xb4\
x90\x25\x10\xf0\xfb\xcc\x19\x66\x79\xc2\x73\x88\x79\x78\x3d\xd5\x36\xf6\x4a\x42\x92\x18\xb0\xa
4\x6c\xf6\xcf\x20\x79\x19\x26\xd8\xca\x92\x1a\x72\xfc\x76\x5f\x86\x73\xc8\x31\xdd\xce\x24\xf3\x

2b\xa7\xb6\xf0\xb5\x24\xac\x14\x1e\x1d\xa2\x13\xc3\xea\x32\xc3\x6c\x54\xe5\xc1\xe4\x2d\x78\x0
1\xa5\xec\xc4\xdb\xca\xcf\x9d\x86\x8b\x6d\xc7\x87\xbe\xa9\x4a\xb3\xca\xad\x94\xa5\x07\x77\xe4
\x08\x07\xfa\xa8\x30\x43\x08\x5b\x9e\x1c\xf5\xb9\x0d\xa5\x75\x29\x48\x80\xd6\x65\xcb\x21\xa0\x
b2\x2b\xa5\x45\x2c\xc2\xe0\xdb\x58\x91\x5b\x0f\x77\xf5\x91\xaa\xe1\x6d\x75\xa3\x5a\xce\x15\x0
e\x0f\xf9\x25\x4a\xdb\xd9\x0c\xd5\x4c\xa8\xe6\xa2\x6d\xab\x99\x95\x1a\xd4\xaa\x1b\x25\x0d\xeb
\xe6\xfd\x4b\xc7\xe0\x65\x72\x7d\xdb\x21\x78\x99\x5c\x77\x8c\xc0\x57\x34\xbd\x81\xd4\x4b\xe9\
xf2\x06\xef\xd1\xdd\xb2\x89\xd6\x00\xaa\x98\x14\xe2\x9e\xa3\xad\x87\x4e\x36\x39\xff\xb2\xa1\xf
3\x16\xb4\x85\xb6\x90\xdf\x94\xda\x26\x0d\x9d\xdb\xcb\x7c\xf1\x51\xc8\x7e\x5f\x95\x9e\xa1\x50\
xfb\xb6\x76\x3a\x11\x35\xa0\x2b\x49\x59\xf1\x89\x75\x51\x10\xa9\x07\x70\x08\x9d\x33\xe2\xcf\x
65\x50\xa3\x68\x97\x70\x0d\x2a\x6f\x3c\x37\x3b\x03\x02\x4f\xc5\xe6\x98\x98\x6e\xf2\x72\xb2\x6
e\x94\xdb\x06\x79\xa4\xbd\xe3\x22\x1c\xc0\x70\x72\x71\x08\x27\xee\x8e\x3e\xfc\xb8\x0e\x31\x32
\x20\x5f\xd1\xdf\x13\x7b\xe0\x11\x81\x5e\x52\xd6\x63\x5f\x9b\xe0\x6c\x23\x02\xe5\x5d\x94\xeb\x
b9\x00\x06\x2a\x0e\x14\x10\x7b\x42\x13\x42\xe2\x3a\xc6\x4c\xcd\xaa\x7a\xa9\xc4\x03\xd8\x44\x
bb\xbb\xf0\x08\x36\xc7\xb8\x21\x91\xf7\xbe\xd8\x0f\x89\x5b\x21\x33\xcd\x91\x19\xa8\x02\xf7\xf5\
x6a\xfa\x40\xe4\xb1\xb4\x8c\xdc\x5e\x8f\x84\x72\xed\xe8\xa3\x58\x7b\xfc\x58\x9e\x51\xd9\x7a\x5
b\x48\x19\x66\x13\x2b\x45\xfa\x4d\x84\xa7\x09\x04\x65\xf0\xe8\xe3\x88\x1a\xe2\x18\x4b\x71\x5d
\x93\xd8\x54\x9d\xcc\xa5\xc3\x2e\x88\xd8\x16\x27\x57\x32\x38\xe4\x65\x1f\x42\x56\x1c\x5d\xc9\
xa4\xa3\x2b\xda\xa0\x2e\xbf\x16\xf5\xa3\xc1\x57\xd8\x06\x0c\x2f\x5c\xc5\xdf\x05\x0a\x0f\xe4\x83
\x28\x04\xb7\xd7\x10\xbe\xe0\x8d\x6c\xd0\xec\xad\x7b\xc6\x77\xa6\x45\x99\x7a\x83\x58\x49\xce\
x2e\xe9\xfc\xd8\x75\xe9\xd6\xc7\x51\x0a\x87\xb0\xcb\x70\xd8\xee\x70\x0c\x5f\x3c\xfd\xb8\xfd\x0
9\x17\x75\x6e\x28\x71\x4c\x25\xa5\xeb\x94\x66\x34\xce\xc5\x66\xff\x6a\xf1\xad\x4a\x1c\x85\xbb\
xe8\xe6\xd1\x86\x86\x7b\x78\x41\x23\x9b\x30\x48\xd7\xb7\xca\xe0\xcb\xab\x74\xa5\x7b\x7a\xeb\
xfb\xef\x9a\xa9\x91\x52\xc5\x5d\x73\xfa\xbb\xf0\xa3\x4c\xbe\x3a\xbf\x76\x25\x41\x54\xe4\xf0\xc2\
x23\x2f\xc5\x78\xbd\x64\x6e\x34\xf7\x03\x52\xfa\xf1\x32\x62\xfe\xfd\xa0\xb9\xe9\x5a\xb9\xb5\xba
\x96\x71\x0a\xd2\x55\x01\xbe\xdc\xc1\xbd\x4a\xc2\xfa\x95\xfb\x41\x25\x2b\xe2\x5b\x78\x00\x79\x
22\x5f\x37\x49\x66\x46\xb1\x4d\x12\x2f\x6d\x28\xee\xf3\x00\x8b\xb0\x1f\x6f\x8a\x13\x2a\x60\x39\
x3d\x57\xf1\xcf\xb1\x1e\xde\x0f\x1f\x71\x31\x16\x8d\x2a\xef\xe4\x37\x71\xd3\x46\x4c\xab\x0b\xf9\
x9b\x9b\xba\x57\x7c\x57\x77\x61\xbb\x1a\xc9\x03\x19\x7c\x9e\xc9\x9d\xb7\x62\x94\x64\x78\x95\
x84\xb4\xb5\x91\xfb\x09\x03\xf5\x09\xd7\xe4\x7b\x8f\x0d\xdd\xe3\xb7\xac\x17\x3b\x22\xaa\x64\x
25\x0d\x25\xc7\x38\x7f\x3c\x6a\x2f\x60\xc4\xe5\x7d\xee\x6d\xe0\x51\x99\xa4\x47\x35\x24\x15\x5
7\x97\xd6\x8e\x9d\x20\x3a\xba\x2d\xae\x82\xc6\x6c\xdc\x9e\x31\x8b\x5f\xfd\xb6\x1c\xbc\x6e\x98\
xc2\xa3\x43\x29\x21\xb7\x42\xe5\x9f\xd4\x45\x42\x6c\x94\x8d\xdb\x44\x04\xdc\x08\xaf\xd8\x54\x
1b\xb7\x77\xd5\xea\xcc\x0f\x9b\x13\x0a\xab\x58\xdf\xd8\x10\x0f\x85\xdc\xe8\x0c\x5f\x3c\x92\x56\
xcd\xf9\x44\x3c\xc4\x74\x6d\xa5\x8d\x60\xb3\x0d\xdd\x49\x04\x10\x9b\x10\xda\x6c\xb9\x2d\xdd\x
b7\xbb\x31\x53\x56\x1d\x2e\x49\x99\xfa\x8a\x6f\xd1\x83\x6a\x74\x6a\x5b\x1f\x15\xfc\xca\x7e\x45
\x99\x94\x15\x4f\x39\xd0\x95\xab\x61\xcc\xf8\x95\xf6\x55\x98\x26\x3c\x5f\xa4\x6a\x01\x4f\x5e\x2
5\x32\x7d\x1b\x46\xb7\xe8\x82\x62\x69\xa6\x71\xa7\x6d\x55\xb2\x59\x90\x7b\xbc\x3d\x32\x3c\xe
0\x7b\xc1\x29\x7b\xd7\x10\x67\x5d\x82\x4d\x81\xc4\xd3\xf5\x9a\xc6\x62\xb1\x54\xed\xa0\x14\xeb
\xa0\x8c\xab\x8f\x20\x92\x2f\xdd\x43\xd3\xa4\x1f\xdb\x8b\x2d\xe6\x6d\xd5\xd1\x8e\xdd\x2f\x94\x
d4\x6b\x16\xec\xbe\x99\x1f\xf6\x0e\x21\xd6\x0c\xdd\xdd\xbb\xde\xef\x4e\x24\x71\xff\xe4\x2e\x10\
x78\xbc\xd5\x95\x92\xb0\xb7\x17\x89\x7b\x89\x39\x81\x0b\xaa\x8e\x76\xb6\xbe\xdb\x70\x8b\x0b\
x16\x77\xf9\x9d\xac\xda\x4b\x16\x41\x3a\xb4\xb9\xc5\x7c\x4a\xea\xe6\xe1\xa1\xe8\xa7\x84\x78\x

67\xdd\x5b\x5e\xdc\x08\x77\xde\x21\xfb\x59\x99\xab\x2e\x16\x87\x87\xee\xdf\x67\xc3\xd2\x13\xd8\xd9\xdb\x81\x5d\xc4\xe3\xa0\xf3\x78\xc2\x0f\x7c\x07\x49\x3d\xd9\x29\x9b\xff\xd5\xf2\x81\x2a\x0e\x0c\xd7\x52\x81\x66\x06\xac\x68\xbe\x48\xda\xe7\x5f\xc4\x99\x81\x58\x73\x41\x92\x72\x84\x14\x97\x15\x20\x60\x3e\x6f\x18\xe2\x77\x7e\x45\x52\xd6\x1a\x6e\x57\xad\x4d\x72\x55\xd5\x46\x03\x05\x97\x76\x77\x71\x43\x75\x7a\x91\x95\x87\x5a\x54\xc2\x11\x6b\x20\x47\x6d\xc8\x50\x86\x91\xbc\xf8\xa6\x30\xc1\x51\xc6\xb7\x44\xa6\xe5\x66\x49\x8d\x05\xbc\xc7\xf8\xa7\x13\xcf\x15\x9f\x18\xab\x5e\x69\xf6\x60\xcb\xe1\x04\xce\x98\x11\xb3\xa9\x03\x64\x45\xc7\x8d\x07\x9a\x96\xb6\x11\xc0\x95\x46\xc4\x54\xbe\x7e\x9e\x5e\x22\x72\xdc\x7b\x2e\x8f\x01\xb3\xf1\x05\x33\x92\xca\x99\x31\xf3\x45\x5a\x0b\x62\x3d\xc0\xc4\x24\x8c\x8f\xe5\x4d\x91\x57\x3c\x0b\x3f\x93\x4a\x5c\xc4\xac\x72\x8f\xb0\x86\x7a\xbc\x56\x75\x7a\x7f\xf4\x62\x57\x11\xfb\xe3\x6d\x2a\x17\x33\xf6\x56\xed\x3d\x90\x22\xea\xfe\x88\x95\xc7\xdd\x53\xf0\x98\xd5\xe2\x5f\xef\xe1\x2e\xb8\x38\xf0\x72\x3a\x88\xd5\x84\x17\xf9\x5c\x8a\x36\x06\xbc\x95\x4f\x9f\x60\xa7\x3e\x33\xd9\xd1\x64\x0a\x52\x36\xfe\x04\x76\x6a\x49\x5e\x77\xe0\xa0\xca\xf7\x2c\xe8\xbd\x03\x43\x83\xdf\x76\xdd\xcf\xde\xfa\x56\x3e\x3d\x87\x99\xef\xb4\x6e\xec\x69\x34\xa4\x79\x79\xf3\xfe\x65\x7e\xd2\x09\x33\xb1\xe2\x22\x47\x96\x97\x37\x6d\xb0\x37\xe2\xee\x8a\x1a\xff\xab\x93\x52\xab\x28\xbe\xcc\x0c\x58\x2f\x2f\x33\x2c\x8d\x7b\x2e\x7a\xf8\x1d\x17\xbb\xfb\x90\xc5\x01\x0e\x8d\x0a\x9b\xc5\xe7\xfd\x2d\x6a\xff\xc6\xbc\x44\x8d\xdf\x77\xbb\x13\xd4\xf7\x82\xb3\xbd\xbd\xdf\xce\xcb\x54\x26\xaa\xdb\xe4\xa5\xbd\x30\xaa\xab\xab\xe1\x45\x6f\x7c\xa3\x1d\xde\xe0\xed\xff\x56\x66\xe2\xf8\xad\x08\x67\xfc\xd6\x08\x67\x34\x91\x51\x5f\x6e\x8e\xc2\xcf\x13\x92\x68\x82\x1a\x3d\x77\x9e\x77\xde\x77\x1e\xe3\x9e\xe6\x58\x77\xd7\xb9\xfe\x9e\xf3\x5b\xdd\x71\xae\xbd\xfa\x38\x16\x57\x1f\xc7\x9a\xab\x8f\xb5\x73\x1c\x5d\x78\x25\x56\x25\x0a\xd1\x0c\xe8\x7d\x43\xb6\xb8\x34\x12\x43\x05\x73\x2f\x5a\x16\x7e\x3a\xde\xff\x08\x39\xcd\x72\x66\x4f\x15\x91\x89\xb5\x97\x66\xf4\x44\xe4\x06\xe9\x4d\xce\xcc\xa4\xee\x4d\x4a\xe7\xd1\x06\x0e\x61\xff\xfd\x60\xef\xc9\xd0\x1c\x9c\x6d\xfc\xe4\x7c\xb8\xaf\x38\xfc\x17\x26\xf9\xd3\x79\x8e\xd7\x36\xef\xbf\x1f\x9c\xbd\x1f\x9d\xef\x0e\x7f\x19\xfd\x79\x5f\x59\xf2\x3b\x3a\xe7\xf1\xc6\xfd\xf7\xbf\x8c\x44\x61\x55\xd1\x28\x2b\x44\xe4\x75\x7a\xe2\x9d\x60\x85\xbd\x27\x83\xe2\xe1\xa7\x13\xef\x44\x59\xef\x7a\x11\xe5\x34\x5b\x7b\x01\x7d\x9d\xbe\x61\x46\x02\x5b\xca\x1e\xfc\xb2\xfb\xe9\xfd\x2f\xd9\xee\xa7\x5f\xb2\xdd\x3f\xef\x5f\xf4\x25\xff\x85\x8d\x21\x66\xb9\x97\x2b\x03\x7c\xed\x41\x14\xdc\x49\xa8\x76\xcb\x51\x7c\x57\xf0\xa4\x72\xc8\xaa\x94\x1d\x4d\x24\x8b\xfc\x1d\x2a\x31\x3d\x49\x80\x6e\x02\xba\x46\xc4\x92\xda\x81\x85\x24\xed\x70\x2f\x1a\x04\x1c\x31\xe9\x18\x64\x7a\x3f\x80\x27\x37\x88\xb2\x13\xef\x84\x15\x7b\xc2\x03\x16\x07\x50\x1c\x50\xdf\x23\x70\xa0\x8a\x52\x77\xcf\x5c\xb8\xa3\x73\xb9\xea\x4f\xb1\xa8\x97\xb9\xc1\x93\xc3\x5f\xae\xcf\x7e\xb9\x1e\x9d\x3f\xf8\xf3\x70\x3f\xd2\x42\xc1\xe3\x01\x15\x91\x2b\x88\x86\xcc\xd9\x95\x01\x6b\x62\xc0\x5a\xbf\xab\xb9\xf8\x88\x03\x0e\x03\x56\xf6\x10\xd6\x56\x3d\x87\x0c\xdf\x5c\xb2\xb3\xd9\x81\x27\x40\x26\x70\x80\xa5\x0e\x61\xc7\x67\x0f\x2c\x38\x90\xaf\xdc\x56\x7d\x84\xcc\xdd\xf3\xd9\x10\xee\xb3\xaa\xd8\xde\x13\x58\x33\x42\xaf\x3a\xa6\x10\x5d\xcb\xc7\x8c\xe0\xfe\x96\x1d\xf3\xfb\x97\xa1\xf1\x50\xf7\x0e\x19\xed\x40\x9e\xf0\x4c\x01\x3b\x23\xc2\x7f\x98\x23\xa2\xba\x69\xb4\xfa\x34\xd8\x51\x58\x09\x03\x76\xfe\x4c\x76\x60\x58\x7b\xc3\xad\x82\xc1\xa0\xe2\xcb\xae\x09\x54\xf7\xac\x8f\x69\xdb\xbd\x43\xc8\xa4\x60\x5e\xdd\x99\xcf\xba\x17\x04\x35\xf0\xf7\xf7\x61\xa7\x0e\x68\x28\x6e\x42\xe2\x43\xf0\x01\xfc\x3d\xfe\xac\xa6\x87\xbe\x2a\xe2\xf1\x77\xff\x73\x3f\x0c\xc5\xfd\x61\x51\x68\xc0\x0e\xc2\xd9\xc1\x55\x06\x1f\xfd\x3f\x04\xc9\x1e\xf8\xcc\xf9\x43\xf7\x91\xf9\x80\x08\x7f\x87\x9f\xb4\xd7\x74\x9c\xeb\x3f\x1f\x71\xb7\x20\x8a\xc8\xbb\x22\x0d\xd3\x2d\x9c\x15\x17\x70\x55\xb3\x0a\x45\x0a\xac\x53\xe6\xf6\xcb\x91\x75\x40\xc7\x5c\x31\x11\x6d\xb

8\xcf\xab\xec\xc2\x60\xc3\xa0\xd2\xab\xa3\x0c\x86\x98\xd3\x21\xbc\x01\x9c\xb5\x70\xad\x33\x68\xc7\x28\x50\x6a\x9f\xe3\xda\x09\x56\x6b\xf1\xbe\xfa\xec\xef\x43\x3b\x8f\xa9\xb8\x4e\xa6\x5e\x78\x7f\x1f\xda\x89\x4b\x77\xc2\xe8\x4a\x05\xd3\x56\x96\x3c\x4d\x5e\xc6\xb9\x5c\x7c\x7f\x1f\xda\x09\x5a\x77\xe8\xc7\x16\x48\x4c\x3e\xde\x2e\x79\x91\x2b\x4b\x4e\x54\x25\xa9\x0a\xcf\x69\xbb\xe4\x52\x0d\xd3\x55\x95\x54\xc2\x6c\xe7\xfa\xdc\x41\xd7\xbf\x51\x96\x51\x5e\xc1\xa4\x55\x12\x2a\x99\xa4\xe0\x12\x9b\x49\x34\xcb\xb2\xa2\x0a\x36\x95\xa9\xbb\xa4\xf2\xac\xa8\x82\x4f\x3c\x39\x65\x8b\xf7\x44\xc1\x28\x8c\x5e\x2b\xba\xa5\xe0\x14\x2e\xa4\x2a\x8a\x2a\x58\x85\xf3\x22\x45\x51\x05\xaf\xf2\xe4\x39\x4e\x1e\x1a\x32\x45\x14\xcc\xaa\x45\x3e\x2b\x35\x21\x0a\x6e\x89\x88\x64\x0d\x05\x26\xfd\x0a\x6e\x15\xeb\x4f\x0d\x04\x2c\x05\xb7\xf2\xe4\x85\xc8\x12\x56\x15\x66\x45\x55\xdc\x4a\xae\x55\x32\x60\x29\xb8\x25\x05\xee\xcb\x2a\xac\xa8\x82\x5b\xc5\xc2\x50\x13\x57\x05\xb7\x2a\x7b\x52\x2f\x5a\xe7\xd6\xf9\x19\x37\x63\xe7\xcc\x52\x0f\x86\x68\xbc\x57\xd9\x05\xfb\x75\x80\x3f\xd0\xb0\x35\x86\x6a\x34\x6a\x23\x0c\x1a\x1c\x4a\x0d\x71\x03\xd7\x88\x5a\xaa\xec\x30\x46\x58\x38\x94\xad\x62\x38\x4c\x44\x61\xc3\xc6\xfd\x2c\x54\x2c\x56\x88\x63\x34\xb5\x45\x49\x48\x57\x23\x71\xdf\x0a\x9b\x7f\x26\x57\x34\xdd\xbf\x64\x33\xb7\xbd\xf9\xb2\x11\xdd\x79\x39\x87\x54\x84\x6e\xf2\xc5\x8d\x01\x51\xce\x7e\x89\x5d\xc5\x6c\xae\xc3\x37\x77\x7a\x29\xe5\xab\xe4\xa2\x51\x6f\xce\x2f\x0a\x6a\xe6\xb3\xea\x89\x06\x88\x95\x28\xe6\xe1\x87\x7c\xcd\x52\xbd\x5c\x19\xe2\xda\xd4\x6f\x98\x1f\x25\x36\x20\x8e\x0c\x48\xc3\xb6\xb3\xaf\x3f\xe8\xbd\x4e\xae\x33\x62\x4a\x89\x57\xdb\x53\xd9\xe2\x9c\x32\x1b\xc7\x9b\x6e\xfd\xa8\x9d\x88\x70\x13\xe8\x26\xb3\x69\xd8\x5a\x19\x56\xe7\xf6\xaa\x88\xb6\x7d\x46\x2f\x90\x36\xa5\x09\xaf\x05\x93\x2b\x73\xb7\xc2\xa8\xaf\xd8\x63\xbe\xce\x2a\x8b\x28\x6e\x03\x42\x6f\x21\x48\xe2\xdc\x8b\x62\x14\x94\x50\xdd\x40\x7b\xf5\x3c\xc6\x1b\x9e\xa2\x98\x41\x50\xd6\x29\xfb\xad\x0a\x85\x28\x16\x5a\x18\xd1\x5b\xad\xa4\x61\xd1\x4c\xcc\xaf\x66\x42\x02\xf6\x24\x4c\xd8\xdf\x87\xdf\x0a\x48\x5e\x90\x5f\x7a\x4b\x35\xc0\x41\x34\x67\x53\x28\x43\xf0\xc8\xab\xc1\x1d\xb6\xe1\x26\x97\xf9\x41\xc7\xf2\xd6\xf7\x1d\xa1\x1f\x65\x1e\xd7\x8d\x82\x70\x50\x46\x85\x42\xbe\xd9\x85\x67\x7b\xc2\xe8\xd0\x87\x22\x3a\xf4\xa1\x88\x0e\x85\xfa\x05\x31\xcc\xf0\x19\xc2\x9e\x32\x07\x18\x94\xf1\xf1\xb6\x60\x32\x52\x88\xb4\x48\x2a\x8c\x35\xa9\x65\xb7\xc8\xe4\x0b\xc5\x2a\x51\x95\x77\x59\x57\xec\x37\x44\x5e\xff\x3e\xe6\x44\x61\x72\x29\x42\x41\x5d\x33\xd8\x82\x31\x8d\x7e\x7a\xb9\x10\x8b\xdf\x34\x12\x59\x7c\x52\xbc\xfa\x07\xf6\x85\xdd\x38\x03\x46\xd6\xdf\x30\xa2\x76\x0e\xdf\x02\x31\x3b\x2e\x43\xee\x5b\x36\xc3\x1e\x54\x37\xfa\x0c\x30\x70\x47\x60\xd8\x77\xb1\x4f\xf1\xe1\xf1\x6e\x4c\x83\x5a\x25\x7e\xeb\xdd\xac\xdb\x9d\x03\x4e\xfe\xec\xef\xc3\x09\xcf\xc4\xe8\xdf\x40\xf6\x31\xdd\x62\x67\x7d\x95\x96\xb2\xc0\xe7\xd1\x21\xc4\x11\xcf\x4c\x79\x99\x2d\x06\xe6\x56\x97\x82\xe0\xde\x84\x8e\xab\x82\xe5\x4f\xb8\xdd\xde\xf6\xa8\x9e\xf5\xbb\xaf\xf8\x6f\x18\x47\xdd\xab\x18\xb1\xdb\xd7\xc8\x56\xbb\x10\xa1\x58\x2a\x62\x06\xa5\x6f\x43\xfb\x17\xb5\x14\x97\x69\x46\xcf\xe2\xa8\xef\x0e\x82\x1e\x0b\xd6\x73\xe2\x41\xb2\x59\x1d\x86\x6a\x5b\x04\x54\xe6\xba\x67\xbb\x7a\x4f\x42\xf7\x5b\x34\x64\x88\x04\xd7\x22\xfb\x68\xcf\x60\xd3\x00\x7c\x5a\x23\x5f\xad\x2e\x1f\x35\xa3\x0c\x2e\xa2\x2b\x1a\x33\x7d\x92\xd3\xbd\x77\x43\x56\x88\xa2\xce\xae\x2b\x7a\x79\x57\xc3\x07\x85\xf1\xfb\xff\xb9\xfb\xba\xed\xb6\x91\x24\xcd\xfb\x7a\x8a\x70\xef\xba\x48\x9a\x00\x09\xd2\x52\xd9\x96\x4c\xeb\xd8\xd5\xaa\x6e\xcd\x71\x51\x5a\x5b\x35\x3d\xbb\x2a\x6d\x1f\x90\x48\x49\x90\x48\x80\x06\x40\x89\x94\xe5\x79\xa7\x79\x85\x79\xb2\x3d\x19\x99\x09\xe4\x2f\x08\xaa\xe4\xde\xe9\xd1\x85\x2d\x01\xc8\xc8\xc8\xbf\xc8\xc8\xc8\x88\x2f\xae\xb9\xa9\x31\x80\xbd\x47\x0a\x42\xa8\xb9\x62\xb7\x3e\x2e\x81\x47\x73\x81\x3c\xea\x76\x99\x60\x30\xc5\x54\x11\x4d\xd6\x15\xd0\xde\x06\x65\xd4\xee\x5c\xc1\x87\x90\xac\x16\x19\xc9\xd1\x5f\x08\x12\x78\x6e\x6d\x70\xe9\x49\x14\xce

\x66\xf2\x42\x51\x50\x5a\x5d\x95\x28\xe0\xad\x61\x41\xb5\x59\xd2\xbb\xec\x95\x17\xb1\x6f\x82\xd7\xaf\x06\x3b\x78\x91\x86\xba\xcc\xd0\x13\x78\xeb\x82\x2f\x3a\x8b\x72\x78\x35\xd8\x71\x8d\x1f\xae\x7a\x05\xa9\x95\x61\xfa\x88\xc1\x4c\x70\x30\xed\x6d\x73\xad\xd5\x0a\xdc\xd5\x51\xe9\x01\xb4\xe9\x94\x79\x32\xac\xd7\x3d\x4a\xed\x1d\xec\x62\xe9\x88\x96\xde\x95\xa9\xda\xd0\x5e\xeb\x23\xf5\xf1\xdc\x73\x77\x45\x10\x22\xc9\x40\xd6\x9d\x91\x8b\xd2\xcd\xca\x54\x8b\xd2\xa8\x66\x99\xb6\x4b\xcf\x96\x03\xb8\xe6\xff\x9b\x6b\xe5\x1c\xf6\x70\x15\xb1\xc1\x41\x48\xda\x0e\x5b\x3a\x9d\x8d\xd8\xb4\x5b\x80\xd3\x8a\x1f\x66\x79\xb5\xa1\xd4\xba\xea\x90\xe1\x63\x37\x61\xac\x6e\x56\x21\x64\xac\xf1\x88\xa5\x2d\x50\x32\xb0\xd6\xdb\xa9\x23\xf0\x99\x25\xd3\x0e\x85\xa4\x55\x34\xf0\x20\xe8\xe1\x3f\x01\xfb\x57\xfc\x17\x0c\x80\x14\x0e\x5d\xbb\x6a\x36\xa2\xb4\x94\xa3\x95\x47\xf0\xbc\x92\xb6\x1b\x22\x05\x98\xb1\xd5\xcf\xa3\xa6\x41\x0a\x9b\x9a\x62\xbf\x01\xb5\xf1\xcb\xaa\xae\x15\xba\x6e\xbd\xd6\x0a\xdd\xb8\xa1\x60\x75\xb1\x4e\x56\x53\x92\xe7\xa6\x4f\xb6\xf4\x53\x39\xd9\xd6\x1f\x0b\xe4\x49\x97\x38\x30\xfa\x80\x43\x55\xd7\xa8\x60\x49\xec\xfb\x8f\xd4\xc3\x55\x0e\xea\x35\xbd\x1b\x69\x9e\xc8\xd9\x2b\x36\x9d\x43\xf0\xce\x66\xf7\xa7\x57\x41\x50\x42\xf0\xee\xfe\x84\xb9\xbc\x2e\xe0\x95\xdd\x4a\xe0\x9e\x04\x78\xc8\xa5\x52\x66\x4e\xc2\x24\x47\xc9\x53\xa7\x77\xd4\xe1\xa6\x51\xe5\x10\x37\x79\x26\xb4\x24\x64\x0b\x9b\xfc\x2a\x37\x8c\xeb\x73\xc4\x38\xbd\x71\xe3\x24\xd6\x4c\x21\xb4\x5b\x2d\x17\x6e\x0f\xc7\x7a\xd9\x22\xce\x18\xcd\x12\xb3\x1d\x5d\xa8\x92\x5e\x85\xac\x7e\xd4\x79\x57\x66\x15\x0f\x71\x8d\xc3\x7c\xd1\xce\x21\xc2\xb9\xd8\x9e\x83\xf3\x0e\xab\xa3\x8b\x7a\xc2\x2e\xe2\x6f\xd0\x26\x11\x45\x64\x83\x6a\x08\x7a\x78\xcc\x75\x65\x31\x70\xfa\x93\x6c\x22\x58\xd2\xa0\xe7\xf5\x1a\x5c\x51\x95\x01\xb6\x3c\x2d\xb5\xde\x34\x50\xff\xa1\x34\xbc\x21\x64\xfc\x8d\xdc\x37\x57\x98\x7c\x61\x9a\x91\x30\x6f\xd2\x1d\x5c\xec\x20\x95\xa6\x71\x92\xab\x1e\xd9\x14\xc0\xa7\x90\x2f\x81\x06\x79\x28\x94\x10\xc8\x4f\x13\x62\xbd\x01\xce\x15\xb6\x39\x6b\xf2\x05\xde\x6c\x20\x45\xdb\x12\x86\x59\xcb\x1b\xd7\x80\x9d\xb2\x26\xdf\x3f\x6f\x76\x68\xdf\x20\xcc\xe1\x11\xd9\xdd\x36\xee\x59\x9b\xf0\xfd\x40\x59\x4c\x12\xca\xfa\x8a\xa5\x12\x13\xe9\x9d\x5c\x49\x36\x1c\x4c\xc8\x1e\x5b\x35\x1e\x56\xac\xf3\x7b\x44\x73\xd3\xb2\x31\x69\xde\xf2\xda\x2a\xad\x1c\xb2\x1c\x2a\x85\xec\x9a\x45\x89\xa9\xce\x59\xee\x8a\xcf\x84\xe2\x61\xcd\xc9\xf5\x43\x4d\x77\x98\x4a\x87\x8e\xd1\xfe\xe9\xf8\xf4\xf8\xf4\x7f\x9f\x1c\xf6\x8f\xc6\x9f\x4f\xdf\x8f\x7f\x3e\x84\x5f\x0f\x4f\xff\x7a\xfc\xe7\xcf\x9b\xa3\xd6\x94\xcb\x62\xb8\xbb\x4a\xf3\x32\x5a\x82\x9b\x83\x27\x79\x3a\x5b\x16\x8a\x55\x3c\xce\xed\x21\x63\xd5\x0d\xc5\x49\x4f\x94\xc3\x3c\xb7\x30\x62\x4f\x60\x04\xf5\xee\x5a\x2b\xc3\x63\x99\xd6\x66\xcb\xcb\x22\x34\xfa\x0e\xbf\xed\xd7\xd6\x85\xa5\xd3\x1a\xc4\xf0\x6d\xec\x0d\x47\x27\xc8\xd1\x79\x21\x2d\x37\x23\x32\x7d\xe1\x4e\x9d\xf0\x4d\x35\xe3\x19\x94\xd3\x0b\xdb\xd4\x56\xba\x71\x4a\xe2\x59\x6d\xaf\x89\x1c\x5d\xb6\x98\x24\xec\x3b\x0f\x79\x65\x47\x02\x8f\x9e\x1c\xb7\xe8\x12\xf9\xc9\x40\x0d\x56\x32\xfb\x20\xce\xe1\x32\x23\x21\xb3\x1d\xf0\x58\x1e\x33\xf8\xb1\xbd\xf6\x60\x22\xe7\xcd\x7a\x01\x7e\x13\xd2\x6a\x8c\x50\x23\xba\x01\x27\x2b\x90\x4a\xaf\x08\xe4\xe1\x9c\x17\x56\xbe\x4c\x33\x76\xda\xd7\xd9\xe0\xb8\xc0\xcc\x0f\xde\x39\x44\x1c\xe6\xe4\x34\xc5\x69\xce\xb0\x43\x24\x2f\xad\xb5\xcd\xeb\x2e\xb6\x18\x60\x05\x70\xb8\x00\x32\xa0\x9d\xe0\xe9\x3e\x3f\x9c\xda\x36\xa3\xa8\x5b\x28\xb5\x80\xd1\xda\xae\xf7\xac\x7d\xa3\xf4\x9d\x6d\xb4\xea\x3c\xfb\x94\xce\xe3\xcc\x9c\x30\x5e\x68\xff\x45\x8b\x8d\x52\x22\xf1\xe0\xd6\xbc\xa3\xc4\xac\x52\x74\xaa\xeb\x50\xd4\xcc\x73\x4f\x8e\x89\x34\x9c\x7c\x12\x9e\x63\xf7\x56\x72\x90\xe6\x0e\xc0\x32\xb6\x2f\x5f\x49\x0a\xf6\x6f\x47\xf2\x19\x36\x6f\x45\x4b\x7c\x08\x75\x0c\xd4\x5d\x55\x89\xad\x4d\x74\xb1\x5a\x65\xfd\x1c\xc1\xf4\xec\xf6\xbc\x0c\x4a\xda\x67\x49\x3f\x03\x9e\xc6\xaa\xcc\x45\x98\xb0\x08\x79\x83\x40\xc2\x05\xa6\x0d\x9e\xd1\xe2\xb6\xeb\x9e\x56\x78\xca\x09\x60\x04\x47\xc6\xd3\x31\x8c\x6

