ETHEREUM

Lightning fast light clients for the future of Ethereum

Zsolt Felföldi

zsfelfoldi@ethereum.org
github.com/zsfelfoldi

An incentivization model for asymmetrical P2P protocols

- originally meant for and first implemented in Geth light client
  - **most parts of the presented model are already implemented and working!**
- designed for fully decentralized networks
- **primarily targeted to use cases requiring a stable and reliable connection**
  - involves long term trust building
  - incentivizes predictable availability and consistently fast response times
- scalable consensus needs a scalable infrastructure
  - our backbone infrastructures are going to depend more and more on light protocols
  - ETH 2.0 beacon chain validators need guaranteed stable access to the "old chain"
- LES incentivization is designed to also serve as a PoC for our future network infrastructure
- this model is applicable to any decentralized service market infrastructure

Real clouds are decentralized

What do I expect from an ideal service market model?

- no central coordination or authority
- resource allocation and load balancing through market signals
- automatic market discovery and reputation management
  - humans have a limited capacity for storing reputation data (big names benefit from this)
- allows free operation whenever possible
- low infrastructure overhead
- capable of considering transaction costs
  - ideally workable with any payment tech including simple on-chain transactions
  - cheaper/faster payment channel solutions allow better performance

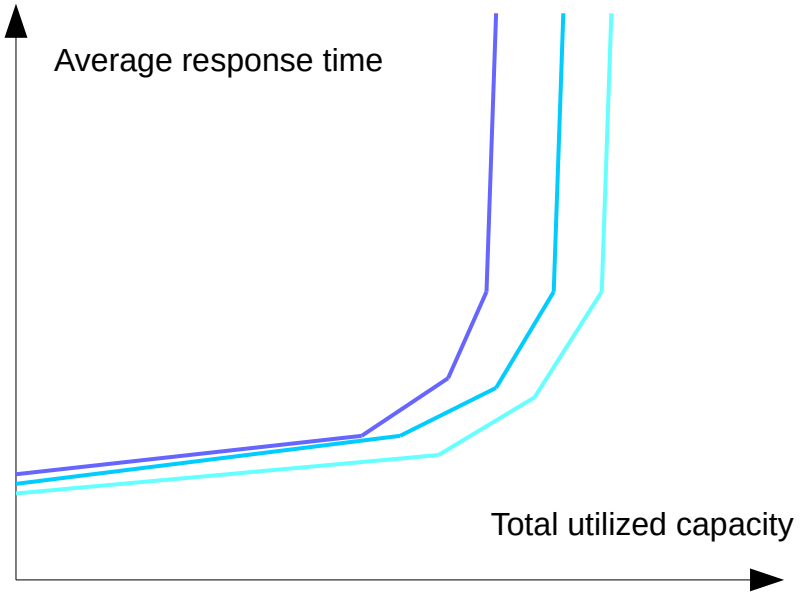
Time is money

- a market mechanism that works on a certain time scale might not work on another
  - demand is heavily fluctuating, supply is short-term inflexible
  - actual utility of service cannot be measured on a short time scale (depends on response times)
  - extreme fluctuation in the actual serving difficulty of requests
  - **gathering information to base decisions on needs time**
  - honest valuation of requests on a low price level is no indication of trustworthiness on a much higher price level
  - we have to avoid extreme price fluctuations
  - **price changes are meaningless on a short time scale!**
  - momentary price cannot be directly used as a feedback for load balancing
  - somewhat analogous with HFT issues: disconnection from fundamental value, destruction of honest market activity

Predictable supply, plannable demand

- **on a short time scale:**
  - how do servers predict and manage their own serving capacity to ensure quick responses?
  - how to implement/incentivize decentralized load balancing and avoid service delays and outages?
- **on a long time scale:**
  - how do clients find the best service for their money?
  - how should servers sell their service and plan ahead for future demand?

The server capacity model



- **Goal: utilize server capacity as much as possible but keep response times low**
  - servers have their internal request cost accounting
  - costs nominated in the server's own service token units
  - token costs are based on physical service bottlenecks (independent of market conditions)
  - real currency price of tokens can change according to the market
  - upper estimates for the cost of each request type are announced
    - **using a fixed price table would be very limiting**
    - servers can charge less than the announced maximum depending on actual conditions
  - capacity is defined as tokens spendable per millisecond

Client side flow control

- a guaranteed capacity and a request cost buffer is assigned to each client (leaky bucket model)
  - served request costs are removed from the buffer
  - the buffer is recharged at a guaranteed rate
  - server can recharge the buffer faster if actual conditions permit that
- actual buffer status is reported in each reply
- total capacity allowance for active connections is limited
- servers monitor their own state (response times, serving queue state) and adjust the total capacity allowance to keep their response times low

How does this model help decentralization?

