Impact of Distributed Ledgers on Provider Networks

An Industry IoT Consortium Whitepaper

20220110

Authors

Dirk Trossen (Huawei Technologies), David Guzman (Huawei Technologies), Abhijeet Kelkar (GEOOWN Consulting), Xinxin Fan (IoTex), Mike McBride (Futurewei Technologies), Lei Zhang (iExec), Ulrich Graf (Huawei Technologies).

We've all heard of the myriad cryptocurrencies, such as Bitcoin and Ethereum, which rely on *distributed ledger technologies* (DLTs), such as blockchain, to provide decentralization and immutability. Industrial companies are using DLTs for everything from shipping-container tracking, vehicle identity and history, to energy trading and farm-to-store tracking.

But what about the *operational impact* of these peer-to-peer distributed ledgers on the service provider network, for example, in terms of network load? Do certain architectures work better than others? Conversely, are there any opportunities for service providers to leverage DLT's to provide additional services or improve on the network's overall performance, for example, managing routing state, user membership information or others? We analyze here how blockchains affect a service provider network.

Types of DLTs running in the network: There are essentially three types of DLTs deployed in today's networks: permissionless (internet-wide cryptocurrencies), permissioned (health, supply chain, government) and hybrid. They operate as an overlay using existing IP infrastructure, with DLT members, called peers,[1] communicating with other peers of the network. When the DLT is open to the public larger-scale DLT infrastructure is typically needed, including solo minor peers, full peers, light peers, mining pools and a variety of protocols such as Bitcoin, Ethereum and Stratum. When a private DLT is deployed, the amount of infrastructure needed to support the DLT system is typically less than a public DLT.

Potential problems: A lot of state information is maintained, and messages transmitted, between DLT peers. Transaction information is required to reach all other peers. Bootstrap nodes maintain IP addresses and port information of all miners. A new DLT peer needs to download routing information, and that needs regular update across all peers. Peers know nothing about other peers' capability to serve requests. Consequently, peers need to contact potential peers, wait for a successful connection, then they can enquire about the necessary capabilities. Peers will communicate if the capabilities match; disconnect if not. Peers may never reply to a connection establishment. Peers map sending of transactions onto single unicast communication.

This behavior leads to both *wasted* and *inefficient communication*. We differentiate between them because inefficiency costs the service provider, while wasted communication leads to expenditure without successful completion of the intended service communication. It therefore affects both service provider and the application using the DLT. This in turn may then lead to additional interoperability, scalability and data privacy challenges that need to be addressed.

Potential opportunities: As network operators address these problems, there is an opportunity to develop and offer new services such as a supply-chain integrity, DLT as a service, identity

[1] A peer may be a *client* that issues a transaction to a set of miners or a *miner* that exchanges information with other miners as a part of a transaction. Peers use the same overlay routing information, regardless of the role they have in the specific transaction (miner or client).

services, energy trading, smart contracts and integration into a variety of verticals. Some of these services will be made more secure by integrating blockchain into IP routing protocols.

1 Layering Architecture Overview

A DLT aims to power large-scale, decentralized applications and achieve good balance among decentralization, scalability and security. A typical DLT architecture comprises the following seven layers, as shown in Figure 1.

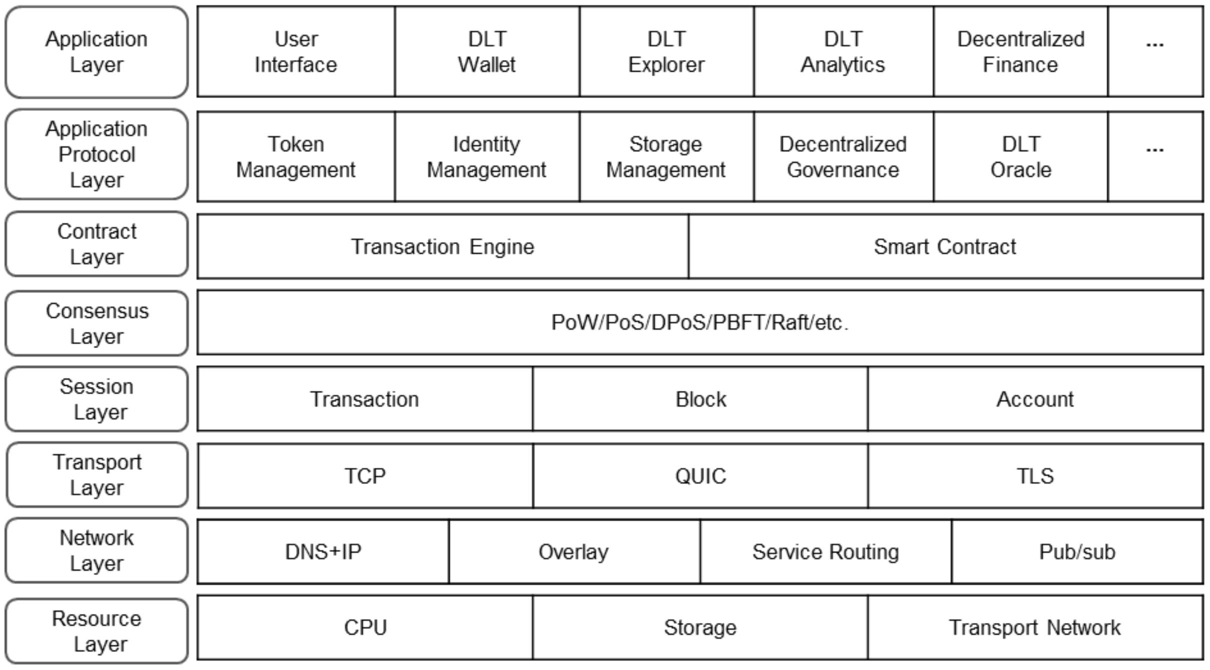

Figure 1. An overview of DLT system architecture.