0\x6c\x3c\x3d\xa2\x35\x2a\x4f\x03\xe8\x33\x3e\x8c\xa7\x81\x41\x21\xb0\xd2\x0d\xac\x74\xc7\x9c\xee\xd8\x78\x6a\xd2\x1d\x5b\xe9\x8e\x39\x5d\xf5\xe9\x11\xa7\x7b\x64\x3c\x35\xfb\xe1\xc8\x4a\xf7\xc8\x42\xf7\x69\x37\x21\x86\x54\x8f\xf7\x4e\xf2\x67\x72\x25\xba\xb8\x2c\x37\x2e\x25\x35\xab\x9a\xbe\x10\x4d\xaa\x4a\xd4\xb1\x73\x31\x33\x06\x3e\xac\xd9\x42\xc6\x5c\x2f\x8d\x04\xe1\x4b\xab\x20\xc4\x1c\x64\x35\x42\xd0\xd3\xf3\x2c\xaa\x5c\x6e\x27\x23\x1b\xf4\xbb\x88\xf3\x11\x91\xa5\x72\x4e\x3f\x75\x50\xac\x5d\xae\x8f\x8b\x3e\x18\x1b\x7a\x15\x7d\xe5\x68\xfd\x55\xf7\xe2\xa3\xa6\x7d\xbc\xf3\xc8\x3e\x0e\x3c\x2a\x06\xb7\xda\x6e\x78\x7c\xd4\x86\x1d\x9d\x7c\x59\x86\x33\x61\x46\x6e\xb4\xa1\x57\x61\x55\xe2\x22\x41\xcf\xce\xac\x74\x1c\x56\xc0\x36\x15\xf2\xa5\x69\x37\xed\xfe\x81\x3d\x99\x9f\x75\xfe\x8b\xab\x9c\xfe\x26\x9d\x13\x4d\x59\x4f\xa9\x74\xbe\xfc\x2e\xf3\xe7\x71\xca\xe6\x96\x73\x88\x57\x72\x4a\xeb\xa0\x13\xe9\xb2\xf1\x7a\xfb\xe9\x8f\x4c\xa4\x77\xdb\x4d\xa3\xc7\xf4\x58\x9a\xd9\x57\x60\xed\x76\xf1\xf8\xde\x3b\xce\x0e\x69\x6d\x5c\x49\xbe\x2c\x48\xd3\x7e\x7c\x65\xed\xc7\x36\x4c\x64\x20\xe8\xfa\xde\xec\x70\x60\x28\x1e\x9d\xa0\x6a\x3f\x4f\xd6\xbb\x21\xe0\xb2\x22\xa5\x2f\xd9\x96\x9d\x15\xe7\xbf\xb0\xf2\x0d\x56\xde\xb3\x67\x42\xd3\x7d\xfa\x66\x48\x79\x6d\xf5\x16\x6c\x6a\x00\xdf\xa1\x70\x88\xf1\xaf\x6d\xda\x02\x3f\xfe\xb8\x41\xd5\x7e\xc7\xf5\xfb\x4a\x41\x1f\x7e\x87\x0e\x60\x59\x60\xb6\x1d\x3b\x16\xd8\xb5\xb9\xb1\xd8\x82\xfc\x3b\xb0\x9d\x90\xcb\xb0\x88\x6f\xc9\x63\x78\xe7\x45\xf9\xb8\x8d\xc9\x65\x93\xa6\xb0\x96\xd0\x33\xc5\x77\x68\x0d\x82\xa8\xfa\xc1\x23\x1a\xf3\x7f\x48\x96\x6e\x3b\xed\xd8\x6f\x22\x0f\xce\x77\x68\xce\x23\xac\x17\x5b\x36\x9c\xd6\x50\xee\x52\xb3\xc6\xbb\xd4\xeb\x3f\xb2\x4b\x7d\x9f\xa1\xaf\xfa\xea\x1f\xb1\x45\x89\x7e\x53\xf7\xa7\x59\xf3\xfd\xe9\xcd\x53\xed\x4f\xbe\xba\x41\x35\xe8\x57\x48\xc0\xc7\xf3\x67\x62\x3c\xb5\x9d\xc3\x7d\x3c\x7f\xfa\x47\xda\x31\xda\xc7\x83\xad\x9f\x18\x8f\x03\xcb\xf9\xdc\x46\x39\xb0\x53\x1e\x73\xca\xfa\xf1\xda\xb7\x1e\xc5\x6d\x84\xc7\x9c\xb0\x7e\x90\xf6\xad\x47\x71\xdf\x7a\x14\xb7\xd1\x3d\xb2\xd0\x7d\x5a\xe5\x1c\xe3\x7b\x9a\xce\x59\xd7\xd4\x64\x44\xe8\x74\xcc\x97\x93\x06\xd3\x91\x27\x80\xbc\xf6\xa0\xf0\x60\xf5\xf1\x74\x6d\xf1\xf0\xe7\x36\x32\xf3\x4d\xc8\x32\xc2\xe9\xa6\x33\x34\x53\x6a\x57\x53\x66\x2a\x7e\xce\x8f\xfa\xd9\x04\xf3\xc2\xe9\x14\x95\x9c\x70\x96\x60\xfc\x67\x21\xba\xdd\x4c\x5c\x41\x8f\xe3\x70\x6c\x89\xee\xff\x1c\x5f\x26\x39\x44\xf1\xc5\x05\xc9\x2c\x44\x43\xcc\x26\x69\xbd\xa1\x59\xe3\xed\x81\x3f\x31\x2f\x65\xc4\x15\x42\x6f\x31\x5b\xe6\xed\x75\x3d\xce\x04\x5e\x5e\x88\x4c\x65\x95\xd2\x62\xf6\xf3\x9a\xa5\xd5\xaf\xff\xc8\x1d\x88\xe1\xc8\xb6\xc7\x73\xe7\x11\x96\xc2\x8d\x38\x81\x02\x78\xd7\x8b\x13\xe0\x86\x24\x7c\x6b\xc9\x66\x8a\x79\xed\xda\x65\x77\x79\xb0\xc6\x2c\xdb\xfa\x3c\xa0\x5f\xad\x60\x0f\x53\xdc\xb9\xb2\x2b\x33\x1e\xec\xa0\x9b\x72\xce\xfd\x06\x69\xff\xaa\x54\x79\x6b\x5a\x74\xcd\x02\x50\x98\x3b\xa3\x07\x2b\x29\x28\x45\x3c\x4b\x59\xcd\xf4\x0d\xa6\xf9\x0b\x33\x82\x0f\xec\x37\x9b\xbc\xf1\x8c\x8b\x06\xed\x5f\xf1\x0f\x57\xee\xfc\x7e\xfd\x3e\x1c\x1d\x1e\x1e\xc2\xab\xdd\x1d\x68\x0f\x83\xe0\x75\x07\x7e\xea\xbd\xdc\x43\xf9\x8c\x82\x38\x60\x30\x30\xa5\x5b\x4b\x91\x56\x47\x76\x07\x4d\xd5\xee\x85\x49\x7d\x0e\x28\xa1\x3d\x7b\x1a\xa4\xda\xeb\xc6\x95\x92\xeb\x6e\x45\xb4\xf2\x6b\xe5\xf5\x5a\x7f\xcd\x66\xae\x04\x59\xae\xaf\xd4\x2a\x77\xf7\xdd\x55\x3c\xbd\x12\x82\x74\x12\x5f\xd2\x83\x83\xd3\xf8\x8d\xe2\x09\x73\x00\x3a\x26\x37\xbb\x14\xfc\x78\x4a\x85\x53\x58\x1b\x55\x41\x49\xf9\xa1\xfd\x1a\xbd\x40\xee\xb7\x44\x06\xc0\x2e\x59\x39\x30\xcd\x0a\x84\x70\x6f\x74\xe7\x5d\xf4\x32\x72\x4b\xb2\xdc\xec\x37\xe0\x17\xbd\x19\xa9\xa0\xce\xe8\xbc\x40\xed\x88\xaa\x3a\x02\x54\xc5\x99\x80\x8f\x0a\xe3\x70\x1f\x26\xbe\xbf\x0f\x45\x5d\x50\x81\xc2\x84\xb5\x0b\xac\x6b\x5a\xd4\xcf\x58\x12\xd1\x72\xcc\x7f\x08\x93\x82\x5b\x5d\xb3\xae\xf1\xda\x85\x8f\x1a\x1f\x64\x29\x26\xe3\x2d\x67\x7c\x2d\x3d\xeb\xc0\x01\x84\xb0\x67\x8d\xf8\x67\x4d\xa5\x54\x26\x2c\xdf\xe0\x04\xde\xc2\xf5\x3e\x4c\xea\xb2\x0e\x0b\xff\x8d\x09\xba\x

6f\xac\xf1\x97\x5a\xcf\x3b\xc6\x2d\x2b\xf1\x96\x15\x70\xfb\x64\xd4\xf8\x81\x6c\x89\x94\xd6\xef\x
c3\x8a\xd6\x77\x00\x88\x8c\x47\x57\x1a\x57\x8b\x59\x5c\x1a\xbf\x43\xda\xb0\x90\x28\xf7\x1d\x3
e\xcd\x3d\xb6\x5a\xd7\x53\x8f\x6d\x2b\x85\x27\xc4\x9b\xb9\x6d\x4f\x70\x80\xae\xb5\xc1\xf0\x75\
xb7\x0f\x1b\xf2\x8e\x8c\xcf\x47\x59\x5e\x4d\x29\x27\xf9\x55\x9a\x15\x3a\x87\x0c\x23\x44\xe4\xb
c\x08\xe5\x3c\x71\x6b\xb9\x14\x84\x79\x85\x6a\x8f\xe9\x7f\x68\x19\xfc\x2e\x2f\xc2\x0c\xfd\xc3\x
4b\x3e\x2d\xd2\x64\xa2\x80\xfb\xf1\x65\xb1\x2a\x61\xbc\x0d\xa1\x39\xe1\xc9\x41\x2d\xd9\x81\xe
4\x3b\xbe\xf5\x94\xc1\xbf\xeb\x61\x6a\xa2\x9a\x6b\x78\x47\x17\x61\x8d\xfc\x62\xa8\x4a\x6c\x5a\
x5d\xbb\xe7\x61\xe5\x70\x73\xbd\x0f\x31\x3d\xc1\x3e\xe3\xee\x36\xfb\x3c\x49\xe6\xc8\xd4\xc7\x
c4\x8f\xef\xe3\x27\x0e\xe8\x04\xac\xb7\xeb\xca\x85\x6a\x43\x4f\xc0\x12\xfe\x88\xb1\x5c\xab\x24\
x39\x12\x38\xe3\x05\x13\x8b\x20\x91\x80\xb5\xc4\x85\xd4\x6c\x6d\xef\xcc\x2a\x6f\x2d\x3a\x17\x
71\x60\x6e\x0f\x7c\x7f\x6d\xc2\x45\x72\x0f\x61\x9b\xbe\xb9\x72\x6b\x18\xfd\x3e\xfc\x92\xce\x66\
xe9\x1d\x65\xd5\xb2\x69\x9b\xea\x19\x86\x96\xb2\x7d\xbc\x1b\x00\x26\xef\x93\x37\x76\x63\x67\
xbf\x0b\xb3\x28\xb7\x5a\xe4\xf9\x0f\x5b\x90\xf6\xfd\xdd\x81\x62\xb3\xe6\xfe\x46\x6b\xb7\xbf\x91\
x50\x6b\x36\x8d\x96\xb4\x18\xa7\x65\xf8\xb3\x9c\x64\xb8\xbb\x02\x1f\xba\x6b\x2a\x39\xcb\xc7\x
3f\xfe\x08\x3e\x7d\xec\x2b\x8f\x75\xd2\x94\x12\xb7\x97\x32\xc4\xc6\xb5\x35\xa9\xb6\x84\x77\xb6
\xf6\x50\x66\xad\x49\xc3\xdb\x04\x16\x87\x42\xcf\x84\xda\xf1\x0f\x9f\x8f\x1d\xcf\xe9\xb9\x50\x3b\
x4f\x43\x00\xcf\x71\x04\xb5\xd3\xb0\x2f\x9e\xfb\x81\xf9\xbd\xad\xde\xc0\x51\x6f\x20\xea\xd5\xe8\
x8c\x45\xbd\x63\xf3\xb9\x8d\xfe\xd8\x41\x7f\x2c\xe8\x6b\xcf\x8f\x1c\xf4\x8f\x1c\xf4\x8f\x1c\xf4\x8f
\x6c\xf4\x9f\xf8\x40\x9d\x46\xcb\x59\x63\x2b\x50\x0f\x43\x92\x78\xf6\x86\x08\x95\xa6\x1c\x52\xc
d\xf8\xf6\xeb\xf1\x9f\x7f\xfb\x78\x5c\x7b\xa1\xcd\x6b\x1d\xb1\x5f\x1b\x1e\xc1\xbf\x78\x60\x39\x5f\
x8b\x93\xb7\xf5\x80\x3d\x68\x7c\xc0\x36\xe5\xa9\x9c\xc7\x1c\xcf\x3a\x9a\xe7\x8d\x57\x66\x0d\x
a7\x1f\xf1\xe3\x8f\x65\x4f\x7c\xb6\xea\x4d\xe5\x54\xe6\x6b\x66\x1a\x7d\xb6\xee\x39\x23\x60\xb6
\x38\xa0\x73\x3e\x57\xe5\x29\x4d\xe6\x12\xd9\x36\xd9\x92\x5d\x2d\x29\x1b\x94\xab\x15\xe7\x6a
\xb5\x25\x57\xab\xfa\xe3\x3b\xd6\x21\x4d\x08\x2a\x63\xdf\xb8\x4f\xd1\xcb\xe9\x2c\x8e\xe2\x30\x
61\x59\x5f\xe2\x34\xd9\xc3\x54\xe7\x79\x7c\x99\xb4\xd7\x1d\x78\x01\x2c\x1a\x60\x05\x7d\x08\x2
7\x79\x7b\xdd\x31\x83\xb5\xfa\x7d\x0c\x0d\xa3\x82\xf2\xcb\x1a\x10\xed\x8d\x64\x84\xca\x82\xb7\
x23\xc8\xe0\x2d\x2f\x68\x94\xcb\x85\x55\x45\x7f\xb1\xb6\xb9\x41\x02\x4f\xc2\x8e\xd7\xfc\x2b\x8f
\x4e\xa3\x40\xbb\x8f\x55\xcb\x5b\x08\x7f\xe9\xe5\xf0\xc2\x78\xe3\x3c\x11\x99\xd5\xc9\xfd\xda\x0
8\xad\x73\xc5\x4c\x5f\x6d\xf8\xd2\x43\xec\x17\xda\xa7\x4f\xed\xcc\xe1\x12\x34\x78\x8d\x42\x22\
xc5\x14\x8f\xb8\x13\x3c\x4f\x63\xcc\x1c\x6c\x94\xac\x6e\x8a\xcc\xe0\x04\x50\x68\x24\x1b\x6e\x5
2\x1a\xbb\xbc\x32\x1f\x57\x7f\xc5\x96\xa6\xe9\xbf\xb6\x95\xbb\x2b\x24\xd0\xb5\x9a\x8d\xbb\x56\
xb3\x71\x17\x65\xbb\x6e\x35\xee\x32\x74\x61\xe3\xa9\xcd\x68\x6c\xa3\x1b\x58\xe9\x8e\x39\x5d\
xdd\x0e\xdc\xb5\xda\x8c\x6d\x74\xc7\x9c\xae\x6e\xf1\xed\x5a\x6d\xc6\x5d\xab\xcd\xd8\x46\xf7\xc
8\xc2\xef\xd3\xce\x3c\xc4\x81\xfd\x83\x26\xe3\xc5\x8c\x5b\x8c\xe9\xd9\xa9\xd9\x76\x55\x3c\x95\
x95\x78\xf8\x4f\x6c\x25\x7e\x0a\x33\x31\x93\x59\xff\x3d\xec\xc4\x7c\x36\x4b\x6e\xbc\x71\xe9\x0
d\x5d\x3d\x74\xa1\x81\x5b\xcc\xc7\x3a\x0c\x39\x3a\x4e\x3e\xc6\x40\xfc\x3d\x2d\xbe\x2e\xe3\xb3
\x66\x05\x5e\xd7\xda\x7d\x21\x44\xd7\xf3\xfa\x69\xf0\x9d\x8d\xac\x4d\x0c\x85\xf0\x4b\x98\xa3\x3
3\x50\x0a\x4b\xbc\x2c\x45\xab\x1f\x15\x40\x09\x44\x29\xf0\x0c\x8f\x79\x03\x5b\xac\xb5\x1b\xc3\
xda\x94\x05\x8f\x33\x99\xd6\x19\x61\x1f\x65\xd5\xdd\x60\x83\xd5\x5f\x0a\xd3\x43\xf8\x07\xac\xa
8\x6a\x95\x8a\xc5\xd3\x22\x18\x1d\x60\xdd\x74\x88\x35\x23\xe0\x2c\x4d\x2e\x49\xc6\x6c\x81\x2
c\x9f\xe2\xa4\xcc\x15\xc5\x8d\x66\x4e\x73\x58\x08\x3e\x9a\x4a\xe9\x78\x15\xb2\x45\xb0\xb4\x11
\x16\x1e\xb7\x20\x9b\xb8\xda\xc9\x6c\x2d\x8c\x6e\x11\x5a\x35\x64\xdb\x1b\xfa\xd0\x87\x6c\x5b\

xbb\x58\x66\xb8\xae\x2b\xe0\x87\xd5\x14\xa6\x61\x02\x13\x02\xf1\x65\x92\x66\x7a\xa0\x62\x65\xce\x0d\xf6\x61\x62\x4f\x71\x10\x32\x0b\xf2\xf4\xcc\xf7\x27\xe7\xa5\x59\xb6\xcb\xed\xb8\x5d\x08\x11\x18\x08\x6d\x77\xd6\x40\x73\xf6\xfd\x73\x9b\xbd\xcb\x72\x44\x08\x6d\x2c\xac\xa6\x65\x3a\xd4\xd0\x32\x15\xba\xdd\x35\x79\x94\x9d\x85\x49\x2f\x66\x64\xe9\x72\x23\x4b\xc0\xad\x2b\x5d\x6e\x5d\x09\x74\x4a\xb8\xb0\x44\x28\x5c\x17\x06\xb0\x48\xf3\x3c\x9e\xcc\x88\x4d\x5d\xfc\x03\xc6\x15\x6b\x64\x89\x05\x4d\xaf\x3e\xba\xc4\x95\x97\xec\xec\xfe\x1c\xbe\x4e\xd2\x74\x46\xc2\xe4\x81\xd1\xff\x06\x7f\x13\x10\x14\x29\x4c\xd3\x65\x52\x08\x37\x35\x1f\xfd\xb1\xd5\x88\x8a\x3d\x74\x36\xf1\x98\xeb\x87\x07\x03\xba\x09\x04\x4e\xa5\xa9\xcc\x74\x84\x77\xed\xaa\xe2\x74\xdf\x3c\x00\xc5\xad\x36\xf1\x6d\xda\x5c\x3d\x15\x0e\x67\xbb\x53\x65\x84\xe0\x90\xba\xac\xfd\x94\xf5\x49\x9c\x84\x19\xbf\x45\xd9\x83\xaf\xf7\x1a\xb4\x2e\x2e\xe2\x7b\x39\xd5\x18\xfd\x63\x04\xcf\x9e\xdd\x57\x7f\x0c\xaa\x5f\xed\x72\xd9\x02\xcc\x3b\x78\xe9\x41\x4b\xb0\xd5\xa2\x3a\x74\x5a\x7c\x48\xd3\x99\x07\xf7\x36\xa9\x57\xf1\x41\x6b\xee\xc0\xbd\x15\x50\xd7\x76\xf4\xae\x8f\xce\xd1\xa3\x72\xcc\xd8\x9d\x5b\x19\xb8\xdf\xc4\xbe\xd0\xe3\x67\xec\x4a\xcf\xf6\xb1\x3a\x3c\xf0\xdd\x75\xe9\xb6\x21\x40\xc7\xc6\x43\x99\x8f\xb4\x09\x24\x9e\xab\xde\xdb\x32\x3d\xc2\xad\x08\x2c\xaf\xea\x36\xc3\xd9\x6d\x03\x82\x13\x87\x03\x89\xc0\x3b\xcc\xba\x92\x80\x0b\x5a\x64\xab\xe0\xa1\x27\xf6\x7c\x9f\x87\xab\x78\xbe\x9c\x6b\xc7\xa5\x68\xa1\x87\xb9\xd9\x61\x3d\x51\x3b\x2c\x52\x08\xcc\x20\x17\xc8\x48\xbe\x20\xd3\x22\xbe\x25\x33\xaa\x6a\xca\xe4\xd3\x79\x5c\x14\xca\x76\xa5\x8a\xaf\x68\x71\x0e\x5f\x85\xdc\xfa\xb3\x0a\xe0\x02\x47\xc2\xbb\x36\xa0\x35\xff\xfa\xfe\xdf\x18\x4e\x63\x1e\xdf\xaa\xd9\x0f\xcf\xb2\xb9\x44\xe5\x93\x92\x25\x51\x23\xf2\xda\x4e\x42\xa6\xc6\xb0\x79\xdb\x1d\x23\xfc\x4f\xca\xb2\xb2\x07\x5f\xa3\x85\x2c\x5b\xdc\xc5\xd4\x54\x2c\x75\x05\xd5\x2e\x37\xca\x65\xf3\x46\xe5\x34\x36\xb5\x52\xb2\x34\xc7\x72\xea\xd9\x37\x5a\xb0\xc4\x7c\x36\x49\xee\xb2\xbe\x98\x4b\x22\x5a\x94\xb9\x0f\x1f\x1e\x94\x64\x8e\x94\x7e\xe0\xd1\xa1\xf4\x60\xb0\xeb\x48\x50\x24\x82\x25\x3c\xf8\xf7\x7f\xc7\x6c\xaa\x72\x84\x04\xc3\x29\xe2\xb4\x2d\x3a\xec\x33\x7b\xee\xc8\xc1\xae\x9e\x37\x12\x0e\xb4\x59\xbc\x27\x32\x47\x36\xb2\xbc\xfd\xa3\x96\x30\xaa\x4b\xcc\x92\x76\xc3\xa7\x94\x5c\x41\x7b\x91\xde\x91\x0c\x85\xde\x20\xe8\xf4\xe0\x33\xfd\x5c\x41\x09\x63\xb0\x5f\xef\x68\x3f\x60\xea\x1c\x09\x8c\x0a\xdf\xbc\x55\xb7\x7b\x99\xf8\x4d\xb5\xa8\xca\x55\xe4\x53\x95\xe9\xf3\xfb\x5f\x0e\xff\x7e\x34\x3e\x3d\xfc\xcb\xe1\x27\xbe\x34\xd5\x67\x1b\x17\xd9\xd1\x05\x03\x21\x91\x67\x2b\xf2\x57\xe5\x74\xe2\x1a\x89\x94\x5d\x5a\x20\x9b\xfc\xe7\x7f\x60\xc0\xf2\x0d\x43\x58\x4d\xe5\x83\xb6\x22\x7b\x84\x7f\xab\x73\xa5\xf3\xab\x50\x4d\x9f\x90\x17\xdd\x8d\xb6\xe6\x8c\x12\xda\x72\xbb\x71\xae\x36\x2c\xa9\xac\xb6\x1b\xd7\x32\x63\xb7\x1f\x96\x49\x27\xcf\xed\x1b\xcb\x58\x78\xc6\x48\x78\x30\xf8\x49\xd1\x4c\x3a\xfa\x66\xda\xef\x63\x47\x0e\x48\x77\x38\x60\x42\x5e\xa4\xb1\xba\xe9\xb0\x1e\x67\x39\xfe\x89\x35\xab\xa5\xbe\xb1\x1e\x40\xc2\x2d\xd1\xd0\x1a\x10\xaa\x08\xc9\xe4\x74\x23\xbd\x61\x1d\x48\xd8\xa5\x05\xcb\xb5\xca\xc0\xd8\x28\x73\xe6\xa4\x7b\x78\x80\x1b\x0e\x65\xa1\x3c\x37\xaf\x01\x0e\x30\x65\xde\x0b\x4e\x49\x20\x0d\x0e\x98\x59\xc5\xf8\x7a\x4f\xcd\x93\x55\x67\x1b\xce\xbf\x64\x45\xdb\x4f\x3a\xe6\x35\x1f\xbe\x81\xb1\xeb\x8d\x7f\xe4\x2c\xc3\xde\x1c\x59\xde\x04\x1d\xf3\xda\x93\x51\xc3\x37\xca\xc5\xea\xe3\x24\x50\xfe\x65\x19\x66\x04\xb2\x34\x2d\x36\x85\xba\xcb\x15\x3c\x69\x5c\x2c\xe3\xe1\x13\x65\x01\x4f\x18\x5f\xb2\xfa\xc0\x98\x32\xeb\x20\x64\x1e\x64\x64\xe1\x6d\x67\xa4\x75\x1a\x05\x73\x66\xbd\x35\x5f\x70\x83\xa4\x25\xcb\xd1\x02\x46\x7a\xb3\xbb\xb0\x63\x7e\x78\x15\xce\x2e\x8c\xad\xb4\x15\xf4\x76\x5b\x16\xf3\x94\x88\x37\xe9\x8f\xc3\x71\x5f\xc8\xb8\xbe\x69\xef\x63\x58\x7c\xfc\xf4\xf2\xf0\x40\x4f\x09\xf8\xef\x56\x37\x7f\xf0\x0c\xef\x4c\xca\xfc\x92\x6d\x4e\x86\x53\x39\xc0\x6b\xd1\x3d\x10\x8e\xab\x03\x61\x99\x94\x09\x9b\x27\xf6\xa3\x24\x46\x89\x41\xf2\x22\

x9e\x87\x85\x96\xc1\x9c\x76\xf4\xaf\x61\x71\xd5\x63\xb3\xbc\xbb\xb2\x5d\xdd\x96\x1f\xc0\x52\x8
0\xce\xf4\x53\x6b\x96\xb2\x7e\x1f\x4e\xc2\x3c\x67\xa8\xf5\x55\x31\x84\x78\xe2\x1b\x18\x9a\xed\
xb8\x0f\x0d\x87\xbd\x64\x7e\x34\x02\x9a\x11\xf7\x1a\x8b\xed\x27\x2f\xa1\x26\xf1\x37\xde\x7c\x4
7\x32\x49\x35\xc3\xf0\xd4\x75\x04\xa4\x12\x8f\x9f\xd6\xba\x98\xa0\xee\x39\x0c\x05\xde\x57\xc2\
x33\x8b\x9a\x45\xd5\x4e\x4b\x2c\xc4\x15\xab\x68\x99\xeb\xb4\x03\x7d\x04\xc4\xf4\x79\xb2\xb4\x
80\xe5\xd9\x7d\xce\xb0\x4e\xec\x1c\x6e\x68\xab\x68\x2f\x17\xf6\xb6\x44\xf2\x75\xb6\x48\x4c\xb0
\xdc\x53\x53\xeb\x3a\x9c\xb1\x50\xff\xe4\x86\x5c\xba\xed\x27\x3d\x04\x9a\x3d\xbe\x68\xb7\x48\x
ab\xc3\xdb\x67\xe7\xc0\x6c\x5a\x66\xac\x40\xbd\x17\x9d\x6c\x9b\x45\x21\x87\x2e\xcf\xe5\x55\x5
3\x6d\x09\x10\x7a\x61\xf5\x2c\xc0\x5b\xee\x69\xba\x9c\x45\x54\xc3\x11\x66\x78\x81\x98\x14\xe7
\x65\x22\xec\x0a\x73\x16\x85\x72\x95\x10\x7b\x8a\x21\xa4\x91\x41\xf6\x94\x7e\x76\x87\x94\xa7\
xe1\x32\x27\x10\x96\x97\xf1\x54\xab\xc4\xaa\xda\xab\x7e\xd1\x41\x31\x7d\x45\x92\x29\xa9\x7c\x
0e\x26\x64\x96\xde\x79\xdc\x03\x59\x22\xa2\x57\xa2\xcc\x77\xa6\x64\xaa\xe9\x3c\xc5\x7c\xca\x
dc\xee\x08\x74\xd2\x66\x3d\xcb\x00\xd2\xf9\x4e\x67\x70\xb4\x70\x2c\x23\x2a\xb2\x5e\x42\x07\x3
f\xb4\x01\x8b\xa2\x24\xbd\x2b\xd2\xc4\xff\x14\x2e\xae\xf2\x34\x81\xb8\x20\x99\x23\x2b\xb3\x8a\
xff\xa7\xbf\x05\x6e\x25\xcf\xec\xb3\x94\xce\x0e\x2a\xe1\x85\x12\x54\xb0\x98\x88\xf2\x9e\xbf\xf0\
xf0\x0c\x34\x60\xd1\x4c\x75\xc8\xaa\x5a\x8e\xf2\xa2\x37\x05\x80\x8e\xb4\x00\x72\x1e\x0d\xd5\x
a6\x4b\xc3\x4a\x07\x7f\x34\x3a\x19\xa6\xbc\xd7\xe9\x6c\x4a\xd4\x76\xaa\x09\xca\x8c\x25\x68\x2
7\x19\xa1\xb3\x35\x4d\x88\x16\xb7\x77\x11\x27\xe1\x4c\xa8\xed\xa2\xa0\x3d\x67\x1e\xaf\x81\xf4
\x2e\x19\xa0\xea\x9b\x37\x6f\xde\x40\x9b\xf8\x3b\x1d\xf0\xfd\x77\x0c\x6b\x95\xfe\xfd\xb2\xe3\x4
1\x9e\x0a\xc9\x9d\xd3\xdf\x4d\x4c\x4b\x33\x83\xba\x54\x05\xd5\x6e\x50\x66\x90\x08\xa6\x69\x9
6\x91\x69\xa1\xbb\x4c\x1a\xa3\x90\x21\x70\x19\x95\xcc\xbe\x6f\xf1\xf8\x10\x3f\x8a\x70\xca\xc1\x
87\x97\x1e\x4a\x85\x41\x2d\x34\x21\xef\xd8\x9d\xe2\x4a\x07\x04\xe6\xb9\x39\xe2\x84\x67\xb2\x
42\x3f\x0a\xda\x64\x1e\x14\xb8\xb3\x6d\xbb\xb1\x57\xd3\x0c\x76\xb0\x77\xd1\x4d\x23\x5c\x2c\xb
2\x34\x9c\x5e\xe1\x4d\x44\x45\x6f\x42\x7f\x09\xb3\x75\x07\x73\x77\xc4\xc9\x12\x6f\xba\xea\xe8\
xd7\xac\x25\xa5\x2f\x13\x4c\x95\x47\x39\x69\xa1\x76\x92\x91\x05\x2a\xfb\xf8\x78\x07\x1f\x37\x4
1\xdc\x3c\x96\x01\x35\xcb\xba\xd1\xb9\xd9\xe3\x97\x02\x45\x0a\x39\x41\x7f\xa8\xb2\x59\xcb\x05
\x47\xb5\xae\x6b\x0c\xaf\x81\xac\xc2\x69\x21\x66\x2f\xbf\x93\x49\xe2\x84\xe4\x38\x2e\x31\x93\x
8d\x64\xb6\xa6\x1a\x27\x09\x37\xa4\x2d\x60\x56\x5b\xda\xd6\x26\x70\x91\xdc\xf6\x51\x78\x50\x
a0\xc5\xc3\xd0\x28\x87\x9e\xe3\x3a\xd8\x5a\x6f\xc1\xe5\x50\xd1\xe9\x91\x2f\xed\x95\x3b\x11\xa
4\xc1\x07\xd5\x99\x9b\x41\x56\x36\x04\x70\x74\xe3\x2d\xc2\x66\xdc\xca\x68\x41\x95\xa1\x9d\xfa
\x5a\xf2\x06\xdf\xd0\x71\xa8\x85\x86\x6c\x8c\xa3\x7c\x74\xa1\xaf\x41\x5c\x68\xc9\x72\x36\xf3\x2
0\xf8\x1a\x78\x3b\xdf\xe8\x82\xdb\xa5\xbf\xbe\xfc\xe6\x49\xd7\x55\x38\xbd\x36\x91\xe7\x6a\x28\
xad\x26\x49\x0b\xae\xb6\x56\x29\x91\xb4\x2b\x42\xba\x73\xcf\x2b\xe4\x57\x4c\xa9\xd4\x64\x5a\x
76\x31\x97\xf4\xb3\xae\x90\x5d\x83\x0e\x3f\x7e\x5f\x85\xd9\xfb\xa2\x1d\xb0\x74\x97\xbb\x0d\x16
\x26\x67\xfb\x94\x9f\xf6\x85\x9d\x89\xad\xd2\xa6\xc0\xc3\xe2\x87\xaf\x02\x7a\xa6\x73\xae\x82\x4
1\x13\xac\xd9\x39\x8c\xe0\x59\xc6\x17\x41\xc6\x17\xc1\x26\x5c\xd0\xda\xd7\x1b\x66\xfb\x53\xa6
\xfa\xdf\xd8\x0b\x03\x0d\xc7\xc9\x83\x79\x53\xe3\x45\x82\xae\x17\xba\x03\x1a\x7d\x6a\x73\x6c\x
7b\x61\x75\x6c\x7b\x61\xc5\x25\xb3\xd1\x0d\xac\x74\x03\x4e\x57\x77\x4b\x7b\x61\x75\x6c\x7b\x
61\x75\x6c\xb3\xd1\x1d\x5b\xe9\x1e\x71\xba\xba\x0b\x9b\x8d\xee\x91\x95\xee\x91\xa5\x1f\x1e\x
67\x6a\x71\x19\x7b\x71\x9e\xfe\x51\xcf\x36\x46\x04\x3d\xb1\x97\xb3\x86\xae\x6d\x53\x0f\x88\x9
4\xf9\x6c\xee\xc1\x6a\xfa\xd1\x83\xd5\x2c\xf5\x60\x75\x15\x7b\xb0\xa6\x7f\xae\xe9\x9f\x6b\xfa\xe
7\xbd\xc5\x66\x82\x69\x91\xd1\x26\xf5\xc1\x9a\x20\xd9\x6d\x86\xd9\xe0\x9c\xd5\xe6\x6e\x73\xaf\

