PhD-FSTM-2023-028
The Faculty of Science, Technology and Medicine

DISSERTATION

Defence held on 15/05/2023 in Luxembourg

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Lucian-Andrei TRESTIOREANU
Born on 13 June 1977 in Ploiesti, Romania

ENABLING RESILIENT AND EFFICIENT COMMUNICATION FOR THE XRP LEDGER AND INTERLEDGER

Dissertation defence committee

Affidavit

I hereby confirm that the Ph.D. thesis entitled "Enabling Resilient and Efficient Communication for the XRP Ledger and Interledger" has been written independently and without any other sources than cited.

Luxembourg, 30 January 2023

Lucian TRESTIOREANU

Name

Acknowledgments

First and foremost, I would like to offer my deepest gratitude to my supervisor, Professor Radu State for making everything possible; for giving me the chance to learn more and become better; for the time and patience spent offering me valuable regular feedback; for his optimism and encouragement to believe, to persevere, to dare, to strive for better. Nelson Mandela said *"It always seems impossible until it's done."*, but Radu shaped me to believe that things are possible, even before they are done.

I want to gratefully thank Professor Cristina Nita-Rotaru from Northeastern University, USA, for her support and guidance; I am forever indebted for all the time spent providing me with valuable advice, for challenging me and making me think about the work from new perspectives, for the drive for excellence.

I would like to extend my thanks to my advisors Dr. Cyril Cassagnes and Dr. Ing. Wazen Shbair for their countless support, advice and assistance along the way. I thank Dr. Aanchal Malhotra from Ripple Labs for the discussions providing valuable insights and advice with respect to XRP and Interledger. I want to thank Professor Raphael Frank, member of my defense committee. I would also want to thank Professor Damien Magoni from LaBRI, Universite de Bordeaux, France, for the valuable discussions and support concerning networking aspects related to the XRP ledger.

A warm thank you to my present and former colleagues from SnT for the fun times and ideas exchanged, especially Professor Patrick Glauner from Deggendorf Institute of Technology, Germany - my former Master thesis advisor, who encouraged me to pursue this path and provided me with advice long after he left SnT; to Dr. Eric Falk, Dr. Christof Ferreira Torres and Flaviene Scheidt de Cristo - with whom I closely collaborated, and to all those who contributed to what I did and who I am today - professionally, and not only. As Sir Isaac Newton put it, if I was able to see further, this was thanks to all the many great people before and around me.

This work was supported by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared. In addition, we thankfully acknowledge the support from the RIPPLE University Blockchain Research Initiative (UBRI) for our research.

Dedications

This work is dedicated to my son, with the hope that it will inspire and motivate him; to my mother, a math teacher, who has been my best mentor and example all along, and who sow in me the seeds that years later made it possible that I walk this path; and to the family that supported and encouraged me.

Last but not least I dedicate it to the whole world - before, around and after me - it would be my greatest reward and satisfaction to contribute, even a little bit, to advancing mankind towards being an even better place.

Index

List of Figures

List of Tables

Abstract

The blockchain technology is relatively new and still evolving. Its development was fostered by an enthusiastic community of developers, which sometimes forgot about the lessons from the past related to security, resilience and efficiency of communication which can impact network scalability, service quality and even service availability. These challenges can be addressed at network level but also at operating system level. At network level, the protocols and the architecture used play a major role, and overlays have interesting advantages like custom protocols and the possibility of arbitrary deployments. This thesis shows how overlay networks can be designed and deployed to benefit the security and performance in communication for consensus-validation based blockchains and blockchain inter-operativity, taking as concrete cases the XRP ledger and respectively the Interledger protocol. XRP Ledger is a consensus-validation based blockchain focused on payments which currently uses a flooding mechanism for peer to peer communication, with a negative impact on scalability. One of the proposed overlays is based on *Named Data Networking*, an Internet architecture using for propagation the data name instead of data location. The second proposed overlay is based on Spines, a solution offering improved latency on lossy paths, intrusion tolerance and resilience to routing attacks. The system component was also interesting to study, and the contribution of this thesis centers around methodologies to evaluate the system performance of a node and increase the security from the system level. The value added by the presented work can be synthesized as follows: i) investigate and propose a *Named Data Networking*-based overlay solution to improve the efficiency of intra-blockchain communication at network level, taking as a working case the XRP Ledger; ii) investigate and propose an overlay solution based on Spines, which improves the security and resilience of inter-blockchain communication at network level, taking as a working case the Interledger protocol; iii) investigate and propose a host-level solution for non-intrusive instrumentation and monitoring which helps improve the performance and security of inter-blockchain communication at the system level of machines running Distributed Ledger infrastructure applications treated as black-boxes, with Interledger Connectors as a concrete case.

1 Introduction

Distributed databases (databases stored distributively on multiple servers) emerged as an improvement over *Centralised Databases* (databases stored on a single central server) because they offered enhanced security and fault tolerance by redundancy. In the distributed database architecture, the trust boundary is set between the distributed database as a whole on one side, and each user on the other side. Because for certain use cases this was a limitation, new technologies like the distributed ledger technology appeared, which enable "database" nodes that don't fully trust each other to cooperate in a byzantine scenario for maintaining and updating the database entries correctly by a consensus mechanism between the participating nodes. The byzantine consensus problem is often explained through the example of the agreement dilemma of the generals of the Byzantine army:

A number of generals of the Byzantine army which can communicate only by messengers must agree on the attack of an enemy city. While one or more of the generals could be traitors sending confusing messages, how could the honest generals reach a secure agreement? [1], [2].

Blockchain is a type of distributed ledger technology on which for example, cryptocurrencies like Bitcoin are based. It is a secure, auditable technology resilient to outages that, in a byzantine environment, enables the recording of transactions in a transparent, immutable, irrevocable, and digital manner.

Even though blockchain technology is relatively new, it has already distinguished as a possibly disruptive technology in several industries: finance, multimedia, healthcare, industry supply chains, energy, the public sector, and more.

However, the blockchain technology is still evolving, with needs for improvement concerning various sub-technologies involved being felt at different levels. For example the network traffic underpinning the blockchain communication can impact the scalability of the blockchain networks, as well as blockchain interoperability. Different aspects related to network traffic can be addressed at the networking level but also at the operating system (system) level. The work presented in this thesis concerns both levels. A large part of the work, presented in Chapters 3 and 5 investigates the usage of overlays as a solution to improve the security, resilience and efficiency of network-level traffic for the XRP Ledger (XRPL)[1] and respectively for blockchain inter-operativity solutions like the Interledger Protocol (ILP) [3]. Chapter 4 presents work at system level to provide better solutions for non-intrusive performance monitoring and profiling, and improved security for the Interledger Protocol taking as a concrete working example two implementations of the Interledger Protocol specification.

This work is *relevant* because the ongoing scaling of the blockchain networks requires secure, resilient and efficient communication, with the same required for the software infrastructure implementing it.

1.1 Research Questions

In the above context, it was interesting to investigate the following main Research Questions which in particular, concern two networks: one implementing a distributed system, the XRP Ledger; and the other implementing a blockchain interoperability solution, specifically the *Interledger Protocol.* Both of them rely on (internet) network communication for a secure, reliable, and efficient service. The below research questions will be refined and clarified in

[1]https://xrpl.org/, valid in January 2023

their respective Chapters.

- *Could overlay architectures, and more specifically, architectures based on Named Data Networking (NDN) [4] and Spines [5], be useful to improve distributed systems like blockchains, at the communication level?*

  Currently there exist many solutions for implementing overlays, like Gossipsub [6], Named Data Networking, Spines, Nebula [7], and Open Overlay [8] to name a few. One of the reasons for choosing to investigate NDN was its push-pull model for data dissemination, which brought along the interesting exercise of finding out how the consensus process of *consensus-validation*-based blockchains could be mapped to NDN. Spines was chosen because it offers security properties like intrusion-tolerance and resilience to various attacks, including Border Gateway Protocol (BGP) [9] attacks. Spines was used for performing remote surgery [10] and it has a good track record of reliability. It was interesting to see how can it be used for blockchain inter-operability.

- *At system level, how to find the real cause of the performance problems of particular implementations (e.g. Interledger applications), and how to assess the performance of the software stack while the application is running, and is treated as a black-box? In this context, to what extent the security and resilience of running applications treated as black-boxes can be improved by acting at the system level?*

More specifically, the research work presented in this thesis focuses on the following aspects:

1) The efficiency of blockchain communication. The growing adoption of Distributed Ledger Technologies and Blockchain in particular, naturally led to the challenge of scaling the networks that they are based on, which highlighted the inherent need for efficient and resilient communication used by the underlying consensus and replication mechanisms. While resilient and efficient communication is one of the main pillars of an efficient blockchain network as a whole, Distributed Ledger Technology is still relatively new and the task of scaling these networks has come with its own challenges towards ensuring these goals. In our concrete case, the XRPL network encountered scaling challenges due to processing and bandwidth overhead induced by the flooding of messages: at scale (around 1000 nodes), because of the sheer number of consensus-related and transaction messages, including duplicates, in the XRPL peer-to-peer network the nodes need ever increasing resources (CPU, memory, bandwidth) to stay in sync. New content distribution concepts like Information Centric Networking (ICN) [11], of which Named Data Networking is investigated here, create new possibilities for achieving this goal, through in-network caching or built-in native multicasting, for example. We modified XRPL to connect to NDN, built an experimental testbed in the lab and evaluated four NDN-based dissemination models on two different topologies, showing that the proposed solution, *XRP-NDN-overlay*, provides a decreased number of messages processed at XRPL node level and improved stability for the XRPL network.

2) Improving the performance and security of blockchain inter-operativity. Payment systems are a critical component of the finance industry and the everyday life of our society. While in many situations payments can still suffer from combinations of low speed, opacity, siloed systems, high costs, or even failures, users expect them to be fast, transparent, cheap, reliable, and global. Recent technologies such as distributed ledgers create opportunities for near-real-time, cheaper, and more transparent payments. However, to achieve a global payment system, payments should be possible not only within one ledger, but also across different ledgers and geographies. One existing solution to enable payments between ledgers is the Interledger Protocol (ILP) [3],[12]. Unfortunately, like many services deployed over the Internet, ILP payments can suffer due to events like lossy links, network failures, and routing misdirections of benign or malicious nature. Also, in the quest towards near-real-time payments, it would greatly benefit from improved link latencies. These challenges must be addressed first, for such systems to be adopted on a large scale. As such, this thesis proposes ***S**ecure **P**ayments with **O**verlay **N**etworks (SPON)*, a service that enables global payments across multiple ledgers by combining the transaction exchange provided by the Interledger protocol with an intrusion-tolerant overlay of relay nodes to achieve (1) improved payment latency, (2) fault-tolerance to benign failures such as node failures and network partitions, and (3) resilience to Border Gateway Protocol (BGP) hijacking attacks. The design takes into account that overlay nodes can be compromised and thus has provisions to address the resilience and fairness of payments in the presence of such compromised overlay nodes. The design goals are discussed, and an implementation based on the Interledger Protocol and

Spines overlay network is presented. We analyze the resilience of SPON and demonstrate through experimental evaluation that it is able to improve payment latency, recover from path outages, withstand network partition attacks, and disseminate payments fairly across multiple ledgers. It is also shown how SPON can be deployed to make the communication between different ledgers resilient to BGP hijacking attacks.

3) Enabling non-intrusive monitoring and performance investigation. Meanwhile, in the current technology state, proper tools and methodologies for monitoring and performance investigation of the infrastructure nodes become more important; relying on static or manual program instrumentation to obtain key performance metrics is unusable for many production environments, while container engines are strengthening their isolation mechanisms. Therefore, non-intrusive monitoring becomes a must-have for the performance analysis of containerized user-space applications in production environments.

The main question addressed in this context is how to find the real cause of your performance problems and how to assess the performance of your software stack?

These capabilities are created and demonstrated by carrying out profiling and tracing of several Interledger connectors using two full-fledged implementations of the Interledger protocol specifications.

1.2 Contributions

The contribution brought by the work presented in this thesis is threefold, and can be synthesized as follows:

- **Network level - optimization of blockchain communication:** Evaluate the utilization of Named Data Networking (NDN) as a communication overlay to identify performance trade-offs to improve the efficiency of communication at the blockchain networking level.

- **System level - blockchain interconnectivity through Interledger:** propose a solution for non-intrusive performance monitoring of Interledger infrastructure (connectors) using eBPF.

- **Network level - blockchain inter-connectivity through Interledger:** design, implement and evaluate *Secure Payments with Overlay Networks (SPON)*, a system which is a solution that enables global payments across multiple ledgers while improving payments latency, offering fault-tolerance to benign network failures and resilience to malicious attacks like BGP hijacking attacks.

2

State of the Art

This chapter presents the current state of the art and background in the areas of blockchain technology and blockchain interoperability solutions. It also highlights some of the known technology limitations in this space and identifies the opportunities for improvement relevant to the topic.

2.1 The Distributed Ledger Technology

Initially data, and in particular bank accounts ledgers, was stored on paper books, or later on digitally, on a server. Because the traditional centralized storage solutions were prone to errors, theft, and loss of data, distributed database solutions have been developed where the same data is stored on multiple machines, improving resilience through redundancy and making forgery more complicated due to the multiple-eyes and multiple-copies nature of Distributed Databases [13]. Normally, on Distributed Databases, all database nodes trust all other database nodes and the limit of trust is between the Distributed Database as a whole and each user.

Distributed Ledger Technology (DLT) appeared as an alternative to Distributed Databases, where data is stored also in a distributed way, but it was designed to work for different trust assumptions [14]. As shown in Figure 2.1, DLT introduces an Adversarial Model, where the nodes which don't fully trust each other cooperate to reach a consensus on the state of the shared data while assuming the possible presence of some malicious nodes. As such, the distributed data can be updated or modified only through a consensus mechanism between participants. DLT uses cryptography means to ensure immutability, non-repudiation, and authorization of the recorded data (transactions) [15].

2.1.1 Overview of DLT

Currently, there are many types of DLT proposed and in production, such as Hashgraphs, Directed Acyclic Graphs (DAG), Holochains, Tempo, and more, with Blockchain technology being just another subtype of DLT.

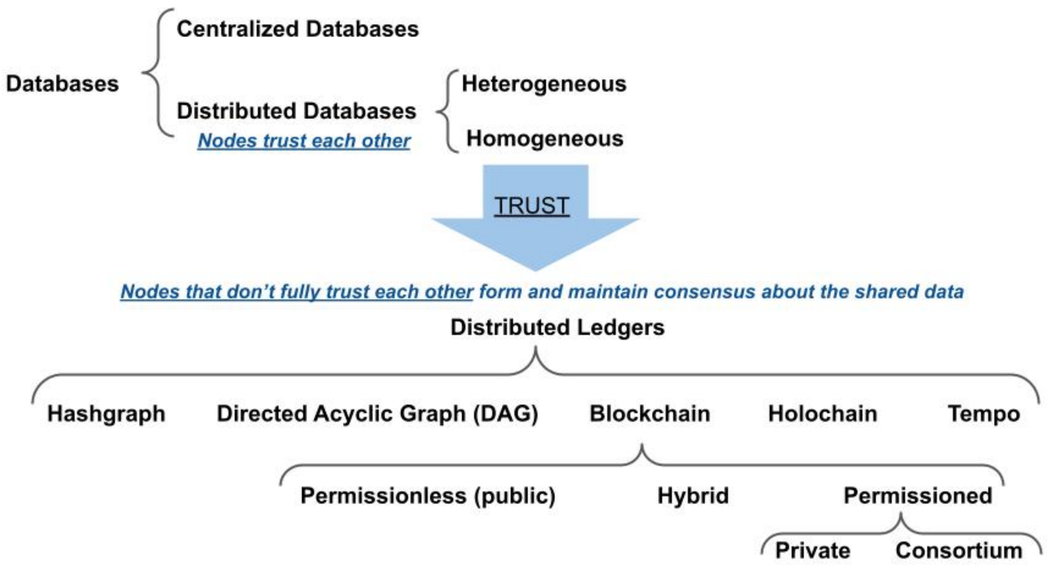


Figure 2.1: Distributed databases, DLT and Blockchains.

Being a DLT, the blockchain technology also replicates the same data distributively on the participating machines or nodes, which do not necessarily fully trust each other but the blockchain is trusted as a group. In the context of other DLTs, blockchains use a specific data structure where the data is stored as a chain of blocks, with each block pointing back to a single parent block (e.g., DAG data structure can be a tree). As such, *blockchain* can be used both to refer to the specific data structure and the specific DLT type. The blocks are linked through cryptographic signatures named hashes (each block points to the previous, or parent block) to guarantee that the recorded information can not be tampered with. To alter the information on some block, all the blocks that followed it should be modified, which, given the consensus mechanisms employed by DLT makes it very hard to achieve. Blockchains can be classified as Private (controlled by an organization), Public (no central authority), or Consortium (controlled by a group). From another perspective, they can be considered permissionless (free to join) or permissioned (working in private enterprise contexts for example).