Resource layer includes all the elements that provide CPU, storage and transport network to support DLT operations.

Network layer realizes the needed network technologies for discovering, connecting and communicating with each other in a secure manner, e.g., as a DNS+IP, overlay, service routing or publish/subscribe solution.

Transport layer realizes the secure end-to-end communication of session information.

Session layer records all transactions and blocks as well as maintains users' accounts.

Consensus layer implements a consensus protocol such as Proof of Work (PoW), Proof of Stake (PoS), Delegated Proof of Stake (DPoS), to reach an agreement among DLT peers regarding DLT transactions.

Contract layer serves as the on-chain computation engine that makes use of a transaction engine, such as a virtual machine, to implement smart contracts on DLTs.

Application protocol layer contain protocols implemented as a set of smart contracts on a DLT, which provide core DLT services such as token management, identity management, storage management, decentralized governance and DLT oracle.[2]

Application layer addresses effective interaction between a user and DLT by providing user-friendly interface, DLT wallet, DLT visualization dashboards such as explorer and analytics, decentralized finance and other DLT-based apps.

Peer-to-peer (P2P) networks disseminate system information and maintain decentralization of the entire system. This architecture allows participants to exchange digital assets on a global scale, without the need for centralized entities or intermediaries. As one of the fundamental components of DLTs, the performance of P2P networks has a significant effect on network service providers and DLT-empowered applications.

2 Details of DLT Network Impact

Characteristics of the DLT P2P network: The topological properties of DLT networks are described through its *network size* and the *node degree* (the number of peers that communicate with other peers for a given DLT transaction). Those characteristics affect the distributed consensus' ability to reach agreement in the face of failures and asynchrony. [1]

Extensive work has been done to measure blockchain networks. For instance, in [2] [3], Ethereum's topological properties are exposed, and [4] [5] address Bitcoin blockchain properties and networking state-of-the-art and challenges. Table 1 summarizes those key properties.

	Network Size	*Node Degree*	*Blockchain Size [GB]*
Bitcoin	~ 656191 [3]	Measured: 47 Max: 257 [3]	348.26 [4]
Ethereum	~508467 [2]	Measured: 117 [6] Max:1000	262.94 [5]

Table 1. DLT Characteristics.

[2] A DLT oracle is a service that provides the smart contract, realized by the DLT, with so-called *off-chain* information; that is information that is not stored as part of the blockchain itself.

[3] *https://www.blockchain.com/charts/n-unique-addresses* [Review: August 2021]

[4] *https://www.statista.com/statistics/647523/worldwide-bitcoin-blockchain-size/* [Review: August 2021]

[5] *https://blockchair.com/ethereum/charts/blockchain-size* [Review: August 2021]

DLT overlay routing entries: To ensure reachability towards any other miner in the overall DLT network, an *overlay network* is established that comprises of IP(v4/v6) end nodes over one or more provider networks. For this, *overlay routing entries* are maintained by each peer, realized as an overlay node in the DLT network, for communicating with other peers. Every overlay routing entry comprises the following elements:

- IPv4 address: 32 bits,
- Port: 2*16 bit (for UDP and TCP port) and
- ETH ID: 512-bit (to uniquely identify a DLT peer).

Given the network size identified in [2] (see Table 1), this results in a *maximum routing table size* of 36MB, totaling a maximum of 18TB (36MB times the total network size of 508K) of data being sent among the peers of the overall DLT overlay network since (part or all of) this table needs to be downloaded by every peer. The table also needs to be maintained upon changes, such as new peers, increasing those numbers further.

DLT interaction patterns: In a DLT network, a new peer must *discover* neighbors and peers with initial information provided through bootstrap nodes before being able to commit transactions or mine blocks. For this, a peer, regardless of its role (miner or client), needs to download an initial part of the overlay routing table. Then, a miner discovers suitable other miners with which to communicate, with a randomized set of first-hop miners, each of which return their next-hop miners to the initiating miner.

A client may then issue a *transaction*[6] into the DLT, again sent to a random set of miners among those that have been discovered using the initial overlay routing table, using the same randomness mechanism as for the initial peer discovery.

One of the first transactions for every miner is to synchronize to the latest *blockchain,*[7] using the neighbor miners discovered in the previous discovery step.

The *commit* transaction will lead to miners committing compute and storage resources for executing suitable computations in relation to the *smart contract*[8] that the system is realizing. Such resources may realize so-called *proofs,*[9] whose result will be committed to the miner's

[6] A *transaction* is defined through a command and command-specific parameters, where the execution of the command will yield a response to the invoking client.

[7] A *blockchain* represents the ledger information of transactions that were previously committed to the miners in the system.

[8] A *smart contract* is represented as a state machine of transactions, such as those related to trading cryptocurrencies. Achieving consensus in such a (distributed) state machine is represented in the (distributed) ledger over the smart contract transactions, where the ledger itself is represented as the blockchain.

[9] *Proofs* involve computational operations over transaction data provided by clients. Committing computational resources to those operations is key to the trustworthiness of the overall consensus achieved by the distributed miners.

neighbors, discovered in the initial step and using the latest blockchain information obtained in the initial blockchain synchronization.

Ultimately clients may read the latest blockchain as a confirmation from miners, again using the randomness methods for contacting a set of miners that may provide said blockchain data.