x3c\xb7\xab\x1c\x74\x6c\x97\xc6\x95\x67\x9c\xa7\x01\x2d\xfe\xe7\x7f\x04\xf6\x10\x17\xe1\x8f\x5
5\x65\x34\xd8\x10\x5c\x6b\xb8\x96\x2b\x50\x98\x68\x51\xa5\xfa\x31\x22\x75\xa0\x01\xfe\x8a\x30
\xcb\xb4\xec\xdb\xed\xf4\xb0\x62\x6e\xac\xc2\xd3\x7c\x35\x15\x11\x4c\xcc\x3a\xca\x19\x5d\x73\x
ff\xf3\xf2\xf1\x6a\xea\xf6\x27\xea\x4d\xb9\xbb\xdc\x88\xbb\xea\x39\xb2\xbf\xd6\xc6\x71\x32\x47\x
67\xd3\xbf\xd1\xec\x14\xab\x63\x9c\x82\x7b\x59\x03\xef\xaf\xf8\xc7\xb9\x15\x17\xb9\x4d\x0e\xc0\
x62\x9d\xa9\x40\xe7\x26\xb0\xb3\xb1\x11\x09\x9b\xc7\x10\x05\x35\x91\x8d\xeb\xba\x3c\x0a\xb6\
xad\xc1\x65\xae\xab\x8f\x42\x52\x6d\x3e\x2b\x1d\x12\xa8\x2b\xbd\x5c\x6b\x2f\x35\x9f\x3a\x69\x7
4\xe5\xe7\xab\xe9\x47\xb7\x13\xd6\x1a\x5f\xd6\xb8\x61\x1d\x26\xf9\x32\x23\x74\x0a\x2f\xd2\x38\
x29\xd0\xd9\x4e\x76\xc5\xc2\xc5\x41\xab\x28\x52\xa0\x2a\x8d\xd3\x07\x8b\x7e\xf3\x16\xeb\xeb\x
c0\xfd\xd4\x1a\x9f\x79\x3f\xf5\x78\xb4\xe5\x47\x8f\xb3\xcd\x04\x18\xfe\x1a\x9b\xcc\x71\x23\x65\x
cc\xdc\xfa\x4a\xd5\x1f\xf9\xba\x8b\x8b\x2b\x1b\xa6\xb7\x8c\xe5\xfd\x11\x1d\xa9\x3e\x7a\x8c\xa1\
xb3\xf3\x7d\x88\x7d\x7f\x1f\xee\x95\xac\x64\x4a\x69\x9e\x8e\xdb\x74\xa6\x12\x02\x14\x46\xf0\xf
9\x7f\x7d\x3a\xb5\xa1\xa5\x56\x35\xaf\xa7\x1f\xf7\xc1\xf7\x31\x47\x5b\x60\x37\x58\x4c\xed\x73\x
6f\x8d\xd1\x3a\x6b\x0c\x81\x7c\x5e\x56\x6a\xf9\xee\x2a\x2e\xbf\xeb\x57\xcc\x3d\x58\xed\x2c\x15
\x54\x3f\xf6\x3c\xcb\x2c\xd5\x85\x1b\x16\xd0\x19\xbb\x2d\x2a\x2b\xe4\x06\x9d\xd3\x6e\xea\xf9\x
c1\xaf\xaf\x58\xaf\x9f\xdd\x98\x3c\xd9\xbe\xa7\x5a\x20\x6d\xc7\x0b\xac\xa7\x8b\xe5\x5f\xd0\x1e\
x70\x90\x67\x5d\x33\x4b\xcb\x02\x6d\x68\xc3\x5c\x62\x0b\xa1\x6c\xa5\x3f\xba\x70\xcf\xa2\x41\x
c1\xe2\x3b\x29\x06\xa1\x8d\xc4\x58\xd2\x7b\x76\xcf\x8c\x94\xe7\x5a\x13\xf0\x31\xe7\xf6\xca\x91
\xaf\x84\xd6\xc6\xb0\xf0\x29\xc9\xe7\x48\xb2\x91\xdd\x93\xb1\x39\xd2\xd9\xb4\xf8\xb5\x4c\x6d\x
43\xd5\xed\xea\x0e\x7a\x2e\xf1\x78\x5f\xc6\x99\x36\xba\x4c\x57\xdc\xeb\xe8\xea\xdd\xd2\xbb\xe
e\xfb\xfa\xc8\x6c\x93\xfe\x16\x37\x7c\xd5\xb1\x20\xbe\x80\x6c\x8e\x97\xdd\x1b\x9c\x61\xf2\x48\x
72\x63\xf9\x6c\xd4\x28\xf9\xb2\x0c\xfe\xa1\x0e\x31\x22\x57\x74\xbb\x03\x95\x07\xa0\x76\xf1\x9d
\x47\xda\x4d\xb9\xb5\x90\x71\xbf\x5e\x57\x6c\x5b\x7f\x18\x67\xd1\xe6\x2e\x31\x82\x84\xaa\x36\
xb3\x94\xc1\xdb\x79\xc5\xc8\x1f\xa2\xa3\x64\x1e\xc9\x4e\x31\x8a\x9b\x0a\xa5\x3f\x10\x5e\x31\x
af\x3d\x39\xe7\x38\xde\xbb\x61\xa1\x3d\x4a\xc1\x10\x71\xd9\x9c\x65\x4d\x73\x91\x96\x3c\x60\x5
e\x37\xf6\x80\xb1\xfa\x1e\xe4\x11\x1c\x48\xae\x39\x65\x9f\xec\x6d\xeb\x0a\x93\xb3\x3b\x82\x8c\
x2c\x32\x92\x93\xa4\xb0\xe2\x02\xcb\x40\x67\x89\xd5\xf3\x80\xea\x0b\x32\xfd\x32\xeb\x0e\x2e\x
4b\xb5\x61\x45\x0a\xd1\x02\x2e\xe2\x15\x89\xdc\x59\xc2\xfe\xeb\x3b\xa6\x69\xf7\x64\xdb\x7b\x9
a\x6d\x20\xb0\xc9\xc3\x4d\x2f\xbe\xfd\x02\xad\x2b\xbf\xcd\x2a\x95\xe8\x34\x72\x60\x13\xa0\x9c\
x98\x71\xbe\x6d\x3d\x86\x46\x0b\xd9\x2d\xd8\xe9\xb4\xf6\x06\xd7\x0d\x77\x4c\x1b\xd0\xd9\x8f\x
06\x3f\xba\xce\xe8\xbb\xef\xbe\x14\x70\x12\xfb\xa8\x42\x57\x4b\x41\x74\xa4\x5c\x89\x7b\xce\x3f\
xfd\xce\x35\x4e\x0b\xb2\x07\x61\xce\x74\xe5\x7f\x09\x6f\xc3\xcf\xd3\x2c\x5e\x14\xad\x5c\xb8\xe
7\x16\xeb\x05\xf1\xa0\xed\x07\x9d\x5e\x91\xfe\x42\x99\xa2\x4a\x71\x9c\x43\x2b\x68\x29\xee\x24
\x93\x65\xc1\xb2\x55\xb6\x7d\xbc\x0e\x0a\x06\x46\x09\x3f\x68\xfd\x93\x2d\x5b\xc6\xfe\xa3\x16\x
ac\xa3\xe8\xe6\xa5\x2a\x0a\x6e\xbf\x48\xed\x25\xb7\x59\x9e\x48\xe1\xd1\x0b\xb3\xe1\x52\x1c\x
9a\xfe\x5b\x07\xa6\xc7\xa8\xb2\x42\x87\xc1\xff\xd7\x15\x4a\x94\x0d\x8b\xaf\xc2\xb9\xb9\xec\xd8\
xd2\x55\x47\x9c\x79\x6f\xb2\x7e\x2a\x74\x77\x2b\xca\xd1\x22\x4b\x17\x24\x2b\x62\xd5\x3f\x94\x
3b\xb5\xfc\x72\xfc\xe9\xd7\xf7\xa7\x90\x4e\xae\xc9\xb4\x80\x76\x4e\x88\x14\x5a\x32\x4d\x93\x8b
\xf8\xb2\xe3\x9a\xbf\xbc\xec\x48\x1e\xb9\x17\xec\x3f\xce\xe2\x67\xb2\x08\xb3\xb0\x48\x33\xd8\x
83\x56\x4f\x5d\xcf\xf8\x73\x99\xa5\xcb\x85\xf2\x95\xe7\xfc\x2a\xbe\x27\xb0\x07\x2f\xcd\xd7\x39\x
99\xa6\x78\x33\xf9\x17\xe9\xbb\xc0\xfc\xee\x22\x0b\x71\xd2\xfd\xc5\xa8\xf4\xf7\xd5\xfb\xa0\xe5\x
01\x47\x9f\x49\x13\x1f\x8d\xf9\xb4\x0f\xf3\x45\x38\x25\x1b\x28\xf1\x0a\xe5\xaf\xbe\xed\xff\x93\xc

9\x21\xb6\xcc\x1e\x27\x88\x1c\x65\x1b\x48\x22\x51\xf2\x11\xa2\xc8\x5e\x74\x2b\x59\x84\x24\x1a\xbb\xb9\xe7\x05\x02\xfa\x3f\x46\x1e\x0d\xb6\x97\x47\xa6\x7f\x00\xbb\xb6\x65\x78\xcb\xb6\x23\x38\x83\x2e\xb5\x5a\x05\xc2\x8c\xb2\x9e\x17\x59\x2f\x5f\xcc\xe2\xa2\xdd\xea\xb5\x3a\xf6\x2f\x2f\x07\x30\x82\x2e\x5b\xd8\xbd\x72\xd9\x39\xbe\x1d\x4a\xdf\x9a\x6b\xd0\x51\x48\x5d\x7b\x23\x50\xea\x12\xcf\xed\x65\xe3\xa4\x38\x09\xd1\xef\x33\xcc\xb2\xb3\xe0\xdc\xfe\x95\x58\x9b\xd2\xa7\x03\xc7\xa7\x02\xa7\xbb\x82\xe1\x76\x56\x5c\x9e\xfe\x58\x99\x03\xc1\x4c\x75\x3d\xba\x27\x1e\xd9\x69\xcc\x08\x3d\x11\x96\x94\xec\xd6\x50\xe0\xe3\x7c\x39\xec\xa0\x15\xef\x72\xe0\xb1\x21\xb9\x1c\x7a\xac\xbf\x63\x0f\x29\xf9\xa6\xb1\x52\x94\xa5\x05\xde\x31\x3f\x4d\xfa\x65\x5d\x08\x31\xad\x82\x7e\xf3\x9c\x96\x79\x78\x80\x4b\xc7\xf5\x77\xd5\xf1\x15\xff\xf9\x72\x92\x17\x19\xfa\x09\xc5\x4e\xb7\x03\xe1\x31\x15\xc3\x5b\x5a\x11\xfd\xa5\x4b\x5b\x55\x6b\xb4\x17\xb5\xd1\x2f\xd5\xd9\xd2\xb5\xd4\x1f\x63\x0f\x39\xae\x6e\x1d\x57\xb6\xac\x9b\x86\xbc\x6f\x1a\xd6\x87\xe3\x1c\x3b\x2a\xa2\x14\x71\x66\x74\xa4\xce\x6a\xf9\x2d\x46\x82\xfe\xdd\xc8\x0a\xc7\x65\x8c\x34\x83\x2d\xb5\x1d\x54\x1c\x8b\xd5\x63\x6c\xb7\xcc\x32\xa9\xac\x50\x73\xcf\xb2\xa7\x3f\x3f\x50\xea\xef\x65\x04\xc5\x39\x4b\x51\xf2\x89\x5c\x1e\xae\x16\x6d\x68\xfd\xfe\x7b\xf4\x95\x36\xee\x72\x08\x5d\x68\x7d\xfb\xfd\xf7\x0f\x2d\x0f\x5a\x97\x2d\x70\xc8\x15\x80\xd6\xff\xfc\xb1\x55\x31\xec\xd8\x8a\xed\x1c\xed\xa9\x6b\xda\xf6\xd1\x9e\xbd\x9b\xed\x46\xc5\xbc\xc8\x1e\xa5\xeb\x31\xab\x7f\x63\x8d\x2f\xcc\x69\xd1\x78\xbe\x98\x91\xb2\x01\x78\x0e\x92\xeb\xa8\xf6\x39\x7a\x20\x22\xac\x1b\x10\xcc\xaf\x7a\x11\x91\x24\x9d\xc7\x09\x7d\xc5\x50\xb9\xa4\x07\xa5\x6b\x47\x08\x8b\x34\x8f\x8b\xf8\x56\xcd\x61\xc3\x21\x1c\x38\x93\x6e\x5c\xf8\x7c\x41\xa6\xf1\x45\x4c\xa2\xd2\xd2\xa9\xd4\x7a\x74\x51\xd9\x40\x65\xfa\x32\x27\x31\x53\x17\x4a\x4a\x2c\x42\xc7\xc6\x2b\x0b\xb6\xbf\x23\x79\xc1\xf9\x4a\xc8\x94\xe4\x79\x98\xad\xa1\x48\x15\xd3\x8d\xe8\x6b\x39\xa0\x13\xbd\x64\x14\xff\x3c\x55\xc3\x9a\x57\xc6\xd2\x07\x36\x72\x0f\xe5\xa0\x94\xf1\x4a\x18\xd4\x89\x1d\xfd\xb6\xba\x6b\xc5\xce\xb5\xb5\xbf\x46\x71\xe2\x03\xdb\xee\xd0\x82\x4a\x6b\x0d\x35\x66\x6e\x9a\x34\x6b\x8a\x6b\xaa\x8c\x56\x58\x53\x65\xc4\xf4\x92\x95\x99\x79\x64\x53\x63\xc2\x2c\xf3\x20\x0a\x3c\x88\x86\x78\xb5\x4f\x56\x0b\x0c\x93\x48\x02\xcf\x01\xbb\x76\x03\x23\x1e\x7c\xf5\x34\x57\xf7\x91\x61\x19\x3d\x1e\x1f\x5a\xe4\x46\x42\x37\xbe\x28\x68\xf8\x75\x44\xbf\x4e\xec\x5f\xdb\x34\xa9\x79\x54\xaa\x6e\xb6\xcd\x88\x47\x9b\x8d\x58\xb0\x99\x29\xbc\x4d\xf3\xee\x3c\xb2\xec\x0d\x25\x99\x1b\xd7\x6e\xfd\x8c\x5d\x8b\x25\x2c\xd9\x4b\xbb\x03\x1d\x84\xa7\xea\xcd\x0a\x64\xbd\x1e\x74\x57\xc4\x9a\xbb\x37\x53\x1e\x85\x3e\x1c\xba\x7d\x6c\x01\x5a\xfa\xdc\x6b\xe1\x06\x72\x03\x07\xd0\x92\xa7\x61\x8b\x9e\x95\xd4\x69\x4d\x25\x3e\xed\xcb\xed\x36\xe0\x7e\x5f\x0f\xe7\xdb\x73\x7d\x77\x74\x41\xc9\x2b\xb9\x81\xd2\xc4\x8f\xa5\x55\xcc\xad\xd7\xc0\xd2\xac\x4b\x72\x93\xae\xee\x65\x4e\xc0\xe5\x58\x36\xa7\x33\xf1\xd9\x0d\x77\x6c\xa3\xff\x55\x56\xec\x44\x84\x95\x0e\xa0\xd3\xbb\x2c\x08\x1b\x8c\x03\x48\xb8\x9a\xbe\x25\x22\x78\xe9\xb4\x50\x81\xc1\xf7\x0a\xe1\x7a\xad\x5f\x0a\x18\x51\x22\xab\x69\x2d\x0a\x78\xcc\x03\x69\xec\x12\x8b\x7d\xcd\xbb\x10\x83\x42\x59\x4c\x68\xe9\x73\x32\x8f\x13\x14\x78\xfa\x24\x28\xae\xc2\x4a\x9a\xcb\x57\x65\xa6\x00\xe6\x4e\x06\x11\xfa\x33\xe4\x15\xcc\xc0\xaa\x47\x4c\xb0\x81\x88\xa7\x89\x81\x93\xe3\xbf\x7d\xfe\xfb\xe9\xe1\xf8\x0c\xda\x54\x0c\xa1\x2f\xff\x73\xd9\x33\x81\x45\xe5\x49\xa0\x04\xf4\xab\x3d\xfc\x57\xf3\xaa\x60\x43\x39\x8f\xd8\xda\x99\xce\x17\xed\x08\x33\x80\x21\xe0\x3c\xe1\xbf\x45\x74\xec\x06\xe2\x66\x42\xe5\x1e\xab\xe7\xa0\x1f\x2a\x69\x81\x04\xc2\x83\x5d\x4c\xe0\x04\x55\x0c\x18\x81\xcf\xfd\x3e\x13\x4b\x28\xa0\x54\xb0\x91\x24\x10\x5d\xa1\xdf\x99\xcb\xa1\x06\x75\x00\x81\x89\x07\x91\x99\x5f\xb0\xec\xe9\x21\x4a\x50\x1e\x69\x20\x90\x00\xa3\x81\x15\x6f\x87\x05\x68\x0f\xb1\xf3\xe8\xd6\x81\xd1\x3d\xce\x2c\xec\x11\x6b\x92\xe5\x05\x4a\xed\xa1\x45\x62\x32\x01\xad\x31\xc3\x78\xc4\x

51\xb1\xf0\xc4\x3a\xce\x42\x0c\xf7\x10\x1d\xe2\x90\xb7\xd7\x41\xca\x42\x49\x5b\xa4\xac\x38\xed\xd6\x79\xc4\x69\x47\x41\xc7\x83\x68\x60\xef\x62\xb6\xdb\x88\x16\x45\x43\xce\x46\x62\xf6\x2f\xeb\x2d\xe3\x4b\xcb\x48\x24\x01\x73\xbd\x1a\xf4\x72\xb0\x78\xda\x10\x78\x31\x82\xe1\xe6\x94\x00\xa5\xbe\x19\xe7\x30\x9d\xa5\x39\xc3\x7c\x59\xd1\x6d\xbe\x1f\x61\x62\xa8\x64\xd0\x8f\x06\x0a\x19\x66\x25\x60\xd3\x6a\xc0\x5a\x4d\x8c\x7c\xa0\xbc\x5f\x56\x9d\x5e\x38\xc9\xdb\x1d\x9c\x2d\x16\xe1\xca\xa8\x04\x4c\xd3\xd8\x48\x05\x97\xfa\xc0\x71\x1a\x39\xa3\x9d\x51\xc9\x4a\xca\x98\xf4\x27\x9c\x3b\x8e\x0c\x67\xb4\x2b\x95\x62\x81\x52\x4c\xeb\xc3\x6a\x95\x93\xd5\xc2\x7a\xcd\x19\x66\x5b\x9d\x1a\xea\x4e\x06\xd3\x34\xb9\x25\x59\x21\x7c\x0b\xb8\x5e\xbb\xc8\xe2\x39\x2a\xf0\x4e\xff\xcd\x94\x97\x6f\x82\x87\x69\x01\xca\xad\x7c\xad\xf0\x48\x70\x15\xe6\x22\xda\x05\xbd\x18\xac\xe0\xd1\xdd\x15\x15\xa9\x2c\x57\xfc\x01\xfe\xfb\x02\xc3\xa4\x79\x72\x92\xe6\xfd\x11\x3e\xd2\x09\x83\xea\x2f\x25\x38\x01\xdb\xbe\x12\xc5\x16\x7a\x74\x01\x89\x9c\xbc\x8d\x6b\x03\x8f\x8e\x3a\xa6\x04\x4f\x8e\xff\xf6\xf7\x93\x4f\x87\x3f\x1f\x7d\x3e\x3a\x1e\x8b\x83\x4d\x20\x14\x8d\x22\xd5\x3e\xb0\xdc\x2c\xbb\x8e\x0c\x89\x0d\x39\xe1\x4d\x10\xbc\x1a\xbc\x79\x33\xdc\xdd\x79\xb5\x13\xbc\x79\x33\xa4\x35\x18\xcf\xec\x96\xe0\xf6\x09\xc9\x2e\xd2\x6c\x9e\xc3\xee\x0e\xcc\xd2\x74\x51\x05\xbe\xe4\xb8\x8b\x60\xbe\x52\x9d\x58\xaf\xe3\x3a\xd1\x2c\xd2\xbb\x76\xa7\x8a\xe8\x32\xce\x30\x89\x76\x84\xd1\xbe\xd7\x0e\x2d\x49\xcd\x99\xe5\x04\xc7\x72\x04\x27\xbd\x45\x7a\xa7\x4c\xe9\xc4\x36\xa7\x6f\x3c\xb0\x24\x63\x8a\x61\x04\xf3\xb0\xb8\x62\x60\xc1\x3c\x3d\xf7\x01\xf8\x54\x4d\xeb\x26\x36\x6b\x84\x7b\x6d\x60\x7c\xb0\xe4\x95\x2a\x02\x85\x29\x7b\xf1\x45\xd5\x46\x0d\x9c\xc2\x86\x03\x2d\x5b\x82\x93\x86\x70\x0c\xc3\x97\x1e\xb4\x44\x25\x2d\xe8\xc0\x8f\x3f\x6a\xcc\x23\x61\x96\xe0\x92\xf6\xd1\xc3\x03\xc4\x36\xa4\x03\x0c\xd0\x4e\xa0\xcf\x82\x85\x2d\x58\x28\x8b\x30\xcb\xc9\x2f\xb3\x34\x2c\x28\x19\x7a\x0a\x42\xdf\xe0\x36\xf3\xae\xa7\x0b\xdb\x85\xbf\x62\x0d\x0e\x17\x9d\xd4\x86\x2e\xdd\x60\xcc\xed\xd7\x8c\x77\xe5\xc1\x20\x74\xd5\x90\x70\x7a\xc5\xd4\xdd\x78\x1a\x63\xd6\x02\xb4\xb2\xa0\x60\xe4\x7a\x64\x7a\x01\x37\x3c\xba\x15\xbf\x16\x30\xc2\x53\x76\x83\x46\xbe\x2c\xc3\x82\xe4\x7a\x1d\x54\x64\x54\xd5\x58\xdc\xb4\x8c\x65\xdc\x85\xb3\xe1\x6b\x0f\x76\x06\xe7\x1e\xc3\x2b\x66\x51\x35\xa5\x85\x45\xaf\x41\x28\xce\xe9\x05\x0c\x5f\xc3\xe5\x32\xcc\x22\x41\x3b\x23\x45\x18\x27\x24\xea\x41\xfb\x37\x14\x0e\x5d\x18\xf4\x76\x79\x9c\xec\x25\x95\x52\x67\x6f\x3c\x18\x0e\xce\x95\x62\x3d\xd5\xb4\x75\xc3\x74\x64\x89\xc3\x03\x9c\xec\x53\x12\xcf\xda\xda\x9b\xbe\xac\x26\x0f\x51\xc7\xdd\x98\x72\xcc\x72\x34\x56\x43\x5c\xed\x3a\x62\xcc\xe2\xc2\x5d\x1e\xe0\xe8\x70\xcd\xf4\x1b\x57\x44\x4d\x05\x47\xee\xd4\x2f\xcb\xcf\xf0\x78\xb6\xae\x12\x96\xbe\x83\x1b\xe8\xc8\x0f\xf0\x6c\xad\x17\xb5\x9c\x3b\x35\x51\x11\xb3\x90\x77\x87\x1e\xfc\x2c\xae\x58\xb3\x8a\x90\x55\x4d\x13\x2b\xbe\x39\xd4\xfa\x4a\x67\x7f\x55\xc3\xbe\xed\xe4\x98\x70\x64\x49\xda\xb9\xc7\xe3\xc3\x1e\x55\xaf\x74\x94\x5c\xbe\x38\x6f\x2a\x67\xac\xb5\xa7\x4e\x12\x43\x0d\x83\x3d\xd9\x73\xfc\x7b\xdc\x7a\x4b\x3e\x94\x56\x5f\x49\xb9\x12\xbb\xf3\x89\x71\x09\x8e\xc6\xc8\x3c\x32\x53\x82\xea\x78\x6f\xca\xb6\x2b\x19\x18\x35\xab\xa2\x9e\x92\xbe\x6a\x8e\xe3\xca\x9e\x87\xdc\x29\xb1\xee\x2f\x36\x61\xe0\xfc\xb3\xf8\x73\x9e\x48\x60\x8a\xdb\x79\x67\xda\x4b\x6e\x76\x06\x95\xcb\x6d\x7f\x0f\xec\x2e\xbd\xcd\x55\xf0\x89\x04\x5f\xb9\xd9\xbd\xd3\xea\x9a\x92\x47\xae\xab\x60\xd9\x89\x73\xb8\xa3\x39\x71\x6a\xc2\xe3\x80\xfb\x72\xaa\x97\xc1\x3b\xff\x00\xe7\x14\x74\x3e\x9f\xa0\xab\x17\xfe\x3a\xc0\x40\x98\x09\xc4\xaa\xd7\x08\x77\xf7\x62\x17\x02\xf8\x61\x9c\xcb\x26\x7f\x9c\x63\x2c\x5c\x9c\x51\xf1\x9a\xe9\xe3\x4a\x33\x8c\xe5\x5e\x55\xa5\xdd\x30\x30\x1b\x96\xd2\x96\xab\x30\x97\x2e\x43\xca\x75\x29\x57\x80\x96\xad\x38\xaf\x6e\x40\xd2\x4c\x4d\xec\x7e\x7a\x4c\x0f\x83\x7f\x3f\x39\xfe\x8c\x1d\x12\x56\x87\x8b\x52\x0b\x94\xcb\x96\x72\x48\xae\x84\xd3\x18\x1f\xfe\xc5\x13\x33\x66\x2b\x19\x31\x3

9\xb7\x1c\x15\xf0\x68\xf0\xd3\x4e\xa3\xa5\x5c\x9e\x75\xb1\xef\x8c\xc5\x34\x31\xd6\x92\x5a\x40\x5b\x3f\x93\x9a\xe5\xc3\xc1\x3d\xe4\xb5\x33\x71\xb8\x50\x58\xcc\xfc\xce\x2b\x84\x9c\x61\x27\xd8\xb1\x96\x92\x1e\xb1\x45\xeb\x28\x79\x5b\x2c\x81\x7c\x04\x91\x31\x4a\xb3\xbf\x75\xef\xce\x5d\xba\x0d\xbb\x9e\x6d\x89\x5a\x2c\x00\x3c\x20\x03\x8f\xd0\x1d\x9b\x17\xc1\x9b\x60\xe5\xde\x51\xfc\xd4\xc6\x91\xf1\xd2\xe3\x70\x6c\xa9\x4b\x33\x38\xbb\xe8\x30\x1a\x1a\xd8\x47\x42\x75\x2f\xd7\x95\xc4\xc4\xe9\x34\x3e\xf1\x60\xe8\xc1\x4f\x3b\x1e\x0c\x77\x3d\x68\xd1\x79\xd2\x72\x9c\x15\xaa\xaa\x09\xbc\x1d\x49\xab\x81\x41\x0b\xbd\x1b\x49\x8b\xcc\x5a\x98\x0a\x43\xd5\x2f\x18\xe7\x0f\xb8\xee\xad\x01\xf6\x80\x3b\x18\x22\xb0\x77\xf5\xf9\xa3\x3a\x9d\xdb\x68\x3e\x84\x39\x69\x3b\xe8\x7a\x30\xa1\xb2\xda\x43\x29\x97\xdb\xb3\x84\xda\x7b\xb8\x44\xd2\x4a\x04\xe8\x4d\xed\x3c\x79\xa2\x6b\xec\xc7\x47\x1f\xf3\xd8\x7d\x6e\xaf\xba\xbb\x4a\x67\xea\x2d\xb3\x81\x44\xad\xc8\x87\xb2\xf4\x48\xfc\xd5\x24\xb9\xbc\xb8\x8b\x31\x63\x28\x3c\x05\xf7\x73\x60\x6c\x8e\x1b\xfa\x81\x9e\xfc\xd8\x3a\xf0\xd0\xa9\x38\x4a\x99\x70\x9c\x4e\xc9\xa2\x10\x5b\x8d\xc0\x45\x74\x35\x0a\xbb\xea\xf8\x82\x35\x29\xfd\x97\xcf\xc7\xe3\xc6\xf9\xfe\x6d\xf7\x3e\xea\xf0\xf5\xe1\xfd\x2c\x0e\x73\xc2\x2c\x38\x1f\xe2\x4b\xe1\x27\x38\x27\xc5\x55\x1a\x49\xb1\x09\xfd\xbe\xc8\x53\xc2\x12\x97\xec\x83\x44\x84\xbd\x62\x3b\x32\x89\x20\x9c\xa4\x92\x73\x00\x2d\x59\xe6\x7f\x1c\x89\x54\xd9\xfb\xa2\x64\xbe\x9c\xd4\x95\xe4\x47\xf0\x35\x6b\x3f\x3d\x09\x95\x25\xe7\xcb\x59\x5d\xc9\x28\xbe\x8d\x23\x82\xe5\xa2\xf8\x76\x5f\x79\x97\x91\x79\x18\x27\x11\x37\x0b\xcd\xd3\x48\x7d\xcd\x53\xc1\xf3\xfc\xf2\xd3\xf9\x42\x7d\xcd\xb2\x06\x89\xa4\x41\x72\x87\x72\xd8\xa4\xe4\x22\xbe\x3c\x9e\x5c\x4b\xf7\xbe\xba\xab\x6b\xbb\xfc\x48\x16\x8e\x7c\xf0\xca\x8f\x59\xb5\xdf\x78\x05\xb4\xd1\x9f\x8e\xfe\xf5\xfd\xe9\x21\xfc\xf5\xf0\xe3\xc9\xe1\x27\xf8\xe5\xb7\xf1\xcf\xa7\x47\xc7\xe3\xcf\xfc\x8b\x72\x5a\x94\x71\xb6\x8a\x85\x0b\x1d\xf7\xe8\x8e\xa6\x46\xd8\x08\x63\x0b\x5e\x5e\x3d\x3c\x20\x1a\xce\x08\x62\x38\x80\x18\xf6\x20\xae\xee\xd5\x24\x46\xca\xa5\x6e\x1a\x54\x98\x37\x0a\xdb\xa8\xd3\x8b\x52\xc7\xe3\x07\x0e\x95\x4f\x75\xb3\x08\x75\x66\x73\x0f\xee\xd5\x4d\x99\xb2\x3f\x50\x1f\x5d\xc3\x08\x42\x7e\xc8\x55\xdf\xd0\xd1\x0d\xa9\xd4\xeb\x42\xab\x25\x75\xb3\xec\x2c\x76\x6d\x46\x84\xe6\x58\x0c\xd3\x8b\xb2\x82\xf2\xcb\x7b\x18\x55\x06\x10\x1f\x72\x6b\x24\xb2\xa8\xe0\xde\xf7\xf7\x91\x5c\x2b\x40\x71\xab\x4b\xee\x0c\xba\x4a\xae\xae\x6f\xca\xd2\xac\xae\x5a\x54\x54\xf3\x9e\xd6\x90\x6b\x04\x2d\xe3\x7c\x40\x86\xc8\x29\x3f\xa7\x11\x79\x5f\xb4\x7d\xff\x9a\x21\x74\xed\xbc\xde\x97\x2b\x17\x92\x4f\xc2\xe0\xba\x46\xd7\xd0\x87\x87\x4a\xcc\x49\xc3\xfd\x33\x5b\x10\xaa\xec\x2e\xa7\x69\xae\x66\x85\x94\x46\x17\x4b\xb1\xec\x62\xa0\x0f\x6e\xe8\xc1\x44\x1d\x2f\xbb\x27\x87\xc8\x8e\x63\x4e\x04\x03\x1e\xf3\x9a\x21\x0a\x78\x86\x71\xcb\xc0\xcb\x9c\xb1\x68\xfd\x7d\xa5\xc3\x6d\x59\x8c\x84\x91\x86\x2a\x28\xd7\x3a\xee\x7d\xf9\x15\x4b\x0d\x52\x21\x23\x54\x3d\xcd\x32\x83\x54\xe0\x08\xd6\x1a\x55\x34\x4d\x9e\xe6\xe3\xe1\x01\xa4\x94\x49\x21\x1c\xc0\x84\x23\xc7\xfa\xd7\x74\x5d\xaa\x94\xec\xe9\x92\xb8\x19\xed\xd9\x08\x24\xde\x63\x8d\x71\xba\x0c\x02\x95\xe3\x1b\xaa\x98\xcd\xac\xbc\x9a\xe9\xe8\x9d\x79\x84\x04\xbf\xf4\xdd\xff\xc5\x16\x0c\x28\xf7\x03\x95\xae\x98\x5b\x96\xb4\xd8\x8c\xb2\xd4\x0b\x37\xf0\x0e\x66\x2e\x5a\x2c\x41\xf5\x8d\x25\x2f\xf5\x4c\x4b\x85\x7c\x00\x37\xb0\xa7\xb7\x4f\xf0\xe1\xca\x83\x5d\x85\xbc\x07\xa5\xe8\x88\x31\x59\xb5\x48\x07\x1c\x8b\xac\xd4\xf1\xb9\x9c\x8b\x9f\xfe\xf9\x8e\x3f\xde\xdc\x0d\x8c\x49\xa9\x13\xca\xa6\xd3\x21\xe1\x3d\x6a\xeb\x07\x90\x97\x2c\xd7\x47\xaa\x4b\xc2\x6c\x49\x18\xd8\x36\x7a\x67\xdc\x52\x45\x5b\xd8\xb1\xe2\x84\x9d\xbd\xbc\x0a\xae\x9a\xb9\xca\xf4\x04\x91\xdf\xe8\x13\x7a\x3c\x15\x90\xd3\x58\x9e\x19\xc5\x31\x4b\xae\xe6\x63\x53\x16\x64\x51\x59\x78\x0d\x40\xd5\xfa\xd6\x80\x74\x07\x2d\x7e\xe3\x4f\x35\x22\xe9\x82\x40\x7e\x27\x90\x30\xb8\x22\x54\x0a\x94\x38\x29\xfe\x95\xd5\x9c\x66\xe3\xf4\x30\xcb\xd2\x2c\xc7\x7b\x8f\x79\x4c\xff\x09\