Each DLT has its own consensus mechanisms, with Proof of Work (PoW), Proof of Authority (PoA), Proof of Stake (PoS), or Practical Byzantine Fault Tolerance (pBFT) being some of the most important. Other examples are *proof of elapsed time*, *proof of activity*, *proof of weight*, *proof of importance*, *leased proof of stake*, *proof of capacity*, or *proof of burn*. Even though incomplete due to inherent graphical limitations, Figure 2.2 attempts to illustrate the DLT landscape [16].

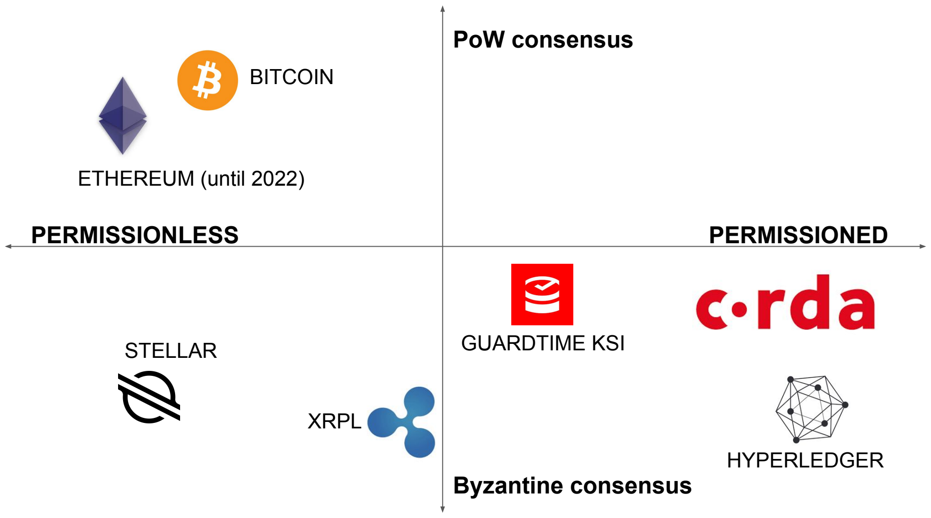


Figure 2.2: The DLT landscape.

Proof of Work (PoW) blockchains.

Well-known blockchains based on the Proof of Work consensus are Bitcoin (BTC) [17], and until recently, Ethereum (ETH). Bitcoin was anonymously proposed under the pseudonym *Satoshi Nakamoto.* The PoW consensus mechanism employed requires a significant but feasible computing effort to verify transactions and add them to the blockchain. This method prevents malicious actors from exploiting the system, however, it is lately criticized for using excessive computing power at scale and as such, being carbon-intensive energy-wise. It is argued that the energy used in the process is lost forever, unlike the gold used to back some FIAT currency[1].

Proof of Stake (PoS) blockchains.

The PoS consensus [18] was designed as an alternative to the energy-intensive PoW, and recently ETH transitioned to PoS for reasons which include the carbon intensiveness of PoW. To the best of the author's knowledge, this mechanism was initially proposed by the user "QuantumMechanic" on "bitcointalk" [2] [19]. PoS is a consensus mechanism where cryptocurrency owners validate block transactions based on the number of staked coins that they offer as collateral (stake) during the process. Also, PoS is argued to be more secure, because the way it structures compensation makes potential attacks on the network less attractive[3].

Proof of Authority (PoA) blockchains.

The proof of authority consensus was originally proposed by Gavin Wood [4], co-founder of Ethereum and Parity Technologies, for private networks [20] in the ETH ecosystem. Suitable for private and consortium settings, for example logistical applications, the PoA consensus is relatively fast due to a design that uses identity as a stake. With a low energy consumption, a small, arbitrarily selected and trustworthy number of nodes validates the transactions while staking their reputation instead of coins. Foregoing decentralization, PoA is often seen as an alternative to improve over centralized solutions in corporate environments. Possible

[1]https://nakamotoinstitute.org/mempool/the-proof-of-work-concept/, valid in January 2023
[2]https://bitcointalk.org/index.php?topic=27787.0, valid in January 2023
[3]https://www.investopedia.com/terms/p/proof-stake-pos.asp, valid in January 2023
[4]https://github.com/ethereum/guide/blob/master/poa.md, valid in January 2023

PoA blockchain examples are Parity, Geth [20], Hyperledger Besu [21, 22], VeChain [5], and Xodex [23].

Practical Byzantine Fault Tolerant (pBFT) blockchains.

As early as 1982, Leslie Lamport showed that, in a synchronous environment, a consensus can be achieved if at most n out of 3n+1 parties involved are dishonest [1]. However, for an asynchronous environment an algorithm was proposed only in 1999 under the name of Practical Byzantine Fault Tolerance [24]. The advantages of the proposed algorithm over BFT are that it can work over the internet which is an asynchronous environment, and that it is optimized to improve the previous response times. This makes it practical also for use cases like DLT.

Federated Byzantine Agreement (FBA) blockchains.

The FBA consensus is part of the Byzantine consensus family. In Byzantine agreement systems, all nodes must agree on the list of participants and require all nodes to process all transactions to reach a consensus. FBA introduces the concept of quorum slices, where a node only needs to trust a specific set of other nodes to take its own decision. Because nodes can decide on their own which other nodes to trust, multiple quorum slices form in the network. For a healthy network, the quorum slices must overlap as shown in Figure 2.3, which means some nodes are trusted in multiple quorum slices which as a result ensures the dissemination of information. The quorum slices must reach a decision among themselves, and when this is reached, a final decision can be taken.

The XRP Ledger (XRPL) [6] is an example blockchain using FBA [25], [26], [27] to achieve high throughput (currently +/- 1500 tx/s), speed (can settle a tx in 3-5s), and low transaction costs (e.g. $0.0000774 in April 2021).

2.1.2 The XRP Ledger (XRPL)

As one of the most established DLTs to date, XRPL is characterized as an open-source, permissionless, and decentralized blockchain system. While the developers of XRPL state

[5]https://www.vechain.org/, valid in January 2023

[6]https://xrpl.org, valid in January 2023

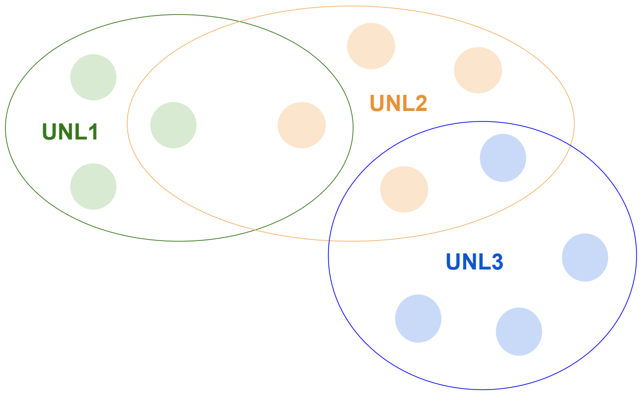


Figure 2.3: FBA quorum slices.

that XRPL is technically a permissionless blockchain, XRPL is arguably perceived by some as permissioned or centralized [28] because, between others, the Default Unique Node List (dUNL) defining the default nodes to be trusted by a node is centrally proposed at this time. It is also considered eco-friendly, with a low energy consumption mainly because by design does not make use of, for example, Proof of Work. All XRPL tokens are pre-generated at the initial launch of the ledger. No new tokens are created after this moment, instead a very small amount of the initially pre-generated coins is burnt for each transaction as an anti-spamming measure.

XRPL node types can be *trackers* or *validators*. Both types receive, relay, and process transactions, but tracking servers are meant to distribute transactions from clients and respond to queries about the ledger while validating servers can do the same functions as the tracking servers, plus they work to advance the ledger history. For performance reasons it is recommended that validators do not receive and distribute transactions from clients or respond to queries.

XRPL consensus [29], [30] uses *message flooding* for communication between nodes and prioritizes safety over liveness which means that in case consensus can not be reached, the building of new ledgers halts until the problem is solved, possibly by manual intervention.

During *flooding*, every incoming packet to a node is re-transmitted through every outgoing link except the one it arrived on. The advantages of flooding are that it is simple to implement, and that, if a packet can be delivered then it will, although probably multiple times. The disadvantages of flooding are message duplication which increases the network load, and poor bandwidth usage - a message with only one destination reaches all nodes; moreover, unless specific measures are taken, duplicate packets may circulate forever.

As such, at scale (around 1000 nodes) XRPL starts suffering from the high number of messages incurred by the flooding mechanism used for the dissemination of messages like transactions and consensus-related messages. The nodes need increasing hardware specifications and bandwidth to keep up with the network (remain in sync with the rest of the XRPL network). As such the XRP Ledger encountered a scalability challenge, which needs to be addressed. This is one of the topics of this thesis.

A literature review showed that most peer-reviewed work related to XRPL focuses mainly on the XRPL consensus protocol which was originally described in 2014 [26]. It was analyzed in [31], [32], [33] and investigated empirically in [34]. In 2020, the authors of [25] identified relatively simple cases where consensus may violate safety and/or liveness; it is argued that the XRPL needs a very close synchronization, interconnection, and fault-free operation between validators.

Other work related to XRPL comprises an analysis of the energy consumption of the XRPL validator nodes [35], crypto-asset network flows [36], a health assessment of the XRPL credit network [37], a proposal for a blockchain benchmarking framework [38], or a topology analysis [39]. XRPL's security is challenged in [40] through a man-in-the-middle attack.

However, to the best of the author's knowledge, there is no previous peer-reviewed work focused on the efficiency of messaging mechanisms involved in the XRPL consensus protocol and transaction propagation. At the same time, the problem of communication efficiency for blockchain networks at scale has lately received increased attention, and different solutions, some making use of overlays, have been investigated:

At the protocol level, a *squelching* mechanism was proposed by *Ripple Labs* to mitigate the impact of message flooding on the XRPL nodes and bandwidth performance [7,8], but it was not yet deployed in production. *Erlay* [41], an efficient transaction relay for Bitcoin, reduces bandwidth by 84% but increases the latency for Tx dissemination by 2.6s (from

[7]https://xrpl.org/blog/2021/message-routing-optimizations-pt-1-proposal-validation-relaying.html, valid in January 2023

[8]https://github.com/XRPLF/rippled/blob/develop/RELEASENOTES.md, valid in January 2023

3.15 to 5.75s). *Perigee* [42] which as the title says, is an efficient peer to peer network design for blockchains, focuses on mitigating the block propagation delay, but not on the message flooding issue. *Epidemic Broadcast Trees* [43] propose broadcast trees embedded on a gossip-based overlay while *Splitstream* [44] distributes the load of forwarding messages evenly between participants.

At the overlay level, *Gossipsub* [6], was proposed for improving transaction and block propagation in ETH2 and Filecoin (IPFS). It disseminates data by forming a separate, dedicated overlay mesh network of *Gossipsub* nodes. Its main advantages are: the node degree can be set and tweaked as necessary between a min and max; it addresses the latency while being resilient to network churn. Security assessments have been performed for a large spectrum of attacks and strong mitigation solutions implemented. It scales to large networks, provides an adjustable trade-off between bandwidth and latency, and the mesh updates in real-time as per nodes' individual decisions.

Named Data Networking [4], [45] is another possible overlay solution, which is investigated in this thesis for practical FBA blockchains like XRPL.

BoNDN [46], proposes tx dissemination for Bitcoin (BTC) through a push model over NDN interests, and a subscribe-push model for block propagation. The model proposed for tx dissemination is similar to our approach *piggyback* where we obtained good results performance-wise, but it is challenged in [47] for using multicast at the NDN level - the authors argue that it is doubtful if in practice, the NDN nodes would enable multicast for the given data-labels. For XRPL which incurs a handful of known-in-advance, static list of validators, we find such a set-up not problematic.

Another design and implementation for propagating over NDN the transactions and blocks for PoW blockchains like the Ethereum blockchain were proposed in [47], while [48] sends blocks over a multi-layer design based on NDN to achieve 74% of [49]'s overhead.

2.1.3 NDN data synchronization protocols

Being content-oriented, the NDN architecture suites for data synchronization, which resulted in the development of different data synchronization protocols, like *Vectorsync [50], Chronosync [51]*, or *Psync [52].*

BlockNDN [49] employs the Chronosync protocol for block broadcasting on a Bitcoin-like blockchain and *State Vector Sync* [53] was also proposed to sync data between multiple nodes over NDN. However, in [47], it is argued that these solutions are not suitable for blockchain networks because, among others, the underlying protocol used was not originally designed for the byzantine environment characteristic of DLT. Moreover for XRPL, if the goal is to minimize the number of messages incurred by the communication, then the synchronization protocols could introduce some additional synchronization messages which could be counter-productive.

2.2 DLT interconnectivity

DLT interconnectivity, in particular blockchain interoperability, refers to the ability of different independent blockchain networks to exchange *assets*, and *data* as a superset of the former, in a secure, trusted way. Exchanging data could be seen as the ability to make advanced inter-blockchain API calls. The motivating idea is that the future will rather have multiple blockchains that need to communicate with each other, than multiple blockchains working in isolation. The simplest analogy would be that someone with an AOL account could not send in the past an email to someone else with an MSN account, as it is the case today between any email providers. Someone with an Ethereum account for example, can not natively (at layer 1) make a payment to someone with an XRP account. Possible blockchain interoperability project examples are Polkadot [54], Interledger [3], Cosmos [55], Cardano [56, 57], Plasma Bridge [58], Quant's Overledger [59], and Lisk [9].

2.2.1 DLT interconnectivity landscape

The best-known types of technical approaches to blockchain interoperability could be classified as i) *notary schemes* (e.g. the "atomic mode" of Interledger [3]), ii) *relay* or *multi-chain* (e.g. Cosmos and Polkadot), and iii) *hashed timelocks*, with the latter enabling in principle only the exchange of assets (e.g. the Lightning network [60, 61, 62]); HTLCs were at some point introduced in the Interledger Protocol as a way to remove the need for the notaries. The

[9]https://lisk.com/learn/undefined/lisk-interoperability, valid January 2023

Interledger Protocol (ILP) v.4 is today blockchain agnostic and the settlement can be either based on Hashed TimeLock Contract (HTLC)[10] [63, 64], unconditional payment channels [11], on-chain transactions, or legacy payment systems.

A Systematization of Knowledge on the topic of Cross Chain Communication (CCC)[12] was published in 2019 [65], as a systematic exposition of protocols for cross-chain communication. The paper shows that (CCC) is not possible unless a trusted third party is involved.

Belchior et. al. [62] extend and generalize the classification to include more types, as shown in Table 2.1. Along the time, ILP went through the notary scheme and hashed time locks to become open to any settlement method and blockchain-agnostic, or even more generally, ledger-agnostic.

Table 2.1: DLT types.

Type	Subtype	Examples
Public connectors	Notary schemes	- ILP "atomic mode" - Binance, Coinbase (centralised) - 0x, Uniswap (decentralised)
	Sidechains and relays	- BTC Relay, Zendoo
	HTLC based	- ILP with HTL settlement, Lightning, XClaim
Blockchain of blockchains		- Polkadot, Cosmos
Hybrid Connectors	Trusted Relays	- Hyperledger Cactus, supporting Hyperledger Besu and Fabric, Corda and Quorum
	Blockchain agnostic	- ILPv.4
	Blockchain migrators	- Hyperledger Burrow based on ETH

With reference to Table 2.1, The *public connectors* family was the first one to emerge [60], to allow crypto token exchange, but this is no longer the only scope of DLT interoperability, and solutions for generalized interoperability have emerged, namely the *blockchain-of-blockchains* and *hybrid connector* types [62].

A brief presentation of the above approaches is provided below.

Public connectors are further categorized into the following:

[10]https://en.bitcoinwiki.org/wiki/Hashed_Timelock_Contracts, valid in January 2023
[11]https://interledger.org/rfcs/0027-interledger-protocol-4/#why-unconditional-payment-channels, valid in January 2023
[12]https://crosschain.mx/, valid in January 2023

Sidechains and relays. One blockchain (the *main chain*) considers a second chain as its *child chain* or *secondary chain* or *side chain.* Through specific protocols, the two blockchains can interoperate and transfer assets. Main chains can also be a sidechain to each other. To enable the asset transfer, a two-way peg is required between the two blockchains. In fact, the asset transfer does not physically happen, instead the assets are only locked and unlocked on the respective chains. Smart contracts are used to enable this process. An example is the Liquid [13] [66] open-source side chain for Bitcoin. Security-wise, if the security of the main chain is compromised, the logic of the sidechain is invalidated. Another limitation is that sidechains do not allow users to embed specific pegging mechanisms into their applications.

Notary schemes. The notary schemes employ a *trusted* federation of notaries to attest to chain A that some event happened on chain B, and can be seen as cryptocurrency exchanges. They can be centralized or decentralized, acting as intermediaries between blockchains, but their applications are limited. One disadvantage of this approach comes from the trust that must be invested in the federation of notaries.

