Design #1 - Promethean Nodes

With Promethean nodes I recommended to setup 3 types of nodes:

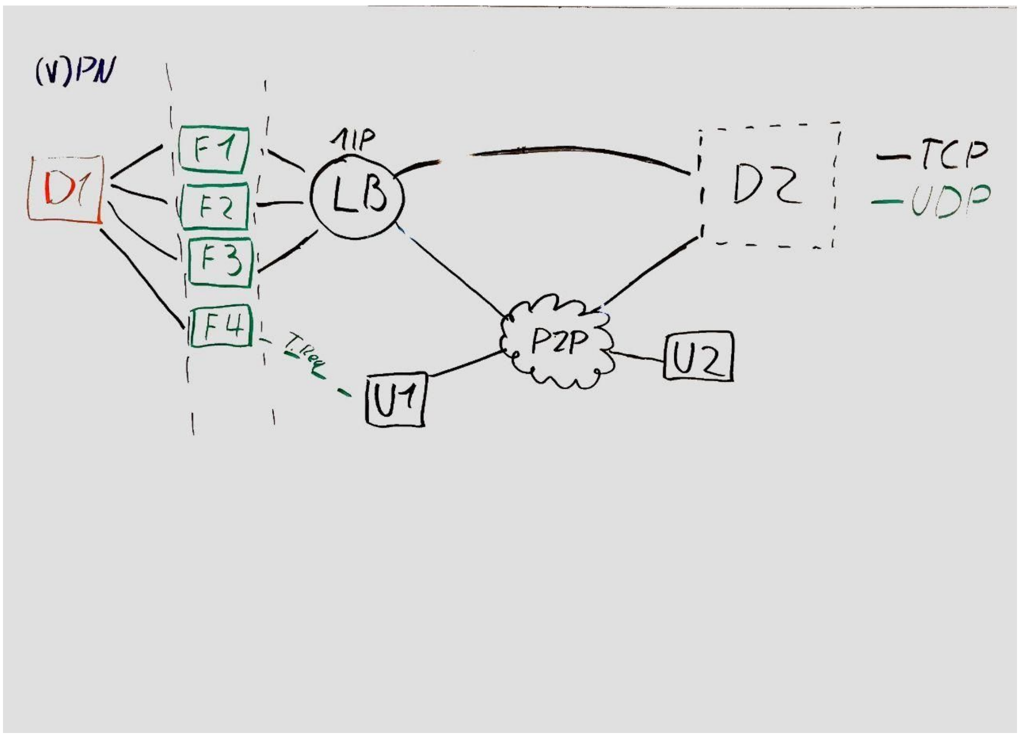
1. UDP acceptor nodes (which only accept transactions via UDP) - these are the only nodes that allow incoming UDP traffic. Multiple nodes can be put behind a network LB.
2. TCP nodes - these are behind a TCP terminating LB
3. Outbound only nodes - those are not actually announced and only make outbound connections (incoming ones are dropped); these help maintain a network connection if all other channels are attacked

Pros:

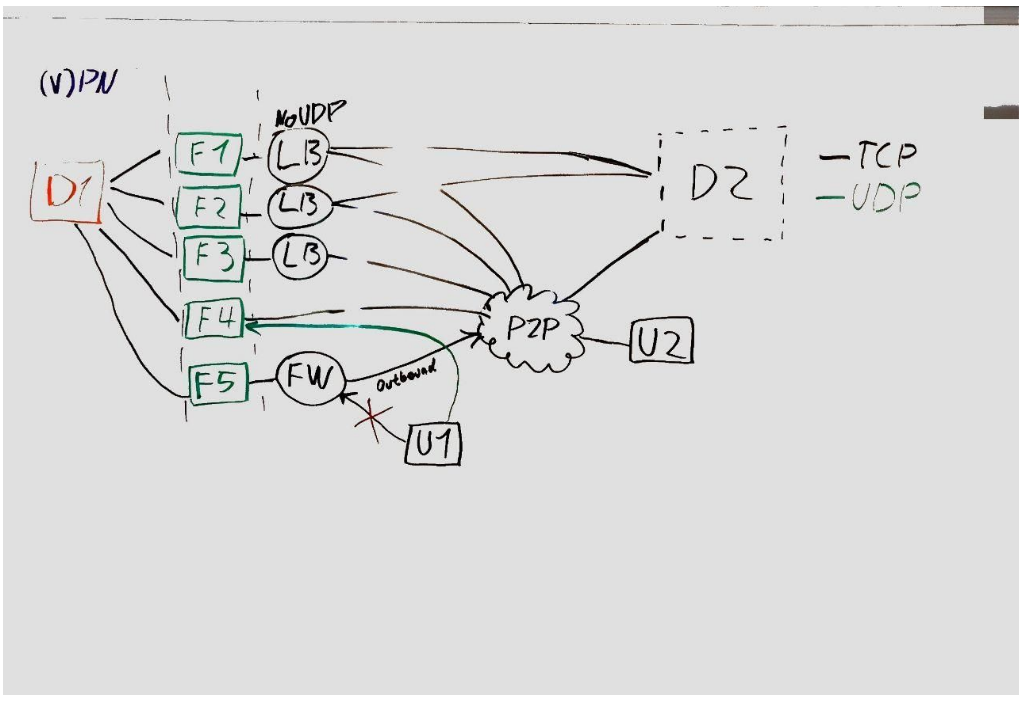
- We don't need special dynamic ACLs for delegate<>delegate point-to-point communication since all communication goes through Promethean nodes
- We can have different types of nodes whose overall diversity increases security
- We can expose single nodes to UDP (since this can not be handled by a global TCP LB)
- relatively portable (can simply be spun up in different cloud environments without very complex configurations)
- "Secure by default"
- no single vendor (since multiple IPs can be announced)