x57\x8a\x90\xe1\x1d\xc7\x6e\x1b\x4a\x0c\xe8\x84\x9e\x22\xde\x8d\x68\x11\x06\xfd\xf6\x76\x44\x
4b\xaa\xfd\x57\xd5\x98\xbf\xa7\x9b\x5a\x3b\x9d\x5c\x5b\x48\x1f\x63\x30\x4b\x6f\x91\xa5\x45\x5a
\xac\x17\xa4\xd4\xb8\x7a\xd3\x70\x36\x63\x85\x46\x23\x68\x9d\xf1\xa8\x17\xa4\x75\xde\x72\x8d\
xd5\xcf\xec\x24\xa0\x6d\x9e\x47\x09\xf7\x36\x2a\x53\x87\x27\x5c\xea\xf2\x0f\x8e\x97\x45\xd9\xe
1\x87\x97\x3d\xe5\x40\xd1\x1a\xee\xee\xb6\xd8\xb9\x61\xf0\x93\x98\x93\x39\x9c\x0d\x76\x3d\x1
8\xec\x9e\xbb\xcb\x5d\x5c\xb4\x18\x70\xf7\x20\x90\x8a\x0d\x3d\xd8\xf5\xa0\x2a\xa7\x8d\x4f\x91\
x7e\x60\xfc\xf0\xa3\x0b\x63\xdf\x13\x5c\x1a\x5b\xc0\xb5\x2a\x94\x99\x5f\x80\xe1\x6f\x1f\x66\xd9\
x47\x53\xf8\x6b\x1e\xf4\xcc\xeb\x3d\x2f\x32\xd3\xdf\xdd\xf0\x0d\xd7\x37\x7c\x9e\xbb\x28\xcb\x3e
\x32\x37\xfe\x72\x2f\xa5\x8f\xe8\x16\x1e\x66\xd9\x19\xfd\xfd\x1c\x5e\x8c\xc4\x98\x68\x5b\x39\xfd
\x04\x25\x50\x00\x98\x92\xfb\xfd\xc7\x93\xbf\xbe\xff\x70\x78\x5a\x42\xc4\x22\x6f\x1c\xd4\x8c\xcb
\x0d\xd7\x9d\xc8\x35\xbc\x55\xd8\xb8\x76\xa5\xc4\x67\x3b\x45\x96\x9d\x5d\x53\x09\x23\x7a\xd9\
x07\xb7\xdb\x7b\x55\x82\x6e\xf7\xe7\x50\x99\x28\xe4\x87\x2e\x14\x0b\xe9\x9b\xee\x48\x54\xdc\
x2f\x2b\x76\xa2\x5f\xf0\x2f\x9f\x8f\xc4\xa7\x9b\x9d\xda\xaa\xdf\xf4\x45\x47\x7b\x46\x4b\xd8\x65\x
2c\x5b\xc7\x71\xde\x26\x1c\xaa\x29\x03\xef\x60\x00\x07\xf2\x38\x05\x1d\xaa\x01\xf6\xf8\x51\x59\
x8e\xc0\xa0\xa7\xe8\x0e\x74\x25\xa6\x05\x9e\xf1\x01\xb4\x98\xfb\x22\xe9\xb6\x64\x38\x60\x0b\x
8f\xb6\xa3\xbe\xb6\x42\x66\x24\xf1\xe0\x5e\xdd\x29\xc6\xba\x89\x50\xdb\x8a\x09\xb7\x08\x7d\xa
d\x4f\x35\x67\x83\x92\xb9\x47\x5d\xb5\xd7\xda\x47\x9c\x0d\xb8\xe7\xf8\xcf\xfa\x64\x67\x36\x84\x
7b\x61\x41\x90\x99\x3b\x71\x9a\x43\xad\xa6\x90\x9a\x65\xcb\x09\xbe\x5f\xb8\x58\xc6\xc6\x76\xb
b\x84\xee\x85\x24\xb1\x4e\x79\xb9\x55\x2d\xda\xbf\xfe\x88\x89\x00\xdf\xaf\x69\x9e\x68\x62\x77\
x44\xbb\x5e\x7e\x2a\x67\x39\x26\x4c\x9c\x58\xeb\x65\x1d\x54\xcd\x19\xaa\x60\x33\x94\x16\x7d\
x36\xe9\xe9\x02\xeb\x66\x7e\x69\x87\x31\xe7\x92\xb4\xbb\x55\xdc\xd0\xae\x55\xae\xe0\x2d\x67\
x3e\x36\x65\xcb\xcb\xde\x84\x4e\xee\xea\xf2\x32\x31\x8f\x02\x87\xff\x76\x72\xfc\xe9\x94\xff\x5d\
x59\x6e\x46\x10\x26\xa8\x3e\x94\x29\x0e\xe9\xd8\xfd\xfa\x67\x36\x64\x2c\xa4\x6c\xbd\x20\xe9\x
05\x44\xe4\x82\x9e\x69\xe8\xbe\x28\xf8\x6f\xd1\x9d\x98\x3d\xef\x85\xf3\x48\xe9\x52\xf6\xb8\xad\
x9a\x3c\xcc\x93\x32\x7c\x03\xa9\xe2\x71\x1a\xb1\x5c\x1e\x0c\xe6\x8c\x24\xb7\x71\x96\x26\x54\x
75\xc9\xb9\xb7\xe9\x72\xb1\x48\xb3\x82\xa5\x2a\x27\x3d\x3a\x59\x33\xa1\x79\xca\x83\xcc\x79\x
66\x9f\x51\xed\xae\xb5\x4c\x18\x47\x11\x32\xad\x96\x57\x18\xd7\x5e\x8d\xf4\x63\x3d\x54\xf9\xb
4\xb2\xf5\xa2\x48\xa1\x03\x45\xb6\x86\xaf\xc0\xff\x1c\x41\x46\xbe\x2c\xe3\x8c\xb4\x5b\xec\x49\x
ab\x43\x5b\x39\x0d\x8b\xe9\x15\xb4\x49\x07\xbe\x7e\x2b\xdb\xfb\x21\x4b\xef\x72\x61\x18\x33\x1
6\xda\xe5\x2c\x9d\x84\xb3\x9e\x3c\x58\x86\x89\xe1\x5b\xa7\x4c\xa6\xf3\xcd\xfb\xfa\x27\x56\xe3\
x9f\xf6\x76\x83\x6f\xe7\xde\x9f\xee\xc8\xe4\xe5\x9f\xf6\xce\xc4\x10\xb4\x39\x63\x1e\x6b\xa2\xc7\
x9b\xd8\xf9\xfa\x03\x15\x59\x7f\x23\x93\x97\x32\xf3\xbd\xfe\x2c\x9e\xf4\x29\x09\x4c\x2f\xd0\xef\x
43\x94\x26\x05\xa4\xb7\x24\xcb\xe2\x88\x70\xee\xa8\xb4\x8b\xc3\xc9\x8c\xfc\x40\xfb\x84\x77\xfb
\x5d\x9c\x44\xe9\x1d\xe6\x14\xd0\xfa\x5d\xf9\xa0\xc7\xaa\x54\xbf\x12\x43\xa1\x7c\x82\xcc\xed\xf
f\xf0\xed\x87\x1f\x8c\xd1\x61\x6f\xb0\xf1\x15\xc7\x7f\xda\x1b\x0e\xbf\x9d\x7f\xf3\xbe\x7e\xf3\xce\
x58\x2f\x9c\x77\x7e\xe8\xf7\xff\x07\xe4\xe9\x32\x9b\x92\x5f\xc3\xc5\x22\x4e\x2e\x7f\xfb\xf4\x71\x
44\x5f\xf6\xae\xf3\xde\x3c\x5c\xfc\xf0\xff\x02\x00\x00\xff\xff\xd2\x48\xd8\x13\xdc\x80\x07\x00")

func web3JsBytes() ([]byte, error) {
return bindataRead(
_web3Js,
"web3.js",

```
)
}

func web3Js() (*asset, error) {
bytes, err := web3JsBytes()
if err != nil {
return nil, err
}

info := bindataFileInfo{name: "web3.js", size: 0, mode: os.FileMode(0), modTime: time.Unix(0, 0)}
a := &asset{bytes: bytes, info: info}
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) {
cannonicalName := strings.Replace(name, "\\", "/", -1)
if f, ok := _bindata[cannonicalName]; 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)
}

// 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
}

// 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) {
cannonicalName := strings.Replace(name, "\\", "/", -1)
if f, ok := _bindata[cannonicalName]; 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)
}

// 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){
"bignumber.js": bignumberJs,
"web3.js":      web3Js,
}

// 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
// AssetDir("") will return []string{"data"}.
func AssetDir(name string) ([]string, error) {
```

```
node := _bintree
if len(name) != 0 {
cannonicalName := strings.Replace(name, "\\", "/", -1)
pathList := strings.Split(cannonicalName, "/")
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{
"bignumber.js": {bignumberJs, map[string]*bintree{}},
"web3.js":      {web3Js, 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
}
err = os.Chtimes(_filePath(dir, name), info.ModTime(), info.ModTime())
if err != nil {
return err
}
return nil
}

// 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 {
cannonicalName := strings.Replace(name, "\\", "/", -1)
return filepath.Join(append([]string{dir}, strings.Split(cannonicalName, "/")...)...)
}

71:F:\git\coin\ethereum\go-ethereum\internal\jsre\deps\deps.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package deps contains the console JavaScript dependencies Go embedded.
package deps
```

```
//go:generate go-bindata -nometadata -pkg deps -o bindata.go bignumber.js web3.js
//go:generate gofmt -w -s bindata.go

72:F:\git\coin\ethereum\go-ethereum\internal\jsre\jsre.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package jsre provides execution environment for JavaScript.
package jsre

import (
crand "crypto/rand"
"encoding/binary"
"fmt"
"io"
"io/ioutil"
"math/rand"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/internal/jsre/deps"
"github.com/robertkrimen/otto"
)

var (
BigNumber_JS = deps.MustAsset("bignumber.js")
Web3_JS      = deps.MustAsset("web3.js")
)

/*
JSRE is a generic JS runtime environment embedding the otto JS interpreter.
It provides some helper functions to
- load code from files
- run code snippets
- require libraries
- bind native go objects
*/
type JSRE struct {
assetPath     string
output        io.Writer
evalQueue     chan *evalReq
stopEventLoop chan bool
closed        chan struct{}
```

```
}

// jsTimer is a single timer instance with a callback function
type jsTimer struct {
timer    *time.Timer
duration time.Duration
interval bool
call     otto.FunctionCall
}

// evalReq is a serialized vm execution request processed by runEventLoop.
type evalReq struct {
fn   func(vm *otto.Otto)
done chan bool
}

// runtime must be stopped with Stop() after use and cannot be used after stopping
func New(assetPath string, output io.Writer) *JSRE {
re := &JSRE{
assetPath:     assetPath,
output:        output,
closed:        make(chan struct{}),
evalQueue:     make(chan *evalReq),
stopEventLoop: make(chan bool),
}
go re.runEventLoop()
re.Set("loadScript", re.loadScript)
re.Set("inspect", re.prettyPrintJS)
return re
}

// randomSource returns a pseudo random value generator.
func randomSource() *rand.Rand {
bytes := make([]byte, 8)
seed := time.Now().UnixNano()
if _, err := crand.Read(bytes); err == nil {
seed = int64(binary.LittleEndian.Uint64(bytes))
}

src := rand.NewSource(seed)
return rand.New(src)
}
```

```
// This function runs the main event loop from a goroutine that is started
// when JSRE is created. Use Stop() before exiting to properly stop it.
// The event loop processes vm access requests from the evalQueue in a
// serialized way and calls timer callback functions at the appropriate time.

// Exported functions always access the vm through the event queue. You can
// call the functions of the otto vm directly to circumvent the queue. These
// functions should be used if and only if running a routine that was already
// called from JS through an RPC call.
func (self *JSRE) runEventLoop() {
defer close(self.closed)

vm := otto.New()
r := randomSource()
vm.SetRandomSource(r.Float64)

registry := map[*jsTimer]*jsTimer{}
ready := make(chan *jsTimer)

newTimer := func(call otto.FunctionCall, interval bool) (*jsTimer, otto.Value) {
delay, _ := call.Argument(1).ToInteger()
if 0 >= delay {
delay = 1
}
timer := &jsTimer{
duration: time.Duration(delay) * time.Millisecond,
call:     call,
interval: interval,
}
registry[timer] = timer

timer.timer = time.AfterFunc(timer.duration, func() {
ready <- timer
})

value, err := call.Otto.ToValue(timer)
if err != nil {
panic(err)
}
return timer, value
}
```

```
setTimeout := func(call otto.FunctionCall) otto.Value {
_, value := newTimer(call, false)
return value
}

setInterval := func(call otto.FunctionCall) otto.Value {
_, value := newTimer(call, true)
return value
}

clearTimeout := func(call otto.FunctionCall) otto.Value {
timer, _ := call.Argument(0).Export()
if timer, ok := timer.(*jsTimer); ok {
timer.timer.Stop()
delete(registry, timer)
}
return otto.UndefinedValue()
}
vm.Set("_setTimeout", setTimeout)
vm.Set("_setInterval", setInterval)
vm.Run(`var setTimeout = function(args) {
if (arguments.length < 1) {
throw TypeError("Failed to execute 'setTimeout': 1 argument required, but only 0 present.");
}
return _setTimeout.apply(this, arguments);
}`)
vm.Run(`var setInterval = function(args) {
if (arguments.length < 1) {
throw TypeError("Failed to execute 'setInterval': 1 argument required, but only 0 present.");
}
return _setInterval.apply(this, arguments);
}`)
vm.Set("clearTimeout", clearTimeout)
vm.Set("clearInterval", clearTimeout)

var waitForCallbacks bool

loop:
for {
select {
case timer := <-ready:
```

```
// execute callback, remove/reschedule the timer
var arguments []interface{}
if len(timer.call.ArgumentList) > 2 {
tmp := timer.call.ArgumentList[2:]
arguments = make([]interface{}, 2+len(tmp))
for i, value := range tmp {
arguments[i+2] = value
}
} else {
arguments = make([]interface{}, 1)
}
arguments[0] = timer.call.ArgumentList[0]
_, err := vm.Call(`Function.call.call`, nil, arguments...)
if err != nil {
fmt.Println("js error:", err, arguments)
}

_, inreg := registry[timer] // when clearInterval is called from within the callback don't reset it
if timer.interval && inreg {
timer.timer.Reset(timer.duration)
} else {
delete(registry, timer)
if waitForCallbacks && (len(registry) == 0) {
break loop
}
}
case req := <-self.evalQueue:
// run the code, send the result back
req.fn(vm)
close(req.done)
if waitForCallbacks && (len(registry) == 0) {
break loop
}
case waitForCallbacks = <-self.stopEventLoop:
if !waitForCallbacks || (len(registry) == 0) {
break loop
}
}
}

for _, timer := range registry {
timer.timer.Stop()
```

```
delete(registry, timer)
}
}

// Do executes the given function on the JS event loop.
func (self *JSRE) Do(fn func(*otto.Otto)) {
done := make(chan bool)
req := &evalReq{fn, done}
self.evalQueue <- req
<-done
}

// stops the event loop before exit, optionally waits for all timers to expire
func (self *JSRE) Stop(waitForCallbacks bool) {
select {
case <-self.closed:
case self.stopEventLoop <- waitForCallbacks:
<-self.closed
}
}

// Exec(file) loads and runs the contents of a file
// if a relative path is given, the jsre's assetPath is used
func (self *JSRE) Exec(file string) error {
code, err := ioutil.ReadFile(common.AbsolutePath(self.assetPath, file))
if err != nil {
return err
}
var script *otto.Script
self.Do(func(vm *otto.Otto) {
script, err = vm.Compile(file, code)
if err != nil {
return
}
_, err = vm.Run(script)
})
return err
}

// Bind assigns value v to a variable in the JS environment
// This method is deprecated, use Set.
func (self *JSRE) Bind(name string, v interface{}) error {
```

```
return self.Set(name, v)
}

// Run runs a piece of JS code.
func (self *JSRE) Run(code string) (v otto.Value, err error) {
self.Do(func(vm *otto.Otto) { v, err = vm.Run(code) })
return v, err
}

// Get returns the value of a variable in the JS environment.
func (self *JSRE) Get(ns string) (v otto.Value, err error) {
self.Do(func(vm *otto.Otto) { v, err = vm.Get(ns) })
return v, err
}

// Set assigns value v to a variable in the JS environment.
func (self *JSRE) Set(ns string, v interface{}) (err error) {
self.Do(func(vm *otto.Otto) { err = vm.Set(ns, v) })
return err
}

// loadScript executes a JS script from inside the currently executing JS code.
func (self *JSRE) loadScript(call otto.FunctionCall) otto.Value {
file, err := call.Argument(0).ToString()
if err != nil {
// TODO: throw exception
return otto.FalseValue()
}
file = common.AbsolutePath(self.assetPath, file)
source, err := ioutil.ReadFile(file)
if err != nil {
// TODO: throw exception
return otto.FalseValue()
}
if _, err := compileAndRun(call.Otto, file, source); err != nil {
// TODO: throw exception
fmt.Println("err:", err)
return otto.FalseValue()
}
// TODO: return evaluation result
return otto.TrueValue()
}
```

```
// Evaluate executes code and pretty prints the result to the specified output
// stream.
func (self *JSRE) Evaluate(code string, w io.Writer) error {
var fail error

self.Do(func(vm *otto.Otto) {
val, err := vm.Run(code)
if err != nil {
prettyError(vm, err, w)
} else {
prettyPrint(vm, val, w)
}
fmt.Fprintln(w)
})
return fail
}

// Compile compiles and then runs a piece of JS code.
func (self *JSRE) Compile(filename string, src interface{}) (err error) {
self.Do(func(vm *otto.Otto) { _, err = compileAndRun(vm, filename, src) })
return err
}

func compileAndRun(vm *otto.Otto, filename string, src interface{}) (otto.Value, error) {
script, err := vm.Compile(filename, src)
if err != nil {
return otto.Value{}, err
}
return vm.Run(script)
}

73:F:\git\coin\ethereum\go-ethereum\internal\jsre\jsre_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package jsre

import (
"io/ioutil"
"os"
"path"
"testing"
```

```
"time"

"github.com/robertkrimen/otto"
)

type testNativeObjectBinding struct{}

type msg struct {
Msg string
}

func (no *testNativeObjectBinding) TestMethod(call otto.FunctionCall) otto.Value {
m, err := call.Argument(0).ToString()
if err != nil {
return otto.UndefinedValue()
}
v, _ := call.Otto.ToValue(&msg{m})
return v
}

func newWithTestJS(t *testing.T, testjs string) (*JSRE, string) {
dir, err := ioutil.TempDir("", "jsre-test")
if err != nil {
t.Fatal("cannot create temporary directory:", err)
}
if testjs != "" {
if err := ioutil.WriteFile(path.Join(dir, "test.js"), []byte(testjs), os.ModePerm); err != nil {
t.Fatal("cannot create test.js:", err)
}
}
return New(dir, os.Stdout), dir
}

func TestExec(t *testing.T) {
jsre, dir := newWithTestJS(t, `msg = "testMsg"`)
defer os.RemoveAll(dir)

err := jsre.Exec("test.js")
if err != nil {
t.Errorf("expected no error, got %v", err)
}
val, err := jsre.Run("msg")
```

```
if err != nil {
t.Errorf("expected no error, got %v", err)
}
if !val.IsString() {
t.Errorf("expected string value, got %v", val)
}
exp := "testMsg"
got, _ := val.ToString()
if exp != got {
t.Errorf("expected '%v', got '%v'", exp, got)
}
jsre.Stop(false)
}

func TestNatto(t *testing.T) {
jsre, dir := newWithTestJS(t, `setTimeout(function(){msg = "testMsg"}, 1);`)
defer os.RemoveAll(dir)

err := jsre.Exec("test.js")
if err != nil {
t.Errorf("expected no error, got %v", err)
}
time.Sleep(100 * time.Millisecond)
val, err := jsre.Run("msg")
if err != nil {
t.Errorf("expected no error, got %v", err)
}
if !val.IsString() {
t.Errorf("expected string value, got %v", val)
}
exp := "testMsg"
got, _ := val.ToString()
if exp != got {
t.Errorf("expected '%v', got '%v'", exp, got)
}
jsre.Stop(false)
}

func TestBind(t *testing.T) {
jsre := New("", os.Stdout)
defer jsre.Stop(false)
```

```
jsre.Bind("no", &testNativeObjectBinding{})

_, err := jsre.Run(`no.TestMethod("testMsg")`)
if err != nil {
t.Errorf("expected no error, got %v", err)
}
}

func TestLoadScript(t *testing.T) {
jsre, dir := newWithTestJS(t, `msg = "testMsg"`)
defer os.RemoveAll(dir)

_, err := jsre.Run(`loadScript("test.js")`)
if err != nil {
t.Errorf("expected no error, got %v", err)
}
val, err := jsre.Run("msg")
if err != nil {
t.Errorf("expected no error, got %v", err)
}
if !val.IsString() {
t.Errorf("expected string value, got %v", val)
}
exp := "testMsg"
got, _ := val.ToString()
if exp != got {
t.Errorf("expected '%v', got '%v'", exp, got)
}
jsre.Stop(false)
}

74:F:\git\coin\ethereum\go-ethereum\internal\jsre\pretty.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package jsre

import (
"fmt"
"io"
"sort"
"strconv"
"strings"
```

```
"github.com/fatih/color"
"github.com/robertkrimen/otto"
)

const (
maxPrettyPrintLevel = 3
indentString        = "  "
)

var (
FunctionColor = color.New(color.FgMagenta).SprintfFunc()
SpecialColor  = color.New(color.Bold).SprintfFunc()
NumberColor   = color.New(color.FgRed).SprintfFunc()
StringColor   = color.New(color.FgGreen).SprintfFunc()
ErrorColor    = color.New(color.FgHiRed).SprintfFunc()
)

// these fields are hidden when printing objects.
var boringKeys = map[string]bool{
"valueOf":             true,
"toString":            true,
"toLocaleString":      true,
"hasOwnProperty":      true,
"isPrototypeOf":       true,
"propertyIsEnumerable": true,
"constructor":         true,
}

// prettyPrint writes value to standard output.
func prettyPrint(vm *otto.Otto, value otto.Value, w io.Writer) {
ppctx{vm: vm, w: w}.printValue(value, 0, false)
}

// prettyError writes err to standard output.
func prettyError(vm *otto.Otto, err error, w io.Writer) {
failure := err.Error()
if ottoErr, ok := err.(*otto.Error); ok {
failure = ottoErr.String()
}
fmt.Fprint(w, ErrorColor("%s", failure))
}
```

```
// jsErrorString adds a backtrace to errors generated by otto.
func jsErrorString(err error) string {
if ottoErr, ok := err.(*otto.Error); ok {
return ottoErr.String()
}
return err.Error()
}

func (re *JSRE) prettyPrintJS(call otto.FunctionCall) otto.Value {
for _, v := range call.ArgumentList {
prettyPrint(call.Otto, v, re.output)
fmt.Fprintln(re.output)
}
return otto.UndefinedValue()
}

type ppctx struct {
vm *otto.Otto
w  io.Writer
}

func (ctx ppctx) indent(level int) string {
return strings.Repeat(indentString, level)
}

func (ctx ppctx) printValue(v otto.Value, level int, inArray bool) {
switch {
case v.IsObject():
ctx.printObject(v.Object(), level, inArray)
case v.IsNull():
fmt.Fprint(ctx.w, SpecialColor("null"))
case v.IsUndefined():
fmt.Fprint(ctx.w, SpecialColor("undefined"))
case v.IsString():
s, _ := v.ToString()
fmt.Fprint(ctx.w, StringColor("%q", s))
case v.IsBoolean():
b, _ := v.ToBoolean()
fmt.Fprint(ctx.w, SpecialColor("%t", b))
case v.IsNaN():
fmt.Fprint(ctx.w, NumberColor("NaN"))
```

```
case v.IsNumber():
s, _ := v.ToString()
fmt.Fprint(ctx.w, NumberColor("%s", s))
default:
fmt.Fprint(ctx.w, "<unprintable>")
}
}

func (ctx ppctx) printObject(obj *otto.Object, level int, inArray bool) {
switch obj.Class() {
case "Array", "GoArray":
lv, _ := obj.Get("length")
len, _ := lv.ToInteger()
if len == 0 {
fmt.Fprintf(ctx.w, "[]")
return
}
if level > maxPrettyPrintLevel {
fmt.Fprint(ctx.w, "[...]")
return
}
fmt.Fprint(ctx.w, "[")
for i := int64(0); i < len; i++ {
el, err := obj.Get(strconv.FormatInt(i, 10))
if err == nil {
ctx.printValue(el, level+1, true)
}
if i < len-1 {
fmt.Fprintf(ctx.w, ", ")
}
}
fmt.Fprint(ctx.w, "]")

case "Object":
// Print values from bignumber.js as regular numbers.
if ctx.isBigNumber(obj) {
fmt.Fprint(ctx.w, NumberColor("%s", toString(obj)))
return
}
// Otherwise, print all fields indented, but stop if we're too deep.
keys := ctx.fields(obj)
if len(keys) == 0 {
```

```
fmt.Fprint(ctx.w, "{}")
return
}
if level > maxPrettyPrintLevel {
fmt.Fprint(ctx.w, "{...}")
return
}
fmt.Fprintln(ctx.w, "{")
for i, k := range keys {
v, _ := obj.Get(k)
fmt.Fprintf(ctx.w, "%s%s: ", ctx.indent(level+1), k)
ctx.printValue(v, level+1, false)
if i < len(keys)-1 {
fmt.Fprintf(ctx.w, ",")
}
fmt.Fprintln(ctx.w)
}
if inArray {
level--
}
fmt.Fprintf(ctx.w, "%s}", ctx.indent(level))

case "Function":
// Use toString() to display the argument list if possible.
if robj, err := obj.Call("toString"); err != nil {
fmt.Fprint(ctx.w, FunctionColor("function()"))
} else {
desc := strings.Trim(strings.Split(robj.String(), "{")[0], " \t\n")
desc = strings.Replace(desc, " (", "(", 1)
fmt.Fprint(ctx.w, FunctionColor("%s", desc))
}

case "RegExp":
fmt.Fprint(ctx.w, StringColor("%s", toString(obj)))

default:
if v, _ := obj.Get("toString"); v.IsFunction() && level <= maxPrettyPrintLevel {
s, _ := obj.Call("toString")
fmt.Fprintf(ctx.w, "<%s %s>", obj.Class(), s.String())
} else {
fmt.Fprintf(ctx.w, "<%s>", obj.Class())
}
```

```
}
}

func (ctx ppctx) fields(obj *otto.Object) []string {
var (
vals, methods []string
seen          = make(map[string]bool)
)
add := func(k string) {
if seen[k] || boringKeys[k] || strings.HasPrefix(k, "_") {
return
}
seen[k] = true
if v, _ := obj.Get(k); v.IsFunction() {
methods = append(methods, k)
} else {
vals = append(vals, k)
}
}
iterOwnAndConstructorKeys(ctx.vm, obj, add)
sort.Strings(vals)
sort.Strings(methods)
return append(vals, methods...)
}

func iterOwnAndConstructorKeys(vm *otto.Otto, obj *otto.Object, f func(string)) {
seen := make(map[string]bool)
iterOwnKeys(vm, obj, func(prop string) {
seen[prop] = true
f(prop)
})
if cp := constructorPrototype(obj); cp != nil {
iterOwnKeys(vm, cp, func(prop string) {
if !seen[prop] {
f(prop)
}
})
}
}

func iterOwnKeys(vm *otto.Otto, obj *otto.Object, f func(string)) {
Object, _ := vm.Object("Object")
```

```
rv, _ := Object.Call("getOwnPropertyNames", obj.Value())
gv, _ := rv.Export()
switch gv := gv.(type) {
case []interface{}:
for _, v := range gv {
f(v.(string))
}
case []string:
for _, v := range gv {
f(v)
}
default:
panic(fmt.Errorf("Object.getOwnPropertyNames returned unexpected type %T", gv))
}
}

func (ctx ppctx) isBigNumber(v *otto.Object) bool {
// Handle numbers with custom constructor.
if v, _ := v.Get("constructor"); v.Object() != nil {
if strings.HasPrefix(toString(v.Object()), "function BigNumber") {
return true
}
}
// Handle default constructor.
BigNumber, _ := ctx.vm.Object("BigNumber.prototype")
if BigNumber == nil {
return false
}
bv, _ := BigNumber.Call("isPrototypeOf", v)
b, _ := bv.ToBoolean()
return b
}

func toString(obj *otto.Object) string {
s, _ := obj.Call("toString")
return s.String()
}

func constructorPrototype(obj *otto.Object) *otto.Object {
if v, _ := obj.Get("constructor"); v.Object() != nil {
if v, _ = v.Object().Get("prototype"); v.Object() != nil {
return v.Object()
```

```
}
}
return nil
}

75:F:\git\coin\ethereum\go-ethereum\internal\web3ext\web3ext.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// package web3ext contains geth specific web3.js extensions.
package web3ext

var Modules = map[string]string{
"admin":      Admin_JS,
"chequebook": Chequebook_JS,
"clique":     Clique_JS,
"debug":      Debug_JS,
"eth":        Eth_JS,
"miner":      Miner_JS,
"net":        Net_JS,
"personal":   Personal_JS,
"rpc":        RPC_JS,
"shh":        Shh_JS,
"swarmfs":    SWARMFS_JS,
"txpool":     TxPool_JS,
}

const Chequebook_JS = `
web3._extend({
  property: 'chequebook',
  methods:
  [
    new web3._extend.Method({
      name: 'deposit',
      call: 'chequebook_deposit',
      params: 1,
      inputFormatter: [null]
    }),
    new web3._extend.Property({
name: 'balance',
getter: 'chequebook_balance',
outputFormatter: web3._extend.utils.toDecimal
}),
```

```
    new web3._extend.Method({
      name: 'cash',
      call: 'chequebook_cash',
      params: 1,
      inputFormatter: [null]
    }),
    new web3._extend.Method({
      name: 'issue',
      call: 'chequebook_issue',
      params: 2,
      inputFormatter: [null, null]
    }),
  ]
});
`


const Clique_JS = `
web3._extend({
  property: 'clique',
  methods:
  [
new web3._extend.Method({
name: 'getSnapshot',
call: 'clique_getSnapshot',
params: 1,
      inputFormatter: [null]
}),
new web3._extend.Method({
name: 'getSnapshotAtHash',
call: 'clique_getSnapshotAtHash',
params: 1
}),
    new web3._extend.Method({
      name: 'getSigners',
      call: 'clique_getSigners',
      params: 1,
      inputFormatter: [null]
    }),
new web3._extend.Method({
name: 'getSignersAtHash',
call: 'clique_getSignersAtHash',
params: 1
```

```
}),
new web3._extend.Method({
name: 'propose',
call: 'clique_propose',
params: 2
}),
new web3._extend.Method({
name: 'discard',
call: 'clique_discard',
params: 1
})
  ],
properties:
[
new web3._extend.Property({
name: 'proposals',
getter: 'clique_proposals'
}),
]
});
`

const Admin_JS = `
web3._extend({
property: 'admin',
methods:
[
new web3._extend.Method({
name: 'addPeer',
call: 'admin_addPeer',
params: 1
}),
new web3._extend.Method({
name: 'removePeer',
call: 'admin_removePeer',
params: 1
}),
new web3._extend.Method({
name: 'exportChain',
call: 'admin_exportChain',
params: 1,
inputFormatter: [null]
```

```
}),
new web3._extend.Method({
name: 'importChain',
call: 'admin_importChain',
params: 1
}),
new web3._extend.Method({
name: 'sleepBlocks',
call: 'admin_sleepBlocks',
params: 2
}),
new web3._extend.Method({
name: 'startRPC',
call: 'admin_startRPC',
params: 4,
inputFormatter: [null, null, null, null]
}),
new web3._extend.Method({
name: 'stopRPC',
call: 'admin_stopRPC'
}),
new web3._extend.Method({
name: 'startWS',
call: 'admin_startWS',
params: 4,
inputFormatter: [null, null, null, null]
}),
new web3._extend.Method({
name: 'stopWS',
call: 'admin_stopWS'
})
],
properties:
[
new web3._extend.Property({
name: 'nodeInfo',
getter: 'admin_nodeInfo'
}),
new web3._extend.Property({
name: 'peers',
getter: 'admin_peers'
}),
```

```
new web3._extend.Property({
name: 'datadir',
getter: 'admin_datadir'
})
]
});
`

const Debug_JS = `
web3._extend({
property: 'debug',
methods:
[
new web3._extend.Method({
name: 'printBlock',
call: 'debug_printBlock',
params: 1
}),
new web3._extend.Method({
name: 'getBlockRlp',
call: 'debug_getBlockRlp',
params: 1
}),
new web3._extend.Method({
name: 'setHead',
call: 'debug_setHead',
params: 1
}),
new web3._extend.Method({
name: 'traceBlock',
call: 'debug_traceBlock',
params: 1
}),
new web3._extend.Method({
name: 'traceBlockByFile',
call: 'debug_traceBlockByFile',
params: 1
}),
new web3._extend.Method({
name: 'traceBlockByNumber',
call: 'debug_traceBlockByNumber',
params: 1
```

```
}),
new web3._extend.Method({
name: 'traceBlockByHash',
call: 'debug_traceBlockByHash',
params: 1
}),
new web3._extend.Method({
name: 'seedHash',
call: 'debug_seedHash',
params: 1
}),
new web3._extend.Method({
name: 'dumpBlock',
call: 'debug_dumpBlock',
params: 1
}),
new web3._extend.Method({
name: 'chaindbProperty',
call: 'debug_chaindbProperty',
params: 1,
outputFormatter: console.log
}),
new web3._extend.Method({
name: 'chaindbCompact',
call: 'debug_chaindbCompact',
}),
new web3._extend.Method({
name: 'metrics',
call: 'debug_metrics',
params: 1
}),
new web3._extend.Method({
name: 'verbosity',
call: 'debug_verbosity',
params: 1
}),
new web3._extend.Method({
name: 'vmodule',
call: 'debug_vmodule',
params: 1
}),
new web3._extend.Method({
```

```
name: 'backtraceAt',
call: 'debug_backtraceAt',
params: 1,
}),
new web3._extend.Method({
name: 'stacks',
call: 'debug_stacks',
params: 0,
outputFormatter: console.log
}),
new web3._extend.Method({
name: 'memStats',
call: 'debug_memStats',
params: 0,
}),
new web3._extend.Method({
name: 'gcStats',
call: 'debug_gcStats',
params: 0,
}),
new web3._extend.Method({
name: 'cpuProfile',
call: 'debug_cpuProfile',
params: 2
}),
new web3._extend.Method({
name: 'startCPUProfile',
call: 'debug_startCPUProfile',
params: 1
}),
new web3._extend.Method({
name: 'stopCPUProfile',
call: 'debug_stopCPUProfile',
params: 0
}),
new web3._extend.Method({
name: 'goTrace',
call: 'debug_goTrace',
params: 2
}),
new web3._extend.Method({
name: 'startGoTrace',
```

```
call: 'debug_startGoTrace',
params: 1
}),
new web3._extend.Method({
name: 'stopGoTrace',
call: 'debug_stopGoTrace',
params: 0
}),
new web3._extend.Method({
name: 'blockProfile',
call: 'debug_blockProfile',
params: 2
}),
new web3._extend.Method({
name: 'setBlockProfileRate',
call: 'debug_setBlockProfileRate',
params: 1
}),
new web3._extend.Method({
name: 'writeBlockProfile',
call: 'debug_writeBlockProfile',
params: 1
}),
new web3._extend.Method({
name: 'writeMemProfile',
call: 'debug_writeMemProfile',
params: 1
}),
new web3._extend.Method({
name: 'traceTransaction',
call: 'debug_traceTransaction',
params: 2,
inputFormatter: [null, null]
}),
new web3._extend.Method({
name: 'preimage',
call: 'debug_preimage',
params: 1,
inputFormatter: [null]
}),
new web3._extend.Method({
name: 'getBadBlocks',
```

```
call: 'debug_getBadBlocks',
params: 0,
}),
new web3._extend.Method({
name: 'storageRangeAt',
call: 'debug_storageRangeAt',
params: 5,
}),
],
properties: []
});
`


const Eth_JS = `
web3._extend({
property: 'eth',
methods:
[
new web3._extend.Method({
name: 'sign',
call: 'eth_sign',
params: 2,
inputFormatter: [web3._extend.formatters.inputAddressFormatter, null]
}),
new web3._extend.Method({
name: 'resend',
call: 'eth_resend',
params: 3,
inputFormatter: [web3._extend.formatters.inputTransactionFormatter,
web3._extend.utils.fromDecimal, web3._extend.utils.fromDecimal]
}),
new web3._extend.Method({
name: 'signTransaction',
call: 'eth_signTransaction',
params: 1,
inputFormatter: [web3._extend.formatters.inputTransactionFormatter]
}),
new web3._extend.Method({
name: 'submitTransaction',
call: 'eth_submitTransaction',
params: 1,
inputFormatter: [web3._extend.formatters.inputTransactionFormatter]
```

```
}),
new web3._extend.Method({
name: 'getRawTransaction',
call: 'eth_getRawTransactionByHash',
params: 1
}),
new web3._extend.Method({
name: 'getRawTransactionFromBlock',
call: function(args) {
return (web3._extend.utils.isString(args[0]) && args[0].indexOf('0x') === 0) ?
'eth_getRawTransactionByBlockHashAndIndex' :
'eth_getRawTransactionByBlockNumberAndIndex';
},
params: 2,
inputFormatter: [web3._extend.formatters.inputBlockNumberFormatter, web3._extend.utils.toHex]
})
],
properties:
[
new web3._extend.Property({
name: 'pendingTransactions',
getter: 'eth_pendingTransactions',
outputFormatter: function(txs) {
var formatted = [];
for (var i = 0; i < txs.length; i++) {
formatted.push(web3._extend.formatters.outputTransactionFormatter(txs[i]));
formatted[i].blockHash = null;
}
return formatted;
}
})
]
});
`

const Miner_JS = `
web3._extend({
property: 'miner',
methods:
[
new web3._extend.Method({
name: 'start',
```

```
call: 'miner_start',
params: 1,
inputFormatter: [null]
}),
new web3._extend.Method({
name: 'stop',
call: 'miner_stop'
}),
new web3._extend.Method({
name: 'setEtherbase',
call: 'miner_setEtherbase',
params: 1,
inputFormatter: [web3._extend.formatters.inputAddressFormatter]
}),
new web3._extend.Method({
name: 'setExtra',
call: 'miner_setExtra',
params: 1
}),
new web3._extend.Method({
name: 'setGasPrice',
call: 'miner_setGasPrice',
params: 1,
inputFormatter: [web3._extend.utils.fromDecimal]
}),
new web3._extend.Method({
name: 'getHashrate',
call: 'miner_getHashrate'
})
],
properties: []
});
`

const Net_JS = `
web3._extend({
property: 'net',
methods: [],
properties:
[
new web3._extend.Property({
name: 'version',
```

```
getter: 'net_version'
})
]
});
`

const Personal_JS = `
web3._extend({
property: 'personal',
methods:
[
new web3._extend.Method({
name: 'importRawKey',
call: 'personal_importRawKey',
params: 2
}),
new web3._extend.Method({
name: 'sign',
call: 'personal_sign',
params: 3,
inputFormatter: [null, web3._extend.formatters.inputAddressFormatter, null]
}),
new web3._extend.Method({
name: 'ecRecover',
call: 'personal_ecRecover',
params: 2
}),
new web3._extend.Method({
name: 'deriveAccount',
call: 'personal_deriveAccount',
params: 3
})
],
properties:
[
new web3._extend.Property({
name: 'listWallets',
getter: 'personal_listWallets'
})
]
})
`
```

```
const RPC_JS = `
web3._extend({
property: 'rpc',
methods: [],
properties:
[
new web3._extend.Property({
name: 'modules',
getter: 'rpc_modules'
})
]
});
`

const Shh_JS = `
web3._extend({
property: 'shh',
methods: [
new web3._extend.Method({
name: 'setMaxMessageLength',
call: 'shh_setMaxMessageLength',
params: 1
}),
new web3._extend.Method({
name: 'setMinimumPoW',
call: 'shh_setMinimumPoW',
params: 1
}),
new web3._extend.Method({
name: 'markTrustedPeer',
call: 'shh_markTrustedPeer',
params: 1
}),
new web3._extend.Method({
name: 'hasKeyPair',
call: 'shh_hasKeyPair',
params: 1
}),
new web3._extend.Method({
name: 'deleteKeyPair',
call: 'shh_deleteKeyPair',
```

```
params: 1
}),
new web3._extend.Method({
name: 'newKeyPair',
call: 'shh_newKeyPair'
}),
new web3._extend.Method({
name: 'getPublicKey',
call: 'shh_getPublicKey',
params: 1
}),
new web3._extend.Method({
name: 'getPrivateKey',
call: 'shh_getPrivateKey',
params: 1
}),
new web3._extend.Method({
name: 'newSymKey',
call: 'shh_newSymKey',
}),
new web3._extend.Method({
name: 'addSymKey',
call: 'shh_addSymKey',
params: 1
}),
new web3._extend.Method({
name: 'generateSymKeyFromPassword',
call: 'shh_generateSymKeyFromPassword',
params: 1
}),
new web3._extend.Method({
name: 'hasSymKey',
call: 'shh_hasSymKey',
params: 1
}),
new web3._extend.Method({
name: 'getSymKey',
call: 'shh_getSymKey',
params: 1
}),
new web3._extend.Method({
name: 'deleteSymKey',
```

```
call: 'shh_deleteSymKey',
params: 1
}),
new web3._extend.Method({
name: 'subscribe',
call: 'shh_subscribe',
params: 2
}),
new web3._extend.Method({
name: 'unsubscribe',
call: 'shh_unsubscribe',
params: 1
}),
new web3._extend.Method({
name: 'post',
call: 'shh_post',
params: 1
}),
new web3._extend.Method({
name: 'publicKey',
call: 'shh_getPublicKey',
params: 1
}),
new web3._extend.Method({
name: 'getFilterMessages',
call: 'shh_getFilterMessages',
params: 1
}),
new web3._extend.Method({
name: 'deleteMessageFilter',
call: 'shh_deleteMessageFilter',
params: 1
}),
new web3._extend.Method({
name: 'newMessageFilter',
call: 'shh_newMessageFilter',
params: 1
})
],
properties:
[
new web3._extend.Property({
```

```
name: 'version',
getter: 'shh_version',
outputFormatter: web3._extend.utils.toDecimal
}),
new web3._extend.Property({
name: 'info',
getter: 'shh_info'
}),
]
});
`

const SWARMFS_JS = `
web3._extend({
property: 'swarmfs',
methods:
[
new web3._extend.Method({
name: 'mount',
call: 'swarmfs_mount',
params: 2
}),
new web3._extend.Method({
name: 'unmount',
call: 'swarmfs_unmount',
params: 1
}),
new web3._extend.Method({
name: 'listmounts',
call: 'swarmfs_listmounts',
params: 0
})
]
});
`

const TxPool_JS = `
web3._extend({
property: 'txpool',
methods: [],
properties:
[
```

```
new web3._extend.Property({
name: 'content',
getter: 'txpool_content'
}),
new web3._extend.Property({
name: 'inspect',
getter: 'txpool_inspect'
}),
new web3._extend.Property({
name: 'status',
getter: 'txpool_status',
outputFormatter: function(status) {
status.pending = web3._extend.utils.toDecimal(status.pending);
status.queued = web3._extend.utils.toDecimal(status.queued);
return status;
}
})
]
});
`
```