While on the notary scheme, some intermediary had to verify for some blockchain that an event took place on another blockchain, in the multichain or relay approach, the blockchains do this task themselves, i.e., one blockchain can read events on another blockchain, and possibly, the other way around. This approach can cover use cases like asset portability and atomic swaps. Asset portability means that some coin, native on Ledger X, can be transferred and used on Ledger Y, with the possibility to return to ledger X. Atomic swaps mean that, if two users A and B have accounts on both ledgers, some payment from A to B will happen only if another payment from B to A happens also.

Hash-locking. On hash locking, two events on chains A and B are triggered by the same event, possibly the preimage of some hash. It is used for cross-chain atomic operations, but can not enable asset portability or cross-chain oracles. A blockchain oracle is an informational gateway for that blockchain to the outside world. A smart contract deployed on some blockchain would have limited functionality if all data that it could access would be the data on its native blockchain. Blockchain oracles provide a way to query data from the outside world, and in this way enable for example much more coverage for smart contracts

[13]https://liquid.net/, valid in January 2023

deployed on some blockchain. Cross-chain oracles would provide the same data to multiple blockchains, which is sometimes not always easy because each blockchain has its own integration requirements [60].

Blockchain of blockchains enables the creation of customized, application-specific blockchains that can inter-operate, and are somewhat similar to the sidechains in the sense that they are composed of a main relay chain and secondary chains that use mechanisms called bridges to connect to other blockchains through pegging or Hashed Time Locks. They are very recent, with Kusama [14] - the test network of Polkadot released in 2019 for example, and Cosmos' main network launched in March 2019 [62]. Cosmos is focused on transferring tokens while Polkadot focuses on the transfer of data and assets, besides tokens. Since none of the present solutions is accepted as wide as a standard, solutions like Polkadot or Cosmos still risk creating fragmentation and siloing of user data and value, also because their users can not connect to any arbitrary blockchain, should they wish so. It is also argued that validator nodes can be compromised, and a costly BFT consensus is used amongst blockchains [67].

Hybrid connectors use a *blockchain abstraction layer* [68] to enable *decentralised Applications* (dApps) to interact with blockchains through several uniform operations instead of different APIs [62],[69].

Trusted relays act as a trusted party that relays data from a source to a target blockchain, without involving a new blockchain. Users work with different APIs and can create arbitrary logic. An example is Linux Foundation's open-source *Hyperledger Cacti*, which resulted from the merge of Hyperledger Cactus and Hyperledger Weaver [15] supported by *Accenture* and *Fujitsu*, which is currently compatible with *Hyperledger Besu*, *Hyperledger Fabric*, *Hyperledger Sawtooth*, *Corda*, *Iroha*, *Go-Ethereum*, and *Quorum.* Security wise it is argued that the centralized Cacti node server could be compromised [67].

Blockchain agnostic protocols make use of an abstraction layer to enable inter-connectivity between virtually any blockchain networks, and the business logic flexibility is somewhat limited. The parties emitting transactions on different blockchains i.e. the connectors are the root of trust, meaning that they are accountable under the law. An example is Interledger

[14]https://kusama.network/, valid in January 2023
[15]https://www.hyperledger.org/use/cactus, valid in January 2023

Protocol v.4 (ILPv4), a protocol also discussed at the World Wide Web Consortium (W3C).

Blockchain migrators. Using this approach, users cam migrate data between blockchains, with smart contract to be implemented in future work. Similarly to pegged sidechains, tokens or smart contracts could be locked on a chain and re-deployed on another one [70].

2.2.2 DLT interconnectivity examples

This section introduces a few of the most relevant blockchain interoperability solutions with an emphasis on the Interledger Protocol which is part of one of this thesis' lines of work.

Polkadot [71] is one of the best-known examples of the *relay chains* or *blockchain of blockchains* family. The term *polkadot* is used to define a token and a protocol. The protocol aims to enable different independent blockchains to exchange money and data. The token is used in governance to enable holders to have a say on the future of Polkadot for example, and in the consensus mechanism as a stake for the transaction verification. Polkadot consensus is based on a *Nominated Proof of Stake* (NPoS) [72] mechanism, and it is made up of a central super-ledger or relay ledger, and multiple parallel chains whose transactions can be processed in parallel which are called *parachains.* The parachains can communicate between themselves through the central relay ledger and do significant heavy computation, thus freeing up the central relay ledger. This is possible due to the sharding technique to split a database into pieces, a technique used here to enable multiple blockchains (the parachains) to process data in parallel [73].

This is how Polkadot can perform many transactions in a short time (approximately 1000 transactions per second). The central relay ledger is performing the consensus and validation of transactions and relaying the communication between the parachains. The parachains are application-specific full-fledged, specialized blockchains. The security of the architecture is ensured by the *validators* on the central relay ledger, which are chosen by the *Nominators*, while *Fishermen* ensure that the validators behave correctly, or otherwise their stake (and nominator's) would be lost to the other stakeholders. *Collator nodes* are lighter than the validators - they just record the valid transactions on the parachain and send them to the relay chain validators. Figure 2.4 illustrates the architecture of Polkadot. The central relay blockchain ensures information exchange between the parachains and as such between the

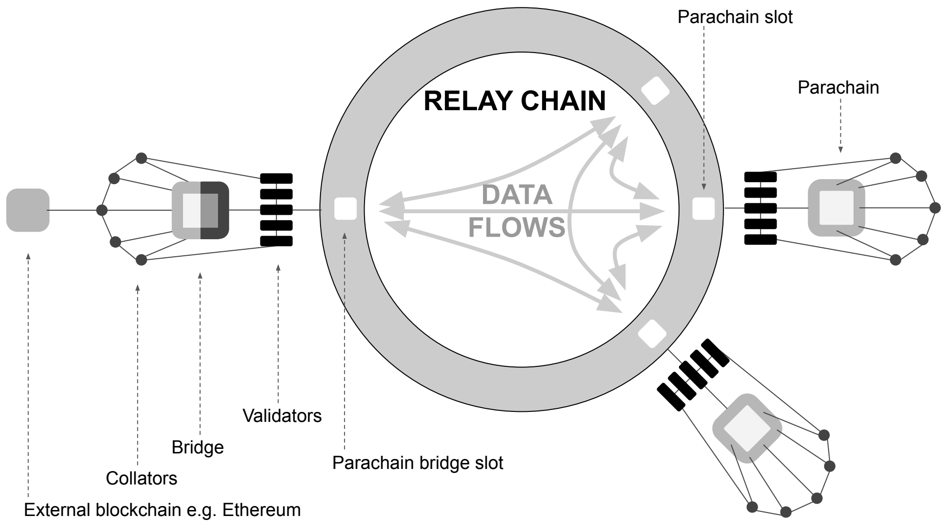


Figure 2.4: Polkadot architecture.

external inter-connected blockchains, hence the name *"blockchain of blockchains"*. Through Polkadot blockchains can exchange tokens, account balances, information from the real world, and interoperability between smart contracts deployed on different blockchains is achieved. The transactions are atomic.

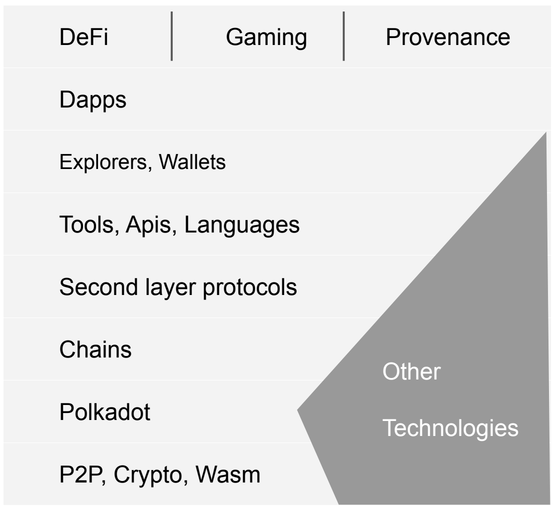


Figure 2.5: Polkadot tech stack.

The Polkadot Tech Stack [74], illustrated in Figure 2.5 is based on open-source technologies and is meant to be used for the development of decentralized applications dApps including DeFi, Gaming, Provenance, and more.

Hyperledger Cactus, renamed *Cacti* after the fusion between *Hyperledger Cactus* and *Hyperledger Weaver* [16], makes use of different plugins to interconnect with different ledgers. The plugins can be built as part of the Hyperledger Cacti project, or as separate projects by private or consortium entities. When interconnecting different blockchains, proofs of the network-wide states of the blockchains must be provided to the outside, and these proofs must be verifiable by others. As such, in Cacti, for each connected external blockchain, there is a group of validators external to the said blockchain (also called ledger connectors) - which work together to reach a consensus agreement (independent from the consensus of the monitored blockchain) on the state of the monitored blockchain, and provide proof of the network-wide state of the blockchain.

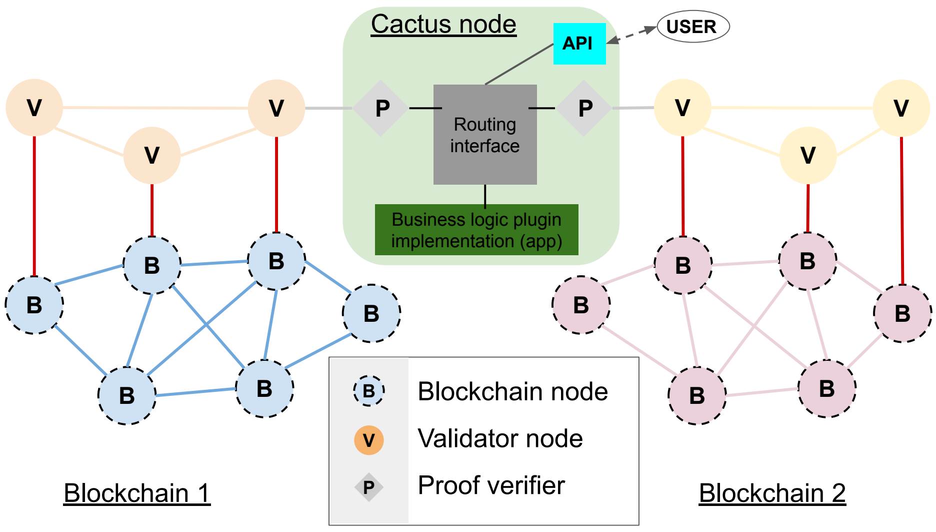


Figure 2.6: Hyperledger Cacti architecture.

The validators are ledger-specific, which means that a new group of validators must be deployed to connect a new blockchain. The advantage of using such validators is that being similar, they are easier to discover and connect to, and as well, the signatures they provide to verify the proofs are easier to verify because they all have the same format. With the condition that they trust the validator nodes, outside *verifier nodes* can request validator signatures and use these to verify the proofs of the connected blockchain [75]. The design of

[16]https://www.hyperledger.org/use/cactus, valid in January 2023

the Cacti architecture is illustrated in Figure 2.6.

The **Interledger Protocol (ILP)** is an interoperability solution proposed to support payments across different ledgers. Because ILP is an important component of the work presented in this thesis, the Interledger protocol will be discussed in more detail below. This work considers the current version of Interledger which is version 4 (ILPv4)[17].

The goal of Interledger is to create an international friction-less payments routing system for sending and receiving money globally. The Interledger protocol is literally a protocol for inter-ledger payments. Its main usage consists of multi-ledger payments, enabled by a set of connectors. To stream payments, the ILP stack provides STREAM, an additional transport protocol that breaks the total amount into small packets of value and then streams them.

Besides lower cost and faster settlement than some classic banking transactions, one of the most interesting aspects of the Interledger Protocol (ILP) is that it will seamlessly manage payments when the sender's currency is different from the receiver's currency, or when the sender's payments network is different from the receiver's payments network.

The ILP ecosystem comprises multiple software components. *Ledgers* keep records of users' accounts and balances, either in fiat or crypto-currencies. In the context of Interledger, a *Ledger* is any accounting system that holds user accounts and balances. It can be linked to cryptocurrencies like Bitcoin, Ethereum and XRP, or to classic banks, PayPal and more. *ILP Connectors* are the transaction intermediaries and hold multiple wallets on different ledgers, such that they can perform currency exchange, and forward packets on behalf of their customers, while receiving a fee. Finally, *Applications* ran by end-users connect to ILP Connectors to perform transactions between each other; examples include Moneyd, or Switch API by Kava Labs.

Figure 2.7 shows how ILP facilitates payments. Consider customers *Alice* and *Bob*, where *Alice* has an account in Euro and wants to pay *Bob*, who has an account in BTC. Connector *C1* has an account in Euro, and an account in XRP, while Connector *C2* has an account in XRP and an account in BTC. C1 and C2 are peered together, i.e. they negotiated also a business relationship. ILP allows Alice to create a payment request in Bob's favor, which will travel from her to C1, C2 and then to Bob. Upon receiving the payment, Bob will send back

[17]https://interledger.org/rfcs/0027-interledger-protocol-4/, valid in January 2023

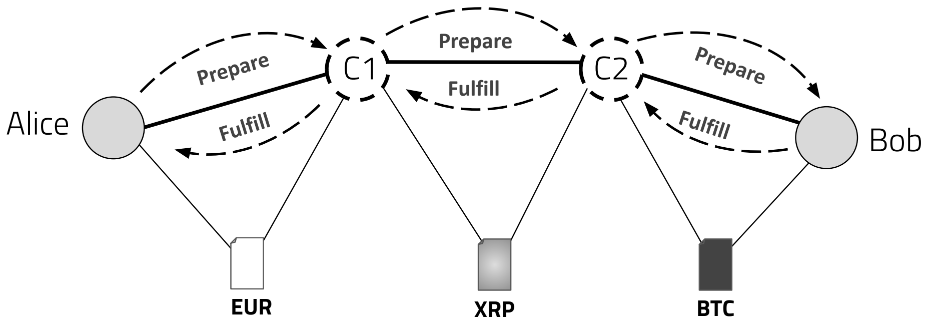


Figure 2.7: Payment with ILP. C1 and C2 are ILP connector nodes.

on the same path a receipt, which will finally reach Alice. The receipt assures all parties that the payment was successful and they settle their balances. As it travels between connectors, the value changed wallets and currencies.

In ILP, money is actually not moved, meaning that ILP doesn't decrease or increase the total amount of electronic money in circulation. A connector swapping both currencies has an account for each payment system it supports. Account balances are open and closed between parties involved in a particular transaction according to the transaction instructions of each payment system involved. The parties are the sender, the intermediaries (connectors), and the receiver. In other words, when the receiver's currency is different from the sender's currency, also, no money is leaving the sender's network and no money enters the receiver's network. In fact, at some point along the chain, some connector with accounts on both payment systems keeps the sender's currency in one wallet (belonging to the same ledger as the sender) and forwards the money towards the receiver, now denominated in the receiver's currency, from its other wallet holding that currency on the second ledger - the same ledger with the receiver's. The main difference with the classic system running today is that with ILP, the end-to-end payment becomes completely seamless thanks to the automation of many parts provided by the Interledger Protocol (ILP) suite.

As represented in Figure 2.8, the system of money transfer over ILP involves recording and manipulating balances at different levels:

- *The Bilateral (ILP) Balance*, kept between two peers.

- *Settlement* on the *payment channel (paychan)*, which involves signing *claims* that are recorded on the payment channel opened between the two peers. The claims settle the transactions between the two peers resulting from adjusting the Bilateral Balance above.
- *On-Ledger recorded transactions*, resulting, for example, from redeeming the previous claims submitted on the payment channel.

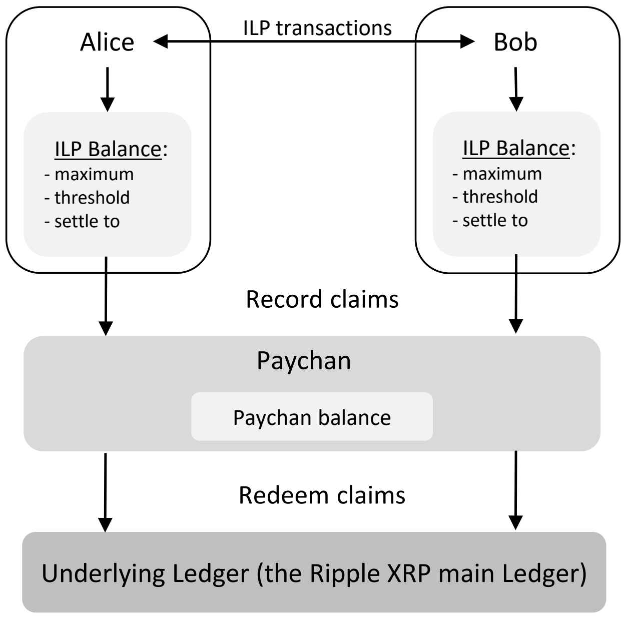


Figure 2.8: The money transfer system.

