ethereum / go-ethereum Public

Code Issues 299 Pull requests 79 Wiki Security 9 Insights

671094279e

go-ethereum / core / forkchoice.go / Jump to

MariusVanDerWijden all: core rework for the merge transition (#23761) … ✕ History

3 contributors

108 lines (99 sloc) | 4.13 KB

```go
// Copyright 2021 The go-ethereum Authors
// This file is part of the go-ethereum library.
//
// The go-ethereum library is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// The go-ethereum library is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with the go-ethereum library. If not, see <http://www.gnu.org/licenses/>.

package core

import (
	crand "crypto/rand"
	"errors"
	"math/big"
	mrand "math/rand"

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

// ChainReader defines a small collection of methods needed to access the local
// blockchain during header verification. It's implemented by both blockchain
// and lightchain.
type ChainReader interface {
	// Config retrieves the header chain's chain configuration.
	Config() *params.ChainConfig

	// GetTd returns the total difficulty of a local block.
	GetTd(common.Hash, uint64) *big.Int
}

// ForkChoice is the fork chooser based on the highest total difficulty of the
// chain(the fork choice used in the eth1) and the external fork choice (the fork
// choice used in the eth2). This main goal of this ForkChoice is not only for
// offering fork choice during the eth1/2 merge phase, but also keep the compatibility
// for all other proof-of-work networks.
type ForkChoice struct {
	chain ChainReader
	rand  *mrand.Rand

	// preserve is a helper function used in td fork choice.
	// Miners will prefer to choose the local mined block if the
	// local td is equal to the extern one. It can be nil for light
	// client
	preserve func(header *types.Header) bool
}

func NewForkChoice(chainReader ChainReader, preserve func(header *types.Header) bool) *ForkChoice
	// Seed a fast but crypto originating random generator
	seed, err := crand.Int(crand.Reader, big.NewInt(math.MaxInt64))
	if err != nil {
		log.Crit("Failed to initialize random seed", "err", err)
	}
	return &ForkChoice{
		chain:    chainReader,
		rand:     mrand.New(mrand.NewSource(seed.Int64())),
		preserve: preserve,
	}
}

// ReorgNeeded returns whether the reorg should be applied
// based on the given external header and local canonical chain.
// In the td mode, the new head is chosen if the corresponding
// total difficulty is higher. In the extern mode, the trusted
// header is always selected as the head.
func (f *ForkChoice) ReorgNeeded(current *types.Header, header *types.Header) (bool, error) {
	var (
		localTD  = f.chain.GetTd(current.Hash(), current.Number.Uint64())
		externTd = f.chain.GetTd(header.Hash(), header.Number.Uint64())
	)
	if localTD == nil || externTd == nil {
		return false, errors.New("missing td")
	}
	// Accept the new header as the chain head if the transition
	// is already triggered. We assume all the headers after the
	// transition come from the trusted consensus layer.
	if ttd := f.chain.Config().TerminalTotalDifficulty; ttd != nil && ttd.Cmp(externTd) <= 0 {
		return true, nil
	}
	// If the total difficulty is higher than our known, add it to the canonical chain
	// Second clause in the if statement reduces the vulnerability to selfish mining.
	// Please refer to http://www.cs.cornell.edu/~ie53/publications/btcProcFC.pdf
	reorg := externTd.Cmp(localTD) > 0
	if !reorg && externTd.Cmp(localTD) == 0 {
		number, headNumber := header.Number.Uint64(), current.Number.Uint64()
		if number < headNumber {
			reorg = true
		} else if number == headNumber {
			var currentPreserve, externPreserve bool
			if f.preserve != nil {
				currentPreserve, externPreserve = f.preserve(current), f.preserve(
			}
			reorg = !currentPreserve && (externPreserve || f.rand.Float64() < 0.5)
		}
	}
	return reorg, nil
}
```