76:F:\git\coin\ethereum\go-ethereum\les\api_backend.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package les

import (
"context"
"math/big"

"github.com/ethereum/go-ethereum/accounts"
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/math"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/state"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/eth/gasprice"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/light"
```

```
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rpc"
)

type LesApiBackend struct {
eth *LightEthereum
gpo *gasprice.Oracle
}

func (b *LesApiBackend) ChainConfig() *params.ChainConfig {
return b.eth.chainConfig
}

func (b *LesApiBackend) CurrentBlock() *types.Block {
return types.NewBlockWithHeader(b.eth.BlockChain().CurrentHeader())
}

func (b *LesApiBackend) SetHead(number uint64) {
b.eth.protocolManager.downloader.Cancel()
b.eth.blockchain.SetHead(number)
}

func (b *LesApiBackend) HeaderByNumber(ctx context.Context, blockNr rpc.BlockNumber)
(*types.Header, error) {
if blockNr == rpc.LatestBlockNumber || blockNr == rpc.PendingBlockNumber {
return b.eth.blockchain.CurrentHeader(), nil
}

return b.eth.blockchain.GetHeaderByNumberOdr(ctx, uint64(blockNr))
}

func (b *LesApiBackend) BlockByNumber(ctx context.Context, blockNr rpc.BlockNumber)
(*types.Block, error) {
header, err := b.HeaderByNumber(ctx, blockNr)
if header == nil || err != nil {
return nil, err
}
return b.GetBlock(ctx, header.Hash())
}

func (b *LesApiBackend) StateAndHeaderByNumber(ctx context.Context, blockNr
rpc.BlockNumber) (*state.StateDB, *types.Header, error) {
```

```
header, err := b.HeaderByNumber(ctx, blockNr)
if header == nil || err != nil {
return nil, nil, err
}
return light.NewState(ctx, header, b.eth.odr), header, nil
}

func (b *LesApiBackend) GetBlock(ctx context.Context, blockHash common.Hash) (*types.Block,
error) {
return b.eth.blockchain.GetBlockByHash(ctx, blockHash)
}

func (b *LesApiBackend) GetReceipts(ctx context.Context, blockHash common.Hash)
(types.Receipts, error) {
return light.GetBlockReceipts(ctx, b.eth.odr, blockHash, core.GetBlockNumber(b.eth.chainDb,
blockHash))
}

func (b *LesApiBackend) GetTd(blockHash common.Hash) *big.Int {
return b.eth.blockchain.GetTdByHash(blockHash)
}

func (b *LesApiBackend) GetEVM(ctx context.Context, msg core.Message, state *state.StateDB,
header *types.Header, vmCfg vm.Config) (*vm.EVM, func() error, error) {
state.SetBalance(msg.From(), math.MaxBig256)
context := core.NewEVMContext(msg, header, b.eth.blockchain, nil)
return vm.NewEVM(context, state, b.eth.chainConfig, vmCfg), state.Error, nil
}

func (b *LesApiBackend) SendTx(ctx context.Context, signedTx *types.Transaction) error {
return b.eth.txPool.Add(ctx, signedTx)
}

func (b *LesApiBackend) RemoveTx(txHash common.Hash) {
b.eth.txPool.RemoveTx(txHash)
}

func (b *LesApiBackend) GetPoolTransactions() (types.Transactions, error) {
return b.eth.txPool.GetTransactions()
}

func (b *LesApiBackend) GetPoolTransaction(txHash common.Hash) *types.Transaction {
```

```
return b.eth.txPool.GetTransaction(txHash)
}

func (b *LesApiBackend) GetPoolNonce(ctx context.Context, addr common.Address) (uint64,
error) {
return b.eth.txPool.GetNonce(ctx, addr)
}

func (b *LesApiBackend) Stats() (pending int, queued int) {
return b.eth.txPool.Stats(), 0
}

func (b *LesApiBackend) TxPoolContent() (map[common.Address]types.Transactions,
map[common.Address]types.Transactions) {
return b.eth.txPool.Content()
}

func (b *LesApiBackend) Downloader() *downloader.Downloader {
return b.eth.Downloader()
}

func (b *LesApiBackend) ProtocolVersion() int {
return b.eth.LesVersion() + 10000
}

func (b *LesApiBackend) SuggestPrice(ctx context.Context) (*big.Int, error) {
return b.gpo.SuggestPrice(ctx)
}

func (b *LesApiBackend) ChainDb() ethdb.Database {
return b.eth.chainDb
}

func (b *LesApiBackend) EventMux() *event.TypeMux {
return b.eth.eventMux
}

func (b *LesApiBackend) AccountManager() *accounts.Manager {
return b.eth.accountManager
}
```

77:F:\git\coin\ethereum\go-ethereum\les\backend.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package les implements the Light Ethereum Subprotocol.
package les

import (
"fmt"
"sync"
"time"

"github.com/ethereum/go-ethereum/accounts"
"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/hexutil"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/eth"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/eth/filters"
"github.com/ethereum/go-ethereum/eth/gasprice"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/internal/ethapi"
"github.com/ethereum/go-ethereum/light"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/node"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/p2p/discv5"
"github.com/ethereum/go-ethereum/params"
rpc "github.com/ethereum/go-ethereum/rpc"
)

type LightEthereum struct {
odr         *LesOdr
relay       *LesTxRelay
chainConfig *params.ChainConfig
// Channel for shutting down the service
shutdownChan chan bool
// Handlers
peers           *peerSet
txPool          *light.TxPool
blockchain      *light.LightChain
```

```
protocolManager *ProtocolManager
serverPool      *serverPool
reqDist         *requestDistributor
retriever       *retrieveManager
// DB interfaces
chainDb ethdb.Database // Block chain database

ApiBackend *LesApiBackend

eventMux       *event.TypeMux
engine         consensus.Engine
accountManager *accounts.Manager

networkId     uint64
netRPCService *ethapi.PublicNetAPI

quitSync chan struct{}
wg       sync.WaitGroup
}

func New(ctx *node.ServiceContext, config *eth.Config) (*LightEthereum, error) {
chainDb, err := eth.CreateDB(ctx, config, "lightchaindata")
if err != nil {
return nil, err
}
chainConfig, genesisHash, genesisErr := core.SetupGenesisBlock(chainDb, config.Genesis)
if _, isCompat := genesisErr.(*params.ConfigCompatError); genesisErr != nil && !isCompat {
return nil, genesisErr
}
log.Info("Initialised chain configuration", "config", chainConfig)

peers := newPeerSet()
quitSync := make(chan struct{})

eth := &LightEthereum{
chainConfig:    chainConfig,
chainDb:        chainDb,
eventMux:       ctx.EventMux,
peers:          peers,
reqDist:        newRequestDistributor(peers, quitSync),
accountManager: ctx.AccountManager,
engine:         eth.CreateConsensusEngine(ctx, config, chainConfig, chainDb),
```

```
shutdownChan:   make(chan bool),
networkId:      config.NetworkId,
}

eth.relay = NewLesTxRelay(peers, eth.reqDist)
eth.serverPool = newServerPool(chainDb, quitSync, &eth.wg)
eth.retriever = newRetrieveManager(peers, eth.reqDist, eth.serverPool)
eth.odr = NewLesOdr(chainDb, eth.retriever)
if eth.blockchain, err = light.NewLightChain(eth.odr, eth.chainConfig, eth.engine, eth.eventMux);
err != nil {
return nil, err
}
// Rewind the chain in case of an incompatible config upgrade.
if compat, ok := genesisErr.(*params.ConfigCompatError); ok {
log.Warn("Rewinding chain to upgrade configuration", "err", compat)
eth.blockchain.SetHead(compat.RewindTo)
core.WriteChainConfig(chainDb, genesisHash, chainConfig)
}

eth.txPool = light.NewTxPool(eth.chainConfig, eth.eventMux, eth.blockchain, eth.relay)
if eth.protocolManager, err = NewProtocolManager(eth.chainConfig, true, config.NetworkId,
eth.eventMux, eth.engine, eth.peers, eth.blockchain, nil, chainDb, eth.odr, eth.relay, quitSync,
&eth.wg); err != nil {
return nil, err
}
eth.ApiBackend = &LesApiBackend{eth, nil}
gpoParams := config.GPO
if gpoParams.Default == nil {
gpoParams.Default = config.GasPrice
}
eth.ApiBackend.gpo = gasprice.NewOracle(eth.ApiBackend, gpoParams)
return eth, nil
}

func lesTopic(genesisHash common.Hash) discv5.Topic {
return discv5.Topic("LES@" + common.Bytes2Hex(genesisHash.Bytes()[0:8]))
}

type LightDummyAPI struct{}

// Etherbase is the address that mining rewards will be send to
func (s *LightDummyAPI) Etherbase() (common.Address, error) {
```

```
return common.Address{}, fmt.Errorf("not supported")
}

// Coinbase is the address that mining rewards will be send to (alias for Etherbase)
func (s *LightDummyAPI) Coinbase() (common.Address, error) {
return common.Address{}, fmt.Errorf("not supported")
}

// Hashrate returns the POW hashrate
func (s *LightDummyAPI) Hashrate() hexutil.Uint {
return 0
}

// Mining returns an indication if this node is currently mining.
func (s *LightDummyAPI) Mining() bool {
return false
}

// APIs returns the collection of RPC services the ethereum package offers.
// NOTE, some of these services probably need to be moved to somewhere else.
func (s *LightEthereum) APIs() []rpc.API {
return append(ethapi.GetAPIs(s.ApiBackend), []rpc.API{
{
Namespace: "eth",
Version:   "1.0",
Service:   &LightDummyAPI{},
Public:    true,
}, {
Namespace: "eth",
Version:   "1.0",
Service:   downloader.NewPublicDownloaderAPI(s.protocolManager.downloader, s.eventMux),
Public:    true,
}, {
Namespace: "eth",
Version:   "1.0",
Service:   filters.NewPublicFilterAPI(s.ApiBackend, true),
Public:    true,
}, {
Namespace: "net",
Version:   "1.0",
Service:   s.netRPCService,
Public:    true,
```

```
},
}...)
}

func (s *LightEthereum) ResetWithGenesisBlock(gb *types.Block) {
s.blockchain.ResetWithGenesisBlock(gb)
}

func (s *LightEthereum) BlockChain() *light.LightChain      { return s.blockchain }
func (s *LightEthereum) TxPool() *light.TxPool              { return s.txPool }
func (s *LightEthereum) Engine() consensus.Engine           { return s.engine }
func (s *LightEthereum) LesVersion() int                    { return
int(s.protocolManager.SubProtocols[0].Version) }
func (s *LightEthereum) Downloader() *downloader.Downloader { return
s.protocolManager.downloader }
func (s *LightEthereum) EventMux() *event.TypeMux           { return s.eventMux }

// Protocols implements node.Service, returning all the currently configured
// network protocols to start.
func (s *LightEthereum) Protocols() []p2p.Protocol {
return s.protocolManager.SubProtocols
}

// Start implements node.Service, starting all internal goroutines needed by the
// Ethereum protocol implementation.
func (s *LightEthereum) Start(srvr *p2p.Server) error {
log.Warn("Light client mode is an experimental feature")
s.netRPCService = ethapi.NewPublicNetAPI(srvr, s.networkId)
s.serverPool.start(srvr, lesTopic(s.blockchain.Genesis().Hash()))
s.protocolManager.Start()
return nil
}

// Stop implements node.Service, terminating all internal goroutines used by the
// Ethereum protocol.
func (s *LightEthereum) Stop() error {
s.odr.Stop()
s.blockchain.Stop()
s.protocolManager.Stop()
s.txPool.Stop()

s.eventMux.Stop()
```

```
time.Sleep(time.Millisecond * 200)
s.chainDb.Close()
close(s.shutdownChan)

return nil
}

78:F:\git\coin\ethereum\go-ethereum\les\distributor.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package light implements on-demand retrieval capable state and chain objects
// for the Ethereum Light Client.
package les

import (
"container/list"
"errors"
"sync"
"time"
)

// ErrNoPeers is returned if no peers capable of serving a queued request are available
var ErrNoPeers = errors.New("no suitable peers available")

// requestDistributor implements a mechanism that distributes requests to
// suitable peers, obeying flow control rules and prioritizing them in creation
// order (even when a resend is necessary).
type requestDistributor struct {
reqQueue         *list.List
lastReqOrder     uint64
peers            map[distPeer]struct{}
peerLock         sync.RWMutex
stopChn, loopChn chan struct{}
loopNextSent     bool
lock             sync.Mutex
}

// distPeer is an LES server peer interface for the request distributor.
// waitBefore returns either the necessary waiting time before sending a request
// with the given upper estimated cost or the estimated remaining relative buffer
// value after sending such a request (in which case the request can be sent
```

```
// immediately). At least one of these values is always zero.
type distPeer interface {
waitBefore(uint64) (time.Duration, float64)
canQueue() bool
queueSend(f func())
}

// distReq is the request abstraction used by the distributor. It is based on
// three callback functions:
// - getCost returns the upper estimate of the cost of sending the request to a given peer
// - canSend tells if the server peer is suitable to serve the request
// - request prepares sending the request to the given peer and returns a function that
// does the actual sending. Request order should be preserved but the callback itself should not
// block until it is sent because other peers might still be able to receive requests while
// one of them is blocking. Instead, the returned function is put in the peer's send queue.
type distReq struct {
getCost func(distPeer) uint64
canSend func(distPeer) bool
request func(distPeer) func()

reqOrder uint64
sentChn  chan distPeer
element  *list.Element
}

// newRequestDistributor creates a new request distributor
func newRequestDistributor(peers *peerSet, stopChn chan struct{}) *requestDistributor {
d := &requestDistributor{
reqQueue: list.New(),
loopChn:  make(chan struct{}, 2),
stopChn:  stopChn,
peers:    make(map[distPeer]struct{}),
}
if peers != nil {
peers.notify(d)
}
go d.loop()
return d
}

// registerPeer implements peerSetNotify
func (d *requestDistributor) registerPeer(p *peer) {
```

```
d.peerLock.Lock()
d.peers[p] = struct{}{}
d.peerLock.Unlock()
}

// unregisterPeer implements peerSetNotify
func (d *requestDistributor) unregisterPeer(p *peer) {
d.peerLock.Lock()
delete(d.peers, p)
d.peerLock.Unlock()
}

// registerTestPeer adds a new test peer
func (d *requestDistributor) registerTestPeer(p distPeer) {
d.peerLock.Lock()
d.peers[p] = struct{}{}
d.peerLock.Unlock()
}

// distMaxWait is the maximum waiting time after which further necessary waiting
// times are recalculated based on new feedback from the servers
const distMaxWait = time.Millisecond * 10

// main event loop
func (d *requestDistributor) loop() {
for {
select {
case <-d.stopChn:
d.lock.Lock()
elem := d.reqQueue.Front()
for elem != nil {
close(elem.Value.(*distReq).sentChn)
elem = elem.Next()
}
d.lock.Unlock()
return
case <-d.loopChn:
d.lock.Lock()
d.loopNextSent = false
loop:
for {
peer, req, wait := d.nextRequest()
```

```
if req != nil && wait == 0 {
chn := req.sentChn // save sentChn because remove sets it to nil
d.remove(req)
send := req.request(peer)
if send != nil {
peer.queueSend(send)
}
chn <- peer
close(chn)
} else {
if wait == 0 {
// no request to send and nothing to wait for; the next
// queued request will wake up the loop
break loop
}
d.loopNextSent = true // a "next" signal has been sent, do not send another one until this one has been received
if wait > distMaxWait {
// waiting times may be reduced by incoming request replies, if it is too long, recalculate it periodically
wait = distMaxWait
}
go func() {
time.Sleep(wait)
d.loopChn <- struct{}{}
}()
break loop
}
}
d.lock.Unlock()
}
}
}

// selectPeerItem represents a peer to be selected for a request by weightedRandomSelect
type selectPeerItem struct {
peer   distPeer
req    *distReq
weight int64
}

// Weight implements wrsItem interface
```

```
func (sp selectPeerItem) Weight() int64 {
return sp.weight
}

// nextRequest returns the next possible request from any peer, along with the
// associated peer and necessary waiting time
func (d *requestDistributor) nextRequest() (distPeer, *distReq, time.Duration) {
checkedPeers := make(map[distPeer]struct{})
elem := d.reqQueue.Front()
var (
bestPeer distPeer
bestReq  *distReq
bestWait time.Duration
sel      *weightedRandomSelect
)

d.peerLock.RLock()
defer d.peerLock.RUnlock()

for (len(d.peers) > 0 || elem == d.reqQueue.Front()) && elem != nil {
req := elem.Value.(*distReq)
canSend := false
for peer, _ := range d.peers {
if _, ok := checkedPeers[peer]; !ok && peer.canQueue() && req.canSend(peer) {
canSend = true
cost := req.getCost(peer)
wait, bufRemain := peer.waitBefore(cost)
if wait == 0 {
if sel == nil {
sel = newWeightedRandomSelect()
}
sel.update(selectPeerItem{peer: peer, req: req, weight: int64(bufRemain*1000000) + 1})
} else {
if bestReq == nil || wait < bestWait {
bestPeer = peer
bestReq = req
bestWait = wait
}
}
checkedPeers[peer] = struct{}{}
}
}
```

```
next := elem.Next()
if !canSend && elem == d.reqQueue.Front() {
close(req.sentChn)
d.remove(req)
}
elem = next
}

if sel != nil {
c := sel.choose().(selectPeerItem)
return c.peer, c.req, 0
}
return bestPeer, bestReq, bestWait
}

// queue adds a request to the distribution queue, returns a channel where the
// receiving peer is sent once the request has been sent (request callback returned).
// If the request is cancelled or timed out without suitable peers, the channel is
// closed without sending any peer references to it.
func (d *requestDistributor) queue(r *distReq) chan distPeer {
d.lock.Lock()
defer d.lock.Unlock()

if r.reqOrder == 0 {
d.lastReqOrder++
r.reqOrder = d.lastReqOrder
}

back := d.reqQueue.Back()
if back == nil || r.reqOrder > back.Value.(*distReq).reqOrder {
r.element = d.reqQueue.PushBack(r)
} else {
before := d.reqQueue.Front()
for before.Value.(*distReq).reqOrder < r.reqOrder {
before = before.Next()
}
r.element = d.reqQueue.InsertBefore(r, before)
}

if !d.loopNextSent {
d.loopNextSent = true
d.loopChn <- struct{}{}
```

```
}

r.sentChn = make(chan distPeer, 1)
return r.sentChn
}

// cancel removes a request from the queue if it has not been sent yet (returns
// false if it has been sent already). It is guaranteed that the callback functions
// will not be called after cancel returns.
func (d *requestDistributor) cancel(r *distReq) bool {
d.lock.Lock()
defer d.lock.Unlock()

if r.sentChn == nil {
return false
}

close(r.sentChn)
d.remove(r)
return true
}

// remove removes a request from the queue
func (d *requestDistributor) remove(r *distReq) {
r.sentChn = nil
if r.element != nil {
d.reqQueue.Remove(r.element)
r.element = nil
}
}

79:F:\git\coin\ethereum\go-ethereum\les\distributor_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package light implements on-demand retrieval capable state and chain objects
// for the Ethereum Light Client.
package les

import (
"math/rand"
"sync"
"testing"
```

```
"time"
)

type testDistReq struct {
cost, procTime, order uint64
canSendTo             map[*testDistPeer]struct{}
}

func (r *testDistReq) getCost(dp distPeer) uint64 {
return r.cost
}

func (r *testDistReq) canSend(dp distPeer) bool {
_, ok := r.canSendTo[dp.(*testDistPeer)]
return ok
}

func (r *testDistReq) request(dp distPeer) func() {
return func() { dp.(*testDistPeer).send(r) }
}

type testDistPeer struct {
sent    []*testDistReq
sumCost uint64
lock    sync.RWMutex
}

func (p *testDistPeer) send(r *testDistReq) {
p.lock.Lock()
defer p.lock.Unlock()

p.sent = append(p.sent, r)
p.sumCost += r.cost
}

func (p *testDistPeer) worker(t *testing.T, checkOrder bool, stop chan struct{}) {
var last uint64
for {
wait := time.Millisecond
p.lock.Lock()
if len(p.sent) > 0 {
rq := p.sent[0]
```

```
wait = time.Duration(rq.procTime)
p.sumCost -= rq.cost
if checkOrder {
if rq.order <= last {
t.Errorf("Requests processed in wrong order")
}
last = rq.order
}
p.sent = p.sent[1:]
}
p.lock.Unlock()
select {
case <-stop:
return
case <-time.After(wait):
}
}
}

const (
testDistBufLimit       = 10000000
testDistMaxCost        = 1000000
testDistPeerCount      = 5
testDistReqCount       = 50000
testDistMaxResendCount = 3
)

func (p *testDistPeer) waitBefore(cost uint64) (time.Duration, float64) {
p.lock.RLock()
sumCost := p.sumCost + cost
p.lock.RUnlock()
if sumCost < testDistBufLimit {
return 0, float64(testDistBufLimit-sumCost) / float64(testDistBufLimit)
} else {
return time.Duration(sumCost - testDistBufLimit), 0
}
}

func (p *testDistPeer) canQueue() bool {
return true
}
```

```
func (p *testDistPeer) queueSend(f func()) {
f()
}

func TestRequestDistributor(t *testing.T) {
testRequestDistributor(t, false)
}

func TestRequestDistributorResend(t *testing.T) {
testRequestDistributor(t, true)
}

func testRequestDistributor(t *testing.T, resend bool) {
stop := make(chan struct{})
defer close(stop)

dist := newRequestDistributor(nil, stop)
var peers [testDistPeerCount]*testDistPeer
for i, _ := range peers {
peers[i] = &testDistPeer{}
go peers[i].worker(t, !resend, stop)
dist.registerTestPeer(peers[i])
}

var wg sync.WaitGroup

for i := 1; i <= testDistReqCount; i++ {
cost := uint64(rand.Int63n(testDistMaxCost))
procTime := uint64(rand.Int63n(int64(cost + 1)))
rq := &testDistReq{
cost:      cost,
procTime:  procTime,
order:     uint64(i),
canSendTo: make(map[*testDistPeer]struct{}),
}
for _, peer := range peers {
if rand.Intn(2) != 0 {
rq.canSendTo[peer] = struct{}{}
}
}

wg.Add(1)
```

```
req := &distReq{
getCost: rq.getCost,
canSend: rq.canSend,
request: rq.request,
}
chn := dist.queue(req)
go func() {
cnt := 1
if resend && len(rq.canSendTo) != 0 {
cnt = rand.Intn(testDistMaxResendCount) + 1
}
for i := 0; i < cnt; i++ {
if i != 0 {
chn = dist.queue(req)
}
p := <-chn
if p == nil {
if len(rq.canSendTo) != 0 {
t.Errorf("Request that could have been sent was dropped")
}
} else {
peer := p.(*testDistPeer)
if _, ok := rq.canSendTo[peer]; !ok {
t.Errorf("Request sent to wrong peer")
}
}
}
wg.Done()
}()
if rand.Intn(1000) == 0 {
time.Sleep(time.Duration(rand.Intn(5000000)))
}
}

wg.Wait()
}
```

80:F:\git\coin\ethereum\go-ethereum\les\execqueue.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package les
```

```
import "sync"

// execQueue implements a queue that executes function calls in a single thread,
// in the same order as they have been queued.
type execQueue struct {
mu        sync.Mutex
cond      *sync.Cond
funcs     []func()
closeWait chan struct{}
}

// newExecQueue creates a new execution queue.
func newExecQueue(capacity int) *execQueue {
q := &execQueue{funcs: make([]func(), 0, capacity)}
q.cond = sync.NewCond(&q.mu)
go q.loop()
return q
}

func (q *execQueue) loop() {
for f := q.waitNext(false); f != nil; f = q.waitNext(true) {
f()
}
close(q.closeWait)
}

func (q *execQueue) waitNext(drop bool) (f func()) {
q.mu.Lock()
if drop {
// Remove the function that just executed. We do this here instead of when
// dequeuing so len(q.funcs) includes the function that is running.
q.funcs = append(q.funcs[:0], q.funcs[1:]...)
}
for !q.isClosed() {
if len(q.funcs) > 0 {
f = q.funcs[0]
break
}
q.cond.Wait()
}
q.mu.Unlock()
return f
```

```
}

func (q *execQueue) isClosed() bool {
return q.closeWait != nil
}

// canQueue returns true if more function calls can be added to the execution queue.
func (q *execQueue) canQueue() bool {
q.mu.Lock()
ok := !q.isClosed() && len(q.funcs) < cap(q.funcs)
q.mu.Unlock()
return ok
}

// queue adds a function call to the execution queue. Returns true if successful.
func (q *execQueue) queue(f func()) bool {
q.mu.Lock()
ok := !q.isClosed() && len(q.funcs) < cap(q.funcs)
if ok {
q.funcs = append(q.funcs, f)
q.cond.Signal()
}
q.mu.Unlock()
return ok
}

// quit stops the exec queue.
// quit waits for the current execution to finish before returning.
func (q *execQueue) quit() {
q.mu.Lock()
if !q.isClosed() {
q.closeWait = make(chan struct{})
q.cond.Signal()
}
q.mu.Unlock()
<-q.closeWait
}

81:F:\git\coin\ethereum\go-ethereum\les\execqueue_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package les
```