Payment Channels are an important feature of nowadays Interledger. Some distributed ledgers are defining their own payment channel concept. Therefore, it is important to keep in mind the definition of the (ILP) *Payment Channel* agreed in the documentation of Interledger[18].

When two ILP nodes connect, they negotiate a *payment channel* according to their needs. *Payment channels* are opened only between direct peers. Their bilateral transactions will

[18]https://github.com/interledger/rfcs/blob/master/0027-interledger-protocol-4/0027-interledger-protocol-4.md, valid in January 2023

afterward be carried on to the payment channel. Payment channels are a solution for faster, cheaper and more secure transactions, especially when the ledger involved is slow or expensive.

Bilateral Ledger. When transacting on a payment channel, each of the two parties holds a *Bilateral (ILP) Ledger*, which records the transactions performed in-between the two, and the balance (this is not the underlying ledger but internal to the application). Most of the transactions are performed off-ledger, thus also improving the speed and transaction costs. Only when the peers redeem their recorded payment channel claims, the specific transaction is recorded on-ledger. On the payment channel each claim is recorded individually, but they can be later redeemed individually or in bulk on the ledger [76].

Settlement is a core concept used in ILP and only occurs between direct Interledger peers. In practice, *settling* is encountered for example while setting up plugins or in relation to payment channels. The main concept, illustrated in Figure 2.8 [77, 78], in practice usually involves a system of Interledger balances and payment channel claims.

In relation to payment channels, *settlement* involves signing a claim for the money owed. Claims do not need to be directly submitted to the ledger, but in the case of the XRP Ledger which is fast and cheap, they can [79, 76]. The process will be reflected in the payment channel balance in Figure 2.8. Multiple claims can be signed on the payment channel, and the payment channel balance will update accordingly. Note that at this point, no amount or transaction has been yet recorded on-ledger (except the initial channel creation and funding), so the ledger accounts for Alice and Bob still show the same balances as before (except for cheap and fast ledgers like the XRP Ledger which makes it possible to submit claims individually if desired, to make the money available faster).

Claims can be redeemed out of the payment channel and into the user ledger account in bulk or individually. The payment channel can be closed or reused.

On-Ledger transfers. On-ledger transfers can be initiated in different ways. The most relevant in ILP is redeeming the claims submitted on the payment channel. Only at this point, the money will show up in the user's wallet. Another possible way to initiate an on-ledger transaction is for example directly using the *ripple-API.*

A real-life scenario is shown in Figure 2.9: If Alice wants to send Bob 10 USD, ILP will find a route for this payment, which in the given case involves Connectors 1 and 2.

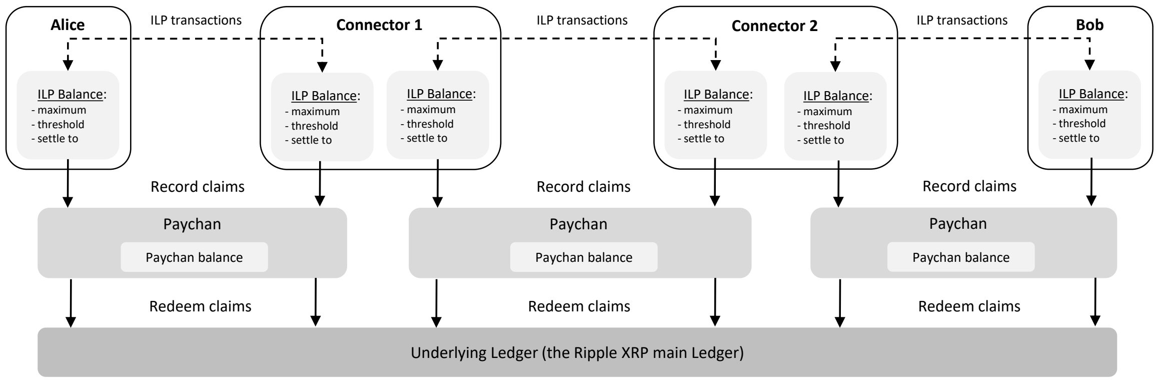


Figure 2.9: The money transfer system in practice.

The *prepare packets* will travel forth from Alice to Bob through Connector 1 and Connector 2, and the *fulfill packets* will travel backward on the same path. Firstly, Connector 2 will pay the 10 dollars to Bob, and settle with Bob, in exchange for the proof from Bob that he got his money. Then Connector 2 will forward the proof to Connector 1 and in exchange, Connector 1 will send the 10 USD to Connector 2 so they can settle with each other. Finally, Connector 1 will pass the proof to Alice which will settle with Connector 1 by paying the 10 USD she wanted to send to Bob. By means of this chain, Alice has in fact sent Bob 10 USD.

2.2.2.1 The Interledger protocol stack

The Interledger architecture is often compared with the Internet architecture, as shown in Table 2.2, also because they adopt a similar layered approach [80]. It involves multiple protocols, of which the most representative are the ***B**ilateral **T**ransfer **P**rotocol* (BTP), the ***I**nterledger **P**rotocol* (ILP), the ***S**treaming **T**ransport for the **R**eal-time **E**xchange of **A**ssets and **M**essages* (STREAM), and the ***S**imple **P**ayment **S**etup **P**rotocol* (SPSP).

SPSP is a protocol for exchanging the required information to set-up an Interledger payment between a payee and a payer. It is the most widely used Interledger Application Layer Protocol today [82]. SPSP makes use of the STREAM protocol to generate the ILP condition and for data encoding.

Because STREAM does not specify how to exchange the required payment details, some

Table 2.2: A parallel between the Internet and Interledger architectures. [80]

Internet architecture		**Interledger architecture**	
L5 Application	**HTTP** SMTP NTP	L5 Application	**SPSP** HTTP-ILP Paytorrent
L4 Transport	**TCP** UDP QUIC [81]	L4 Transport	IPR PSK **STREAM**
L3 Internetwork	**IP**	L3 Interledger	**ILP**
L2 Network	PPP **Ethernet** WiFi	L2 Link	**BTP**
L1 Physical	Copper **Fiber** Radio	L1 Ledger	**Blockchains**, Central Ledgers

other protocol or application has to implement this: SPSP is a protocol for exchanging payment details between the *sender* and the *receiver*, i.e. *ILP address* and *shared secret* [83]. In other words, SPSP is a means for exchanging the server details needed to establish a STREAM connection.

Payment pointers are persistent payment end-point identifiers on Interledger. They are a standardized identifier for accounts which can receive payments [84]. *Payment pointers* can uniquely identify invoices and pull-payment agreements.

STREAM is a Transport Protocol working with ILPv4. Application-level protocols like SPSP make use of the STREAM protocol to send money. STREAM splits payments into packets, sends them over ILP, and reassembles them automatically. It can be used to stream micropayments or larger discrete payments and messages. STREAM is inspired by the QUIC Internet Transport Protocol [81] and was preceded by the *"preshared key v2" (PSK2)* transport protocol.

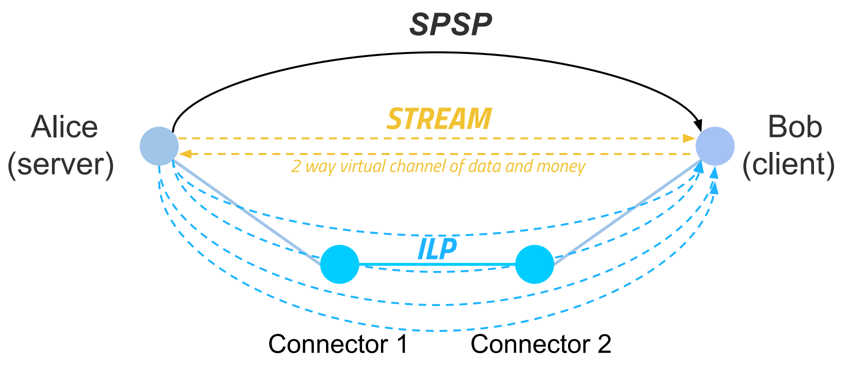


Figure 2.10: STREAM is a logical, bidirectional channel over ILP. [80]

As illustrated in Figure 2.10 with yellow, a STREAM connection establishes a logical two-way, virtual channel of data and money between the payer and payee. STREAM packets can be sent as the data field of different Interledger packets like *ILP Prepare* (type 12 ILP packet), *Fulfill* (type 13 ILP packet), or *Reject* packets (type 14 ILP packet), after being encoded and encrypted. The logical connection is used to send authenticated ILP packets between the "client" and "server" (the blue connections in Figure 2.10). One STREAM logical connection can spawn up to eight ILP physical Interledger packet streams, in the case of Figure 2.10 - the four blue flows. Either the payer or the payee can be the server or the client. STREAM provides authentication, encryption, flow control (ensure one party doesn't send more than the other can process), and congestion control (avoid flooding the network over its processing power).

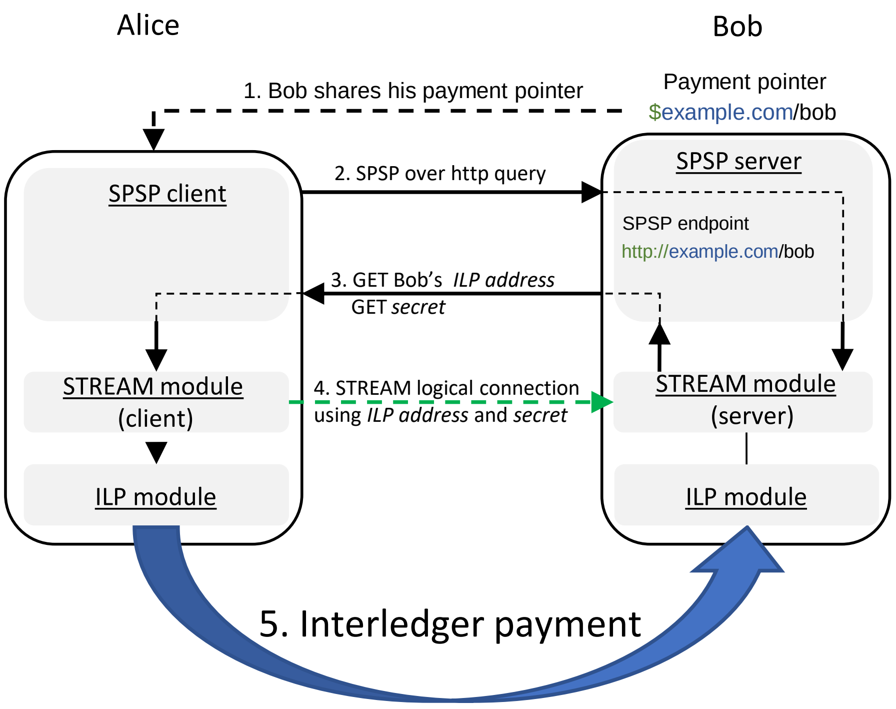


Figure 2.11: The STREAM protocol flow.

Figure 2.11 illustrates the unfolding of the STREAM protocol. STREAM servers are waiting for clients to connect over ILP. The servers connect to a specific plugin on the local machine and wait for the ILP packets. Usually, the *ilp-plugin* is used to connect to Moneyd. The server generates a unique *ILP address* and *shared secret*, which will be used to encrypt data and generate fulfillments for ILP packets in relation to a specific client. The *request* for the address and secret, and the *response*, are not handled by STREAM, but for example by SPSP. After a client has the ILP address and secret (obtained with SPSP for example), it can connect to the STREAM server by using these credentials [85, 86].

STREAM facilitates micro-payments use cases like *Coil* [87], where users pay a flat fee, then Coil sends micropayments to content creators for each second users enjoyed their content; or like *ILP torrent*, which only communicates the list of peers if the fee for the file was paid. STREAM's micropayments feature also helps mitigate the risks associated with transferring large amounts by lowering the amounts in-flight.

The temporal interaction between the components of the Interledger Protocol (comprising BTP, ILP, STREAM, and SPSP), as well as the application-level infrastructure involved is illustrated in Figure 2.12.

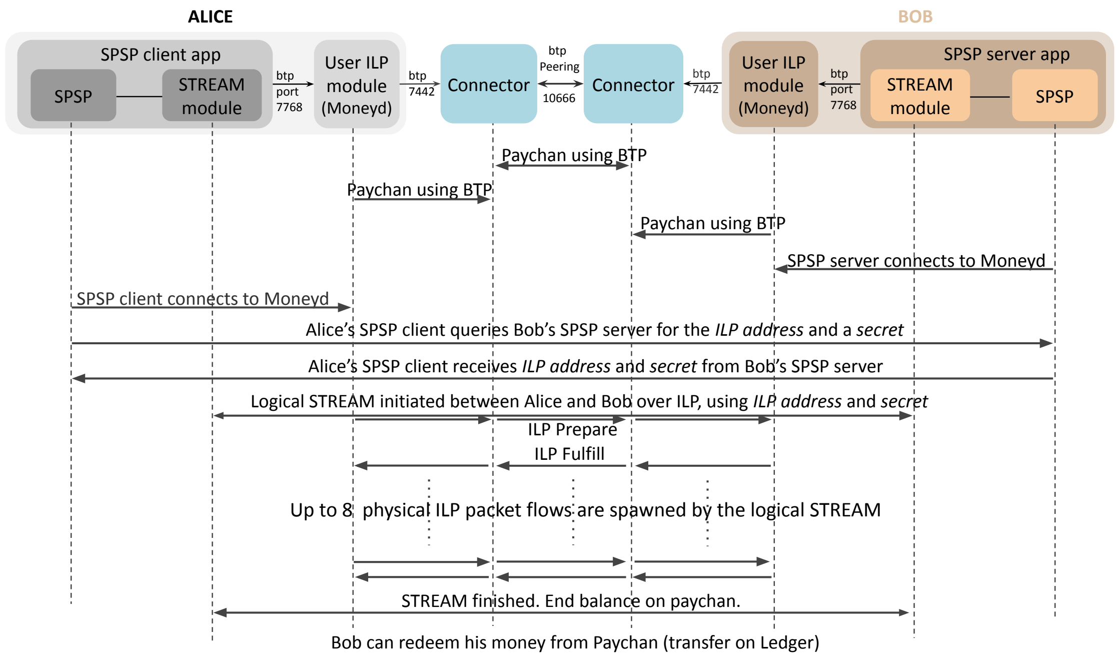

ALICE
BOB
SPSP client app
SPSP
STREAM module
btp port 7768
User ILP module (Moneyd)
btp 7442
Connector
btp Peering 10666
Connector
btp 7442
User ILP module (Moneyd)
btp port 7768
SPSP server app
STREAM module
SPSP
Paychan using BTP
Paychan using BTP
Paychan using BTP
SPSP server connects to Moneyd
SPSP client connects to Moneyd
Alice's SPSP client queries Bob's SPSP server for the ILP address and a secret
Alice's SPSP client receives ILP address and secret from Bob's SPSP server
Logical STREAM initiated between Alice and Bob over ILP, using ILP address and secret
ILP Prepare
ILP Fulfill
Up to 8 physical ILP packet flows are spawned by the logical STREAM
STREAM finished. End balance on paychan.
Bob can redeem his money from Paychan (transfer on Ledger)

Figure 2.12: Unfolding of the ILP protocol.

The advanced relationship between protocols can be better understood by referring to Figure 2.13 [88, 89], which summarizes all the information provided above.

The annotations used in the Figure 2.13 are explained as follows [88, 89].