The above patterns of initial discovery and transaction commitment to the DLT overlay network are realized through a sequence of communication, as illustrated in Figure 2 and reverse-engineered in this case from Ethereum:

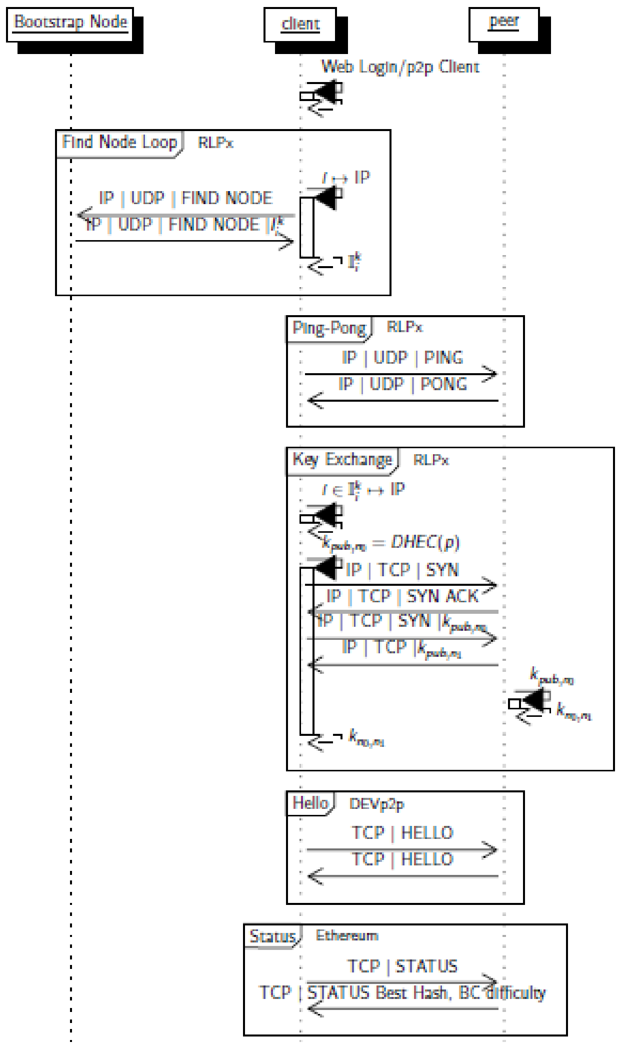

Figure 2. Ethereum case.

The first block in the figure deals with obtaining the initial list of peers to be contacted, for example, to initiate transactions or the discovery of neighboring miners. Usually, this initial list comprises only of a few–perhaps dozens–peers.

The second block deals with discovery of the contacted peer, determining whether the overlay may respond. Here *non-reachable peers* are filtered out, while the initial list of peers is being enriched towards the local *pool of peers* being used for transactions later on. A typical size of this local pool of peers is usually a few thousand (up to 10k), significantly less than the overall DLT network size.

To increase this pool size, peers may repeat the first and second block with repeated requests to the bootstrap node for another initial set of peers albeit at the expense of the communication costs for discovering this larger pool.

The third block establishes a secure transport connection, for example TLS1.2/1.3 mechanisms, for any miner previously not contacted, while the fourth block is used for those peers with which a transport connection already exists. The final block indicates the actual exchange of the command, such as synchronization, discovery and transaction, resulting in a response. The response may well be negative if constraining parameters, such as hash function to be used or DLT proof being needed, do not match.

Given the overlay nature of the DLT network, sending commands is done peer-to-peer by using the overlay routing information that was provided to them initially by the bootstrap node and enriched through the discovery step.

Furthermore, DLTs rely on *diffusion* of the command to a randomized *set of miners* with unicast responses from them being received at the initiating peer. This randomized set of miners is determined from the peer-local pool of peers, as determined through the discovery step, while the diffusion is controlled through the randomly chosen number *N* of miners to be contacted among those miners that have been successfully discovered.[10]

This number *N* is chosen for any new transaction and any peer attempting to issue a transaction to the DLT network. Any failed communication with a miner is compensated by determining another miner to replace the failed one. This process of trial-and-error is repeated until the initial chosen number *N* has been reached, which may lead to a significantly larger number of miners being contacted.[11]

The interaction patterns above exhibit the following aspects:

- Peers need to obtain initial knowledge of the DLT overlay network (via the bootstrap node), enriched by the discovery step to determine the pool of peers with which the peer may communicate in the future.
- The selection of miners, within the discovered pool of peers, to communicate with is randomized to provide protection against collusion of a set of miners, while the communication between a peer and other miners is realized through repeated unicast transfers.
- Miners may or may not be reachable when being contacted (through their IP address information as part of the overlay routing table). For instance, they may reside behind a firewall or be switched off. While the first step filters out those peers already in the discovery step, transient peers may change reachability even after initial positive discovery, leading to failed communication.
- Miners may or may not provide suitable information (or no information at all), despite being reachable. For instance, miners that have been requested to send the latest blockchain may not have the information available, and so respond negatively.

[10] The number of miners set to be used is determined for each transaction and is represented as the node degree presented previously. Each DLT platform employs proprietary algorithms for determining this node degree with platforms like Ethereum, for instance, selecting 117 random peers on average [3].

[11] Although the discovery step attempts to filter out those peers that are not responsive, further constraints may filter out those discovered peers later on, such as transient peers disappearing after the initial discovery or discovered peers responding to communication but not working on the smart contract that transaction relates to, therefore negatively responding to the transaction.

3 WASTED AND INEFFICIENT COMMUNICATION

The first two aspects above lead to *inefficient communication.*[12] First, the initial knowledge of the overlay network needs to be transmitted to every peer, with systems like Ethereum pushing on the order of a maximum of 30GB of data to any new member of the DLT, not accounting for any updates to this information. Second, transmitting the command to a set of miners for fulfilling the smart contract increases the required network capacity per transaction and therefore the cost of transaction.

Here, the randomization of the miner peer sets per transaction poses a significant challenge for using more efficient network technologies, such as IP multicast, since the peer-receiver sets for the transaction command change per transaction (and are different per client) and are likely different for each initiating peer. Removing those inefficiencies may lead to significant cost reductions in operating DLTs.