```
import (
"testing"
)

func TestExecQueue(t *testing.T) {
var (
N        = 10000
q        = newExecQueue(N)
counter  int
execd    = make(chan int)
testexit = make(chan struct{})
)
defer q.quit()
defer close(testexit)

check := func(state string, wantOK bool) {
c := counter
counter++
qf := func() {
select {
case execd <- c:
case <-testexit:
}
}
if q.canQueue() != wantOK {
t.Fatalf("canQueue() == %t for %s", !wantOK, state)
}
if q.queue(qf) != wantOK {
t.Fatalf("canQueue() == %t for %s", !wantOK, state)
}
}

for i := 0; i < N; i++ {
check("queue below cap", true)
}
check("full queue", false)
for i := 0; i < N; i++ {
if c := <-execd; c != i {
t.Fatal("execution out of order")
}
}
```

```
q.quit()
check("closed queue", false)
}

82:F:\git\coin\ethereum\go-ethereum\les\fetcher.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package les implements the Light Ethereum Subprotocol.
package les

import (
"math/big"
"sync"
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/common/mclock"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/light"
"github.com/ethereum/go-ethereum/log"
)

const (
blockDelayTimeout = time.Second * 10 // timeout for a peer to announce a head that has already
been confirmed by others
maxNodeCount      = 20               // maximum number of fetcherTreeNode entries remembered for
each peer
)

// lightFetcher
type lightFetcher struct {
pm    *ProtocolManager
odr   *LesOdr
chain *light.LightChain

maxConfirmedTd  *big.Int
peers           map[*peer]*fetcherPeerInfo
lastUpdateStats *updateStatsEntry

lock       sync.Mutex // qwerqwerqwe
```

```
deliverChn chan fetchResponse
reqMu      sync.RWMutex
requested  map[uint64]fetchRequest
timeoutChn chan uint64
requestChn chan bool // true if initiated from outside
syncing    bool
syncDone   chan *peer
}

// fetcherPeerInfo holds fetcher-specific information about each active peer
type fetcherPeerInfo struct {
root, lastAnnounced *fetcherTreeNode
nodeCnt             int
confirmedTd         *big.Int
bestConfirmed       *fetcherTreeNode
nodeByHash          map[common.Hash]*fetcherTreeNode
firstUpdateStats    *updateStatsEntry
}

// fetcherTreeNode is a node of a tree that holds information about blocks recently
// announced and confirmed by a certain peer. Each new announce message from a peer
// adds nodes to the tree, based on the previous announced head and the reorg depth.
// There are three possible states for a tree node:
// - announced: not downloaded (known) yet, but we know its head, number and td
// - intermediate: not known, hash and td are empty, they are filled out when it becomes known
// - known: both announced by this peer and downloaded (from any peer).
// This structure makes it possible to always know which peer has a certain block,
// which is necessary for selecting a suitable peer for ODR requests and also for
// canonizing new heads. It also helps to always download the minimum necessary
// amount of headers with a single request.
type fetcherTreeNode struct {
hash             common.Hash
number           uint64
td               *big.Int
known, requested bool
parent           *fetcherTreeNode
children         []*fetcherTreeNode
}

// fetchRequest represents a header download request
type fetchRequest struct {
hash    common.Hash
```

```
amount  uint64
peer    *peer
sent    mclock.AbsTime
timeout bool
}

// fetchResponse represents a header download response
type fetchResponse struct {
reqID   uint64
headers []*types.Header
peer    *peer
}

// newLightFetcher creates a new light fetcher
func newLightFetcher(pm *ProtocolManager) *lightFetcher {
f := &lightFetcher{
pm:             pm,
chain:          pm.blockchain.(*light.LightChain),
odr:            pm.odr,
peers:          make(map[*peer]*fetcherPeerInfo),
deliverChn:     make(chan fetchResponse, 100),
requested:      make(map[uint64]fetchRequest),
timeoutChn:     make(chan uint64),
requestChn:     make(chan bool, 100),
syncDone:       make(chan *peer),
maxConfirmedTd: big.NewInt(0),
}
pm.peers.notify(f)
go f.syncLoop()
return f
}

// syncLoop is the main event loop of the light fetcher
func (f *lightFetcher) syncLoop() {
f.pm.wg.Add(1)
defer f.pm.wg.Done()

requesting := false
for {
select {
case <-f.pm.quitSync:
return
```

```
// when a new announce is received, request loop keeps running until
// no further requests are necessary or possible
case newAnnounce := <-f.requestChn:
f.lock.Lock()
s := requesting
requesting = false
var (
rq    *distReq
reqID uint64
)
if !f.syncing && !(newAnnounce && s) {
rq, reqID = f.nextRequest()
}
syncing := f.syncing
f.lock.Unlock()

if rq != nil {
requesting = true
_, ok := <-f.pm.reqDist.queue(rq)
if !ok {
f.requestChn <- false
}

if !syncing {
go func() {
time.Sleep(softRequestTimeout)
f.reqMu.Lock()
req, ok := f.requested[reqID]
if ok {
req.timeout = true
f.requested[reqID] = req
}
f.reqMu.Unlock()
// keep starting new requests while possible
f.requestChn <- false
}()
}
}
case reqID := <-f.timeoutChn:
f.reqMu.Lock()
req, ok := f.requested[reqID]
if ok {
```

```
delete(f.requested, reqID)
}
f.reqMu.Unlock()
if ok {
f.pm.serverPool.adjustResponseTime(req.peer.poolEntry, time.Duration(mclock.Now()-req.sent),
true)
req.peer.Log().Debug("Fetching data timed out hard")
go f.pm.removePeer(req.peer.id)
}
case resp := <-f.deliverChn:
f.reqMu.Lock()
req, ok := f.requested[resp.reqID]
if ok && req.peer != resp.peer {
ok = false
}
if ok {
delete(f.requested, resp.reqID)
}
f.reqMu.Unlock()
if ok {
f.pm.serverPool.adjustResponseTime(req.peer.poolEntry, time.Duration(mclock.Now()-req.sent),
req.timeout)
}
f.lock.Lock()
if !ok || !(f.syncing || f.processResponse(req, resp)) {
resp.peer.Log().Debug("Failed processing response")
go f.pm.removePeer(resp.peer.id)
}
f.lock.Unlock()
case p := <-f.syncDone:
f.lock.Lock()
p.Log().Debug("Done synchronising with peer")
f.checkSyncedHeaders(p)
f.syncing = false
f.lock.Unlock()
}
}
}

// registerPeer adds a new peer to the fetcher's peer set
func (f *lightFetcher) registerPeer(p *peer) {
p.lock.Lock()
```

```
p.hasBlock = func(hash common.Hash, number uint64) bool {
return f.peerHasBlock(p, hash, number)
}
p.lock.Unlock()

f.lock.Lock()
defer f.lock.Unlock()

f.peers[p] = &fetcherPeerInfo{nodeByHash: make(map[common.Hash]*fetcherTreeNode)}
}

// unregisterPeer removes a new peer from the fetcher's peer set
func (f *lightFetcher) unregisterPeer(p *peer) {
p.lock.Lock()
p.hasBlock = nil
p.lock.Unlock()

f.lock.Lock()
defer f.lock.Unlock()

// check for potential timed out block delay statistics
f.checkUpdateStats(p, nil)
delete(f.peers, p)
}

// announce processes a new announcement message received from a peer, adding new
// nodes to the peer's block tree and removing old nodes if necessary
func (f *lightFetcher) announce(p *peer, head *announceData) {
f.lock.Lock()
defer f.lock.Unlock()
p.Log().Debug("Received new announcement", "number", head.Number, "hash", head.Hash,
"reorg", head.ReorgDepth)

fp := f.peers[p]
if fp == nil {
p.Log().Debug("Announcement from unknown peer")
return
}

if fp.lastAnnounced != nil && head.Td.Cmp(fp.lastAnnounced.td) <= 0 {
// announced tds should be strictly monotonic
p.Log().Debug("Received non-monotonic td", "current", head.Td, "previous", fp.lastAnnounced.td)
```

```
go f.pm.removePeer(p.id)
return
}

n := fp.lastAnnounced
for i := uint64(0); i < head.ReorgDepth; i++ {
if n == nil {
break
}
n = n.parent
}
if n != nil {
// n is now the reorg common ancestor, add a new branch of nodes
// check if the node count is too high to add new nodes
locked := false
for uint64(fp.nodeCnt)+head.Number-n.number > maxNodeCount && fp.root != nil {
if !locked {
f.chain.LockChain()
defer f.chain.UnlockChain()
locked = true
}
// if one of root's children is canonical, keep it, delete other branches and root itself
var newRoot *fetcherTreeNode
for i, nn := range fp.root.children {
if core.GetCanonicalHash(f.pm.chainDb, nn.number) == nn.hash {
fp.root.children = append(fp.root.children[:i], fp.root.children[i+1:]...)
nn.parent = nil
newRoot = nn
break
}
}
fp.deleteNode(fp.root)
if n == fp.root {
n = newRoot
}
fp.root = newRoot
if newRoot == nil || !f.checkKnownNode(p, newRoot) {
fp.bestConfirmed = nil
fp.confirmedTd = nil
}

if n == nil {
```

```
break
}
}
if n != nil {
for n.number < head.Number {
nn := &fetcherTreeNode{number: n.number + 1, parent: n}
n.children = append(n.children, nn)
n = nn
fp.nodeCnt++
}
n.hash = head.Hash
n.td = head.Td
fp.nodeByHash[n.hash] = n
}
}
if n == nil {
// could not find reorg common ancestor or had to delete entire tree, a new root and a resync is
needed
if fp.root != nil {
fp.deleteNode(fp.root)
}
n = &fetcherTreeNode{hash: head.Hash, number: head.Number, td: head.Td}
fp.root = n
fp.nodeCnt++
fp.nodeByHash[n.hash] = n
fp.bestConfirmed = nil
fp.confirmedTd = nil
}

f.checkKnownNode(p, n)
p.lock.Lock()
p.headInfo = head
fp.lastAnnounced = n
p.lock.Unlock()
f.checkUpdateStats(p, nil)
f.requestChn <- true
}

// peerHasBlock returns true if we can assume the peer knows the given block
// based on its announcements
func (f *lightFetcher) peerHasBlock(p *peer, hash common.Hash, number uint64) bool {
f.lock.Lock()
```

```
defer f.lock.Unlock()

if f.syncing {
// always return true when syncing
// false positives are acceptable, a more sophisticated condition can be implemented later
return true
}

fp := f.peers[p]
if fp == nil || fp.root == nil {
return false
}

if number >= fp.root.number {
// it is recent enough that if it is known, is should be in the peer's block tree
return fp.nodeByHash[hash] != nil
}
f.chain.LockChain()
defer f.chain.UnlockChain()
// if it's older than the peer's block tree root but it's in the same canonical chain
// as the root, we can still be sure the peer knows it
//
// when syncing, just check if it is part of the known chain, there is nothing better we
// can do since we do not know the most recent block hash yet
return core.GetCanonicalHash(f.pm.chainDb, fp.root.number) == fp.root.hash &&
core.GetCanonicalHash(f.pm.chainDb, number) == hash
}

// requestAmount calculates the amount of headers to be downloaded starting
// from a certain head backwards
func (f *lightFetcher) requestAmount(p *peer, n *fetcherTreeNode) uint64 {
amount := uint64(0)
nn := n
for nn != nil && !f.checkKnownNode(p, nn) {
nn = nn.parent
amount++
}
if nn == nil {
amount = n.number
}
return amount
}
```

```
// requestedID tells if a certain reqID has been requested by the fetcher
func (f *lightFetcher) requestedID(reqID uint64) bool {
f.reqMu.RLock()
_, ok := f.requested[reqID]
f.reqMu.RUnlock()
return ok
}

// nextRequest selects the peer and announced head to be requested next, amount
// to be downloaded starting from the head backwards is also returned
func (f *lightFetcher) nextRequest() (*distReq, uint64) {
var (
bestHash   common.Hash
bestAmount uint64
)
bestTd := f.maxConfirmedTd
bestSyncing := false

for p, fp := range f.peers {
for hash, n := range fp.nodeByHash {
if !f.checkKnownNode(p, n) && !n.requested && (bestTd == nil || n.td.Cmp(bestTd) >= 0) {
amount := f.requestAmount(p, n)
if bestTd == nil || n.td.Cmp(bestTd) > 0 || amount < bestAmount {
bestHash = hash
bestAmount = amount
bestTd = n.td
bestSyncing = fp.bestConfirmed == nil || fp.root == nil || !f.checkKnownNode(p, fp.root)
}
}
}
}
if bestTd == f.maxConfirmedTd {
return nil, 0
}

f.syncing = bestSyncing

var rq *distReq
reqID := genReqID()
if f.syncing {
rq = &distReq{
```

```
getCost: func(dp distPeer) uint64 {
return 0
},
canSend: func(dp distPeer) bool {
p := dp.(*peer)
fp := f.peers[p]
return fp != nil && fp.nodeByHash[bestHash] != nil
},
request: func(dp distPeer) func() {
go func() {
p := dp.(*peer)
p.Log().Debug("Synchronisation started")
f.pm.synchronise(p)
f.syncDone <- p
}()
return nil
},
}
} else {
rq = &distReq{
getCost: func(dp distPeer) uint64 {
p := dp.(*peer)
return p.GetRequestCost(GetBlockHeadersMsg, int(bestAmount))
},
canSend: func(dp distPeer) bool {
p := dp.(*peer)
f.lock.Lock()
defer f.lock.Unlock()

fp := f.peers[p]
if fp == nil {
return false
}
n := fp.nodeByHash[bestHash]
return n != nil && !n.requested
},
request: func(dp distPeer) func() {
p := dp.(*peer)
f.lock.Lock()
fp := f.peers[p]
if fp != nil {
n := fp.nodeByHash[bestHash]
```

```
if n != nil {
n.requested = true
}
}
f.lock.Unlock()

cost := p.GetRequestCost(GetBlockHeadersMsg, int(bestAmount))
p.fcServer.QueueRequest(reqID, cost)
f.reqMu.Lock()
f.requested[reqID] = fetchRequest{hash: bestHash, amount: bestAmount, peer: p, sent:
mclock.Now()}
f.reqMu.Unlock()
go func() {
time.Sleep(hardRequestTimeout)
f.timeoutChn <- reqID
}()
return func() { p.RequestHeadersByHash(reqID, cost, bestHash, int(bestAmount), 0, true) }
},
}
}
return rq, reqID
}

// deliverHeaders delivers header download request responses for processing
func (f *lightFetcher) deliverHeaders(peer *peer, reqID uint64, headers []*types.Header) {
f.deliverChn <- fetchResponse{reqID: reqID, headers: headers, peer: peer}
}

// processResponse processes header download request responses, returns true if successful
func (f *lightFetcher) processResponse(req fetchRequest, resp fetchResponse) bool {
if uint64(len(resp.headers)) != req.amount || resp.headers[0].Hash() != req.hash {
req.peer.Log().Debug("Response content mismatch", "requested", len(resp.headers), "reqfrom",
resp.headers[0], "delivered", req.amount, "delfrom", req.hash)
return false
}
headers := make([]*types.Header, req.amount)
for i, header := range resp.headers {
headers[int(req.amount)-1-i] = header
}
if _, err := f.chain.InsertHeaderChain(headers, 1); err != nil {
if err == consensus.ErrFutureBlock {
return true
```

```
}
log.Debug("Failed to insert header chain", "err", err)
return false
}
tds := make([]*big.Int, len(headers))
for i, header := range headers {
td := f.chain.GetTd(header.Hash(), header.Number.Uint64())
if td == nil {
log.Debug("Total difficulty not found for header", "index", i+1, "number", header.Number, "hash",
header.Hash())
return false
}
tds[i] = td
}
f.newHeaders(headers, tds)
return true
}

// newHeaders updates the block trees of all active peers according to a newly
// downloaded and validated batch or headers
func (f *lightFetcher) newHeaders(headers []*types.Header, tds []*big.Int) {
var maxTd *big.Int
for p, fp := range f.peers {
if !f.checkAnnouncedHeaders(fp, headers, tds) {
p.Log().Debug("Inconsistent announcement")
go f.pm.removePeer(p.id)
}
if fp.confirmedTd != nil && (maxTd == nil || maxTd.Cmp(fp.confirmedTd) > 0) {
maxTd = fp.confirmedTd
}
}
if maxTd != nil {
f.updateMaxConfirmedTd(maxTd)
}
}

// checkAnnouncedHeaders updates peer's block tree if necessary after validating
// a batch of headers. It searches for the latest header in the batch that has a
// matching tree node (if any), and if it has not been marked as known already,
// sets it and its parents to known (even those which are older than the currently
// validated ones). Return value shows if all hashes, numbers and Tds matched
// correctly to the announced values (otherwise the peer should be dropped).
```

```
func (f *lightFetcher) checkAnnouncedHeaders(fp *fetcherPeerInfo, headers []*types.Header, tds
[]*big.Int) bool {
var (
n      *fetcherTreeNode
header *types.Header
td     *big.Int
)

for i := len(headers) - 1; ; i-- {
if i < 0 {
if n == nil {
// no more headers and nothing to match
return true
}
// we ran out of recently delivered headers but have not reached a node known by this peer yet,
continue matching
td = f.chain.GetTd(header.ParentHash, header.Number.Uint64()-1)
header = f.chain.GetHeader(header.ParentHash, header.Number.Uint64()-1)
} else {
header = headers[i]
td = tds[i]
}
hash := header.Hash()
number := header.Number.Uint64()
if n == nil {
n = fp.nodeByHash[hash]
}
if n != nil {
if n.td == nil {
// node was unannounced
if nn := fp.nodeByHash[hash]; nn != nil {
// if there was already a node with the same hash, continue there and drop this one
nn.children = append(nn.children, n.children...)
n.children = nil
fp.deleteNode(n)
n = nn
} else {
n.hash = hash
n.td = td
fp.nodeByHash[hash] = n
}
}
```

```
// check if it matches the header
if n.hash != hash || n.number != number || n.td.Cmp(td) != 0 {
// peer has previously made an invalid announcement
return false
}
if n.known {
// we reached a known node that matched our expectations, return with success
return true
}
n.known = true
if fp.confirmedTd == nil || td.Cmp(fp.confirmedTd) > 0 {
fp.confirmedTd = td
fp.bestConfirmed = n
}
n = n.parent
if n == nil {
return true
}
}
}
}

// checkSyncedHeaders updates peer's block tree after synchronisation by marking
// downloaded headers as known. If none of the announced headers are found after
// syncing, the peer is dropped.
func (f *lightFetcher) checkSyncedHeaders(p *peer) {
fp := f.peers[p]
if fp == nil {
p.Log().Debug("Unknown peer to check sync headers")
return
}
n := fp.lastAnnounced
var td *big.Int
for n != nil {
if td = f.chain.GetTd(n.hash, n.number); td != nil {
break
}
n = n.parent
}
// now n is the latest downloaded header after syncing
if n == nil {
p.Log().Debug("Synchronisation failed")
```

```
go f.pm.removePeer(p.id)
} else {
header := f.chain.GetHeader(n.hash, n.number)
f.newHeaders([]*types.Header{header}, []*big.Int{td})
}
}

// checkKnownNode checks if a block tree node is known (downloaded and validated)
// If it was not known previously but found in the database, sets its known flag
func (f *lightFetcher) checkKnownNode(p *peer, n *fetcherTreeNode) bool {
if n.known {
return true
}
td := f.chain.GetTd(n.hash, n.number)
if td == nil {
return false
}

fp := f.peers[p]
if fp == nil {
p.Log().Debug("Unknown peer to check known nodes")
return false
}
header := f.chain.GetHeader(n.hash, n.number)
if !f.checkAnnouncedHeaders(fp, []*types.Header{header}, []*big.Int{td}) {
p.Log().Debug("Inconsistent announcement")
go f.pm.removePeer(p.id)
}
if fp.confirmedTd != nil {
f.updateMaxConfirmedTd(fp.confirmedTd)
}
return n.known
}

// deleteNode deletes a node and its child subtrees from a peer's block tree
func (fp *fetcherPeerInfo) deleteNode(n *fetcherTreeNode) {
if n.parent != nil {
for i, nn := range n.parent.children {
if nn == n {
n.parent.children = append(n.parent.children[:i], n.parent.children[i+1:]...)
break
}
```

```
}
}
for {
if n.td != nil {
delete(fp.nodeByHash, n.hash)
}
fp.nodeCnt--
if len(n.children) == 0 {
return
}
for i, nn := range n.children {
if i == 0 {
n = nn
} else {
fp.deleteNode(nn)
}
}
}
}

// updateStatsEntry items form a linked list that is expanded with a new item every time a new head with a higher Td
// than the previous one has been downloaded and validated. The list contains a series of maximum confirmed Td values
// and the time these values have been confirmed, both increasing monotonically. A maximum confirmed Td is calculated
// both globally for all peers and also for each individual peer (meaning that the given peer has announced the head
// and it has also been downloaded from any peer, either before or after the given announcement).
// The linked list has a global tail where new confirmed Td entries are added and a separate head for each peer,
// pointing to the next Td entry that is higher than the peer's max confirmed Td (nil if it has already confirmed
// the current global head).
type updateStatsEntry struct {
time mclock.AbsTime
td   *big.Int
next *updateStatsEntry
}

// updateMaxConfirmedTd updates the block delay statistics of active peers. Whenever a new highest Td is confirmed,
```

```
// adds it to the end of a linked list together with the time it has been confirmed. Then checks which peers have
// already confirmed a head with the same or higher Td (which counts as zero block delay) and updates their statistics.
// Those who have not confirmed such a head by now will be updated by a subsequent checkUpdateStats call with a
// positive block delay value.
func (f *lightFetcher) updateMaxConfirmedTd(td *big.Int) {
if f.maxConfirmedTd == nil || td.Cmp(f.maxConfirmedTd) > 0 {
f.maxConfirmedTd = td
newEntry := &updateStatsEntry{
time: mclock.Now(),
td:   td,
}
if f.lastUpdateStats != nil {
f.lastUpdateStats.next = newEntry
}
f.lastUpdateStats = newEntry
for p := range f.peers {
f.checkUpdateStats(p, newEntry)
}
}
}

// checkUpdateStats checks those peers who have not confirmed a certain highest Td (or a larger one) by the time it
// has been confirmed by another peer. If they have confirmed such a head by now, their stats are updated with the
// block delay which is (this peer's confirmation time)-(first confirmation time). After blockDelayTimeout has passed,
// the stats are updated with blockDelayTimeout value. In either case, the confirmed or timed out updateStatsEntry
// items are removed from the head of the linked list.
// If a new entry has been added to the global tail, it is passed as a parameter here even though this function
// assumes that it has already been added, so that if the peer's list is empty (all heads confirmed, head is nil),
// it can set the new head to newEntry.
func (f *lightFetcher) checkUpdateStats(p *peer, newEntry *updateStatsEntry) {
now := mclock.Now()
fp := f.peers[p]
if fp == nil {
```

```
p.Log().Debug("Unknown peer to check update stats")
return
}
if newEntry != nil && fp.firstUpdateStats == nil {
fp.firstUpdateStats = newEntry
}
for fp.firstUpdateStats != nil && fp.firstUpdateStats.time <= now-
mclock.AbsTime(blockDelayTimeout) {
f.pm.serverPool.adjustBlockDelay(p.poolEntry, blockDelayTimeout)
fp.firstUpdateStats = fp.firstUpdateStats.next
}
if fp.confirmedTd != nil {
for fp.firstUpdateStats != nil && fp.firstUpdateStats.td.Cmp(fp.confirmedTd) <= 0 {
f.pm.serverPool.adjustBlockDelay(p.poolEntry, time.Duration(now-fp.firstUpdateStats.time))
fp.firstUpdateStats = fp.firstUpdateStats.next
}
}
}
```

83:F:\git\coin\ethereum\go-ethereum\les\flowcontrol\control.go

```
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package flowcontrol implements a client side flow control mechanism
package flowcontrol

import (
"sync"
"time"

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

const fcTimeConst = time.Millisecond

type ServerParams struct {
BufLimit, MinRecharge uint64
}

type ClientNode struct {
params   *ServerParams
bufValue uint64
lastTime mclock.AbsTime
```

```
lock     sync.Mutex
cm       *ClientManager
cmNode   *cmNode
}

func NewClientNode(cm *ClientManager, params *ServerParams) *ClientNode {
node := &ClientNode{
cm:       cm,
params:   params,
bufValue: params.BufLimit,
lastTime: mclock.Now(),
}
node.cmNode = cm.addNode(node)
return node
}

func (peer *ClientNode) Remove(cm *ClientManager) {
cm.removeNode(peer.cmNode)
}

func (peer *ClientNode) recalcBV(time mclock.AbsTime) {
dt := uint64(time - peer.lastTime)
if time < peer.lastTime {
dt = 0
}
peer.bufValue += peer.params.MinRecharge * dt / uint64(fcTimeConst)
if peer.bufValue > peer.params.BufLimit {
peer.bufValue = peer.params.BufLimit
}
peer.lastTime = time
}

func (peer *ClientNode) AcceptRequest() (uint64, bool) {
peer.lock.Lock()
defer peer.lock.Unlock()

time := mclock.Now()
peer.recalcBV(time)
return peer.bufValue, peer.cm.accept(peer.cmNode, time)
}

func (peer *ClientNode) RequestProcessed(cost uint64) (bv, realCost uint64) {
```

```
peer.lock.Lock()
defer peer.lock.Unlock()

time := mclock.Now()
peer.recalcBV(time)
peer.bufValue -= cost
peer.recalcBV(time)
rcValue, rcost := peer.cm.processed(peer.cmNode, time)
if rcValue < peer.params.BufLimit {
bv := peer.params.BufLimit - rcValue
if bv > peer.bufValue {
peer.bufValue = bv
}
}
return peer.bufValue, rcost
}

type ServerNode struct {
bufEstimate uint64
lastTime    mclock.AbsTime
params      *ServerParams
sumCost     uint64            // sum of req costs sent to this server
pending     map[uint64]uint64 // value = sumCost after sending the given req
lock        sync.RWMutex
}

func NewServerNode(params *ServerParams) *ServerNode {
return &ServerNode{
bufEstimate: params.BufLimit,
lastTime:    mclock.Now(),
params:      params,
pending:     make(map[uint64]uint64),
}
}

func (peer *ServerNode) recalcBLE(time mclock.AbsTime) {
dt := uint64(time - peer.lastTime)
if time < peer.lastTime {
dt = 0
}
peer.bufEstimate += peer.params.MinRecharge * dt / uint64(fcTimeConst)
if peer.bufEstimate > peer.params.BufLimit {
```

```
peer.bufEstimate = peer.params.BufLimit
}
peer.lastTime = time
}

// safetyMargin is added to the flow control waiting time when estimated buffer value is low
const safetyMargin = time.Millisecond

func (peer *ServerNode) canSend(maxCost uint64) (time.Duration, float64) {
peer.recalcBLE(mclock.Now())
maxCost += uint64(safetyMargin) * peer.params.MinRecharge / uint64(fcTimeConst)
if maxCost > peer.params.BufLimit {
maxCost = peer.params.BufLimit
}
if peer.bufEstimate >= maxCost {
return 0, float64(peer.bufEstimate-maxCost) / float64(peer.params.BufLimit)
}
return time.Duration((maxCost - peer.bufEstimate) * uint64(fcTimeConst) /
peer.params.MinRecharge), 0
}

// CanSend returns the minimum waiting time required before sending a request
// with the given maximum estimated cost. Second return value is the relative
// estimated buffer level after sending the request (divided by BufLimit).
func (peer *ServerNode) CanSend(maxCost uint64) (time.Duration, float64) {
peer.lock.RLock()
defer peer.lock.RUnlock()

return peer.canSend(maxCost)
}

// QueueRequest should be called when the request has been assigned to the given
// server node, before putting it in the send queue. It is mandatory that requests
// are sent in the same order as the QueueRequest calls are made.
func (peer *ServerNode) QueueRequest(reqID, maxCost uint64) {
peer.lock.Lock()
defer peer.lock.Unlock()

peer.bufEstimate -= maxCost
peer.sumCost += maxCost
if reqID >= 0 {
peer.pending[reqID] = peer.sumCost
```

```
}
}

// GotReply adjusts estimated buffer value according to the value included in
// the latest request reply.
func (peer *ServerNode) GotReply(reqID, bv uint64) {

peer.lock.Lock()
defer peer.lock.Unlock()

if bv > peer.params.BufLimit {
bv = peer.params.BufLimit
}
sc, ok := peer.pending[reqID]
if !ok {
return
}
delete(peer.pending, reqID)
cc := peer.sumCost - sc
peer.bufEstimate = 0
if bv > cc {
peer.bufEstimate = bv - cc
}
peer.lastTime = mclock.Now()
}

84:F:\git\coin\ethereum\go-ethereum\les\flowcontrol\manager.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package flowcontrol implements a client side flow control mechanism
package flowcontrol

import (
"sync"
"time"

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

const rcConst = 1000000

type cmNode struct {
```

```
node                    *ClientNode
lastUpdate              mclock.AbsTime
serving, recharging     bool
rcWeight                uint64
rcValue, rcDelta, startValue int64
finishRecharge          mclock.AbsTime
}

func (node *cmNode) update(time mclock.AbsTime) {
dt := int64(time - node.lastUpdate)
node.rcValue += node.rcDelta * dt / rcConst
node.lastUpdate = time
if node.recharging && time >= node.finishRecharge {
node.recharging = false
node.rcDelta = 0
node.rcValue = 0
}
}

func (node *cmNode) set(serving bool, simReqCnt, sumWeight uint64) {
if node.serving && !serving {
node.recharging = true
sumWeight += node.rcWeight
}
node.serving = serving
if node.recharging && serving {
node.recharging = false
sumWeight -= node.rcWeight
}

node.rcDelta = 0
if serving {
node.rcDelta = int64(rcConst / simReqCnt)
}
if node.recharging {
node.rcDelta = -int64(node.node.cm.rcRecharge * node.rcWeight / sumWeight)
node.finishRecharge = node.lastUpdate + mclock.AbsTime(node.rcValue*rcConst/(-node.rcDelta))
}
}

type ClientManager struct {
lock                    sync.Mutex
```

```
nodes                       map[*cmNode]struct{}
simReqCnt, sumWeight, rcSumValue uint64
maxSimReq, maxRcSum             uint64
rcRecharge                  uint64
resumeQueue                    chan chan bool
time                        mclock.AbsTime
}

func NewClientManager(rcTarget, maxSimReq, maxRcSum uint64) *ClientManager {
cm := &ClientManager{
nodes:       make(map[*cmNode]struct{}),
resumeQueue: make(chan chan bool),
rcRecharge:  rcConst * rcConst / (100*rcConst/rcTarget - rcConst),
maxSimReq:   maxSimReq,
maxRcSum:    maxRcSum,
}
go cm.queueProc()
return cm
}

func (self *ClientManager) Stop() {
self.lock.Lock()
defer self.lock.Unlock()

// signal any waiting accept routines to return false
self.nodes = make(map[*cmNode]struct{})
close(self.resumeQueue)
}

func (self *ClientManager) addNode(cnode *ClientNode) *cmNode {
time := mclock.Now()
node := &cmNode{
node:          cnode,
lastUpdate:    time,
finishRecharge: time,
rcWeight:      1,
}
self.lock.Lock()
defer self.lock.Unlock()

self.nodes[node] = struct{}{}
self.update(mclock.Now())
```

```
return node
}

func (self *ClientManager) removeNode(node *cmNode) {
self.lock.Lock()
defer self.lock.Unlock()

time := mclock.Now()
self.stop(node, time)
delete(self.nodes, node)
self.update(time)
}

// recalc sumWeight
func (self *ClientManager) updateNodes(time mclock.AbsTime) (rce bool) {
var sumWeight, rcSum uint64
for node := range self.nodes {
rc := node.recharging
node.update(time)
if rc && !node.recharging {
rce = true
}
if node.recharging {
sumWeight += node.rcWeight
}
rcSum += uint64(node.rcValue)
}
self.sumWeight = sumWeight
self.rcSumValue = rcSum
return
}

func (self *ClientManager) update(time mclock.AbsTime) {
for {
firstTime := time
for node := range self.nodes {
if node.recharging && node.finishRecharge < firstTime {
firstTime = node.finishRecharge
}
}
if self.updateNodes(firstTime) {
for node := range self.nodes {
```

```
if node.recharging {
node.set(node.serving, self.simReqCnt, self.sumWeight)
}
}
} else {
self.time = time
return
}
}
}

func (self *ClientManager) canStartReq() bool {
return self.simReqCnt < self.maxSimReq && self.rcSumValue < self.maxRcSum
}

func (self *ClientManager) queueProc() {
for rc := range self.resumeQueue {
for {
time.Sleep(time.Millisecond * 10)
self.lock.Lock()
self.update(mclock.Now())
cs := self.canStartReq()
self.lock.Unlock()
if cs {
break
}
}
close(rc)
}
}

func (self *ClientManager) accept(node *cmNode, time mclock.AbsTime) bool {
self.lock.Lock()
defer self.lock.Unlock()

self.update(time)
if !self.canStartReq() {
resume := make(chan bool)
self.lock.Unlock()
self.resumeQueue <- resume
<-resume
self.lock.Lock()
```

```
if _, ok := self.nodes[node]; !ok {
return false // reject if node has been removed or manager has been stopped
}
}
self.simReqCnt++
node.set(true, self.simReqCnt, self.sumWeight)
node.startValue = node.rcValue
self.update(self.time)
return true
}

func (self *ClientManager) stop(node *cmNode, time mclock.AbsTime) {
if node.serving {
self.update(time)
self.simReqCnt--
node.set(false, self.simReqCnt, self.sumWeight)
self.update(time)
}
}

func (self *ClientManager) processed(node *cmNode, time mclock.AbsTime) (rcValue, rcCost
uint64) {
self.lock.Lock()
defer self.lock.Unlock()

self.stop(node, time)
return uint64(node.rcValue), uint64(node.rcValue - node.startValue)
}

85:F:\git\coin\ethereum\go-ethereum\les\handler.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package les implements the Light Ethereum Subprotocol.
package les

import (
"encoding/binary"
"errors"
"fmt"
"math/big"
"net"
"sync"
```

```
"time"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/state"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/eth"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/p2p/discover"
"github.com/ethereum/go-ethereum/p2p/discv5"
"github.com/ethereum/go-ethereum/params"
"github.com/ethereum/go-ethereum/rlp"
"github.com/ethereum/go-ethereum/trie"
)

const (
softResponseLimit = 2 * 1024 * 1024 // Target maximum size of returned blocks, headers or node
data.
estHeaderRlpSize  = 500           // Approximate size of an RLP encoded block header

ethVersion = 63 // equivalent eth version for the downloader

MaxHeaderFetch       = 192 // Amount of block headers to be fetched per retrieval request
MaxBodyFetch         = 32  // Amount of block bodies to be fetched per retrieval request
MaxReceiptFetch      = 128 // Amount of transaction receipts to allow fetching per request
MaxCodeFetch         = 64  // Amount of contract codes to allow fetching per request
MaxProofsFetch       = 64  // Amount of merkle proofs to be fetched per retrieval request
MaxHeaderProofsFetch = 64  // Amount of merkle proofs to be fetched per retrieval request
MaxTxSend            = 64  // Amount of transactions to be send per request

disableClientRemovePeer = false
)