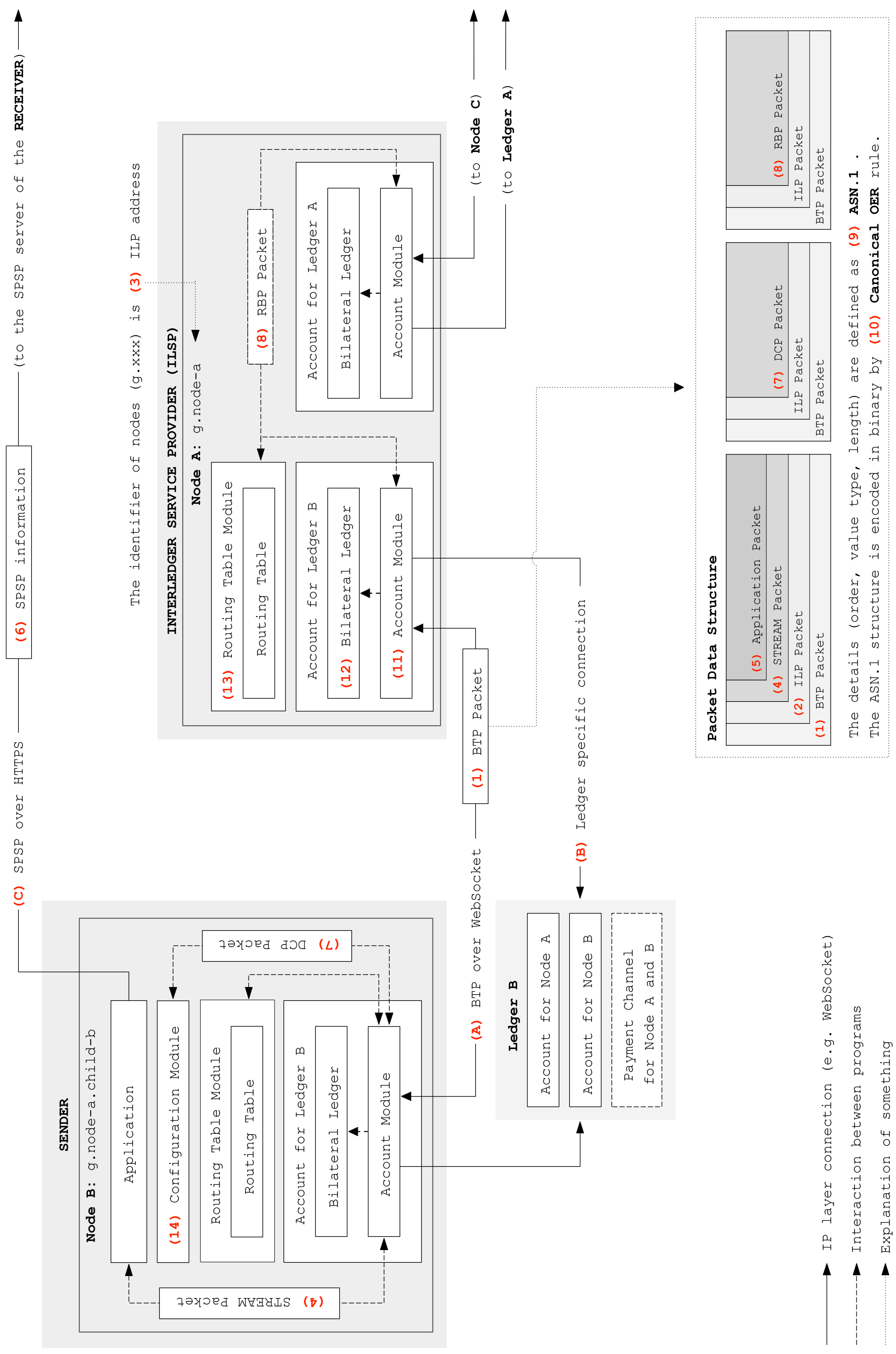


Figure 2.13: Interledger Protocols and details. Advanced diagram. [88, 89]

CONNECTIONS

(A) *BTP over WebSocket* is used to establish the connection between two nodes to exchange payments, configuration and routing information by creating secure communication channels between the peers. BTP currently works over WebSocket Secure connections.

(B) The *Ledger specific connection* is established from Nodes to Ledgers as a means to settle their payments. If the settlement is done over a blockchain, the nodes could use payment channels because they are faster and cheaper than direct blockchain settlements.

(C) Prior to effectively starting a payment, the *SPSP protocol*, running over HTTPS, enables an *Application* to exchange the required payment information such as a *shared secret* or a *destination address* with the payment *Receiver* running an *SPSP Server* at their end of the connection.

PROTOCOLS

Link Layer Protocol

(1) *BTP* - The Bilateral Transfer Protocol 2.0 (BTP 2.0) is used as a carrier for ILP packets and messages between two nodes.

Core ILP Protocols

(2) *ILP* - The *Interledger Protocol V4 (ILPv4)* is used for sending payment packets over multi-hop routes. Other protocols, e.g. for node configuration or routing, can also work over ILP by encapsulating their information in the ILP packets which are exchanged between nodes.

(3) The *ILP Address* is used for nodes identification.

Transport Layer Protocol

(4) *STREAM* is a *Multiplexed Money and Data Transport* for ILP. Built on top of ILP, it enables a bidirectional transfer of money and data between applications.

Application Layer Protocols

(5) *Application Packet* - Applications can implement custom protocols over STREAM by encapsulating their protocol data in the data field inside the STREAM packet.

(6) *SPSP* - Prior to issuing a payment, the Simple Payment Setup Protocol (SPSP) is used to establish the required end-to-end payment details such as a *shared secret* or *destination* *address.*

Connector to Connector Protocol

(7) *DCP* - The Interledger Dynamic Configuration Protocol v1 (ILDCPv1) works over ILP to enable the exchange of node information like ILP addresses. The protocol data is encapsulated inside an extensible data field in the ILP packets.

(8) *RBP* works over ILP by encapsulating the data required for sharing the routing information needed to create the routing tables.

DATA STRUCTURE AND ENCODING

(9) *ASN.1* defines the concrete structure of data such as the order, the type, or its length.

(10) *Canonical OER* - Because ASN.1 doesn't specify an encoding rule, i.e. how packets are encoded in binary, *canonical OER* [19] is used to define the encoding.

FUNCTIONS

The functions can have different forms, according to implementation.

(11) The *Account Module* manages the connection to a given peer node and to the ledger used for settlement with the given node.

(12) *Bilateral Ledger* is implemented as the Bilateral Transfer Protocol 2.0 (BTP 2.0) to keep track of payment balances, e.g. for not-yet-settled payments, between two peered nodes.

(13) The *Routing Table Module* processes the received routing information to build the best routes.

(14) The *Configuration Module* is implemented over the *Interledger Dynamic Configuration Protocol v1* (ILDCP v1), to request to a parent node the *ILP address* and other information that a child (user) node needs to use.

(15) The *SPSP Server* executes the *SPSP protocol* over HTTPS to provide the *payer* with necessary payment details like *shared secret* and *destination address* and it may, or may not, be run on the *payee* node.

[19]https://gist.github.com/sappenin/100a475fc7175a164a949985b05fa696, valid in January 2023

However, because they work over the internet, the DLT interoperability solutions of which Interledger is part of, can suffer from lossy paths, network faults and partitions, or DoS attacks like route hijacking. Previous work used relays to solve some of these individual problems.

The *security* of the interconnection is very important. SABRE [90] was for example proposed to address BGP routing attacks against Bitcoin (BTC). SABRE relies on the BGP path selection to ensure through the placement of a few nodes (less than 10) that most BTC nodes will not be partitioned by a BGP hijacking attack. This is achieved by relaying all the traffic through this very small set of relays that must be equipped with sophisticated hardware to sustain the high load of the BTC network. Changes in BGP peering relationships together with cost changes will impact the correct functioning of SABRE. SABRE also relies on the fact that many BTC clients are within a very small number of ASes, and as such, scaling it for inter-ledger communication to cover clients spread across many different locations may not be a straightforward task. SABRE does not employ custom protocols to improve performance. Finally, SABRE requires that relay nodes do not get compromised and follow the protocol correctly. Partially because of a low relay/client ratio SABRE [91] uses a software-hardware co-design to sustain high loads. It does not consider compromised relay nodes.

With respect to *performance*, another stream of ideas aims to improve blockchain transaction rate by speeding up block propagation: Falcon [92] and Fibre [93] use relays for fast dissemination of BTC blocks. BloXroute [94] also uses relays for fast dissemination of blocks, for several ledgers. All of them focus on blocks and not payments, are vulnerable to routing attacks, and together with SABRE, assume that the relay nodes are not compromised and follow the protocol correctly.

Falcon achieves gains through minimal validation; hand-optimized, dynamical topology and 10 servers deployed world-wide use cut-through routing for fast block propagation: data is propagated as soon as any portion of a block has been received. The blocks are transmitted as they are being actively received by the network. It is orthogonal to block compression, however, if blocks are compressed, there should still be benefits. Falcon has the disadvantage that a block can be validated only after receiving all required packets [95].

Fibre uses Forward Error Correction to enable nodes to reconstruct data in advance even if some parts have been lost on the way [95].

Both Falcon and Fibre are centralized solutions that put control in the hands of the entities that control them, and thus indirectly in the hands of the governments where they reside. As such censorship possibilities are introduced. They are operated voluntarily by small groups and are dependent on them [94].

bloXroute has a provably-neutral network design and the goal to provide the abstraction to connected nodes of being connected to all p2p nodes by the use of cut-through routing [94]. It provides encryption, is ignorant to packet content and seeks to treat all blocks (or payments) fairly but it assumes that the overlay nodes can not be compromised; it also sends audit control packets which offer the possibility to check censorship on origin or content, and show that the network is neutral. Together with Falcon, bloXroute considers the *incentivization* of overlay operators.

Spines [5] proposes a *Soft Realtime Link protocol* enabling localized retransmissions to increase packet delivery ratio [96] and can protect against BGP hijacking.

Nebula [7] and Open Overlay [8] provide *security groups* and *access control lists* but are not intrusion-tolerant.

The above are synthesized in the following two tables: Table [97] presents comparatively the most important characteristics of selected blockchain inter-connectivity solutions. Table [98] illustrates the same for chosen blockchain accelerator and blockchain security solutions.

Table 2.3: Blockchain inter-operativity solutions.

Solution	**Polkadot**	**Hyperledger Cacti**	**Interledger**
Approach	sidechain + relay nodes	notary scheme	relay nodes
Use case	move data and tokens	move data and tokens	pay across ledgers
Limits	- validators can be compromised - costly BFT consensus - limited number of parachains available; slots sold via auction	- central server can be compromised	- complex architecture - connectors can be compromised
Security	- shared state between relay chain and parachains - already hacked, millions lost	- tendermint consensus - 100 validators	conditional transfers
Strong points	- flexibility (modular framework) - shared security model	- extensible plugin architecture - secure by default - toll free	micropayments
Open source?	yes	yes	yes - Interledger Foundation approves commits
Last git commit	January 2023 complex ecosystem	January 2023	January 2023 implementation dependant (ILP is a protocol)
Governance	token holders weight + referenda [99]	- under Linux Foundation - open governance - set of maintainers [100]	Interledger foundation

Table 2.4: Blockchain message dissemination solutions.

Solution	**SABRE**	**Falcon**	**Fibre**	**Bloxroute**
Purpose	- BGP security	- fast block dissemination	- fast block dissemination	- fast block dissemination
Limits	- affected by changes in cost of inter-AS agreements and changes in BGP peering relationships - designed for cases when most clients are in a small number of AS-es - performance was not a goal	- focus only on block propagation (not payments) - blocks validated only after receiving all packets - commercial project - no performance analysis on block propagation [101]	- focus only on block propagation (not payments)	- focus only on block propagation (not payments) - tiered subscription model
Strong points	- BGP security for BTC at relatively low cost - partially deployable - provides some protection even with only 2 relays	- overlay operators incentives - cut-through forwarding on relay nodes	- Forward Error Correction reconstructs data in advance if some parts have been lost - UDP, data compression [101]	- treats all blocks fairly - sends audit packets to ensure correctness
Incentive	- not defined	- overlay operator incentives	- BTC health	- overlay operator incentive (token)
Risks	- relays can be compromised	- routing attack vulnerability - relays can be compromised	- routing attack vulnerability - relays can be compromised	- routing attack vulnerability - relays can be compromised
Gover-nance	- just technical solution - multiple SABRE systems belonging to different entities can coexist	- Cornell University research team - preceded bloXroute [102]	Matt Corallo (BTC team)	- founders (Cornell University research team)
Open source	- partial (some pseudocode in paper)	no	yes	- no [103] - gateway is open source

3

XRP-NDN Overlay: Improving the Communication Efficiency of Consensus-Validation Based Blockchains with an NDN Overlay

3.1 Introduction

The scalability of blockchain networks is an active problem in the community and also in research. Previous work highlighted how latency and bandwidth limit scalability.

There is also, a loose current of opinion that generally, at most two of the following three properties can be achieved for blockchains: decentralization, scalability, and security [104]. Security is reinforced in a permissionless network by broadcasting all transactions and blocks between all the participants, of whom a majority must agree on the state of the blockchain. This process however is expensive and hinders scalability. On the other hand, security can also be achieved by setting a fixed number of trusted participants to control the state of the blockchain, which makes the blockchain permissioned or centralised.

Here, we focus on the scalability problem [105] for permissionless blockchains, which states that starting from a *well-provisioned* blockchain, we can obtain a new well-provisioned one with proportionally increased *load* and *nodes*, where:

- *Load* is the stream of transactions, measured in transactions/second (TPS).
- A blockchain is *well-provisioned* for some *load* if the *max_throughput* of that blockchain is higher than the respective *load*.
- *Max_throughput* is the *load* that can be processed with bounded latency.

While different aspects of Distributed Ledger Technology (DLT) research have lately benefited from increased attention from the community, the underlying communication protocols, often relying on flooding mechanisms due to the one-to-many and many-to-many communication needs of DLT, have received somewhat less attention. The task of scaling the blockchain networks while maintaining their performance and resilience comes with its own specific challenges, one of these being to maintain or even improve the efficiency and resilience of the underlying communication when working at scale. Each blockchain technology type has its specifics, and per our current understanding, a one-size-fits-all solution is far from possible. For example in the case of Ethereum (ETH), which until recently was a Proof of Work (PoW) blockchain, Gossipsub [6] was proposed to improve its communication layer, while [47] proposed a Named Data Networking (NDN)-based design for the ETH block propagation. Authors of [48] propose and experiment an NDN-based design for block propagation obtaining 74% less overhead than [49], but the fairness of their evaluation versus IP network communication is not clear. Nevertheless, the community effort was mainly directed towards PoW-type DLTs, with other types like consensus-validation based blockchains, of which XRPL is part of, receiving less attention. On XRPL, the size of the messages involved in the consensus protocol, e.g. validations and proposals is small enough (approximately 0.5kB) as to not pose a challenge regarding message size itself. Instead, in the context of the near-real-time communication needs (a new ledger is created every 3 to 5 seconds), it is rather the overwhelming number of messages and the processing incurred at each node, inherent to the flooding model of communication used in XRPL, that finally challenges the scalability goals of the XRPL network.

This is aggravated by the strong interconnectivity between the nodes in the XRPL network, as shown by the analysis we performed with Nem [106] after scanning and capturing the XRPL network topology: the network consisted of 892 nodes of which 152 were identified as validators. The *average distance* between nodes was 2.37. The average shortest path is the sum of all shortest paths between vertex couples divided by the total number of vertex couples. We also found a topology *diameter* of 5 (the greatest distance between any pair of vertices, or the graph "compactness").

Between these nodes, a giant bi-connected component of 867 nodes was identified, together with 25 smaller ones. The giant component was analysed further: the component had 9172 edges. The *mean degree* of the nodes was 21, while the *maximum degree* was 296, with multiple similarly high-degree nodes, which means that some nodes act as hubs. The *mean distance* was 2.34, the *median distance* 2.37, with a *diameter* of 4 and a *radius* of 3. The results are summarised in Table 3.1.

Table 3.1: Topology analysis of the XRPL production network.

Metric	**nodes**	**edges**	**validators**	**diameter**	**radius**	**avg_dist.**	**med_dist.**	**avg_deg.**	**max_deg.**
Full XRPL	892	9197	158	5	-	2.37	-	-	-
Giant component	867	9172	-	4	3	2.34	2.37	21.15	296

At scale, the overhead incurred by the flooding of messages increases the requirements for the communication channels e.g. bandwidth (BW), the hardware used by the nodes (CPU and memory), and the energy and financial burden, which if not addressed, could finally result in a degradation of the overall network performance.

In a previous work, [107] showed that on XRPL, the number of proposal and validation messages can represent 72% of all messages.

To mitigate the flooding overhead, different approaches to improve the message dissemination efficiency could be considered, e.g. improving the efficiency of the dissemination protocol itself, or external solutions such as overlays.

As such, in the case of XRPL, the problem can be stated, in a broad manner: How can the performance burden added to the nodes' CPUs, memory, and bandwidth by the high number of messages incurred by the flooding dissemination model used at scale, could be alleviated? Of the possible solutions, we focus on improving the dissemination method to decrease the number of messages, and deviating (some of) these messages through an overlay where we can make use of specific properties to achieve this goal.

NDN [4] is a proposal for a Future Internet Architecture which instead of delivering packets to a given destination (IP), fetches the data by name, offering for example *content caching* to improve delivery speed and reduce congestion. The process of DLT data dissemination can also benefit from native, built-in *multicast* available on NDN.

Data can be disseminated over NDN in at least the following two main ways:

1) In the classic, native pull-based approach, nodes request and receive by name the pieces of data that they are interested in; this, combined with NDN's native in-network caching, could increase communication efficiency by decreasing the overall number of messages exchanged.

2) In another approach, also taken by [46] for example on DLT, the data can be encapsulated in the Interest Packet and disseminated with multicast on pre-determined paths (interested nodes must enable multi-cast for the respective Interest). Moreover, current work to decrease the number of duplicate messages on NDN multicasting [108], could further improve communication efficiency of XRPL.

The contribution is two-fold: i) there has been no prior work on this topic focusing on consensus-validation based DLTs, and ii) this work evaluates over multiple practical implementation models, to find the best approach that benefits a concrete case such as the case of XRPL.

3.2 Background

This section presents the background related to the two main technologies on which this work is centered around: the Named Data Networking as an overlay solution, and the XRP Ledger with a focus on messaging that pertains to being ported to NDN.