The last two aspects lead to *wasted communication*. Unlike inefficient communication that still results in a positive result from a service perspective, though inefficiently, wasted communication does not contribute positively to the overall outcome of the service, regardless of how (in)efficient it is. Such wastage affects the completion time and costs of running DLTs due to the incurred bandwidth usage and prolonged completion of the transaction when attempting to diffuse the transaction to the needed number *N* of miners, where *N* is determined as part of the DLT diffusion process, as explained before.

In the following, we attempt to quantify the waste occurring in a DLT system.

In [2] and [3], the underlying Ethereum topological properties were measured, reporting IP addressing issues in the overlay routing entries due to IP aliasing, proxies, firewalls etc. A surprising finding was the total wasted communications while executing an operation in the Ethereum network, synchronizing, transacting or mining. In a network of ~508467 peers, only ~73847 peers were active out of which ~10856 were successfully contacted; those represent *routable peers* from an overlay network perspective).

This results in *only 2.13% of all peers being useful*. The last facts impact strongly on the actual P2P communication since with high probability a number of peers equal to the node degree are contacted (Ethereum: 17, Bitcoin:117) out of which only 2.13% are efficient.

Ethereum synchronization experiment: To deepen our understanding of wasted communication, we conducted an experiment to discover miners in the Ethereum network to synchronize the entire blockchain (262.94 GB), i.e., executing the first two steps of our DLT interaction patterns.

[12] *Inefficient communication* is characterized by a successful response but inefficient realization of the communication leading up to it. For instance, using repeated unicast communication for sending a command to set of miners is inefficient (compared to using multicast) but may still be successful in result.

We want to estimate the behavior from the entire network G taking samples from the reduced network G(W). We know in advance that |G| = ~508k members in the DLT network.

Our experiment does the following:

1. Contact the bootstrap node to obtain *initial set* of miners -> *N*.
2. Perform the discovery step with those *N* miners to get *first hop neighborhood information.*
3. Then synchronize the latest blockchain over a random selection from the neighborhood set to establish oneself as a miner. The samples of responses taken are randomized due to the Ethereum's addressing scheme and its discovery protocol.

For the installed peer, we use an official go-ethereum release with default configuration, as shown in Figure 3:

Network	mainnet
Version	V1.10.2-stable
Local IP address	77.9.109.90
Ports (UDP/TCP)	30303,30313
Bootstrap Nodes	8 (v4UDP Discover Protocol)

Figure 3. go-ethereum release with default configuration.

The peer was compiled in debug mode with extra loggers in the network layer (go-ethereum/p2p/netutil/net.go and ../common.go), dialers (../p2p/dial.go), readers (../interfaces.go), and discovers (../p2p/discover/v4_udp.go). The client probe interface requires no extra configuration. The client uses the Go SDK 1.13.8, mainly UDP Sockets (src/net/udpsock.go) and TCP sockets (src/net/tcpsock.go).

A miner can receive transactions and blocks while executing our steps above, but they are wasted since the miner may not have fully synced yet.

We took samples between February 2021 and July 2021, it means running the go-ethereum's discovery protocol for short periods of time (~1hr/weekend). The peer downloaded ~40GB of the entire block chain, in full synchronization mode from a peer as shown in Figure 4.

The experiment allowed us to discover around five thousand peers, spread globally, as shown in Figure 5.

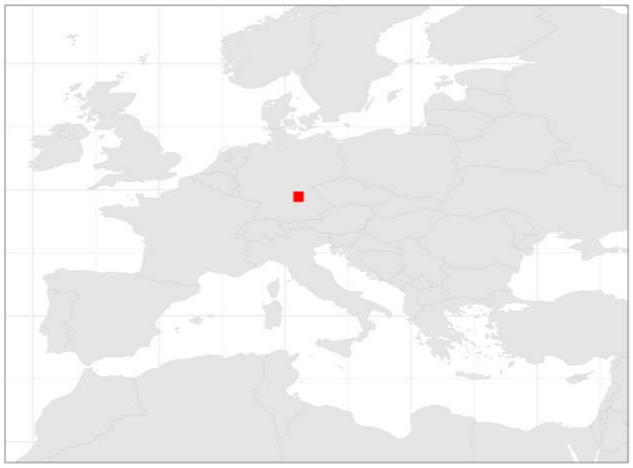
Figure 4. go-ethereum peer download.

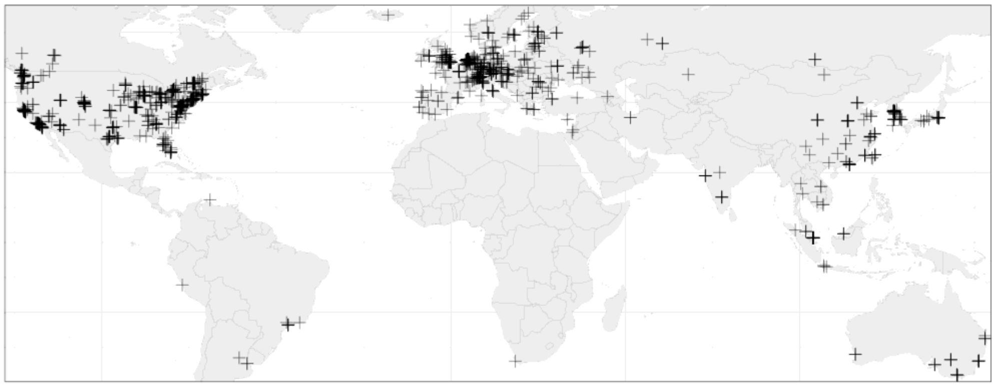
Figure 5. Five thousand peers spread globally.