// errIncompatibleConfig is returned if the requested protocols and configs are
// not compatible (low protocol version restrictions and high requirements).
var errIncompatibleConfig = errors.New("incompatible configuration")
```

```
func errResp(code errCode, format string, v ...interface{}) error {
return fmt.Errorf("%v - %v", code, fmt.Sprintf(format, v...))
}

type hashFetcherFn func(common.Hash) error

type BlockChain interface {
HasHeader(hash common.Hash) bool
GetHeader(hash common.Hash, number uint64) *types.Header
GetHeaderByHash(hash common.Hash) *types.Header
CurrentHeader() *types.Header
GetTdByHash(hash common.Hash) *big.Int
InsertHeaderChain(chain []*types.Header, checkFreq int) (int, error)
Rollback(chain []common.Hash)
Status() (td *big.Int, currentBlock common.Hash, genesisBlock common.Hash)
GetHeaderByNumber(number uint64) *types.Header
GetBlockHashesFromHash(hash common.Hash, max uint64) []common.Hash
LastBlockHash() common.Hash
Genesis() *types.Block
}

type txPool interface {
// AddTransactions should add the given transactions to the pool.
AddBatch([]*types.Transaction) error
}

type ProtocolManager struct {
lightSync   bool
txpool      txPool
txrelay     *LesTxRelay
networkId   uint64
chainConfig *params.ChainConfig
blockchain  BlockChain
chainDb     ethdb.Database
odr         *LesOdr
server      *LesServer
serverPool  *serverPool
lesTopic    discv5.Topic
reqDist     *requestDistributor
retriever   *retrieveManager

downloader *downloader.Downloader
```

```
fetcher    *lightFetcher
peers      *peerSet

SubProtocols []p2p.Protocol

eventMux *event.TypeMux

// channels for fetcher, syncer, txsyncLoop
newPeerCh   chan *peer
quitSync    chan struct{}
noMorePeers chan struct{}

syncMu   sync.Mutex
syncing  bool
syncDone chan struct{}

// wait group is used for graceful shutdowns during downloading
// and processing
wg *sync.WaitGroup
}

// NewProtocolManager returns a new ethereum sub protocol manager. The Ethereum sub
protocol manages peers capable
// with the ethereum network.
func NewProtocolManager(chainConfig *params.ChainConfig, lightSync bool, networkId uint64,
mux *event.TypeMux, engine consensus.Engine, peers *peerSet, blockchain BlockChain, txpool
txPool, chainDb ethdb.Database, odr *LesOdr, txrelay *LesTxRelay, quitSync chan struct{}, wg
*sync.WaitGroup) (*ProtocolManager, error) {
// Create the protocol manager with the base fields
manager := &ProtocolManager{
lightSync:   lightSync,
eventMux:    mux,
blockchain:  blockchain,
chainConfig: chainConfig,
chainDb:     chainDb,
odr:         odr,
networkId:   networkId,
txpool:      txpool,
txrelay:     txrelay,
peers:       peers,
newPeerCh:   make(chan *peer),
quitSync:    quitSync,
```

```
wg:          wg,
noMorePeers: make(chan struct{}),
}
if odr != nil {
manager.retriever = odr.retriever
manager.reqDist = odr.retriever.dist
}
// Initiate a sub-protocol for every implemented version we can handle
manager.SubProtocols = make([]p2p.Protocol, 0, len(ProtocolVersions))
for i, version := range ProtocolVersions {
// Compatible, initialize the sub-protocol
version := version // Closure for the run
manager.SubProtocols = append(manager.SubProtocols, p2p.Protocol{
Name:    "les",
Version: version,
Length:  ProtocolLengths[i],
Run: func(p *p2p.Peer, rw p2p.MsgReadWriter) error {
var entry *poolEntry
peer := manager.newPeer(int(version), networkId, p, rw)
if manager.serverPool != nil {
addr := p.RemoteAddr().(*net.TCPAddr)
entry = manager.serverPool.connect(peer, addr.IP, uint16(addr.Port))
}
peer.poolEntry = entry
select {
case manager.newPeerCh <- peer:
manager.wg.Add(1)
defer manager.wg.Done()
err := manager.handle(peer)
if entry != nil {
manager.serverPool.disconnect(entry)
}
return err
case <-manager.quitSync:
if entry != nil {
manager.serverPool.disconnect(entry)
}
return p2p.DiscQuitting
}
},
NodeInfo: func() interface{} {
return manager.NodeInfo()
```

```
},
PeerInfo: func(id discover.NodeID) interface{} {
if p := manager.peers.Peer(fmt.Sprintf("%x", id[:8])); p != nil {
return p.Info()
}
return nil
},
})
}
if len(manager.SubProtocols) == 0 {
return nil, errIncompatibleConfig
}

removePeer := manager.removePeer
if disableClientRemovePeer {
removePeer = func(id string) {}
}

if lightSync {
manager.downloader = downloader.New(downloader.LightSync, chainDb, manager.eventMux,
blockchain.HasHeader, nil, blockchain.GetHeaderByHash,
nil, blockchain.CurrentHeader, nil, nil, nil, blockchain.GetTdByHash,
blockchain.InsertHeaderChain, nil, nil, blockchain.Rollback, removePeer)
manager.peers.notify((*downloaderPeerNotify)(manager))
manager.fetcher = newLightFetcher(manager)
}

return manager, nil
}

// removePeer initiates disconnection from a peer by removing it from the peer set
func (pm *ProtocolManager) removePeer(id string) {
pm.peers.Unregister(id)
}

func (pm *ProtocolManager) Start() {
if pm.lightSync {
go pm.syncer()
} else {
go func() {
for range pm.newPeerCh {
}
```

```
}()
}
}

func (pm *ProtocolManager) Stop() {
// Showing a log message. During download / process this could actually
// take between 5 to 10 seconds and therefor feedback is required.
log.Info("Stopping light Ethereum protocol")

// Quit the sync loop.
// After this send has completed, no new peers will be accepted.
pm.noMorePeers <- struct{}{}

close(pm.quitSync) // quits syncer, fetcher

// Disconnect existing sessions.
// This also closes the gate for any new registrations on the peer set.
// sessions which are already established but not added to pm.peers yet
// will exit when they try to register.
pm.peers.Close()

// Wait for any process action
pm.wg.Wait()

log.Info("Light Ethereum protocol stopped")
}

func (pm *ProtocolManager) newPeer(pv int, nv uint64, p *p2p.Peer, rw p2p.MsgReadWriter)
*peer {
return newPeer(pv, nv, p, newMeteredMsgWriter(rw))
}

// handle is the callback invoked to manage the life cycle of a les peer. When
// this function terminates, the peer is disconnected.
func (pm *ProtocolManager) handle(p *peer) error {
p.Log().Debug("Light Ethereum peer connected", "name", p.Name())

// Execute the LES handshake
td, head, genesis := pm.blockchain.Status()
headNum := core.GetBlockNumber(pm.chainDb, head)
if err := p.Handshake(td, head, headNum, genesis, pm.server); err != nil {
p.Log().Debug("Light Ethereum handshake failed", "err", err)
```

```
return err
}
if rw, ok := p.rw.(*meteredMsgReadWriter); ok {
rw.Init(p.version)
}
// Register the peer locally
if err := pm.peers.Register(p); err != nil {
p.Log().Error("Light Ethereum peer registration failed", "err", err)
return err
}
defer func() {
if pm.server != nil && pm.server.fcManager != nil && p.fcClient != nil {
p.fcClient.Remove(pm.server.fcManager)
}
pm.removePeer(p.id)
}()
// Register the peer in the downloader. If the downloader considers it banned, we disconnect
if pm.lightSync {
p.lock.Lock()
head := p.headInfo
p.lock.Unlock()
if pm.fetcher != nil {
pm.fetcher.announce(p, head)
}

if p.poolEntry != nil {
pm.serverPool.registered(p.poolEntry)
}
}

stop := make(chan struct{})
defer close(stop)
go func() {
// new block announce loop
for {
select {
case announce := <-p.announceChn:
p.SendAnnounce(announce)
case <-stop:
return
}
}
```

```
}()

// main loop. handle incoming messages.
for {
if err := pm.handleMsg(p); err != nil {
p.Log().Debug("Light Ethereum message handling failed", "err", err)
return err
}
}
}

var reqList = []uint64{GetBlockHeadersMsg, GetBlockBodiesMsg, GetCodeMsg, GetReceiptsMsg,
GetProofsMsg, SendTxMsg, GetHeaderProofsMsg}

// handleMsg is invoked whenever an inbound message is received from a remote
// peer. The remote connection is torn down upon returning any error.
func (pm *ProtocolManager) handleMsg(p *peer) error {
// Read the next message from the remote peer, and ensure it's fully consumed
msg, err := p.rw.ReadMsg()
if err != nil {
return err
}
p.Log().Trace("Light Ethereum message arrived", "code", msg.Code, "bytes", msg.Size)

costs := p.fcCosts[msg.Code]
reject := func(reqCnt, maxCnt uint64) bool {
if p.fcClient == nil || reqCnt > maxCnt {
return true
}
bufValue, _ := p.fcClient.AcceptRequest()
cost := costs.baseCost + reqCnt*costs.reqCost
if cost > pm.server.defParams.BufLimit {
cost = pm.server.defParams.BufLimit
}
if cost > bufValue {
recharge := time.Duration((cost - bufValue) * 1000000 / pm.server.defParams.MinRecharge)
p.Log().Error("Request came too early", "recharge", common.PrettyDuration(recharge))
return true
}
return false
}
```

```
if msg.Size > ProtocolMaxMsgSize {
return errResp(ErrMsgTooLarge, "%v > %v", msg.Size, ProtocolMaxMsgSize)
}
defer msg.Discard()

var deliverMsg *Msg

// Handle the message depending on its contents
switch msg.Code {
case StatusMsg:
p.Log().Trace("Received status message")
// Status messages should never arrive after the handshake
return errResp(ErrExtraStatusMsg, "uncontrolled status message")

// Block header query, collect the requested headers and reply
case AnnounceMsg:
p.Log().Trace("Received announce message")

var req announceData
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "%v: %v", msg, err)
}
p.Log().Trace("Announce message content", "number", req.Number, "hash", req.Hash, "td",
req.Td, "reorg", req.ReorgDepth)
if pm.fetcher != nil {
pm.fetcher.announce(p, &req)
}

case GetBlockHeadersMsg:
p.Log().Trace("Received block header request")
// Decode the complex header query
var req struct {
ReqID uint64
Query getBlockHeadersData
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "%v: %v", msg, err)
}

query := req.Query
if reject(query.Amount, MaxHeaderFetch) {
return errResp(ErrRequestRejected, "")
```

```
}

hashMode := query.Origin.Hash != (common.Hash{})

// Gather headers until the fetch or network limits is reached
var (
bytes   common.StorageSize
headers []*types.Header
unknown bool
)
for !unknown && len(headers) < int(query.Amount) && bytes < softResponseLimit {
// Retrieve the next header satisfying the query
var origin *types.Header
if hashMode {
origin = pm.blockchain.GetHeaderByHash(query.Origin.Hash)
} else {
origin = pm.blockchain.GetHeaderByNumber(query.Origin.Number)
}
if origin == nil {
break
}
number := origin.Number.Uint64()
headers = append(headers, origin)
bytes += estHeaderRlpSize

// Advance to the next header of the query
switch {
case query.Origin.Hash != (common.Hash{}) && query.Reverse:
// Hash based traversal towards the genesis block
for i := 0; i < int(query.Skip)+1; i++ {
if header := pm.blockchain.GetHeader(query.Origin.Hash, number); header != nil {
query.Origin.Hash = header.ParentHash
number--
} else {
unknown = true
break
}
}
case query.Origin.Hash != (common.Hash{}) && !query.Reverse:
// Hash based traversal towards the leaf block
if header := pm.blockchain.GetHeaderByNumber(origin.Number.Uint64() + query.Skip + 1);
header != nil {
```

```
if pm.blockchain.GetBlockHashesFromHash(header.Hash(), query.Skip+1)[query.Skip] ==
query.Origin.Hash {
query.Origin.Hash = header.Hash()
} else {
unknown = true
}
} else {
unknown = true
}
case query.Reverse:
// Number based traversal towards the genesis block
if query.Origin.Number >= query.Skip+1 {
query.Origin.Number -= (query.Skip + 1)
} else {
unknown = true
}

case !query.Reverse:
// Number based traversal towards the leaf block
query.Origin.Number += (query.Skip + 1)
}
}

bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + query.Amount*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, query.Amount, rcost)
return p.SendBlockHeaders(req.ReqID, bv, headers)

case BlockHeadersMsg:
if pm.downloader == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received block header response message")
// A batch of headers arrived to one of our previous requests
var resp struct {
ReqID, BV uint64
Headers   []*types.Header
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
```

```
if pm.fetcher != nil && pm.fetcher.requestedID(resp.ReqID) {
pm.fetcher.deliverHeaders(p, resp.ReqID, resp.Headers)
} else {
err := pm.downloader.DeliverHeaders(p.id, resp.Headers)
if err != nil {
log.Debug(fmt.Sprint(err))
}
}

case GetBlockBodiesMsg:
p.Log().Trace("Received block bodies request")
// Decode the retrieval message
var req struct {
ReqID  uint64
Hashes []common.Hash
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Gather blocks until the fetch or network limits is reached
var (
bytes  int
bodies []rlp.RawValue
)
reqCnt := len(req.Hashes)
if reject(uint64(reqCnt), MaxBodyFetch) {
return errResp(ErrRequestRejected, "")
}
for _, hash := range req.Hashes {
if bytes >= softResponseLimit {
break
}
// Retrieve the requested block body, stopping if enough was found
if data := core.GetBodyRLP(pm.chainDb, hash, core.GetBlockNumber(pm.chainDb, hash));
len(data) != 0 {
bodies = append(bodies, data)
bytes += len(data)
}
}
bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)
return p.SendBlockBodiesRLP(req.ReqID, bv, bodies)
```

```
case BlockBodiesMsg:
if pm.odr == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received block bodies response")
// A batch of block bodies arrived to one of our previous requests
var resp struct {
ReqID, BV uint64
Data      []*types.Body
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
deliverMsg = &Msg{
MsgType: MsgBlockBodies,
ReqID:   resp.ReqID,
Obj:     resp.Data,
}

case GetCodeMsg:
p.Log().Trace("Received code request")
// Decode the retrieval message
var req struct {
ReqID uint64
Reqs  []CodeReq
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Gather state data until the fetch or network limits is reached
var (
bytes int
data  [][]byte
)
reqCnt := len(req.Reqs)
if reject(uint64(reqCnt), MaxCodeFetch) {
return errResp(ErrRequestRejected, "")
}
for _, req := range req.Reqs {
```

```
// Retrieve the requested state entry, stopping if enough was found
if header := core.GetHeader(pm.chainDb, req.BHash, core.GetBlockNumber(pm.chainDb,
req.BHash)); header != nil {
if trie, _ := trie.New(header.Root, pm.chainDb); trie != nil {
sdata := trie.Get(req.AccKey)
var acc state.Account
if err := rlp.DecodeBytes(sdata, &acc); err == nil {
entry, _ := pm.chainDb.Get(acc.CodeHash)
if bytes+len(entry) >= softResponseLimit {
break
}
data = append(data, entry)
bytes += len(entry)
}
}
}
}
bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)
return p.SendCode(req.ReqID, bv, data)

case CodeMsg:
if pm.odr == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received code response")
// A batch of node state data arrived to one of our previous requests
var resp struct {
ReqID, BV uint64
Data      [][]byte
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
deliverMsg = &Msg{
MsgType: MsgCode,
ReqID:   resp.ReqID,
Obj:     resp.Data,
}
```

```
case GetReceiptsMsg:
p.Log().Trace("Received receipts request")
// Decode the retrieval message
var req struct {
ReqID  uint64
Hashes []common.Hash
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Gather state data until the fetch or network limits is reached
var (
bytes    int
receipts []rlp.RawValue
)
reqCnt := len(req.Hashes)
if reject(uint64(reqCnt), MaxReceiptFetch) {
return errResp(ErrRequestRejected, "")
}
for _, hash := range req.Hashes {
if bytes >= softResponseLimit {
break
}
// Retrieve the requested block's receipts, skipping if unknown to us
results := core.GetBlockReceipts(pm.chainDb, hash, core.GetBlockNumber(pm.chainDb, hash))
if results == nil {
if header := pm.blockchain.GetHeaderByHash(hash); header == nil || header.ReceiptHash !=
types.EmptyRootHash {
continue
}
}
// If known, encode and queue for response packet
if encoded, err := rlp.EncodeToBytes(results); err != nil {
log.Error("Failed to encode receipt", "err", err)
} else {
receipts = append(receipts, encoded)
bytes += len(encoded)
}
}
bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)
return p.SendReceiptsRLP(req.ReqID, bv, receipts)
```

```
case ReceiptsMsg:
if pm.odr == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received receipts response")
// A batch of receipts arrived to one of our previous requests
var resp struct {
ReqID, BV uint64
Receipts  []types.Receipts
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
deliverMsg = &Msg{
MsgType: MsgReceipts,
ReqID:   resp.ReqID,
Obj:     resp.Receipts,
}

case GetProofsMsg:
p.Log().Trace("Received proofs request")
// Decode the retrieval message
var req struct {
ReqID uint64
Reqs  []ProofReq
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Gather state data until the fetch or network limits is reached
var (
bytes  int
proofs proofsData
)
reqCnt := len(req.Reqs)
if reject(uint64(reqCnt), MaxProofsFetch) {
return errResp(ErrRequestRejected, "")
}
for _, req := range req.Reqs {
```

```
if bytes >= softResponseLimit {
break
}
// Retrieve the requested state entry, stopping if enough was found
if header := core.GetHeader(pm.chainDb, req.BHash, core.GetBlockNumber(pm.chainDb,
req.BHash)); header != nil {
if tr, _ := trie.New(header.Root, pm.chainDb); tr != nil {
if len(req.AccKey) > 0 {
sdata := tr.Get(req.AccKey)
tr = nil
var acc state.Account
if err := rlp.DecodeBytes(sdata, &acc); err == nil {
tr, _ = trie.New(acc.Root, pm.chainDb)
}
}
if tr != nil {
proof := tr.Prove(req.Key)
proofs = append(proofs, proof)
bytes += len(proof)
}
}
}
}
bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)
return p.SendProofs(req.ReqID, bv, proofs)

case ProofsMsg:
if pm.odr == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received proofs response")
// A batch of merkle proofs arrived to one of our previous requests
var resp struct {
ReqID, BV uint64
Data      [][]rlp.RawValue
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
```

```
deliverMsg = &Msg{
MsgType: MsgProofs,
ReqID:   resp.ReqID,
Obj:     resp.Data,
}

case GetHeaderProofsMsg:
p.Log().Trace("Received headers proof request")
// Decode the retrieval message
var req struct {
ReqID uint64
Reqs  []ChtReq
}
if err := msg.Decode(&req); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
// Gather state data until the fetch or network limits is reached
var (
bytes  int
proofs []ChtResp
)
reqCnt := len(req.Reqs)
if reject(uint64(reqCnt), MaxHeaderProofsFetch) {
return errResp(ErrRequestRejected, "")
}
for _, req := range req.Reqs {
if bytes >= softResponseLimit {
break
}

if header := pm.blockchain.GetHeaderByNumber(req.BlockNum); header != nil {
if root := getChtRoot(pm.chainDb, req.ChtNum); root != (common.Hash{}) {
if tr, _ := trie.New(root, pm.chainDb); tr != nil {
var encNumber [8]byte
binary.BigEndian.PutUint64(encNumber[:], req.BlockNum)
proof := tr.Prove(encNumber[:])
proofs = append(proofs, ChtResp{Header: header, Proof: proof})
bytes += len(proof) + estHeaderRlpSize
}
}
}
}
```

```
bv, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)
return p.SendHeaderProofs(req.ReqID, bv, proofs)

case HeaderProofsMsg:
if pm.odr == nil {
return errResp(ErrUnexpectedResponse, "")
}

p.Log().Trace("Received headers proof response")
var resp struct {
ReqID, BV uint64
Data      []ChtResp
}
if err := msg.Decode(&resp); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
p.fcServer.GotReply(resp.ReqID, resp.BV)
deliverMsg = &Msg{
MsgType: MsgHeaderProofs,
ReqID:   resp.ReqID,
Obj:     resp.Data,
}

case SendTxMsg:
if pm.txpool == nil {
return errResp(ErrUnexpectedResponse, "")
}
// Transactions arrived, parse all of them and deliver to the pool
var txs []*types.Transaction
if err := msg.Decode(&txs); err != nil {
return errResp(ErrDecode, "msg %v: %v", msg, err)
}
reqCnt := len(txs)
if reject(uint64(reqCnt), MaxTxSend) {
return errResp(ErrRequestRejected, "")
}

if err := pm.txpool.AddBatch(txs); err != nil {
return errResp(ErrUnexpectedResponse, "msg: %v", err)
}
```

```
_, rcost := p.fcClient.RequestProcessed(costs.baseCost + uint64(reqCnt)*costs.reqCost)
pm.server.fcCostStats.update(msg.Code, uint64(reqCnt), rcost)

default:
p.Log().Trace("Received unknown message", "code", msg.Code)
return errResp(ErrInvalidMsgCode, "%v", msg.Code)
}

if deliverMsg != nil {
err := pm.retriever.deliver(p, deliverMsg)
if err != nil {
p.responseErrors++
if p.responseErrors > maxResponseErrors {
return err
}
}
}
return nil
}

// NodeInfo retrieves some protocol metadata about the running host node.
func (self *ProtocolManager) NodeInfo() *eth.EthNodeInfo {
return &eth.EthNodeInfo{
Network:    self.networkId,
Difficulty: self.blockchain.GetTdByHash(self.blockchain.LastBlockHash()),
Genesis:    self.blockchain.Genesis().Hash(),
Head:       self.blockchain.LastBlockHash(),
}
}

// downloaderPeerNotify implements peerSetNotify
type downloaderPeerNotify ProtocolManager

func (d *downloaderPeerNotify) registerPeer(p *peer) {
pm := (*ProtocolManager)(d)

requestHeadersByHash := func(origin common.Hash, amount int, skip int, reverse bool) error {
reqID := genReqID()
rq := &distReq{
getCost: func(dp distPeer) uint64 {
peer := dp.(*peer)
return peer.GetRequestCost(GetBlockHeadersMsg, amount)
```

```
},
canSend: func(dp distPeer) bool {
return dp.(*peer) == p
},
request: func(dp distPeer) func() {
peer := dp.(*peer)
cost := peer.GetRequestCost(GetBlockHeadersMsg, amount)
peer.fcServer.QueueRequest(reqID, cost)
return func() { peer.RequestHeadersByHash(reqID, cost, origin, amount, skip, reverse) }
},
}
_, ok := <-pm.reqDist.queue(rq)
if !ok {
return ErrNoPeers
}
return nil
}
requestHeadersByNumber := func(origin uint64, amount int, skip int, reverse bool) error {
reqID := genReqID()
rq := &distReq{
getCost: func(dp distPeer) uint64 {
peer := dp.(*peer)
return peer.GetRequestCost(GetBlockHeadersMsg, amount)
},
canSend: func(dp distPeer) bool {
return dp.(*peer) == p
},
request: func(dp distPeer) func() {
peer := dp.(*peer)
cost := peer.GetRequestCost(GetBlockHeadersMsg, amount)
peer.fcServer.QueueRequest(reqID, cost)
return func() { peer.RequestHeadersByNumber(reqID, cost, origin, amount, skip, reverse) }
},
}
_, ok := <-pm.reqDist.queue(rq)
if !ok {
return ErrNoPeers
}
return nil
}

pm.downloader.RegisterPeer(p.id, ethVersion, p.HeadAndTd, requestHeadersByHash,
```

requestHeadersByNumber, nil, nil, nil)
}

func (d *downloaderPeerNotify) unregisterPeer(p *peer) {
pm := (*ProtocolManager)(d)
pm.downloader.UnregisterPeer(p.id)
}

86:F:\git\coin\ethereum\go-ethereum\les\handler_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package les

import (
"math/rand"
"testing"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/eth/downloader"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/p2p"
"github.com/ethereum/go-ethereum/rlp"
"github.com/ethereum/go-ethereum/trie"
)

func expectResponse(r p2p.MsgReader, msgcode, reqID, bv uint64, data interface{}) error {
type resp struct {
ReqID, BV uint64
Data   interface{}
}
return p2p.ExpectMsg(r, msgcode, resp{reqID, bv, data})
}

// Tests that block headers can be retrieved from a remote chain based on user queries.
func TestGetBlockHeadersLes1(t *testing.T) { testGetBlockHeaders(t, 1) }

func testGetBlockHeaders(t *testing.T, protocol int) {
db, _ := ethdb.NewMemDatabase()
pm := newTestProtocolManagerMust(t, false, downloader.MaxHashFetch+15, nil, nil, nil, db)

```
bc := pm.blockchain.(*core.BlockChain)
peer, _ := newTestPeer(t, "peer", protocol, pm, true)
defer peer.close()

// Create a "random" unknown hash for testing
var unknown common.Hash
for i := range unknown {
unknown[i] = byte(i)
}
// Create a batch of tests for various scenarios
limit := uint64(MaxHeaderFetch)
tests := []struct {
query  *getBlockHeadersData // The query to execute for header retrieval
expect []common.Hash        // The hashes of the block whose headers are expected
}{
// A single random block should be retrievable by hash and number too
{
&getBlockHeadersData{Origin: hashOrNumber{Hash: bc.GetBlockByNumber(limit / 2).Hash()},
Amount: 1},
[]common.Hash{bc.GetBlockByNumber(limit / 2).Hash()},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 1},
[]common.Hash{bc.GetBlockByNumber(limit / 2).Hash()},
},
// Multiple headers should be retrievable in both directions
{
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 3},
[]common.Hash{
bc.GetBlockByNumber(limit / 2).Hash(),
bc.GetBlockByNumber(limit/2 + 1).Hash(),
bc.GetBlockByNumber(limit/2 + 2).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Amount: 3, Reverse: true},
[]common.Hash{
bc.GetBlockByNumber(limit / 2).Hash(),
bc.GetBlockByNumber(limit/2 - 1).Hash(),
bc.GetBlockByNumber(limit/2 - 2).Hash(),
},
},
// Multiple headers with skip lists should be retrievable
{
```

```
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Skip: 3, Amount: 3},
[]common.Hash{
bc.GetBlockByNumber(limit / 2).Hash(),
bc.GetBlockByNumber(limit/2 + 4).Hash(),
bc.GetBlockByNumber(limit/2 + 8).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: limit / 2}, Skip: 3, Amount: 3, Reverse:
true},
[]common.Hash{
bc.GetBlockByNumber(limit / 2).Hash(),
bc.GetBlockByNumber(limit/2 - 4).Hash(),
bc.GetBlockByNumber(limit/2 - 8).Hash(),
},
},
// The chain endpoints should be retrievable
{
&getBlockHeadersData{Origin: hashOrNumber{Number: 0}, Amount: 1},
[]common.Hash{bc.GetBlockByNumber(0).Hash()},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: bc.CurrentBlock().NumberU64()},
Amount: 1},
[]common.Hash{bc.CurrentBlock().Hash()},
},
// Ensure protocol limits are honored
/*{
&getBlockHeadersData{Origin: hashOrNumber{Number: bc.CurrentBlock().NumberU64() - 1},
Amount: limit + 10, Reverse: true},
bc.GetBlockHashesFromHash(bc.CurrentBlock().Hash(), limit),
},*/
// Check that requesting more than available is handled gracefully
{
&getBlockHeadersData{Origin: hashOrNumber{Number: bc.CurrentBlock().NumberU64() - 4},
Skip: 3, Amount: 3},
[]common.Hash{
bc.GetBlockByNumber(bc.CurrentBlock().NumberU64() - 4).Hash(),
bc.GetBlockByNumber(bc.CurrentBlock().NumberU64()).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: 4}, Skip: 3, Amount: 3, Reverse: true},
[]common.Hash{
bc.GetBlockByNumber(4).Hash(),
```

```
bc.GetBlockByNumber(0).Hash(),
},
},
// Check that requesting more than available is handled gracefully, even if mid skip
{
&getBlockHeadersData{Origin: hashOrNumber{Number: bc.CurrentBlock().NumberU64() - 4},
Skip: 2, Amount: 3},
[]common.Hash{
bc.GetBlockByNumber(bc.CurrentBlock().NumberU64() - 4).Hash(),
bc.GetBlockByNumber(bc.CurrentBlock().NumberU64() - 1).Hash(),
},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: 4}, Skip: 2, Amount: 3, Reverse: true},
[]common.Hash{
bc.GetBlockByNumber(4).Hash(),
bc.GetBlockByNumber(1).Hash(),
},
},
// Check that non existing headers aren't returned
{
&getBlockHeadersData{Origin: hashOrNumber{Hash: unknown}, Amount: 1},
[]common.Hash{},
}, {
&getBlockHeadersData{Origin: hashOrNumber{Number: bc.CurrentBlock().NumberU64() + 1},
Amount: 1},
[]common.Hash{},
},
}
// Run each of the tests and verify the results against the chain
var reqID uint64
for i, tt := range tests {
// Collect the headers to expect in the response
headers := []*types.Header{}
for _, hash := range tt.expect {
headers = append(headers, bc.GetHeaderByHash(hash))
}
// Send the hash request and verify the response
reqID++
cost := peer.GetRequestCost(GetBlockHeadersMsg, int(tt.query.Amount))
sendRequest(peer.app, GetBlockHeadersMsg, reqID, cost, tt.query)
if err := expectResponse(peer.app, BlockHeadersMsg, reqID, testBufLimit, headers); err != nil {
t.Errorf("test %d: headers mismatch: %v", i, err)
```

```
}
}
}

// Tests that block contents can be retrieved from a remote chain based on their hashes.
func TestGetBlockBodiesLes1(t *testing.T) { testGetBlockBodies(t, 1) }

func testGetBlockBodies(t *testing.T, protocol int) {
db, _ := ethdb.NewMemDatabase()
pm := newTestProtocolManagerMust(t, false, downloader.MaxBlockFetch+15, nil, nil, nil, db)
bc := pm.blockchain.(*core.BlockChain)
peer, _ := newTestPeer(t, "peer", protocol, pm, true)
defer peer.close()

// Create a batch of tests for various scenarios
limit := MaxBodyFetch
tests := []struct {
random    int          // Number of blocks to fetch randomly from the chain
explicit  []common.Hash // Explicitly requested blocks
available []bool        // Availability of explicitly requested blocks
expected  int           // Total number of existing blocks to expect
}{
{1, nil, nil, 1},         // A single random block should be retrievable
{10, nil, nil, 10},       // Multiple random blocks should be retrievable
{limit, nil, nil, limit}, // The maximum possible blocks should be retrievable
//{limit + 1, nil, nil, limit},                 // No more than the possible block count should be
returned
{0, []common.Hash{bc.Genesis().Hash()}, []bool{true}, 1},      // The genesis block should be
retrievable
{0, []common.Hash{bc.CurrentBlock().Hash()}, []bool{true}, 1}, // The chains head block should be
retrievable
{0, []common.Hash{{}}, []bool{false}, 0},                     // A non existent block should not be returned

// Existing and non-existing blocks interleaved should not cause problems
{0, []common.Hash{
{},
bc.GetBlockByNumber(1).Hash(),
{},
bc.GetBlockByNumber(10).Hash(),
{},
bc.GetBlockByNumber(100).Hash(),
{},
```

```
}, []bool{false, true, false, true, false, true, false}, 3},
}
// Run each of the tests and verify the results against the chain
var reqID uint64
for i, tt := range tests {
// Collect the hashes to request, and the response to expect
hashes, seen := []common.Hash{}, make(map[int64]bool)
bodies := []*types.Body{}

for j := 0; j < tt.random; j++ {
for {
num := rand.Int63n(int64(bc.CurrentBlock().NumberU64()))
if !seen[num] {
seen[num] = true

block := bc.GetBlockByNumber(uint64(num))
hashes = append(hashes, block.Hash())
if len(bodies) < tt.expected {
bodies = append(bodies, &types.Body{Transactions: block.Transactions(), Uncles:
block.Uncles()})
}
break
}
}
}
for j, hash := range tt.explicit {
hashes = append(hashes, hash)
if tt.available[j] && len(bodies) < tt.expected {
block := bc.GetBlockByHash(hash)
bodies = append(bodies, &types.Body{Transactions: block.Transactions(), Uncles:
block.Uncles()})
}
}
reqID++
// Send the hash request and verify the response
cost := peer.GetRequestCost(GetBlockBodiesMsg, len(hashes))
sendRequest(peer.app, GetBlockBodiesMsg, reqID, cost, hashes)
if err := expectResponse(peer.app, BlockBodiesMsg, reqID, testBufLimit, bodies); err != nil {
t.Errorf("test %d: bodies mismatch: %v", i, err)
}
}
}
```

```
// Tests that the contract codes can be retrieved based on account addresses.
func TestGetCodeLes1(t *testing.T) { testGetCode(t, 1) }

func testGetCode(t *testing.T, protocol int) {
// Assemble the test environment
db, _ := ethdb.NewMemDatabase()
pm := newTestProtocolManagerMust(t, false, 4, testChainGen, nil, nil, db)
bc := pm.blockchain.(*core.BlockChain)
peer, _ := newTestPeer(t, "peer", protocol, pm, true)
defer peer.close()

var codereqs []*CodeReq
var codes [][]byte

for i := uint64(0); i <= bc.CurrentBlock().NumberU64(); i++ {
header := bc.GetHeaderByNumber(i)
req := &CodeReq{
BHash:  header.Hash(),
AccKey: crypto.Keccak256(testContractAddr[:]),
}
codereqs = append(codereqs, req)
if i >= testContractDeployed {
codes = append(codes, testContractCodeDeployed)
}
}

cost := peer.GetRequestCost(GetCodeMsg, len(codereqs))
sendRequest(peer.app, GetCodeMsg, 42, cost, codereqs)
if err := expectResponse(peer.app, CodeMsg, 42, testBufLimit, codes); err != nil {
t.Errorf("codes mismatch: %v", err)
}
}