- clients do load balancing based on server feedback
- lower capacity means smaller quantity but not lower quality (more or less guaranteed response times)
- a client can have multiple low capacity server connections instead of one high capacity one
  - reduced risk of losing all connections at the same time

Service Tokens

- servers can sell some of their future capacity
  - only a limited amount
  - it is the server's interest to keep its tokens spendable and their purchasing power more or less stable

Limited token supply and automatic expiration

- total outstanding token amount is limited to ensure they are spendable in a limited time frame
  - token price increases from a base price up to infinity before the total limit is reached
- tokens do expire, ensuring that they are being spent in a limited time frame
- buying tokens in advance is a commitment to a planned service usage and is rewarded by the server with a more reliable connection and with not exposing the client to further price fluctuations
- servers can simultaneously sell tokens with different expiration times at different prices as long as total supply is limited

Token sale mechanism

- servers can accept multiple currencies and payment methods
- transferred funds are not automatically converted to service tokens
  - once payment confirmed client can instantaneously buy tokens at any moment (including handshake)

Token pricing and free service

Connection priority

- if the total capacity allowance is reached the server has to prioritize between clients (there is eventually going to be some kind of competition)
- competing with commitment
  - priority is based on *(token_balance / capacity)*
  - commitment to use the service is rewarded by commitment to provide the service

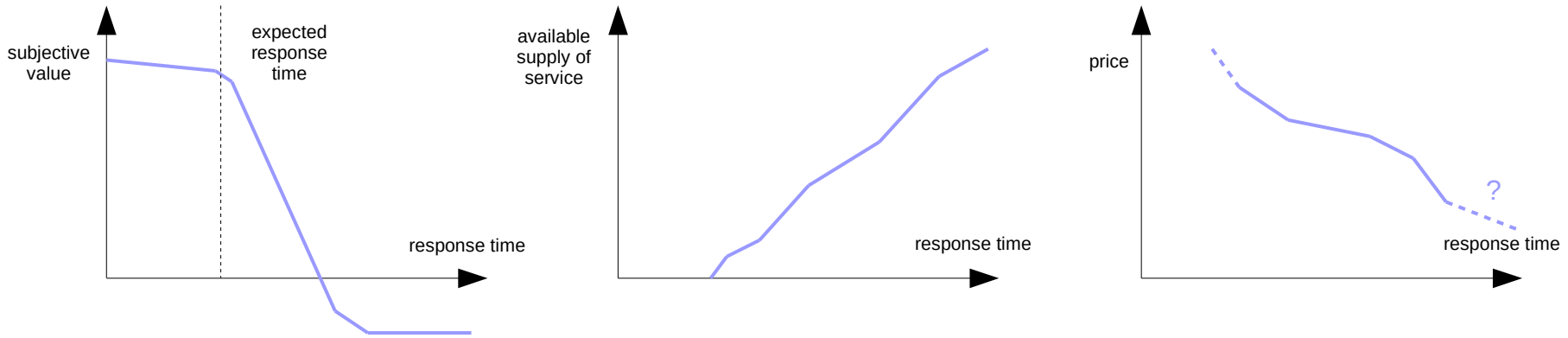
Base token price control and free service

- priority scheme handles occasional congestions but all tokens should be spendable eventually
- no priority is needed if total capacity allowance is not reached by connected clients with a positive balance
  - during these periods (“free periods”) free service is available (which is expected by new clients before they consider paying)
  - spendability of all tokens is also guaranteed by regular free periods
- servers should ensure regular free periods by controlling the base token price (slowly and gradually lower it in free periods and raise it in non-free periods)

Effects of expiration time

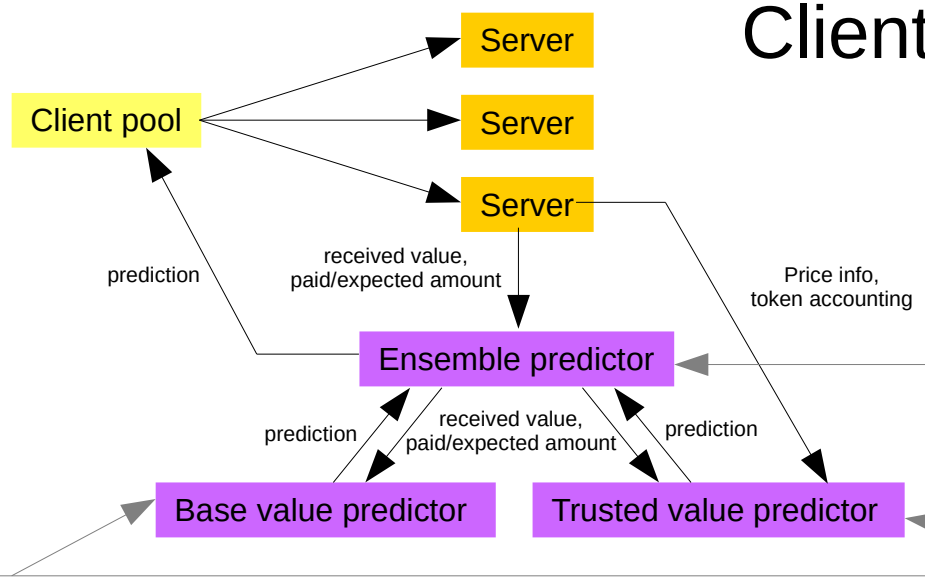
- this constant controls the time scale on which the client valuation and reputation algorithms can operate
- too quick expiration makes it impossible for clients to measure token stability (trust level stays low)
- too long expiration means it's hard to compete with old clients, takes a long time to build up trust levels