We could classify the peers according to the operations they execute, shown in Figure 6 in regards to their global distribution:

1. Peers that do not reply to the initial PING-PONG, i.e., are not reachable for some reason, shown as red triangles.
2. Peers that executed the steps of PING-PONG, Key Negotiation, HELLO, start sending useful data and the data is dropped by the client (unsync peer) are colored in yellow. We discovered two reasons for discarding data here:
    a. Possible churn or join latency may lead to received TRANSACTIONS & COMMITs that are wasted. In other words, since entire blockchain is 296GB, it is likely that many miners may send a blockchain that is not the latest one, which will be discarded.
    b. It may also be that none of the direct nor the discovered peers actually have the latest blockchain–particularly a problem if the rate of transaction is larger than the rate of replication across the miners' (replicated) blockchains.

3. Peers which are only executing the main signaling, i.e., PING-PONG, Key Exchange[13] and HELLO, but never engage in data exchange, are colored in green.
4. Peers which successfully executed our protocol interactions, i.e., the PING-PONG, Key Negotiation, HELLO and start sending useful data (blocks or transactions), are colored in blue—these are our useful (or good) peers: miners that provide up-to-date blockchains.

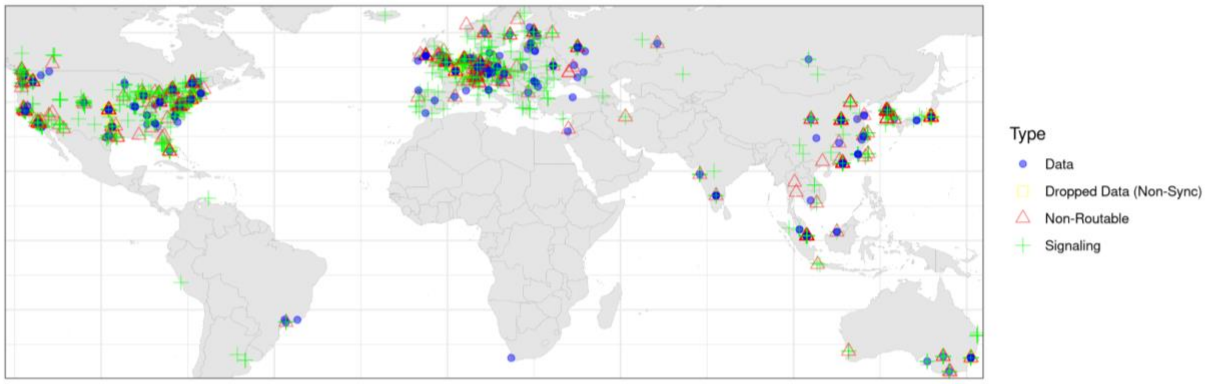

Figure 6. Peers classified by operations.

From our experiment, we can determine that *good peers* account for only ~16% of all peers (with active discovery in ETH). All other peers waste about ~42% of traffic by not providing any useful data. Figure 7 and

Figure 8 illustrate this result from our experiments.

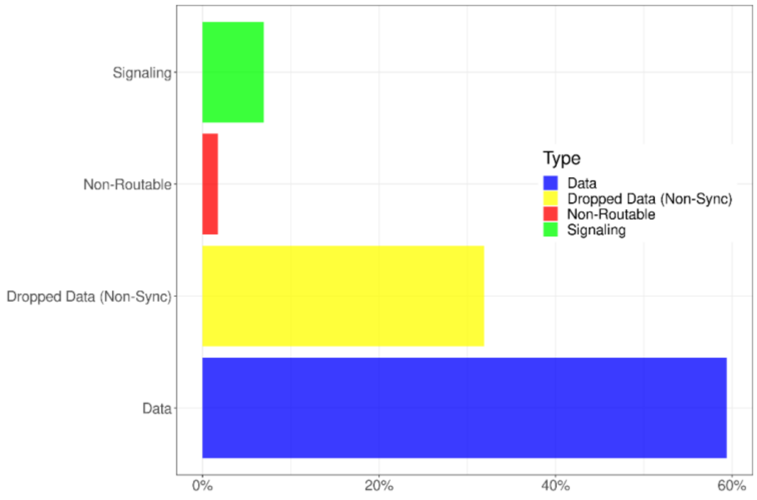

Figure 7. Percentage of packets.

The findings from our experiments show the waste of communication during the discovery step, needing to filter out up to 84% of those nodes that are part of the DLT overlay but not responding positively to the discovery. This wastes network resources every time the discovery step is

[13] The exchange of encryption keys is usually based on available transport protocol solutions such as TLSV1.2 or 1.3; most DLTs started enforcing the use of the recent, most secure TLS1.3 for best security.

performed by any DLT peer. Also, if previously discovered peers disconnect from another peer, another peer must be discovered to replace the lost one, which leads to similar waste than during the initial discovery.

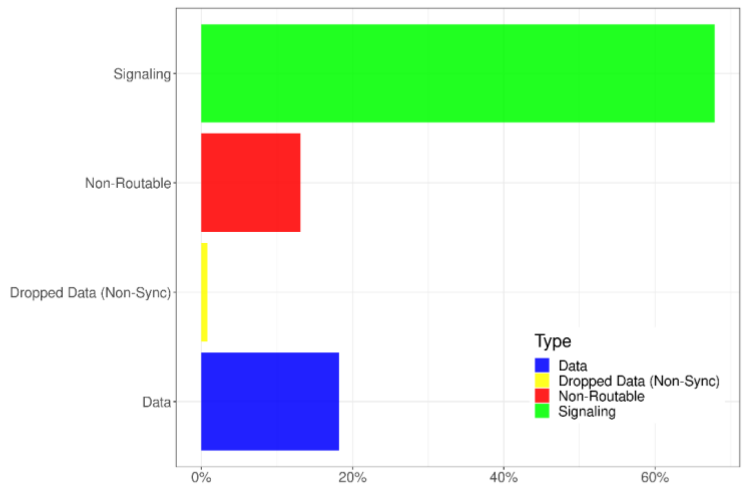

Figure 8. Percentage of endpoints.

In addition, successful communication is significantly reduced, and that impacts the DLT service. Waste may occur when a peer contacts a previously discovered peer, which cannot provide suitable information for the transaction (e.g., by not performing operations for the desired smart contract of the transaction).