// Tests that the transaction receipts can be retrieved based on hashes.
func TestGetReceiptLes1(t *testing.T) { testGetReceipt(t, 1) }

func testGetReceipt(t *testing.T, protocol int) {
// Assemble the test environment
db, _ := ethdb.NewMemDatabase()
pm := newTestProtocolManagerMust(t, false, 4, testChainGen, nil, nil, db)
bc := pm.blockchain.(*core.BlockChain)
```

```
peer, _ := newTestPeer(t, "peer", protocol, pm, true)
defer peer.close()

// Collect the hashes to request, and the response to expect
hashes, receipts := []common.Hash{}, []types.Receipts{}
for i := uint64(0); i <= bc.CurrentBlock().NumberU64(); i++ {
block := bc.GetBlockByNumber(i)

hashes = append(hashes, block.Hash())
receipts = append(receipts, core.GetBlockReceipts(db, block.Hash(), block.NumberU64()))
}
// Send the hash request and verify the response
cost := peer.GetRequestCost(GetReceiptsMsg, len(hashes))
sendRequest(peer.app, GetReceiptsMsg, 42, cost, hashes)
if err := expectResponse(peer.app, ReceiptsMsg, 42, testBufLimit, receipts); err != nil {
t.Errorf("receipts mismatch: %v", err)
}
}

// Tests that trie merkle proofs can be retrieved
func TestGetProofsLes1(t *testing.T) { testGetReceipt(t, 1) }

func testGetProofs(t *testing.T, protocol int) {
// Assemble the test environment
db, _ := ethdb.NewMemDatabase()
pm := newTestProtocolManagerMust(t, false, 4, testChainGen, nil, nil, db)
bc := pm.blockchain.(*core.BlockChain)
peer, _ := newTestPeer(t, "peer", protocol, pm, true)
defer peer.close()

var proofreqs []ProofReq
var proofs [][]rlp.RawValue

accounts := []common.Address{testBankAddress, acc1Addr, acc2Addr, {}}
for i := uint64(0); i <= bc.CurrentBlock().NumberU64(); i++ {
header := bc.GetHeaderByNumber(i)
root := header.Root
trie, _ := trie.New(root, db)

for _, acc := range accounts {
req := ProofReq{
BHash: header.Hash(),
```

```
Key:   acc[:],
}
proofreqs = append(proofreqs, req)

proof := trie.Prove(crypto.Keccak256(acc[:]))
proofs = append(proofs, proof)
}
}
// Send the proof request and verify the response
cost := peer.GetRequestCost(GetProofsMsg, len(proofreqs))
sendRequest(peer.app, GetProofsMsg, 42, cost, proofreqs)
if err := expectResponse(peer.app, ProofsMsg, 42, testBufLimit, proofs); err != nil {
t.Errorf("proofs mismatch: %v", err)
}
}

87:F:\git\coin\ethereum\go-ethereum\les\helper_test.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// This file contains some shares testing functionality, common to  multiple
// different files and modules being tested.

package les

import (
"crypto/ecdsa"
"crypto/rand"
"math/big"
"sync"
"testing"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/consensus/ethash"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/core/vm"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/event"
"github.com/ethereum/go-ethereum/les/flowcontrol"
"github.com/ethereum/go-ethereum/light"
"github.com/ethereum/go-ethereum/p2p"
```

```
"github.com/ethereum/go-ethereum/p2p/discover"
"github.com/ethereum/go-ethereum/params"
)

var (
testBankKey, _  =
crypto.HexToECDSA("b71c71a67e1177ad4e901695e1b4b9ee17ae16c6668d313eac2f96dbcda3f
291")
testBankAddress = crypto.PubkeyToAddress(testBankKey.PublicKey)
testBankFunds   = big.NewInt(1000000)

acc1Key, _ =
crypto.HexToECDSA("8a1f9a8f95be41cd7ccb6168179afb4504aefe388d1e14474d32c45c72ce7b
7a")
acc2Key, _ =
crypto.HexToECDSA("49a7b37aa6f6645917e7b807e9d1c00d4fa71f18343b0d4122a4d2df64dd6f
ee")
acc1Addr   = crypto.PubkeyToAddress(acc1Key.PublicKey)
acc2Addr   = crypto.PubkeyToAddress(acc2Key.PublicKey)

testContractCode        =
common.Hex2Bytes("606060405260cc8060106000396000f360606040526000357c01000000000
000000000000000000000000000000000000000000000000000009004806360cd2685146041578063c16
431b914606b57603f565b005b6055600480803590602001909190505060a9565b6040518082815
260200191505060405180910390f35b608860048080359060200190919080359060200190919050
50608a565b005b806000600050836064811015600257909001600005b50819055505b5050565b6
000600060005082606481101560025790900160005b5054905060c7565b91905056")
testContractAddr         common.Address
testContractCodeDeployed = testContractCode[16:]
testContractDeployed     = uint64(2)

testBufLimit = uint64(100)

bigTxGas = new(big.Int).SetUint64(params.TxGas)
)

/*
contract test {

    uint256[100] data;

    function Put(uint256 addr, uint256 value) {
```

```
        data[addr] = value;
    }

    function Get(uint256 addr) constant returns (uint256 value) {
        return data[addr];
    }
}
*/

func testChainGen(i int, block *core.BlockGen) {
signer := types.HomesteadSigner{}

switch i {
case 0:
// In block 1, the test bank sends account #1 some ether.
tx, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBankAddress), acc1Addr,
big.NewInt(10000), bigTxGas, nil, nil), signer, testBankKey)
block.AddTx(tx)
case 1:
// In block 2, the test bank sends some more ether to account #1.
// acc1Addr passes it on to account #2.
// acc1Addr creates a test contract.
tx1, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBankAddress), acc1Addr,
big.NewInt(1000), bigTxGas, nil, nil), signer, testBankKey)
nonce := block.TxNonce(acc1Addr)
tx2, _ := types.SignTx(types.NewTransaction(nonce, acc2Addr, big.NewInt(1000), bigTxGas, nil,
nil), signer, acc1Key)
nonce++
tx3, _ := types.SignTx(types.NewContractCreation(nonce, big.NewInt(0), big.NewInt(200000),
big.NewInt(0), testContractCode), signer, acc1Key)
testContractAddr = crypto.CreateAddress(acc1Addr, nonce)
block.AddTx(tx1)
block.AddTx(tx2)
block.AddTx(tx3)
case 2:
// Block 3 is empty but was mined by account #2.
block.SetCoinbase(acc2Addr)
block.SetExtra([]byte("yeehaw"))
data :=
common.Hex2Bytes("C16431B900000000000000000000000000000000000000000000000000000000
0000000000010000000000000000000000000000000000000000000000000000000000000001")
tx, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBankAddress), testContractAddr,
```

```
big.NewInt(0), big.NewInt(100000), nil, data), signer, testBankKey)
block.AddTx(tx)
case 3:
// Block 4 includes blocks 2 and 3 as uncle headers (with modified extra data).
b2 := block.PrevBlock(1).Header()
b2.Extra = []byte("foo")
block.AddUncle(b2)
b3 := block.PrevBlock(2).Header()
b3.Extra = []byte("foo")
block.AddUncle(b3)
data :=
common.Hex2Bytes("C16431B90000000000000000000000000000000000000000000000000000
0000000000020000000000000000000000000000000000000000000000000000000000000002")
tx, _ := types.SignTx(types.NewTransaction(block.TxNonce(testBankAddress), testContractAddr,
big.NewInt(0), big.NewInt(100000), nil, data), signer, testBankKey)
block.AddTx(tx)
}
}

func testRCL() RequestCostList {
cl := make(RequestCostList, len(reqList))
for i, code := range reqList {
cl[i].MsgCode = code
cl[i].BaseCost = 0
cl[i].ReqCost = 0
}
return cl
}

// newTestProtocolManager creates a new protocol manager for testing purposes,
// with the given number of blocks already known, and potential notification
// channels for different events.
func newTestProtocolManager(lightSync bool, blocks int, generator func(int, *core.BlockGen),
peers *peerSet, odr *LesOdr, db ethdb.Database) (*ProtocolManager, error) {
var (
evmux  = new(event.TypeMux)
engine = ethash.NewFaker()
gspec  = core.Genesis{
Config: params.TestChainConfig,
Alloc:  core.GenesisAlloc{testBankAddress: {Balance: testBankFunds}},
}
genesis = gspec.MustCommit(db)
```

```
chain       BlockChain
)
if peers == nil {
peers = newPeerSet()
}

if lightSync {
chain, _ = light.NewLightChain(odr, gspec.Config, engine, evmux)
} else {
blockchain, _ := core.NewBlockChain(db, gspec.Config, engine, evmux, vm.Config{})
gchain, _ := core.GenerateChain(gspec.Config, genesis, db, blocks, generator)
if _, err := blockchain.InsertChain(gchain); err != nil {
panic(err)
}
chain = blockchain
}

pm, err := NewProtocolManager(gspec.Config, lightSync, NetworkId, evmux, engine, peers, chain,
nil, db, odr, nil, make(chan struct{}), new(sync.WaitGroup))
if err != nil {
return nil, err
}
if !lightSync {
srv := &LesServer{protocolManager: pm}
pm.server = srv

srv.defParams = &flowcontrol.ServerParams{
BufLimit:    testBufLimit,
MinRecharge: 1,
}

srv.fcManager = flowcontrol.NewClientManager(50, 10, 1000000000)
srv.fcCostStats = newCostStats(nil)
}
pm.Start()
return pm, nil
}

// newTestProtocolManagerMust creates a new protocol manager for testing purposes,
// with the given number of blocks already known, and potential notification
// channels for different events. In case of an error, the constructor force-
// fails the test.
```

```
func newTestProtocolManagerMust(t *testing.T, lightSync bool, blocks int, generator func(int,
*core.BlockGen), peers *peerSet, odr *LesOdr, db ethdb.Database) *ProtocolManager {
pm, err := newTestProtocolManager(lightSync, blocks, generator, peers, odr, db)
if err != nil {
t.Fatalf("Failed to create protocol manager: %v", err)
}
return pm
}

// testTxPool is a fake, helper transaction pool for testing purposes
type testTxPool struct {
pool  []*types.Transaction        // Collection of all transactions
added chan<- []*types.Transaction // Notification channel for new transactions

lock sync.RWMutex // Protects the transaction pool
}

// AddTransactions appends a batch of transactions to the pool, and notifies any
// listeners if the addition channel is non nil
func (p *testTxPool) AddBatch(txs []*types.Transaction) {
p.lock.Lock()
defer p.lock.Unlock()

p.pool = append(p.pool, txs...)
if p.added != nil {
p.added <- txs
}
}

// GetTransactions returns all the transactions known to the pool
func (p *testTxPool) GetTransactions() types.Transactions {
p.lock.RLock()
defer p.lock.RUnlock()

txs := make([]*types.Transaction, len(p.pool))
copy(txs, p.pool)

return txs
}

// newTestTransaction create a new dummy transaction.
func newTestTransaction(from *ecdsa.PrivateKey, nonce uint64, datasize int) *types.Transaction {
```

```
tx := types.NewTransaction(nonce, common.Address{}, big.NewInt(0), big.NewInt(100000),
big.NewInt(0), make([]byte, datasize))
tx, _ = types.SignTx(tx, types.HomesteadSigner{}, from)

return tx
}

// testPeer is a simulated peer to allow testing direct network calls.
type testPeer struct {
net p2p.MsgReadWriter // Network layer reader/writer to simulate remote messaging
app *p2p.MsgPipeRW    // Application layer reader/writer to simulate the local side
*peer
}

// newTestPeer creates a new peer registered at the given protocol manager.
func newTestPeer(t *testing.T, name string, version int, pm *ProtocolManager, shake bool)
(*testPeer, <-chan error) {
// Create a message pipe to communicate through
app, net := p2p.MsgPipe()

// Generate a random id and create the peer
var id discover.NodeID
rand.Read(id[:])

peer := pm.newPeer(version, NetworkId, p2p.NewPeer(id, name, nil), net)

// Start the peer on a new thread
errc := make(chan error, 1)
go func() {
select {
case pm.newPeerCh <- peer:
errc <- pm.handle(peer)
case <-pm.quitSync:
errc <- p2p.DiscQuitting
}
}()
tp := &testPeer{
app:  app,
net:  net,
peer: peer,
}
// Execute any implicitly requested handshakes and return
```

```
if shake {
td, head, genesis := pm.blockchain.Status()
headNum := pm.blockchain.CurrentHeader().Number.Uint64()
tp.handshake(t, td, head, headNum, genesis)
}
return tp, errc
}

func newTestPeerPair(name string, version int, pm, pm2 *ProtocolManager) (*peer, <-chan error,
*peer, <-chan error) {
// Create a message pipe to communicate through
app, net := p2p.MsgPipe()

// Generate a random id and create the peer
var id discover.NodeID
rand.Read(id[:])

peer := pm.newPeer(version, NetworkId, p2p.NewPeer(id, name, nil), net)
peer2 := pm2.newPeer(version, NetworkId, p2p.NewPeer(id, name, nil), app)

// Start the peer on a new thread
errc := make(chan error, 1)
errc2 := make(chan error, 1)
go func() {
select {
case pm.newPeerCh <- peer:
errc <- pm.handle(peer)
case <-pm.quitSync:
errc <- p2p.DiscQuitting
}
}()
go func() {
select {
case pm2.newPeerCh <- peer2:
errc2 <- pm2.handle(peer2)
case <-pm2.quitSync:
errc2 <- p2p.DiscQuitting
}
}()
return peer, errc, peer2, errc2
}
```

```
// handshake simulates a trivial handshake that expects the same state from the
// remote side as we are simulating locally.
func (p *testPeer) handshake(t *testing.T, td *big.Int, head common.Hash, headNum uint64,
genesis common.Hash) {
var expList keyValueList
expList = expList.add("protocolVersion", uint64(p.version))
expList = expList.add("networkId", uint64(NetworkId))
expList = expList.add("headTd", td)
expList = expList.add("headHash", head)
expList = expList.add("headNum", headNum)
expList = expList.add("genesisHash", genesis)
sendList := make(keyValueList, len(expList))
copy(sendList, expList)
expList = expList.add("serveHeaders", nil)
expList = expList.add("serveChainSince", uint64(0))
expList = expList.add("serveStateSince", uint64(0))
expList = expList.add("txRelay", nil)
expList = expList.add("flowControl/BL", testBufLimit)
expList = expList.add("flowControl/MRR", uint64(1))
expList = expList.add("flowControl/MRC", testRCL())

if err := p2p.ExpectMsg(p.app, StatusMsg, expList); err != nil {
t.Fatalf("status recv: %v", err)
}
if err := p2p.Send(p.app, StatusMsg, sendList); err != nil {
t.Fatalf("status send: %v", err)
}

p.fcServerParams = &flowcontrol.ServerParams{
BufLimit:    testBufLimit,
MinRecharge: 1,
}
}

// close terminates the local side of the peer, notifying the remote protocol
// manager of termination.
func (p *testPeer) close() {
p.app.Close()
}

88:F:\git\coin\ethereum\go-ethereum\les\metrics.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.
```

```
package les

import (
"github.com/ethereum/go-ethereum/metrics"
"github.com/ethereum/go-ethereum/p2p"
)

var (
/*propTxnInPacketsMeter     = metrics.NewMeter("eth/prop/txns/in/packets")
propTxnInTrafficMeter     = metrics.NewMeter("eth/prop/txns/in/traffic")
propTxnOutPacketsMeter    = metrics.NewMeter("eth/prop/txns/out/packets")
propTxnOutTrafficMeter    = metrics.NewMeter("eth/prop/txns/out/traffic")
propHashInPacketsMeter    = metrics.NewMeter("eth/prop/hashes/in/packets")
propHashInTrafficMeter    = metrics.NewMeter("eth/prop/hashes/in/traffic")
propHashOutPacketsMeter   = metrics.NewMeter("eth/prop/hashes/out/packets")
propHashOutTrafficMeter   = metrics.NewMeter("eth/prop/hashes/out/traffic")
propBlockInPacketsMeter   = metrics.NewMeter("eth/prop/blocks/in/packets")
propBlockInTrafficMeter   = metrics.NewMeter("eth/prop/blocks/in/traffic")
propBlockOutPacketsMeter  = metrics.NewMeter("eth/prop/blocks/out/packets")
propBlockOutTrafficMeter  = metrics.NewMeter("eth/prop/blocks/out/traffic")
reqHashInPacketsMeter     = metrics.NewMeter("eth/req/hashes/in/packets")
reqHashInTrafficMeter     = metrics.NewMeter("eth/req/hashes/in/traffic")
reqHashOutPacketsMeter    = metrics.NewMeter("eth/req/hashes/out/packets")
reqHashOutTrafficMeter    = metrics.NewMeter("eth/req/hashes/out/traffic")
reqBlockInPacketsMeter    = metrics.NewMeter("eth/req/blocks/in/packets")
reqBlockInTrafficMeter    = metrics.NewMeter("eth/req/blocks/in/traffic")
reqBlockOutPacketsMeter   = metrics.NewMeter("eth/req/blocks/out/packets")
reqBlockOutTrafficMeter   = metrics.NewMeter("eth/req/blocks/out/traffic")
reqHeaderInPacketsMeter   = metrics.NewMeter("eth/req/headers/in/packets")
reqHeaderInTrafficMeter   = metrics.NewMeter("eth/req/headers/in/traffic")
reqHeaderOutPacketsMeter  = metrics.NewMeter("eth/req/headers/out/packets")
reqHeaderOutTrafficMeter  = metrics.NewMeter("eth/req/headers/out/traffic")
reqBodyInPacketsMeter     = metrics.NewMeter("eth/req/bodies/in/packets")
reqBodyInTrafficMeter     = metrics.NewMeter("eth/req/bodies/in/traffic")
reqBodyOutPacketsMeter    = metrics.NewMeter("eth/req/bodies/out/packets")
reqBodyOutTrafficMeter    = metrics.NewMeter("eth/req/bodies/out/traffic")
reqStateInPacketsMeter    = metrics.NewMeter("eth/req/states/in/packets")
reqStateInTrafficMeter    = metrics.NewMeter("eth/req/states/in/traffic")
reqStateOutPacketsMeter   = metrics.NewMeter("eth/req/states/out/packets")
reqStateOutTrafficMeter   = metrics.NewMeter("eth/req/states/out/traffic")
reqReceiptInPacketsMeter  = metrics.NewMeter("eth/req/receipts/in/packets")
```

```
reqReceiptInTrafficMeter  = metrics.NewMeter("eth/req/receipts/in/traffic")
reqReceiptOutPacketsMeter = metrics.NewMeter("eth/req/receipts/out/packets")
reqReceiptOutTrafficMeter = metrics.NewMeter("eth/req/receipts/out/traffic")*/
miscInPacketsMeter  = metrics.NewMeter("les/misc/in/packets")
miscInTrafficMeter  = metrics.NewMeter("les/misc/in/traffic")
miscOutPacketsMeter = metrics.NewMeter("les/misc/out/packets")
miscOutTrafficMeter = metrics.NewMeter("les/misc/out/traffic")
)

// meteredMsgReadWriter is a wrapper around a p2p.MsgReadWriter, capable of
// accumulating the above defined metrics based on the data stream contents.
type meteredMsgReadWriter struct {
p2p.MsgReadWriter     // Wrapped message stream to meter
version           int // Protocol version to select correct meters
}

// newMeteredMsgWriter wraps a p2p MsgReadWriter with metering support. If the
// metrics system is disabled, this function returns the original object.
func newMeteredMsgWriter(rw p2p.MsgReadWriter) p2p.MsgReadWriter {
if !metrics.Enabled {
return rw
}
return &meteredMsgReadWriter{MsgReadWriter: rw}
}

// Init sets the protocol version used by the stream to know which meters to
// increment in case of overlapping message ids between protocol versions.
func (rw *meteredMsgReadWriter) Init(version int) {
rw.version = version
}

func (rw *meteredMsgReadWriter) ReadMsg() (p2p.Msg, error) {
// Read the message and short circuit in case of an error
msg, err := rw.MsgReadWriter.ReadMsg()
if err != nil {
return msg, err
}
// Account for the data traffic
packets, traffic := miscInPacketsMeter, miscInTrafficMeter
packets.Mark(1)
traffic.Mark(int64(msg.Size))
```

```
return msg, err
}

func (rw *meteredMsgReadWriter) WriteMsg(msg p2p.Msg) error {
// Account for the data traffic
packets, traffic := miscOutPacketsMeter, miscOutTrafficMeter
packets.Mark(1)
traffic.Mark(int64(msg.Size))

// Send the packet to the p2p layer
return rw.MsgReadWriter.WriteMsg(msg)
}

89:F:\git\coin\ethereum\go-ethereum\les\odr.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package les

import (
"context"

"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/light"
"github.com/ethereum/go-ethereum/log"
)

// LesOdr implements light.OdrBackend
type LesOdr struct {
db        ethdb.Database
stop      chan struct{}
retriever *retrieveManager
}

func NewLesOdr(db ethdb.Database, retriever *retrieveManager) *LesOdr {
return &LesOdr{
db:        db,
retriever: retriever,
stop:      make(chan struct{}),
}
}

func (odr *LesOdr) Stop() {
```

```
close(odr.stop)
}

func (odr *LesOdr) Database() ethdb.Database {
return odr.db
}

const (
MsgBlockBodies = iota
MsgCode
MsgReceipts
MsgProofs
MsgHeaderProofs
)

// Msg encodes a LES message that delivers reply data for a request
type Msg struct {
MsgType int
ReqID   uint64
Obj     interface{}
}

// Retrieve tries to fetch an object from the LES network.
// If the network retrieval was successful, it stores the object in local db.
func (self *LesOdr) Retrieve(ctx context.Context, req light.OdrRequest) (err error) {
lreq := LesRequest(req)

reqID := genReqID()
rq := &distReq{
getCost: func(dp distPeer) uint64 {
return lreq.GetCost(dp.(*peer))
},
canSend: func(dp distPeer) bool {
p := dp.(*peer)
return lreq.CanSend(p)
},
request: func(dp distPeer) func() {
p := dp.(*peer)
cost := lreq.GetCost(p)
p.fcServer.QueueRequest(reqID, cost)
return func() { lreq.Request(reqID, p) }
},
```

```
}

if err = self.retriever.retrieve(ctx, reqID, rq, func(p distPeer, msg *Msg) error { return
lreq.Validate(self.db, msg) }); err == nil {
// retrieved from network, store in db
req.StoreResult(self.db)
} else {
log.Debug("Failed to retrieve data from network", "err", err)
}
return
}

90:F:\git\coin\ethereum\go-ethereum\les\odr_requests.go
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

// Package light implements on-demand retrieval capable state and chain objects
// for the Ethereum Light Client.
package les

import (
"encoding/binary"
"errors"
"fmt"

"github.com/ethereum/go-ethereum/common"
"github.com/ethereum/go-ethereum/core"
"github.com/ethereum/go-ethereum/core/types"
"github.com/ethereum/go-ethereum/crypto"
"github.com/ethereum/go-ethereum/ethdb"
"github.com/ethereum/go-ethereum/light"
"github.com/ethereum/go-ethereum/log"
"github.com/ethereum/go-ethereum/rlp"
"github.com/ethereum/go-ethereum/trie"
)

var (
errInvalidMessageType  = errors.New("invalid message type")
errMultipleEntries     = errors.New("multiple response entries")
errHeaderUnavailable   = errors.New("header unavailable")
errTxHashMismatch      = errors.New("transaction hash mismatch")
errUncleHashMismatch   = errors.New("uncle hash mismatch")
errReceiptHashMismatch = errors.New("receipt hash mismatch")
```

```
errDataHashMismatch    = errors.New("data hash mismatch")
errCHTHashMismatch     = errors.New("cht hash mismatch")
)

type LesOdrRequest interface {
GetCost(*peer) uint64
CanSend(*peer) bool
Request(uint64, *peer) error
Validate(ethdb.Database, *Msg) error
}

func LesRequest(req light.OdrRequest) LesOdrRequest {
switch r := req.(type) {
case *light.BlockRequest:
return (*BlockRequest)(r)
case *light.ReceiptsRequest:
return (*ReceiptsRequest)(r)
case *light.TrieRequest:
return (*TrieRequest)(r)
case *light.CodeRequest:
return (*CodeRequest)(r)
case *light.ChtRequest:
return (*ChtRequest)(r)
default:
return nil
}
}

// BlockRequest is the ODR request type for block bodies
type BlockRequest light.BlockRequest

// GetCost returns the cost of the given ODR request according to the serving
// peer's cost table (implementation of LesOdrRequest)
func (r *BlockRequest) GetCost(peer *peer) uint64 {
return peer.GetRequestCost(GetBlockBodiesMsg, 1)
}

// CanSend tells if a certain peer is suitable for serving the given request
func (r *BlockRequest) CanSend(peer *peer) bool {
return peer.HasBlock(r.Hash, r.Number)
}
```

```
// Request sends an ODR request to the LES network (implementation of LesOdrRequest)
func (r *BlockRequest) Request(reqID uint64, peer *peer) error {
peer.Log().Debug("Requesting block body", "hash", r.Hash)
return peer.RequestBodies(reqID, r.GetCost(peer), []common.Hash{r.Hash})
}

// Valid processes an ODR request reply message from the LES network
// returns true and stores results in memory if the message was a valid reply
// to the request (implementation of LesOdrRequest)
func (r *BlockRequest) Validate(db ethdb.Database, msg *Msg) error {
log.Debug("Validating block body", "hash", r.Hash)

// Ensure we have a correct message with a single block body
if msg.MsgType != MsgBlockBodies {
return errInvalidMessageType
}
bodies := msg.Obj.([]*types.Body)
if len(bodies) != 1 {
return errMultipleEntries
}
body := bodies[0]

// Retrieve our stored header and validate block content against it
header := core.GetHeader(db, r.Hash, r.Number)
if header == nil {
return errHeaderUnavailable
}
if header.TxHash != types.DeriveSha(types.Transactions(body.Transactions)) {
return errTxHashMismatch
}
if header.UncleHash != types.CalcUncleHash(body.Uncles) {
return errUncleHashMismatch
}
// Validations passed, encode and store RLP
data, err := rlp.EncodeToBytes(body)
if err != nil {
return err
}
r.Rlp = data
return nil
}
```

```
// ReceiptsRequest is the ODR request type for block receipts by block hash
type ReceiptsRequest light.ReceiptsRequest

// GetCost returns the cost of the given ODR request according to the serving
// peer's cost table (implementation of LesOdrRequest)
func (r *ReceiptsRequest) GetCost(peer *peer) uint64 {
return peer.GetRequestCost(GetReceiptsMsg, 1)
}

// CanSend tells if a certain peer is suitable for serving the given request
func (r *ReceiptsRequest) CanSend(peer *peer) bool {
return peer.HasBlock(r.Hash, r.Number)
}

// Request sends an ODR request to the LES network (implementation of LesOdrRequest)
func (r *ReceiptsRequest) Request(reqID uint64, peer *peer) error {
peer.Log().Debug("Requesting block receipts", "hash", r.Hash)
return peer.RequestReceipts(reqID, r.GetCost(peer), []common.Hash{r.Hash})
}

// Valid processes an ODR request reply message from the LES network
// returns true and stores results in memory if the message was a valid reply
// to the request (implementation of LesOdrRequest)
func (r *ReceiptsRequest) Validate(db ethdb.Database, msg *Msg) error {
log.Debug("Validating block receipts", "hash", r.Hash)

// Ensure we have a correct message with a single block receipt
if msg.MsgType != MsgReceipts {
return errInvalidMessageType
}
receipts := msg.Obj.([]types.Receipts)
if len(receipts) != 1 {
return errMultipleEntries
}
receipt := receipts[0]

// Retrieve our stored header and validate receipt content against it
header := core.GetHeader(db, r.Hash, r.Number)
if header == nil {
return errHeaderUnavailable
}
if header.ReceiptHash != types.DeriveSha(receipt) {
```

```
return errReceiptHashMismatch
}
// Validations passed, store and return
r.Receipts = receipt
return nil
}

type ProofReq struct {
BHash       common.Hash
AccKey, Key []byte
FromLevel   uint
}

// ODR request type for state/storage trie entries, see LesOdrRequest interface
type TrieRequest light.TrieRequest

// GetCost returns the cost of the given ODR request according to the serving
// peer's cost table (implementation of LesOdrRequest)
func (r *TrieRequest) GetCost(peer *peer) uint64 {
return peer.GetRequestCost(GetProofsMsg, 1)
}

// CanSend tells if a certain peer is suitable for serving the given request
func (r *TrieRequest) CanSend(peer *peer) bool {
return peer.HasBlock(r.Id.BlockHash, r.Id.BlockNumber)
}

// Request sends an ODR request to the LES network (implementation of LesOdrRequest)
func (r *TrieRequest) Request(reqID uint64, peer *peer) error {
peer.Log().Debug("Requesting trie proof", "root", r.Id.Root, "key", r.Key)
req := &ProofReq{
BHash:  r.Id.BlockHash,
AccKey: r.Id.AccKey,
Key:    r.Key,
}
return peer.RequestProofs(reqID, r.GetCost(peer), []*ProofReq{req})
}

// Valid processes an ODR request reply message from the LES network
// returns true and stores results in memory if the message was a valid reply
// to the request (implementation of LesOdrRequest)
func (r *TrieRequest) Validate(db ethdb.Database, msg *Msg) error {
```

```
log.Debug("Validating trie proof", "root", r.Id.Root, "key", r.Key)

// Ensure we have a correct message with a single proof
if msg.MsgType != MsgProofs {
return errInvalidMessageType
}
proofs := msg.Obj.([][]rlp.RawValue)
if len(proofs) != 1 {
return errMultipleEntries
}
// Verify the proof and store if checks out
if _, err := trie.VerifyProof(r.Id.Root, r.Key, proofs[0]); err != nil {
return fmt.Errorf("merkle proof verification failed: %v", err)
}
r.Proof = proofs[0]
return nil
}

type CodeReq struct {
BHash  common.Hash
AccKey []byte
}

// ODR request type for node data (used for retrieving contract code), see LesOdrRequest
interface
type CodeRequest light.CodeRequest

// GetCost returns the cost of the given ODR request according to the serving
// peer's cost table (implementation of LesOdrRequest)
func (r *CodeRequest) GetCost(peer *peer) uint64 {
return peer.GetRequestCost(GetCodeMsg, 1)
}

// CanSend tells if a certain peer is suitable for serving the given request
func (r *CodeRequest) CanSend(peer *peer) bool {
return peer.HasBlock(r.Id.BlockHash, r.Id.BlockNumber)
}

// Request sends an ODR request to the LES network (implementation of LesOdrRequest)
func (r *CodeRequest) Request(reqID uint64, peer *peer) error {
peer.Log().Debug("Requesting code data", "hash", r.Hash)
req := &CodeReq{
```

```
BHash:  r.Id.BlockHash,
AccKey: r.Id.AccKey,
}
return peer.RequestCode(reqID, r.GetCost(peer), []*CodeReq{req})
}

// Valid processes an ODR request reply message from the LES network
// returns true and stores results in memory if the message was a valid reply
// to the request (implementation of LesOdrRequest)
func (r *CodeRequest) Validate(db ethdb.Database, msg *Msg) error {
log.Debug("Validating code data", "hash", r.Hash)

// Ensure we have a correct message with a single code element
if msg.MsgType != MsgCode {
return errInvalidMessageType
}
reply := msg.Obj.([][]byte)
if len(reply) != 1 {
return errMultipleEntries
}
data := reply[0]

// Verify the data and store if checks out
if hash := crypto.Keccak256Hash(data); r.Hash != hash {
return errDataHashMismatch
}
r.Data = data
return nil
}

type ChtReq struct {
ChtNum, BlockNum, FromLevel uint64
}

type ChtResp struct {
Header *types.Header
Proof  []rlp.RawValue
}

// ODR request type for requesting headers by Canonical Hash Trie, see LesOdrRequest interface
type ChtRequest light.ChtRequest
```

```
// GetCost returns the cost of the given ODR request according to the serving
// peer's cost table (implementation of LesOdrRequest)
func (r *ChtRequest) GetCost(peer *peer) uint64 {
return peer.GetRequestCost(GetHeaderProofsMsg, 1)
}

// CanSend tells if a certain peer is suitable for serving the given request
func (r *ChtRequest) CanSend(peer *peer) bool {
peer.lock.RLock()
defer peer.lock.RUnlock()

return r.ChtNum <= (peer.headInfo.Number-light.ChtConfirmations)/light.ChtFrequency
}

// Request sends an ODR request to the LES network (implementation of LesOdrRequest)
func (r *ChtRequest) Request(reqID uint64, peer *peer) error {
peer.Log().Debug("Requesting CHT", "cht", r.ChtNum, "block", r.BlockNum)
req := &ChtReq{
ChtNum:   r.ChtNum,
BlockNum: r.BlockNum,
}
return peer.RequestHeaderProofs(reqID, r.GetCost(peer), []*ChtReq{req})
}

// Valid processes an ODR request reply message from the LES network
// returns true and stores results in memory if the message was a valid reply
// to the request (implementation of LesOdrRequest)
func (r *ChtRequest) Validate(db ethdb.Database, msg *Msg) error {
log.Debug("Validating CHT", "cht", r.ChtNum, "block", r.BlockNum)

// Ensure we have a correct message with a single proof element
if msg.MsgType != MsgHeaderProofs {
return errInvalidMessageType
}
proofs := msg.Obj.([]ChtResp)
if len(proofs) != 1 {
return errMultipleEntries
}
proof := proofs[0]

// Verify the CHT
var encNumber [8]byte
```

```
binary.BigEndian.PutUint64(encNumber[:], r.BlockNum)

value, err := trie.VerifyProof(r.ChtRoot, encNumber[:], proof.Proof)
if err != nil {
return err
}
var node light.ChtNode
if err := rlp.DecodeBytes(value, &node); err != nil {
return err
}
if node.Hash != proof.Header.Hash() {
return errCHTHashMismatch
}
// Verifications passed, store and return
r.Header = proof.Header
r.Proof = proof.Proof
r.Td = node.Td

return nil
}
```