3.2.1 Named Data Networking (NDN)

Named Data Networking (NDN) is a proposed *Future Internet* architecture evolved from the 2006 *Content-Centric Networking* (CCN) project, by Van Jacobson [109].

Noticing that today's Internet is rather used as an information distribution network, NDN is not delivering packets to a given destination address (IP), but it fetches the data which is identified by a given name. NDN distinguishes itself from other architectures through the following:

1. In NDN, the *data* is named by the applications, and *Consumers* request data by its *name.* As such, the process is consumer-driven.

2. *Data packets* are cryptographically signed by their respective *Producers.* As a result of this *data-centric* approach, the data can be verified by consumers no matter how it was received.

3. Routers record each data request *(Interest packet)* and erase it once data is received. As such, smart strategies can be used for forwarding, and loops eliminated.

NDN offers *content caching* to improve delivery speed and reduce congestion, a *simpler configuration* of the network devices, and *data-level security.*

On NDN, the *Producer* creates new data, and the *Consumer* is *interested* to receive or *"consume"* the new data produced by the *Producer.* From these two roles derive the packet types:

1. The *Interest packet*, normally sent by a *Consumer* to ask for some data piece produced by a *Producer.*

2. The *Data packet*, which is normally created by a *Producer* and sent back to a *Consumer* as a response to an *Interest* packet sent by the *Consumer.*

Besides the above, other relevant NDN building blocks are:

1. *Content Store (CS)* which stores for some period the data packets which it has already seen in order to immediately serve them in case of a new request.

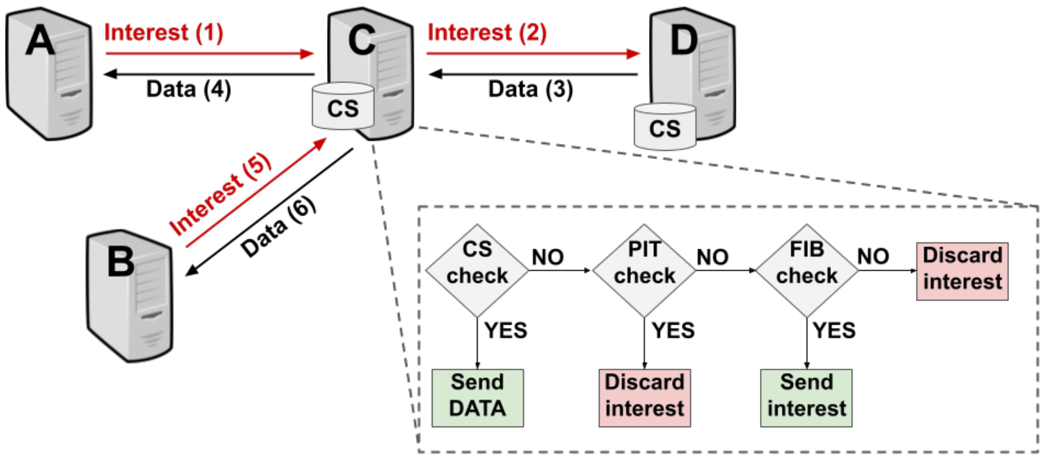


Figure 3.1: The basic mechanisms of NDN.

2. The *Pending Interest Table (PIT)* is a storage for unfulfilled Interest packets.

3. The *Forward Information Base (FIB)*, similar to a routing table, helps an NDN node decide where and how to route some packet.

As illustrated in Figure 3.1, NDN is working as follows:

1) The *Consumer* node A is interested to receive some piece of *Data* and sends the *Interest(1)*.

2) Upon receiving the interest, the *Router* node C:

- Checks its own *Content Store* to see if *data* is already available locally. If data is available it answers with the data, or in case the data is not available locally it proceeds to next check. In the Figure 3.1, data is not available locally, so C proceeds to the next check:

- It looks in its own *Pending Interest Table* (PIT) to see if a similar interest was already received. If a similar interest is already received, the router discards the newly received interest, or if it can't find an entry (this is a new interest), it then proceeds to the next check. In Figure 3.1, this is a fresh new Interest, so C proceeds to the next check:

- It checks its *Forwarding Information Base (FIB)* to see if it knows how to route the interest further. If it does, it forwards the Interest, or if it can't find an entry it drops it. In Figure 3.1, C has a routing entry in its FIB so it forwards the *Interest(2)* to *Producer* node D.

- Upon receiving the Interest, D answers by sending the *Data(3)* which then is stored by node C in its CS, AND forwarded to node A as the *Data(4)* packet.

3) When *Consumer* node B sends the *Interest(5)* for same *Data*, the data is found by C in its CS, and is directly forwarded to B as the *Data(6)* packet in Figure 3.1.

3.2.2 The XRP Ledger

XRPL topology. As shown in Section 3.1, the XRPL network consists of around 1000 nodes which are strongly connected. The network is forecast to continue to grow. In the context of this work, we were interested to shift the transport of the XRPL consensus messages to the NDN overlay. As such, the XRPL consensus was studied and is presented on short below, together with the identified consensus message types that pertain to the NDN overlay.

XRPL consensus algorithm. XRPL's consensus algorithm is classified as part of the Federated Byzantine Agreement (FBA) family. The two main phases of XRP's blockchain building process are called *"Consensus"* and *"Validation"* [1]:

During *Consensus*, new transactions are received and the participants agree on: the transaction set to apply over the previous agreed-upon ledger AND on the closing time of the newly created ledger. The *transaction set* is a set of transactions with a unique ID, on which the consensus protocol is to reach agreement whether to include it in the current ledger being built. *Validation* means that the validators agree on the generated ledger (validate it), based on the ledgers built by chosen validators.

XRPL validators do not explicitly know all the other participating validators, and consensus wise a validator only communicates with those from its own Unique Node List (UNL), where UNL is defined as a set of validators that an individual validator does not necessarily consider to be all honest, but instead it trusts not to collude to fraud.

Before presenting the XRPL consensus, some specific terms should be introduced first:

[1]https://github.com/ripple/rippled/blob/develop/docs/consensus.md, valid in January 2023,

- *A validator's position* is the validator's current belief on what is the current candidate transaction set to be included in the ledger being built AND the *close time*.
- *Close time:* each validator calculates its own close time when it closes the open ledger. The exact close time is rounded to the nearest multiple of the current effective close time resolution.
- *Proposal messages*, which can be: an *initial proposal* which is the initial position taken by a validator before taking into consideration other validators' proposals, and *updated proposals* where validators update their position after receiving proposals from other validators.

The main phases of the *consensus* [1], also shown in Figure 3.2, are [29]:

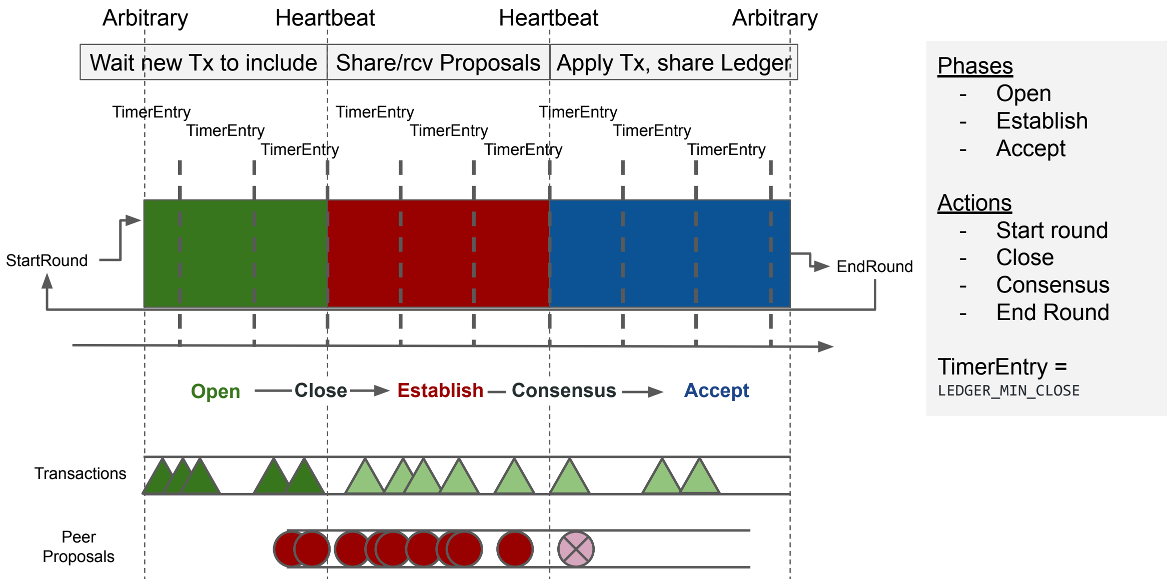


Figure 3.2: The XRPL consensus protocol. [a]

[a]https://github.com/XRPLF/rippled/blob/develop/docs/consensus.md

1. *"Open" phase* - New tx's are received. To reach as many nodes as fast as possible, a flooding mechanism is currently being used for tx propagation. The transaction messages pertain to the NDN overlay.

2. *"Close" state* - The current ledger won't accept new tx's, instead the *Consensus* protocol advances towards closing the current ledger. Tx's received after this moment will be recorded and applied to next ledger. There are multiple ways a ledger can be closed:

    - *Case 1:* In the typical behavior, the validator has received transactions in the open ledger AND more than the minimum predefined time for closing a ledger has passed.
    - *Case 2:* No transactions have been received AND an appropriately longer waiting time has passed. The longer waiting time increases the opportunity to receive and add some tx into the next ledger and avoids useless work to close an empty ledger.
    - *Case 3:* More than half of the participants closed already, which could mean that the validator is remaining behind and should *close*, to catch up.

3. *"Establish" phase* - Based on an increasing threshold for inclusion, the participant validators work towards agreeing with a super-majority of participants on the current tx set by exchanging *proposal* messages and adding or removing tx's. They also agree on the effective *close time*, which is part of the validator's position and is shared with peers in its proposals. In this phase, flooding is also being used for the propagation of the proposal messages. As such, proposal messages also pertain to be disseminated over NDN.

4. *"Consensus reached" state* - Participants agreed on the tx set to include in the current ledger. The consensus is declared when ALL below conditions are true:

    - A *minimum consensus time* was spent during *establish.*
    - At least 75% of peers proposed, OR this establish phase is *minimum consensus time* longer than the previous round's establish phase.
    - A *minimum consensus percentage* of validators (self included) have the same position.

5. *"Accept" phase* - Participants apply the agreed upon tx set in canonical order and share the result.

6. *"End round" state* - the current round is finished, and the participants move to *Validate* this ledger.

During *Validation*, the validator nodes share their results as signed messages containing the hash of the calculated ledger. These messages are called *validations* and allow participants to check if they obtained identical results; then, they can declare the ledger "final". The propagation of validation messages also uses a flooding mechanism. Validations pertain to the usage of the NDN overlay and have been used for the experimental evaluation. Validators compare their results and declare the ledger validated IF enough trusted validators agree. This number of validators is also called "quorum".

The deterministic finite automaton of XRPL consensus is presented in Figure 3.3 [34]. While the main flow is already explained above, the Figure also illustrates some failsafe mechanisms built into the protocol in the case of errors, for example when a validator for some reason finds itself working on the wrong ledger. In this case the validator would bow out of the *consensus* and work the current round in *observing mode.* During this time the validator may receive the correct ledger from the network, in which case would move to he *switched ledger* state and continue to work in observing mode. Starting next round, if all is well, the validator will switch to *proposing mode*, or if not, it will go back to *wrong ledger* mode.

Summarising, flooding is the main dissemination protocol on XRPL, an approach which ensures robustness. The main types of flooded data are *Transactions* (Tx), *Proposals*, and *Validations.* The efficiency of the propagation of these message types needs optimisation and these messages pertain to the proposed *XRPL-NDN overlay.*

3.3 Design and Implementation

This section describes *XRPL-NDN Overlay*, a solution for improving the communication efficiency in the case of the blockchains based on consensus-validation. We work on the concrete case of the XRP ledger.

NDN was chosen as overlay because it offers: i) in-network caching of data, which on large networks can lower the overall number of messages in-flight at a given moment, and

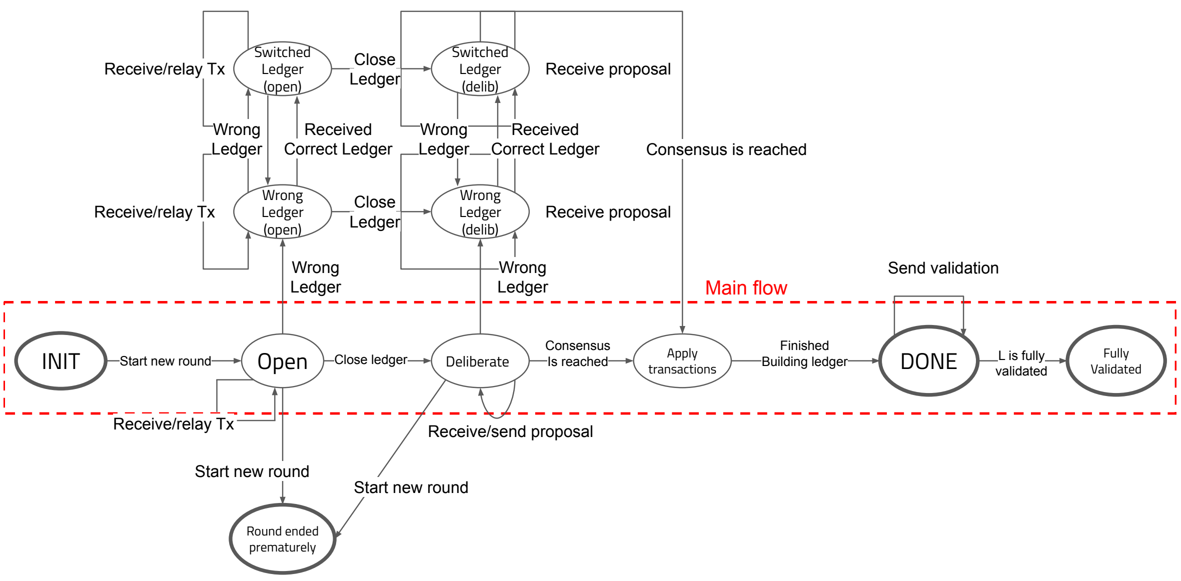


Figure 3.3: The deterministic finite automaton (DFA) of the XRPL consensus protocol.

ii) native multicasting, which could also soon benefit from mechanisms of reducing message duplicates [108].

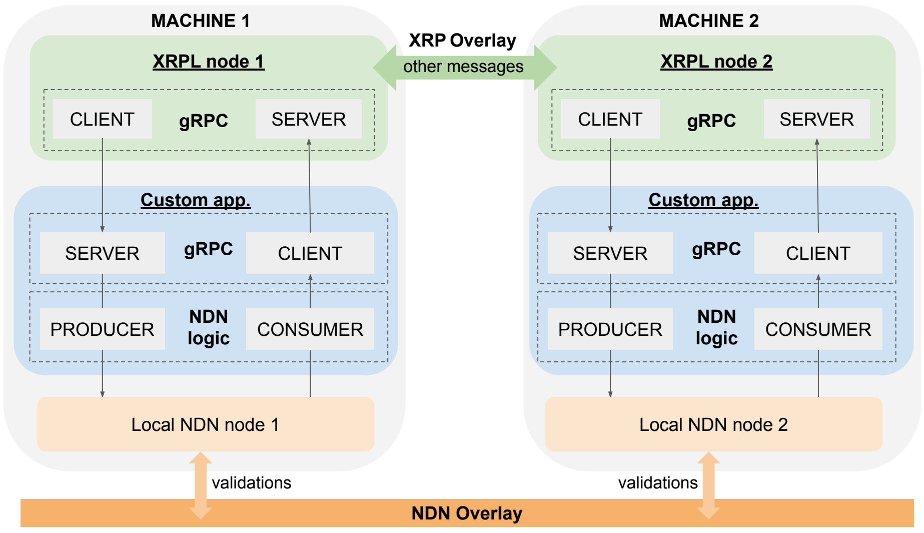


Figure 3.4: General architecture of XRP-NDN Overlay.

The architecture used by *XRP-NDN Overlay* is shown in Figure 3.4. Each XRPL node runs a gRPC server and client which enable it to send and receive validations through gRPC. On each machine running XRPL, an NDN node is also ran, enabling the XRPL node to connect to the NDN overlay. A standalone App (illustrated in blue) talks to the XRPL node through a gRPC server-client pair on the one hand, and to the local NDN node for the dissemination of validations via the NDN overlay. The same App implements various NDN dissemination models, which are described next.

With a goal to decrease the load on the XRPL nodes by decreasing the number of messages processed by them, this work seeks to answer the following questions:

Q1 What models could be used to map the XRPL consensus protocol to the NDN communication environment?

Q2 How would the models considered compare between each other and with the baseline (unmodified XRPL communication)?