Such wasted communication is compensated by issuing the transaction to other miners until the desired spread of transactions, as defined by the DLT diffusion process, has been achieved—those repeated transactions will, however, face the same issues, thereby amplifying waste and prolonging the completion time of the overall transaction. This, in turn, impacts the needed communication for successfully completing the transaction set, thereby increasing the amount of energy per (successful) transaction that needs to be expended, reflected ultimately in the overall costs to provide a DLT service.

Conversely, any avoidance of any such repeated transaction approach would impact the trustworthiness of the transaction, since the (successful) communication with all miners (not a subset defined by the successful communication that may or may not happen at that time for the particular client and its selected miner set) is key to ensuring that the smart contract is being successfully realized.

With that in mind, higher energy and therefore cost expenditure is the direct consequence of the waste we can observe when realizing DLTs; an aspect any service provider will need to consider in costing the deployment of the DLT service. Conversely, prolonging the transaction completion time may negatively affect the DLT application by limiting the maximum rate of transactions for a given DLT network size.

Note that this consequence of the wasted communication adds to the inefficiencies stemming from, for example, the repeated unicast operations to miners for transactions, increasing DLT operational costs further.

4 Opportunities of DLT to the Network

Privacy-preserving computation: Trustworthiness and privacy are primary concerns as companies connect their manufacturing and logistic infrastructures to the industrial internet network. They want to reap the benefit of automated asset management, process control and predictive maintenance. But to do so effectively, companies need to facilitate information sharing among trustworthy partners while complying with data protection and privacy-preserving regulations.

Distributed ledgers offer a viable solution by enabling their participants to discover one another and establish peer-to-peer trust relations without a centralized intermediary. Confidential computing-based blockchain allows the complex network logic to execute inside a trusted execution environment, and preserve the privacy of sensitive networking data. Meanwhile, trusted execution attestation can be signed with a secret key protected inside the blockchain-based confidential computing environment, and the signature can then be verified in a trusted way via blockchain on-chain smart contracts.

This hybrid approach improves privacy on the network-related data and supports a trusted verification for network related execution.

Securing routing protocols: It's a constant battle to protect network infrastructure from malicious attacks. Breaches such as customer records being exposed online, ransomware attacks causing disruption of services and Border Gateway Protocol (BGP) route-hijacking cause problems for companies across the globe.

Network protocols, such as the Border Gate Protocol,[14] can be made more secure through integration of their control plane with blockchain. BGP is the routing protocol that provides connectivity through the exchange of routes between the various autonomous systems (ASs) throughout the world. Security wasn't integrated into the protocol, however, and has relied on trust between the various internet operators. Various mechanisms have been created to help make it more secure such as BGPSec, Mutually Agreed Norms for Routing Security (MANRS) and Resource Public Key Infrastructure (RPKI). Other innovative solutions include integrating blockchain into routing protocols to secure the database holding the routes and AS numbers to prevent tampering. RPKI could, for instance, leverage blockchain in the Regional Internet Registries (RIR's) to ensure the integrity of route ownership by verifying routes in the Blockchain prior to sending or receiving the routing prefix of an autonomous system to ensure secured inter-

[14] *https://datatracker.ietf.org/doc/html/rfc4271*

domain routing in the internet. Or blockchain could be used to secure the distributed network topology. These are among various innovations being researched and tested.

5 Proposed Solutions

Service providers are not without possibilities when it comes to addressing these problems. There are two different angles to address the problems:

- Are there other technology solutions that may fulfil the application intentions in a different, better way?
- If the answer to this question is no, are there network technologies that could improve DLT operations to avoid the waste and inefficiency that we observed?

We start with two answers to the first question, followed by a solution to improve DLT operations through network level innovations.

DLTs use diffusion of transactions (containing the smart contracts) to a set of miners to realize P2P distribution, which in turn leads to the wasteful and inefficient communication observed in our experiments. As an alternative, the Identity/Process/Communication/Transaction (IPCT) framework relies instead on direct, not diffusion-based, communication.

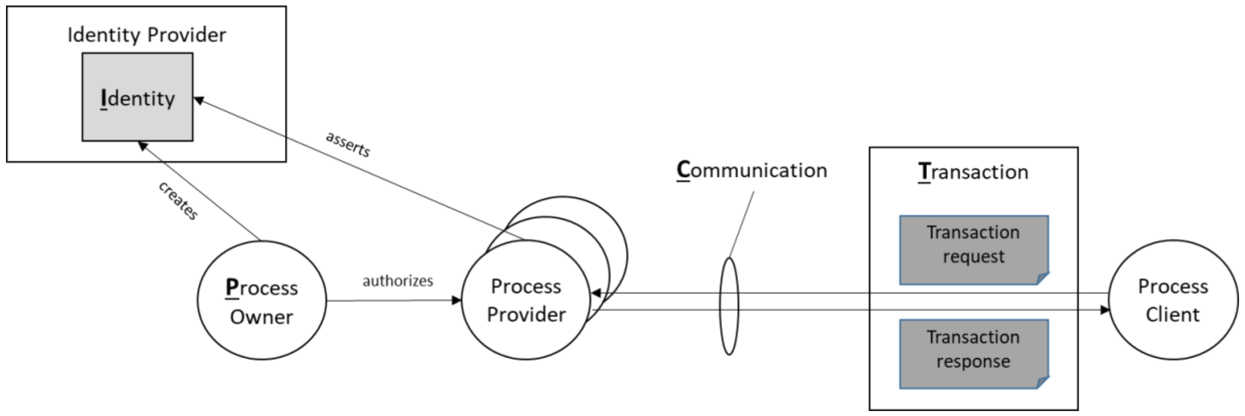

Figure 9. The identity is tied to the process which, in turn, realizes the transaction.