Subjective value measurement



- subjective value depends on
  - type and quantity of requests served
  - response times (compared to client expectations and market realities)
  - guaranteed capacity (if response times are consistent within the guaranteed limits)
- measurement module collects received amount vs. response time statistics for each request type and each server
  - best available price to value ratios can be calculated as a function of expected response time
  - client adjusts its expectations to market realities based on user settings
- client maps the available service market as a price vs. response time tradeoff
  - making quick responses valuable incentivizes actual geographical decentralization
  - convenient and easy to understand for users
  - allows different types of nodes to provide competitive service for different use cases
    - header syncing at startup is less sensitive to response times
    - light clients can use their unused bandwidth to serve block headers

Client strategy



- evaluate and update both predictors
- initially return the base value, then allow more and more deviation from it if the trusted predictor consistently gives better predictions

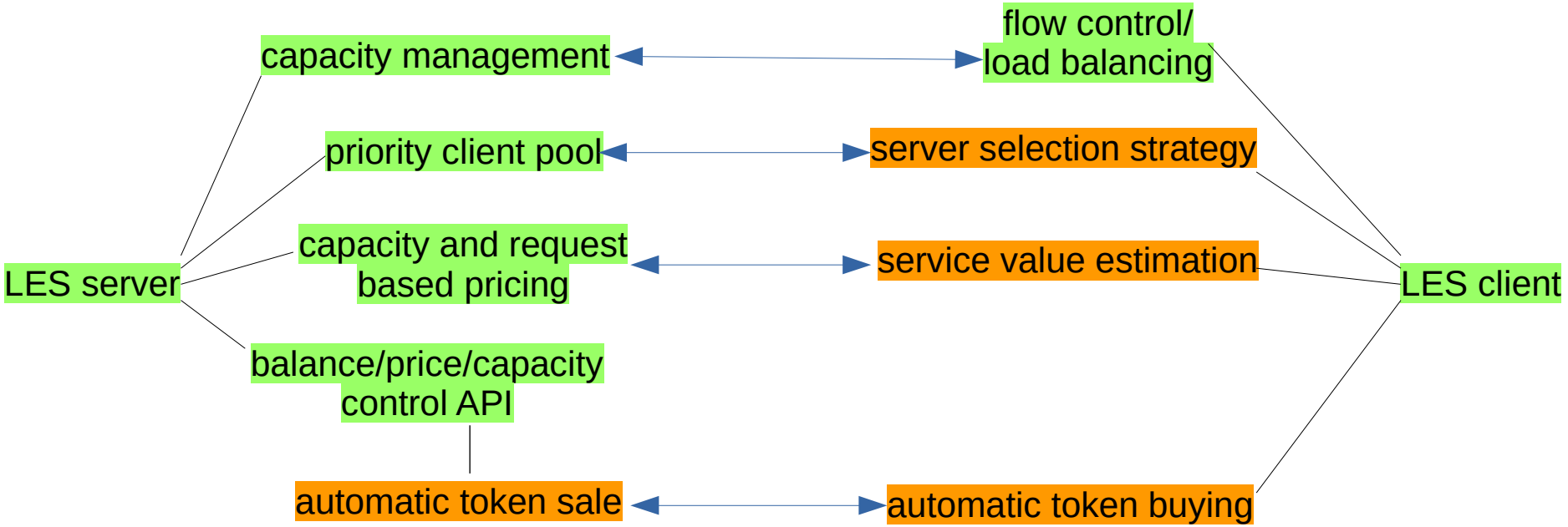
- based only on measured value and sent/expected payments
- cautious strategy and no quick price feedback
- expects free service first
- cannot be tricked into paying for nothing

- based on the pricing info announced by the server
- used only after enough value has been received
- limited deviation allowed from the base layer

Goal: best value to price ratio

- when a client has no more free connections it collects a number of payment requests
- a predictor tries to predict the subjective value received in case the sum is paid
- the offer with the best price to value ratio is accepted
- later (the next time the same server asks for payment) the total subjective value received since the last payment is fed back into the predictor

Current state of implementation



- green: implemented and tried
- orange: design work mostly done, implementation WIP
- **Phase 1 (ETA: now)**
  - server priority mechanism and API ready
  - you can already play with the API and grant priority to your friends or customers
- **Phase 2 (ETA: Dec 2019**)
  - token sale implemented, server side is ready, anyone can spin up their prioritized server and accept payments
  - clients capable of giving hints on service quality and quantity, manual payment whitelists still needed
  - paid service becomes available, market begins to form, lots of useful data can be collected for the final phase
- **Phase 3 (ETA: Q2-Q3 2020, research and refinement probably going on much longer)**
  - fully automated market discovery and payment strategy with easy client configuration

Thank you for your attention!

Zsolt Felföldi

zsfelfoldi@ethereum.org
github.com/zsfelfoldi