Cons:

- High implementation complexity



Load balancing between multiple Promethean nodes (F1-3) and a single node that accepts UDP (optionally behind a network LB). For outbound only connections consult the other graphic.



A single IP + LB for each node (if node IDs are relevant as in cosmos); F4 allows incoming UDP traffic; F5 has firewall rules set (ideally ACLs on the edge) that only allow outbound connections to be established.

Design #2 - LB DDoS Mitigation

Global LBs + single IP mean vendor lock-in *“SPoF”*

Global LBs can only handle TCP traffic. UDP traffic can only be handled regionally (e.g. with Google) which would require an additional IP for incoming UDP connections.

For direct delegate<>delegate connections ACLs would need to be set to allow direct traffic to the validator. That however makes it hard to also configure a TCP LB in cloud environments.

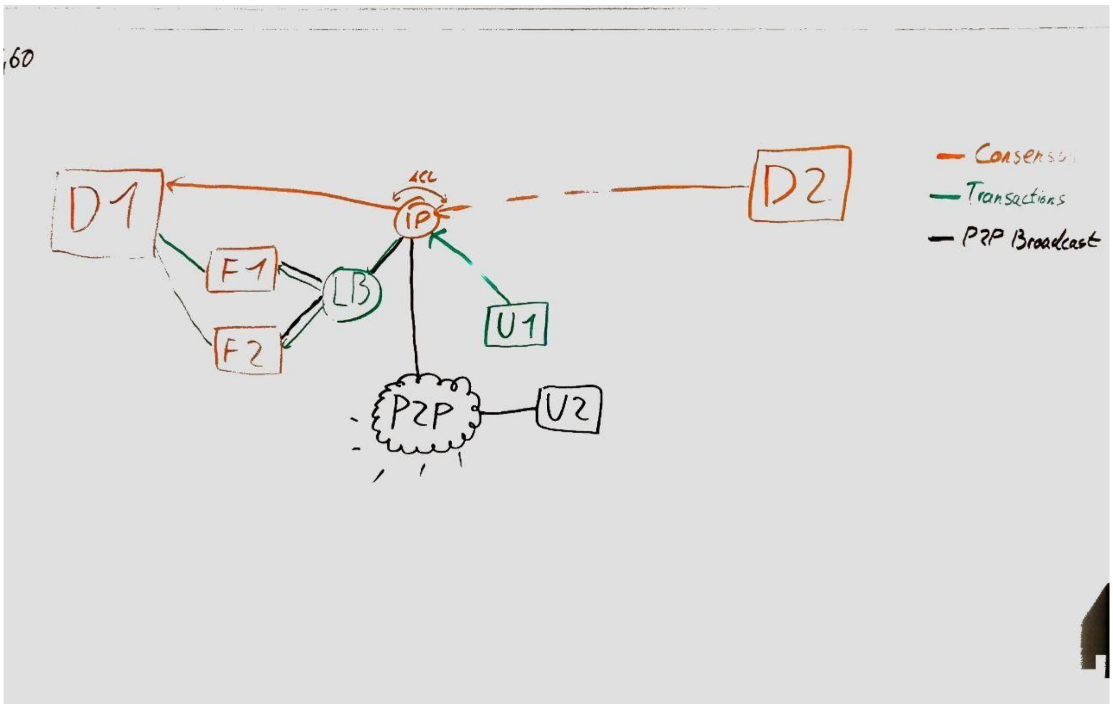
Pros:

- No added complexity in the core implementation

Cons:

- A single IP makes it hard to do both transaction ingest and delegate<>delegate consensus connections.
- A single IP partially locks you to one route/vendor/cloud provider
- Susceptible to BGP hijacking (single IP)
- No delegate<>delegate trust (if someone DDoSes you then it’s probably your competitor)

This setup locks a delegate to a single cloud validator and makes it hard to run the system on premise without very special hardware while it is almost impossible to realize such a setup in the cloud. You need a cloud provider with a highly sophisticated routing/firwall/offering like Google Cloud, AWS, Azure or Alibaba. Other clouds cannot be used since they don’t offer these advanced features.



Design decisions - UDP (WIP)

When do we want to use UDP ?

UDP is useful in environments where head of line blocking is a major problem. A good example for such a protocol is a consensus protocol where head of line blocking can prevent a delegate from catching up or participating in consensus.

UDP is also much more lightweight than TCP

Enhanced Promethean Arch

Based on https://docs.google.com/document/d/1k6V2aCaFW9xx7PlM93-61XvUPUvX8DK7gW-P1Ma8g0w/edit

Promethean Nodes

A Promethean node is a Light Client, that participates in the P2P network.
Its role is to verify incoming P2P messages and relay them to other nodes e.g. the Delegate.
The logic is extremely simplified to add almost no latency.
Ideally, the Promethean nodes hold no history and either sync state from other Full Nodes in the network (untrusted) or private FullNodes (trusted) if they require state at all.

Tx-Acceptor

The acceptor is an instance of a Promethean node that participates in the public P2P network. It accepts transactions via HTTP/2 and batch forwards them to the Delegate.
IPs of public TX-Acceptors are broadcasted via *key-adv*.

CS-Gateway

The CS-Gateway is an instance of a Promethean node that only accepts end-to-end connections from other delegates. This is enforced by only sharing the IP with only delegates over an end-to-end encrypted channel.
Since IPv4 is easy to scan we recommend using IPv6 as an address contains enough entropy to make scans unrealistic.

The IPs of CS-gateways are E2E exchanged via *priv-key-adv* messages.

CS-gateways rate-limit consensus traffic between delegates and conduct simple plausibility checks. Since these checks do not need to be state related CS-gateways “can” be completely stateless and only act as a L7 proxy.

Since the CS-gateways are not known to the public (or even behind a FW with specific ACLs) so any kind of DoS from the public network can be excluded.
However other delegates can still try to take the gateway down using L3 attacks. To mitigate this risk delegates are free to run multiple CS-gateways for different delegates and announce different ones to each delegate. That way an attack against the CS-gateway would only prevent the attacker from participating in P2P consensus communication with the target but not hurt it since there are still open connections to all other delegates using other CS-gateways depending on the setup.

Also depending on the required level of security a simple CS-Gateway (with only stateless checks like signature validation and Rate-limiting) could even be implemented using Serverless functions like AWS Lambda or Google Cloud Functions.
This could also be implemented as a first layer of security in front of a single stateful CS-Gateway that then checks the authenticity of the actual message content before relaying them to the Delegate node.

Messages

Type of message	**Protocol Type**
Consensus - Pre-prepare	HTTP Propagation + pptp
Consensus - Prepare	HTTP Point-to-Point
Consensus - Prepare reject	HTTP Point-to-Point
Consensus - Post-prepare proof	HTTP Propagation + pptp
Consensus - Commit	HTTP Point-to-Point
Consensus - Post-commit proof	HTTP Propagation + pptp
Request - Transaction request	HTTP ptp or HTTP Propagation
Request - Transaction reply	HTTP Propagation
Batched Request - batched transaction request	HTTP Propagation
Bootstrap - bootstrap request	HTTP Point-to-Point
Bootstrap - bootstrap response	HTTP Point-to-Point
Peer discovery - request	HTTP Point-to-Point
Peer discovery - reply	HTTP Point-to-Point
Peer discovery - handshake	HTTP Point-to-Point
Recall - request	HTTP Propagation
Blacklist	HTTP Propagation
Key adv	HTTP Propagation

pptp = This is the private point-to-point network between delegates. All consensus related messages are exchanged here. The Proofs, however, are broadcasted to the public network as well. That way the delegate does not have to listen to/receive consensus messages from the public P2P network thereby reducing the DDoS surface.

Key exchange for pptp

Direct exchange

see Diagram