As shown in Figure 9, a *process owner* creates an *identity*, e.g., for the *transaction* or generally the smart contract, with an *identity provider*. With this, the identity is tied to the *process*, which in turn realizes the transaction. For said realization, a process provider may assert the identity of the process (thereby linking the provider to the specific process) and engage in communication with one or more process clients by replying to any incoming transaction requests from process clients through suitable responses.

A number of systems can be used for identity-based realization of transactions through dedicated process providers. A web-service-based approach, using identity frameworks such as the recently developed W3C approach on decentralized identifiers (DIDs[15]) would entirely remove the need for any centralized identity provider. Differences in an IPCT realization may be driven by the stakeholders realizing it and the roles those stakeholders may want to assume. For instance, if relying on an explicit identity provider, this may favor approaches where such entity is required, while a DID-based identity approach may favor stakeholders that do not want a logically centralized role.

Not using a randomized set of miners, that may not be in the right state for communicating successfully, removes the wasteful communication. This is because the linkage to the contract's identity provides a strong binding between the process client and a possible process provider that realizes the transaction. Hence, as long as a process provider for a given identity can be found by the IPCT system, successful communication is provided.

IPCT communication is realized as a P2P unicast communication between a process client and a process provider rather than a diffusion-based distribution to a randomized set of miners, thereby removing the inefficiencies that come with the unicast replication in DLTs.

5.1 Publish-Subscribe Systems

Publish-subscribe systems[16] [17] [18] are realizations of the IPCT framework. Here, the identity is associated with a *topic* within an information graph; this graph may be tree-based or it may feature more advanced directed acyclic graph (DAG) structures.

The process-bound communication between the process client and process provider (within the IPCT communication model) is realized through a *publish-subscribe semantic*, where a subscriber may request the data associated with a given topic, while the publisher of the topic will ultimately provide the data to any subscriber requesting it. For this, a *rendezvous system* is used that matches the interest expressed by the subscriber (in the form of the topic) against the announcement of data (under the topic) by any publisher.

The matching may be future-based as well, so that subscribers may express interest in data related to a topic for which a publisher may not yet exist at the time of the subscription. Data associated with a given topic may be *mutable* in that it may change over the course of time, with new data versions being sent to all subscribers once becoming available at the publisher(s).

Data may also be *immutable*, leading to a single data transfer from a publisher to one or more subscribers, after which the subscription may be terminated (since no new data will be generated). This difference in data semantics allows for realizing versioning systems, where the topic identifies the version of data the subscriber may want to access, or channel semantics,

[15] *https://www.w3.org/TR/did-core/*
[16] *https://streamr.network/*
[17] *https://kafka.apache.org/*
[18] *https://cloud.google.com/pubsub/*

where the topic identifies a channel of communication over which data is frequently streamed from the publisher to one or more subscribers.

Publishers may not be the owner of the data, following the separation of process owner and process provider in the IPCT framework. Publishers may simply provide (even encrypted) data, ensuring the delegation of the provisioning task through appropriate certification of the topic, such as using certificate-bound topic names or tying the encryption of the data into the certification authority of the process owner.

Communication from the publisher to the set of subscribers may be realized through successive unicast, similar to how DLT peers communicate in existing IP networks, or through utilizing network-level multicast with the publisher sending a single data item that is received by all subscribers. When utilizing, for instance, IP multicast techniques for the network-level realization of such multicast, topics and subscriber sets must be relatively long-lived due to the significant group membership costs in IP multicast. For that reason, most publish-subscribe systems utilize unicast replication instead.

While this may inefficiently realize duplication to more than one subscriber, the publish-subscribe system does not rely on diffusion for information distribution since each subscriber explicitly subscribes to the topic and therefore the data that is associated with it. As a consequence, wastage, as observed in DLTs, does not occur. Conversely, the needed trust relations between publishers and subscribers require the use of (often centralized) certificate systems, unlike the diffusion-based DLT system, which allows for untrusted peers to communicate as part of the distributed consensus process realized in the DLT.

5.2 CONSTRAINT-BASED SERVICE ROUTING

Routing in the internet today is realized through the internet protocol, using locator-based addresses to forward packets from a source to a destination. Additional information, such as ports and service-protocol-specific information (such as URLs), are used to invoke service interactions. The DLT interaction patterns constitute service interactions with miners in a DLT providing their services to other miners and clients alike, while the needed IP locator information and port information of every miner is provided to peers of the DLT during the bootstrap process for routing DLT interactions.

Service routing[19] advocates forwarding packets based on service, rather than locator information. Here, services are identified in so-called *service requests*, with the forwarding operations leading to packets being sent to one of possibly many network locations where *service instances* realize the identified service. Service routing also supports multicast invocation by sending the service

[19] René Glebke, Dirk Trossen, Ike Kunze, Zhe Lou, Jan Rüth, Mirko Stoffers and Klaus Wehrle, „Service-based Forwarding via Programmable Dataplanes“, Workshop on Semantic Addressing and Routing for Future Networks, 2021, available at *https://hpsr2021.ieee-hpsr.org/sarnet-21/*

request to more than one service instance at the network level, thereby reducing inefficiencies in one-to-many invocation as they occur in DLTs.

Service-specific *constraints* can be advertised together with the service identifier, ultimately aiding the selection of the 'best' (in terms of those service-specific constraints) service instance from the set of available ones; the overall solution for such routing is termed *constraint-based service routing* (CBSR).

To support service interactions that build ephemeral state[20] after an initial service request, CBSR uses standard IP routing where clients continue to interact with a specific service instance through its IP address rather than initiating a new service request; we term requests that are tied to an interaction state an *affinity request*. Should the client want to communicate again with any service instance, it issues again a service request, possibly leading to an interaction with another service instance of the same service it interacted before.

