Notes (in progress) about P2P Web

Rasmus Erik Voel Jensen

2017

Abstract

Random ramblings and notes during the P2P Web.

Presentation notes

Purpose:

- Infrastructure for no-server HTML5 apps => a decentralized trustless computer for the web

In short:

- Network topology: kademlia like, - address = hash of pubkey
- State/storage: each node stores a neighbourhood around its own address, saved in blockchain merkeltree
- Operations: changes to state are verifiable, and verified by nodes in neighbourhood
- Balance: nodes gets paid for doing tasks for the network, and can use this to buy tasks in the network. Also pay/payout for state blockchain.
- Tasks: stored, nodes assigned to tasks in deterministic random part of storage, proof-of-result stored, result stored, verification/value, balance updated.

Additional notes:

- WebPlatform: computations in webassembly. WebRTC as transport (thus modified kademlia). Crypto-algorithms from crypto.subtle.
- Neighbourhood size and amount of state per node - determined by node density (global minimum density / local density). Fixed amount of memory per node.
- Mutable references in blockchain (using balance to keep alive)
- Autonomous processes (using balance to keep alive)
- Not entire blockchain stored, only parts needed by the node

- Stake in computation tasks
- Balance/trade between processes
- Introduced 'errors' in blockchain, and bounties for finding/proving them.
- Binary/Quad merkel tree for proofs
- Pub/private-key derived from entropy source
- Task types: computation, storage, storage-transfer, find node with certain data
- Node trust / reliability proof via blockchain
- Block-tree rather than chain
- Computational task level of validation
- Result safety: added to state by any node in neighbourhood by proof of distance of computing-node to task.
- Computational task: computing time bound, and cost calculation.
- consensus algorithm: CRDT, additional data in timeinterval: after last block, before timed signature from other deterministic random node
- Tagged overlay network - opt-in part of infrastructure for tunable bandwidth requirements
- Network simulation (core optimised for low memory)
- Bandwidth optimised, - number of significant bits per node-id, stream compression, only send diffs etc.

Explore/ideas:

- Performance characteristics of current WebRTC implementations
- Performance effects of design choices for Kademlia-like algorithm on top of WebRTC (instead of UDP)
- Verifiable "computational" tasks, and economy based "computation".
- (Survey p2p overlay networks)
- WebRTC bootstrapping options (decentralised signalling server vs. actual node)
- Infrastructure deployment - bootstrap-code + load signed version of code from network, - partly test within network before full deploy.

Articles:

- delegate-verify-public-verifiable-computing-2012
- embracing-peer-next-door-kademlia-2008
- ethereum-yellowpaper-2014
- golem-whitepaper-2016
- handling-churn-dht-2004
- improving-lookup-kademlia-2010
- ipfs-2014
- kademlia-2002
- noninteractive-verifiable-computing-2010

- performance-chord-kademlia-churn-2014
- performance-evaluation-kademlia-2010
- pinoccchio-nearly-practical-verifiable-computing-2013
- s-kademlia-2007
- subsecond-lookup-kademlia-2011
- survey-simulators-overlay-networks-2017
- trustless-computing-what-not-how-2016
- webassembly-2017

Description of algorithm:

- nodes connected in kademlia-like structure
- regular state snapshot (blockchain merkel-dag)
  - divide-and-conqueor consensus algorithm, verifying credit updates in neighbourhood.
  - each node stores the state of a neighbourhood around its own address, as well as the path to the root. The neighbourhood size is fixed for all, ensuring good redundancy of data for
- content of state
  - list of entities(nodes)
    - id
    - balance/credits (updated by work, tasks, cost of staying in blockchain, and transfers)
    - state (+ proof)
    - tasks - scheduled for execution - wager
    - result of previous scheduled task
    - work
      - stake
      - result
      - proof-of-work
    - state
- verifiable tasks
- task types
  - computation
  - storage
    - data
    - key/value
  - random verifications (of proof-of-stake tasks)
  - blockchain verification
- entities
  - nodes
  - nodes with stake
  - accounts (pub-key)
  - autonomous
- computational process

  - task gets stored in blockchain
  - task gets assigned to a number of bcrandom nodes
  - task gets done, and proof-of-work gets stored in the blockchain
  - task result gets released
  - result+proof-of-work get validated + signed into blockchain
  - balance is updated

Design criteria

- low bandwidth
- low memory footprint (useful for large simulation, as well as embeded systems)
- low code footprint
- tagging of hosts
- connect to arbitrary host
- foundation for other p2p applications

Parts

Rough Roadmap:

start with simulator in C.

- network-abstraction dummy-implementation
- wasm C example communicate with network-abstraction
- simulate
- network abstraction websocket-only - localhost - node
- wasm-
- end-to-end test network abstraction api
- spin up nodes
- simple in-complete overlay network
- add webrtc-network-abstraction+websocket-client

Old notes

- webrtc+wss connection abstraction

- kademlie-like network (webrtc+wss)

- network simulator (for tests etc.)

- deterministic addresses

- overlay network - ability to connect to any
- rough design notes
- simple signalling server (in node and php)
- call signalling-server and webrtc from asm.js
- implement kademlia-like algorithm on top of signalling server
- authentification
- addresses are hash of public-key
- possible to generate public key from passphrase
- "blockchain" clock
- list of peers "clock"
- per node ledger
- computational tasks in "blockchain"

network abstraction

API

Connection id is a 16 bit unsigned integer.

Calls from network to overlay

- `connected(connection_id)`
- `disconnected(connection_id)`
- `message(connection_id, data)`

Calls from overlay to network

- `send(connection_id, data)`
- `begin_signalling(connection_id)`
- `receive_signalling(connection_id)`
- `disconnect(connection_id)`
- (`topology(connection_id) -> uint64`)

(data is written to a shared buffer)

Connection types:

- public websocket https endpoint (or http-localhost during local testing)
- webrtc p2p

simulator

address

initially implemented unsecure

- `typedef address uint256`
- `distance(a, b) -> float 32`
- `generateAddress(uint256 entropy_source, securityLevel) -> address`
- later === `sha256(public_key(generat_dsa_keypair(scrypt_or_similar_hardening(entr num_rounds))))`
- `randomRandomAddress() -> uint256`
- … verify address

The address should actually be the hash of a dsa public key derrived from the entropy source.

mochable with fast version during simulation

overlay-network

knowledge self:

- list of addresses of active connections
- tags per connection
- path-bits per connection per own-tag
- own tags

Path: nearest addresses with one bit b_i flipped.

Version 0.1 base infrastructure

- network abstraction (make-connection)
- browser
- node.js wss server
- load wasm-core over network
- docker image for wss-server + caddy

Notes

- page size

- typically 4K (getpagesize() is 4K on my linux, and that looks like common size via https://en.wikipedia.org/wiki/Page_%28computer_memory%29)
- webassembly 64K page size
- minimise memory usage (for ability to run large simulations).
- i.e. 64K per nodes => 100K nodes in memory simulation ~ 6G memory