With respect to Q1, four possible models for sending XRPL validations over NDN were evaluated:

1. *"Polling"*: Each validator maintains a *"sequence_number"*, i.e., to each newly created validation it associates an increasing *"sequence_number"*. The nodes interested to receive a validation from this validator, will send periodic interests asking *"what is the last sequence of your validations?"*. If the sequence is unchanged, they do nothing, and if the sequence increased, they ask for the new validation. As the interval between ledgers on XRPL is normally 3-5 seconds, a 200ms polling interval was chosen in order to ensure the propagation of any fresh validation is not sensibly delayed. The process is illustrated in Figure 3.5.

2. *"Announce-pull"*: A validator which has created a new validation, would send a multicast interest to let all nodes know the new sequence of its new validation. The interested nodes will pull the validation with the given sequence. The process is illustrated in Figure 3.6.

3. *"Advanced-request"*: On XRPL, because a *Consumer* knows in advance the identity of the originating *Producer* (a validator on its UNL), and because the interval between

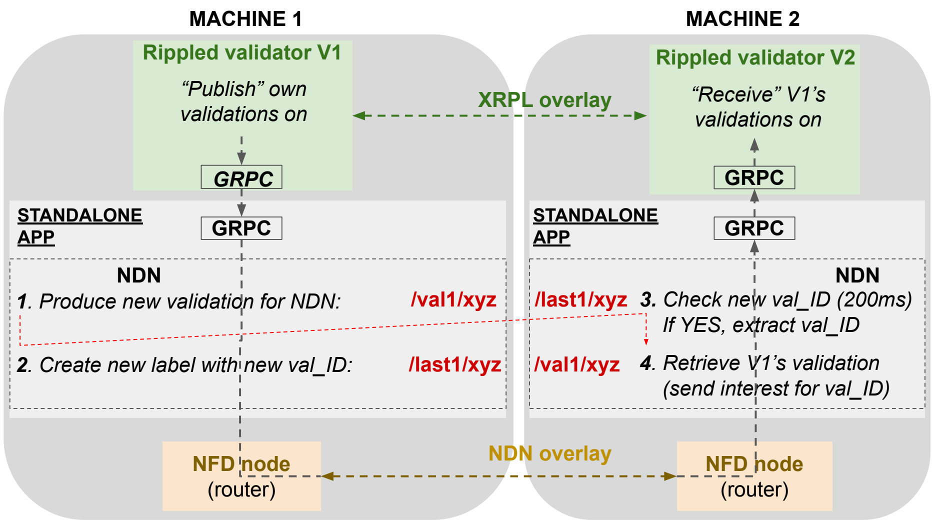

MACHINE 1
MACHINE 2
Rippled validator V1
"Publish" own validations on
GRPC
Rippled validator V2
"Receive" V1's validations on
GRPC
XRPL overlay
STANDALONE APP
GRPC
STANDALONE APP
GRPC
NDN
1. Produce new validation for NDN: /val1/xyz
2. Create new label with new val_ID: /last1/xyz
NDN
/last1/xyz 3. Check new val_ID (200ms) If YES, extract val_ID
/val1/xyz 4. Retrieve V1's validation (send interest for val_ID)
NFD node (router)
NDN overlay
NFD node (router)

Figure 3.5: The *polling* model.

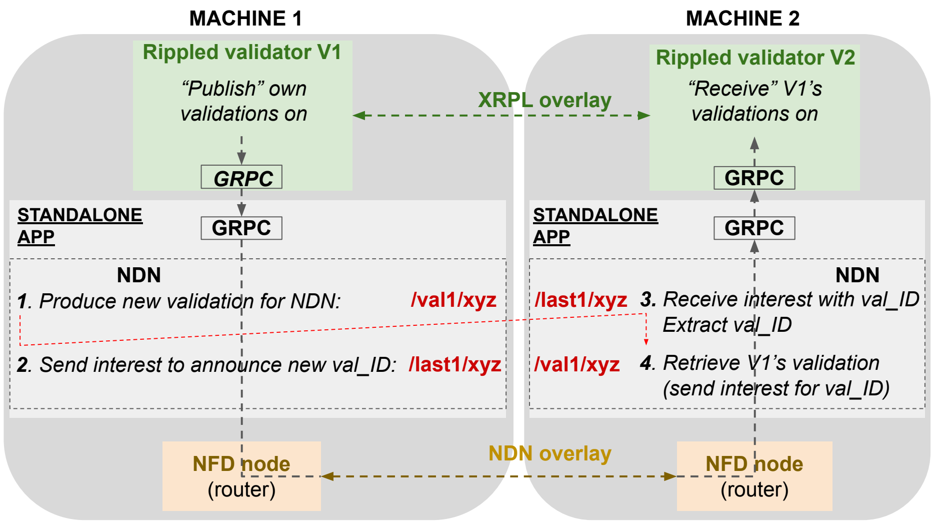

MACHINE 1
MACHINE 2
Rippled validator V1
"Publish" own validations on
GRPC
Rippled validator V2
"Receive" V1's validations on
GRPC
XRPL overlay
STANDALONE APP
GRPC
STANDALONE APP
GRPC
NDN
1. Produce new validation for NDN: /val1/xyz
2. Send interest to announce new val_ID: /last1/xyz
NDN
/last1/xyz 3. Receive interest with val_ID Extract val_ID
/val1/xyz 4. Retrieve V1's validation (send interest for val_ID)
NFD node (router)
NDN overlay
NFD node (router)

Figure 3.6: The *announce-pull* model.

validations is somewhat predictable (3-5s in real-life), it is possible to consider the announcement of a new validation made even before the validation is produced. Thus, the time required to forward the interest to source can be eliminated by proactively requesting the validation in advance, as illustrated in Figure 3.7. This pull-based approach can ensure that the data is served as soon as it is available.

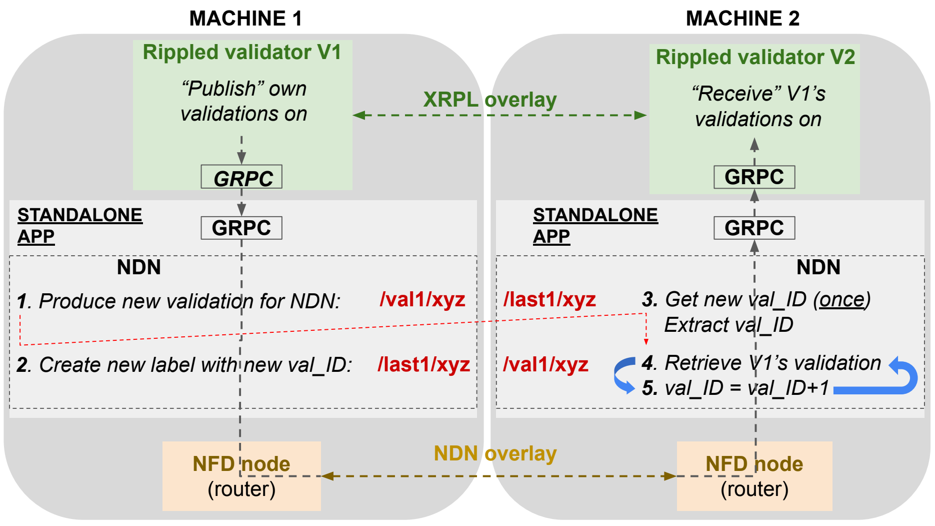


Figure 3.7: The *advance-request* model.

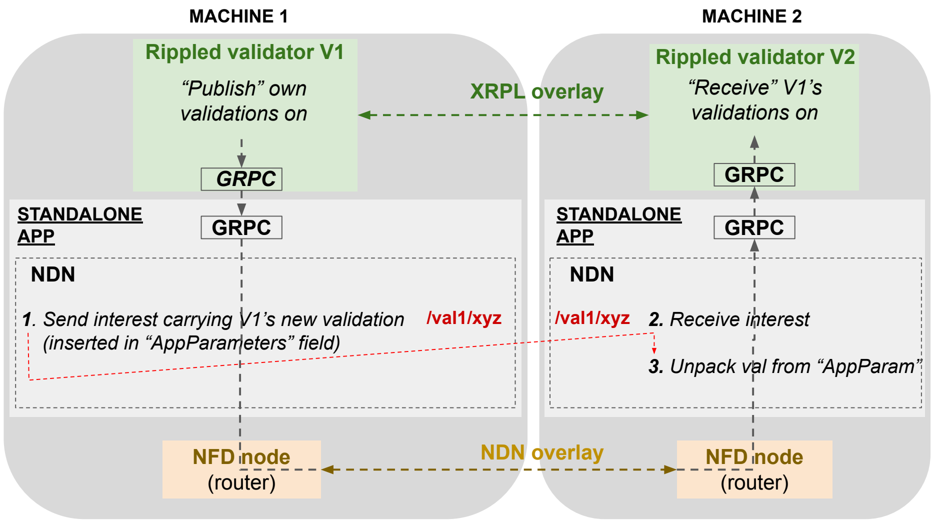


Figure 3.8: The *piggyback* model.

4. *"Piggybacking on Interest"*: We notice that it is feasible to send the XRPL validations directly over NDN Interests by encapsulating them in the field named *appParameters* [2] from the Interest packet. Normally this field is meant to carry custom extra data which could eventually be necessary to disambiguate, or help define completely, the request expressed by the Interest message, and the data format is similar to the *Data Packet* response issued by the content *Producer*. In this model, the validator which has created a new validation will encapsulate it in an Interest message and send it directly with multicast to all nodes. This solution can help reduce at minimum possible the number of messages exchanged at the NDN overlay level because for the dissemination of a validation, only one message (the Interest) is sent, instead of the regular exchanges *Consumer-Producer*, e.g. compared to the *announce-pull model*. This approach can also help latency-wise in some cases (no two way request-response here), because the data caching available in the pull model can help latency-wise especially when multiple nodes request same data on a same data path. The process is illustrated in Figure 3.8.

To answer Q2, we used the following metrics:

M1 *XRPL node load*: How do our NDN models compare with each other and with the baseline concerning the number of validations in/out at XRPL executable level? This metric is important because the number of messages at node level directly impacts the performance of the node (CPU, memory) because all messages must be processed - a decision must be taken regarding each of the messages.

M2 *Network load*: How do our NDN models compare with each other and with the baseline in regards to the amount of physical messages and bytes required to travel in order to propagate a validation to all nodes? This was measured at machine NIC level and is important because it reflects the network bandwidth usage. Other NDN traffic adds on top of the traffic of validations at the XRPL executable level.

M3 *XRPL network stability*: How do our NDN models affect the inter-arrival times of the validation messages? This metric shows if the XRPL network as a whole is working

[2] https://docs.named-data.net/NDN-packet-spec/current/interest.html, valid in January 2023

normally. The validation inter-arrival time on a healthy XRPL network is 3 to 5 seconds. We consider network stability (M3) eliminatory, which means that if a model does not behave acceptably under M3 it will not be investigated further.

Besides the four models proposed above, we also investigated:

1. The validation messages for UNL validators in the live XRPL network, presented below. This gave us information about the behavior of XRPL messaging in real-life: what is the normal state of the XRPL network in real life, in regular conditions and under regular load? For example, this gave an idea about **M3** in real-life conditions.

2. The same for a private network of unmodified XRPL validators fully connected in-between each other, which is the *baseline*.

3.4 Evaluation

To conduct the evaluation we used both a real testbed deployed in the lab, as well as the XRPL live production network. The version of XRPL used was v_1.7 for the *baseline*, and we modified it to be able to divert the validation messages through gRPC towards the NDN overlay.

To build the NDN overlay, we used the *NDNts* typescript library [110],[111]. The experiments were performed on the below three topologies, which can reveal if topology influences performance. The topologies are illustrated in Figure 3.9.

1. *Star* - seven NDN nodes linked in a star formation, of which the three nodes at the edges are also XRPL nodes. The central node is the most stressed traffic-wise so it could potentially be a bottleneck.

2. *Triangle (tri)* - six NDN nodes linked in a triangle formation. The three edge nodes are also XRPL nodes. This topology is more balanced: the three middle nodes share the traffic more fairly.

3. *Baseline* is made of three unmodified XRPL nodes fully connected. This topology is a natural choice for a fair comparison with the other two topologies: at XRP logical level

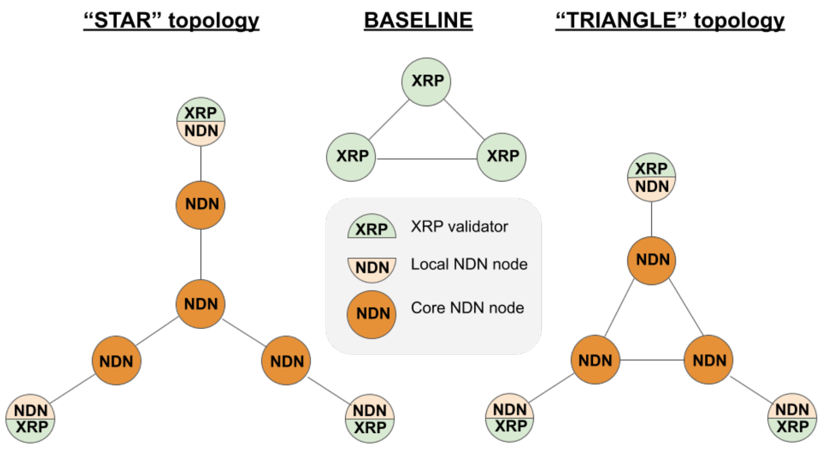


Figure 3.9: The experimental topologies.

(message-wise) it is the closest equivalent to the other two.

M1: For the unmodified version of the XRPL node executable, we used *Rippled Monitor*[3] and *Grafana* to collect statistics regarding the total number of validations coming in and going out from a node, as well as the total bytes incurred by these messages. For the modified version of the XRPL node working over NDN, we counted the number of validations in/out of the node with our own tool.

M2: We used *vnstat* [112] and *tshark* [113] to count the number of bytes/packets at the local machine NIC level.

M3: We parsed specific lines in the XRPL log and extracted the necessary info to analyze the inter-arrival times per validator. To facilitate the readability of our time-series figures, we plotted:

- in *orange color*, the *rolling mean* (rm) over the previous 20 data-points (which means generally over 1-2 minutes depending on the interarrival times).

- in *green color*, the $rm(20)$ plus 2 times the *rolling standard deviation* ($rSTD$) computed over the same 20 data-points: $rm(20) + 2 * rSTD(20)$.

- in *red color*, the same $rm(20)$ from which we substract 2 times the $rSTD(20)$, i.e.: $rm(20) - 2 * rSTD(20)$.

[3]https://github.com/ripple/rippledmon, valid in January 2023

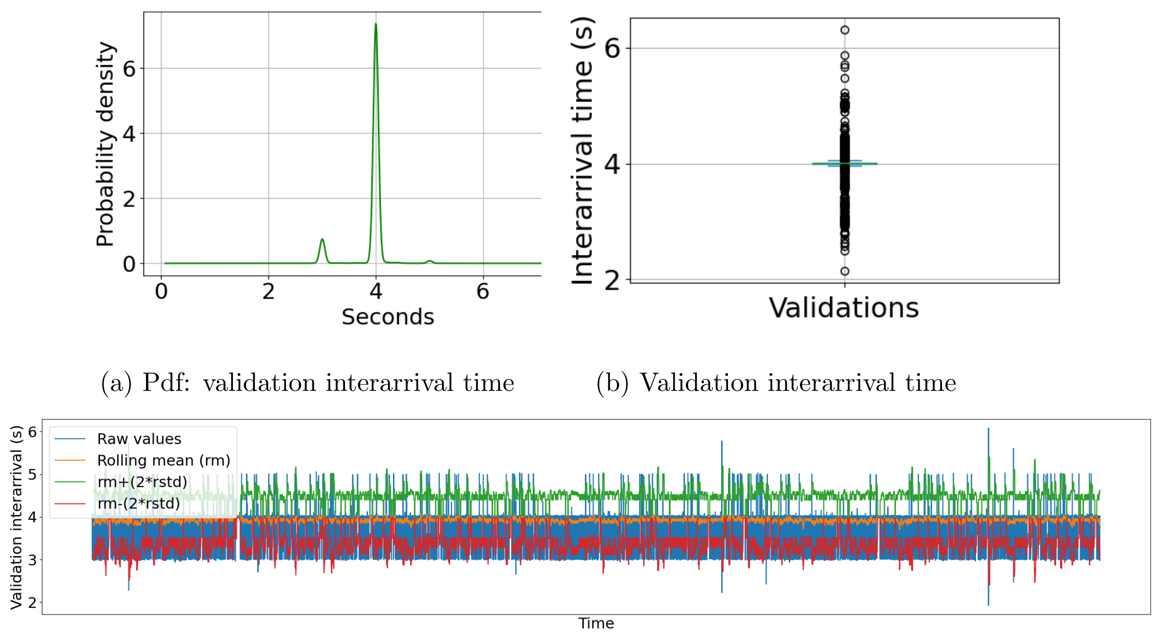
3.4.1 RESULTS

This Section presents the experimental methodology and the results, which will subsequently be discussed in Section 3.5.