CBSR can be deployed in existing IPv6 networks by utilizing IPv6 Extension Header for capturing the service identifier and constraint information, while CBSR-enabled routers (CSR: Constraint-based Service Router) may be deployed in a shim layer between transport and network (here IPv6) layer. CBSR may even be deployed across domains, using the existing internet to interconnect between those domains.

The paradigm of routing service requests instead of locator-addressed IP packets can significantly reduce wastefulness observed in DLTs. For this, we first interpret all miners as service instances of a DLT service, for example, *mydlt.org*. With this, a peer issuing a service request to *mydlt.org* will now communicate with any other miner that has previously announced its willingness to serve requests for the DLT service.

Service communication can now take place without any of the usually required bootstrapping of the DLT overlay network, thereby removing the need for any peer to download GBs of overlay routing data. Instead miners solicit their participation in the DLT to the (CBSR-enabled) network, while peers can issue requests immediately without the need for downloading any (overlay) routing information. Unlike the discovery process in an overlay-based DLT, which determines a small subset of all peers in the overlay DLT, a peer in CBSR will be able to use all other peers in the DLT diffusion process, thereby using a larger potential resource pool for its transaction.

CBSR also removes the disconnections that can be observed in the DLT discovery process. This is done by expressing the aspects, which cause miners to disconnect as constraints that are accounted for when selecting one of many service instances during (CBSR) forwarding operations. For instance, the constraint of wanting to communicate with a miner that provides

[20] Ephemeral state may be created in a service transaction as a result of the initial transaction, then used in subsequent transactions. Ephemeral state has usually a limited lifetime, bound to a sequence of transactions in which it is used, while outside of those transaction, the state is no longer needed.

GPU support can be expressed in CBSR as a *capability constraint*, against which service requests can be matched so that only miners are contacted that have previously solicited support for GPUs in their service announcement.

The same applies for other capabilities, such as support for specific hash or security functions. As a consequence, the filtering of good from bad peers during the discovery process is not needed at all, allowing a peer to communicate directly with other peers, leading to communication with only those that will respond according to the constraints they have advertised.

Most importantly, such constraints can also be dynamic (by associating timeouts to their validity and being re-advertised as new constraints once changing at service level). This ensures that peers send transactions only to miners currently working on the same smart contract with which the initiating request is associated.

Although those constraints are dynamic and therefore race conditions may occur (requests are being sent to a miner having indicated support for smart contract *X*, while said miner has already advertised that it has moved to smart contract *X+1*), simulations have shown that such conditions are rare and matching peer requests with suitable miners working on the relevant proof will happen almost all the time.

Through this expression of aspects that lead to disconnections in current DLTs as constraints in CBSR, wasteful communication is almost entirely reduced to zero, leading to

- Improved usage of network resources by removing the need for explicit overlay routing downloads.
- Avoidance of failed communication with DLT miners by explicitly capturing the constraints[21] leading to successful communication in the CBSR routing decisions, removing the need for filtering out the good nodes during the discovery step in the overlay DLT.
- Improved information resilience by achieving the required number of DLT interactions for the desired resilience level in shorter timeframe than existing DLTs over IP networks.
- Faster convergence time, if defining a desired level of resilience, contrasted against the need for repeated communication in existing DLTs until the needed number of DLT interactions has been successfully completed.

The use of the service routing model further removes the inefficiencies identified in DLTs, specifically the need for pushing DLT overlay information to every client and miner alike, while

[21] For this, we differentiate static from dynamic constraints. The former represent, for example, the support for specific security features or HW capabilities, while the latter represent the commitment to specific smart contracts and therefore the ability to serve transactions that are related to the smart contract the client or miner is referring to in its transaction.

also supporting efficient network-level multicast. As a result, efficiency and therefore overall cost of communication is further improved, in addition to tackling wasted communication.

6 Conclusion

The increasing use of DLT has led us to question the possible impact on provider networks. Looking closer at the interactions in a typical DLT, we have observed that inefficiency and waste are important factors that may negatively impact further adoption of DLTs. Understanding the reasons behind both is paramount for thinking of alternatives, both for the use of DLTs (if that is possible within the remit of the considered application) and the realization over provider networks through novel network solutions.

This whitepaper serves as a starting point for a wider dialogue on the use and impact of DLTs, the relation with and use within provider networks, and the opportunities for addressing the identified problems.

7 Bibliography

[1] Howard, H. (2019). Technical Report. 935.

[2] Gao, Y., Shi, J., Wang, X., Tan, Q., Zhao, C., Yin, Z. (2019). Topology Measurement and Analysis on Ethereum P2P Network.

[3] Wang, T., Zhao, C., Yang, Q., Zhang, S., Member, S. (n.d.). Ethna: Analyzing the Underlying Peer-to-Peer Network of Ethereum Blockchain. 1–15

[4] Dotan, M. (2021). Survey on Blockchain Networking: Context, 54(5).

[5] Garay, J. A. (2017). Basic Properties of the Blockchain. 991(2016), 2016.

[6] Miller, A., Litton, J., Pachulski, A., Gupta, N., Spring, N., Bhattacharjee, B., & Levin, D. (n.d.). Discovering Bitcoin's Public Topology and Influential Nodes.

[7] Baumann, A., Fabian, B., & Lischke, M. (2013). Exploring the Bitcoin Network. 369–374.

8 Authors & Legal Notice

This document is a work product of the Industry IoT Consortium's Industrial Digital Ledger Focus Group, led by Mike McBride (Futurewei Technologies) and Xinxin Fan (IoTeX).

Authors: The following persons contributed substantial written content to this document: Dirk Trossen (Huawei), David Guzman (Huawei), Abhijeet Kelkar (GEOOWN Consulting), Xinxin Fan (IoTex), Mike McBride (Futurewei Technologies), Lei Zhang (iExec) and Ulrich Graf (Huawei).

Technical Editor: Stephen Mellor (IIC staff) oversaw the process of organizing the contributions of the Authors into an integrated document.