3.4.1.1 Production validators on the XRPL Livenet

The main goal of analysing the behavior of XRPL validations on the production network was to get an idea of **M3** in a real-life scenario.

We deployed and connected an XRPL validator node on the live XRPL network. From this node, we listen for incoming validations from each of the approximately 35 XRPL validators on the official UNL. We record only the first received validation from each of these *trusted validators*, and drop the duplicate messages.

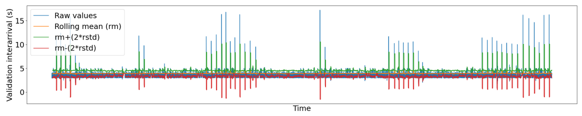


(a) Pdf: validation interarrival time (b) Validation interarrival time

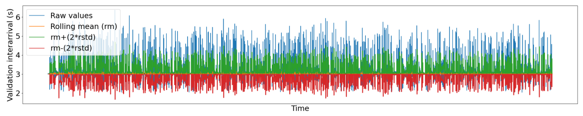
(c) Time series: validation interarrival time

Figure 3.10: Typical validation interarrival time on XRPL *livenet*.

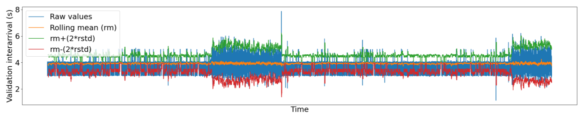
M1, M2: We didn't collect any data because in the case of the livenet, it is not possible to perform a fair comparison with our models using these metrics. The main reasons are the number of nodes involved, the topology and the real-life internet environment that we can



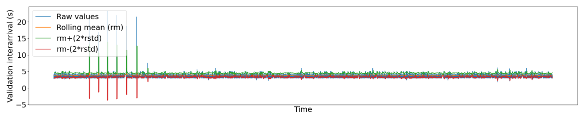
(a) "Intermittent" behavior



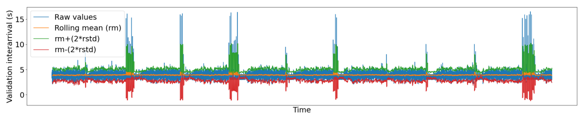
(b) "Atypical" behavior



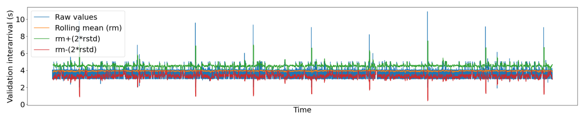
(c) "One-off" behavior



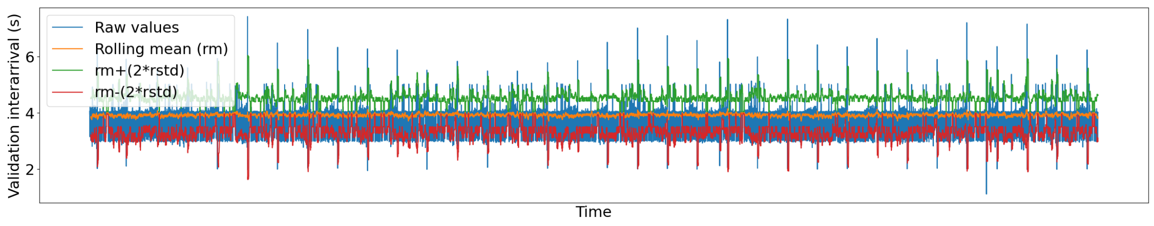
(d) "One-off" behavior



(e) "Regular intervals" behavior



(f) "Regular intervals" behavior



(g) "Regular intervals" behavior.

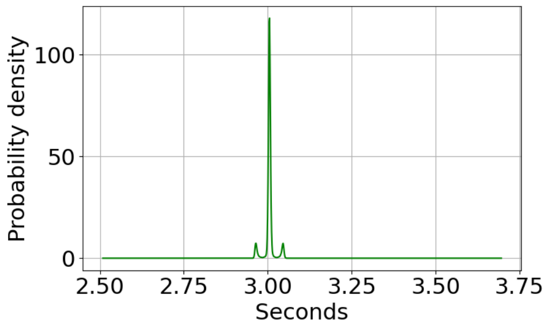
Figure 3.11: Behaviors of validators on the XRPL *livenet*, different from the typical behavior (time series).

not recreate for our models.

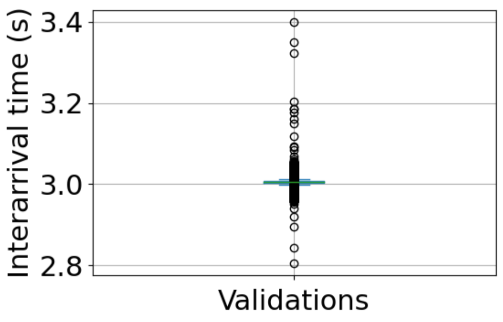
M3: We notice that for the XRPL validators which we "listen" to, while the seemingly default behaviour would be to receive incoming validations spaced at between 3 to 5 seconds (with the mean approximately around 3.92s, median around 4s, quantile(0.25) around 3.98s and quantile(0.75) around 4.02s, as shown in Figures 3.10c, 3.10a, 3.10b), there are validators which exhibit atypical, one-off, irregular or seemingly regular disruptions in the default pattern, of which we illustrate some cases in Figures 3.11g, 3.11f, 3.11e, 3.11c, 3.11d, 3.11a, 3.11b.

3.4.1.2 Baseline - private network of unmodified XRPL validators

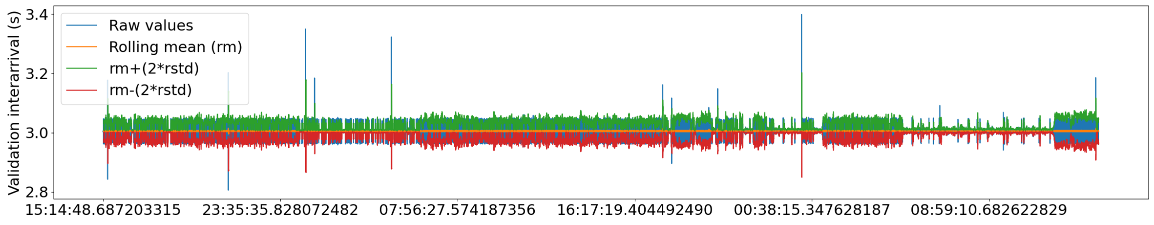
On our dedicated testbed we set up a private network of unmodified, fully connected XRPL validators. From one of these, we record as above, the intervals between the first arrived unique validations from all other validator nodes, dropping duplicates.



(a) Pdf: validation interarrival time



(b) Validation interarrival time



(c) Time series: validation interarrival time

Figure 3.12: Validation interarrival time: *Baseline* (private XRPL).

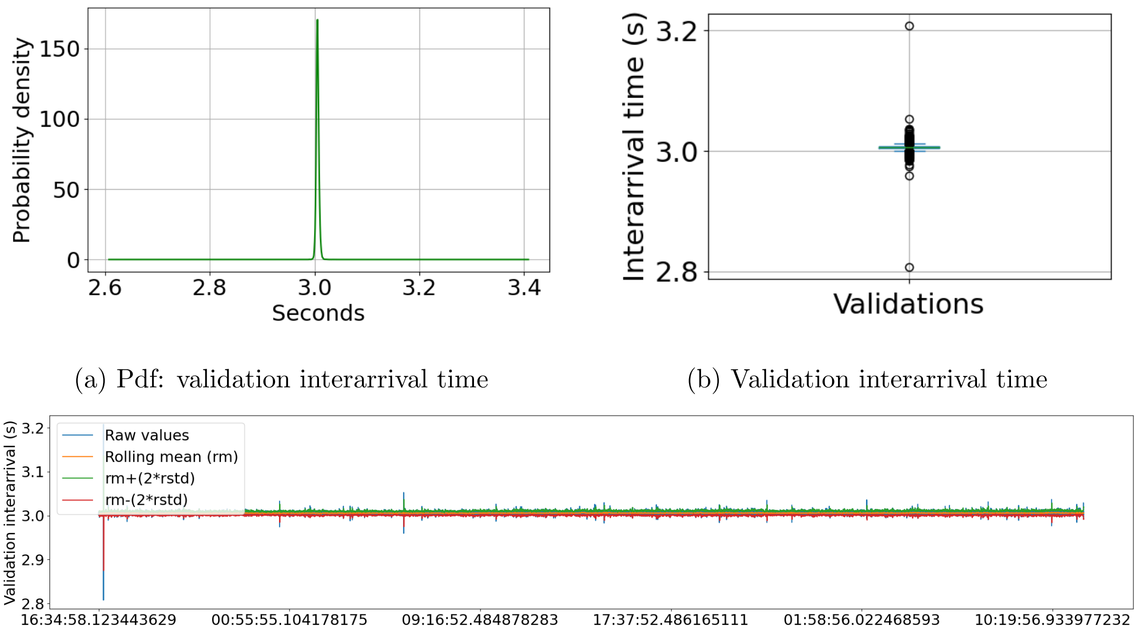
M1: Using *RippledMon* and *Grafana* we were able to compute the ratio of total validations in+out to ledgers created. It turns out that for the baseline topology from Figure 3.9(middle), there are on average 7.34 validations travelling in/out from an XRPL node to create one ledger, and a total number of 17845 validations over 2 hours.

M2: Over 10 minutes, *Tshark* recorded 14420 packets. *Vnstat* reported an average rate of 58kbit/s over 5 minutes.

M3: We notice a strong tendency for interarrival times spaced sharply at around 3 seconds with a mean, median, quantile(0.25) and quantile(0.75) all around 3.00s, as shown in Figures 3.12a, 3.12b and 3.12. These figures will be used later to compare the different models.

3.4.1.3 The "Piggibacking on Interest" model

Under M1, there was a total number of 3 validations in+out of the XRPL node, per ledger created. M2 was evaluated using *tshark*, which recorded over 10 minutes a number of 13713 packets, and *vnstat* showed an average of 80kbit/s over 5 minutes. M3 is better than the baseline, as shown in Figure 3.13.



(a) Pdf: validation interarrival time

(b) Validation interarrival time

(c) Time series: validation interarrival time

Figure 3.13: Validation interarrival time: *Piggyback(tri)* model.

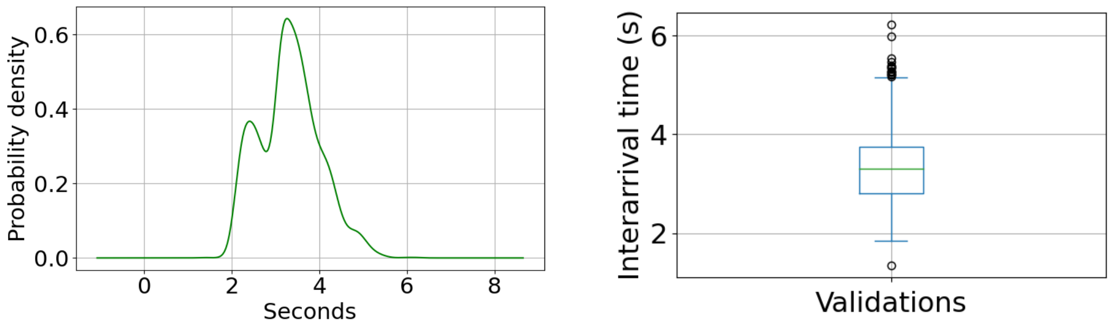
The experiments were carried out on the ”triangle” topology from Figure 3.9. The probability distribution plots show a slightly better performance of the piggyback model versus the baseline, while in regards to the number of validations processed at the XRPL node, the piggyback model is clearly better with 3 validations/ledger versus 7.34, meaning that our model is 2.44 times better for *this experimental setup*.

3.4.1.4 The ”Polling” model

This was our first validation dissemination model, mostly to see how the XRPL and NDN would work out together. We carried out the experiments related to the polling model on the *triangle* topology.

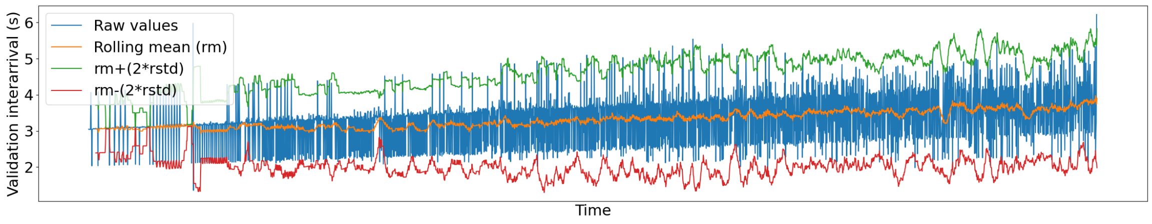
M3: The inter-arrival times from Figure 3.14 are generally not better than the baseline. This, together with the high number of messages incurred at NDN level by the continuous polling make this set-up unfeasible for a real-life usage.

Because this model performed worse than the *Baseline* and the *Piggyback* model (described below) in regards to validation interarrival times (Table 3.2), we didn't collect any further metrics (M1 and M2). However, this model could be further improved to use for example adaptive polling intervals.



(a) Pdf: validation interarrival time

(b) Validation interarrival time



(c) Time series: validation interarrival time

Figure 3.14: Validation interarrival time: *Polling(tri)* model.

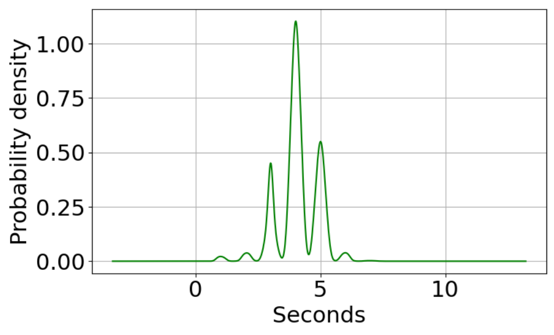
3.4.1.5 The "Announce-pull" model

We experimented with both the *star* and *triangle* topologies. This was the second model we experimented with, as an improvement over the first one.

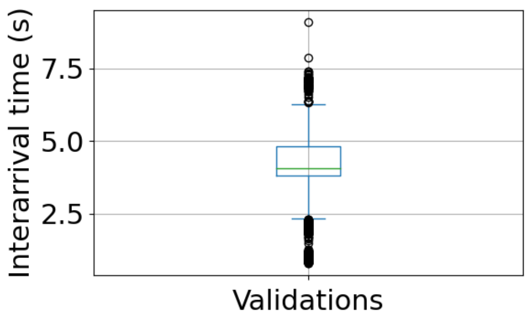
On the *triangle* topology, overall, this model showed a more stable behavior, however without getting close to the baseline regarding M3 (rm and $rSTD$), as shown in Figure 3.15.

On the *star* topology, we present our results in Figure 3.16.

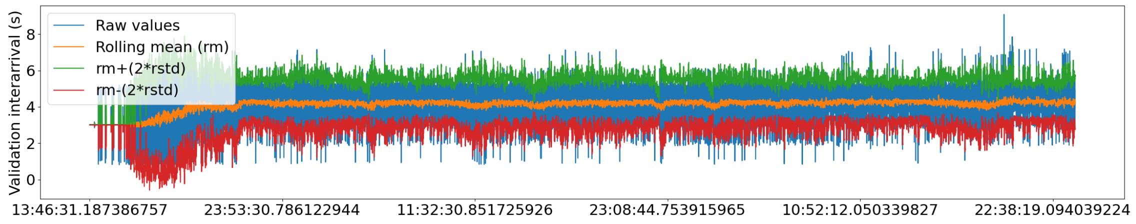
Figures 3.12, 3.13, 3.14, 3.15 show that under M3, both the *Announce* and the *Polling* models performed worse than the *Baseline* and than the *Piggyback* model with respect to the



(a) Pdf: validation interarrival time



(b) Validation interarrival time



(c) Time series: validation interarrival time

Figure 3.15: Validation interarrival time: *Announce-Pull(tri)* model.

validation interarrival times (Table 3.2). This is why, we didn't collect any further metrics (concerning M1 and M2).

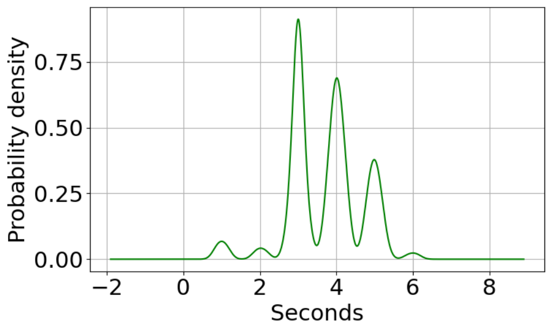
3.4.1.6 The "Advanced-request" model

The results obtained on the *triangle* topology are presented in Figure 3.17. This model was not investigated further as the performance under M3 was not satisfactory.

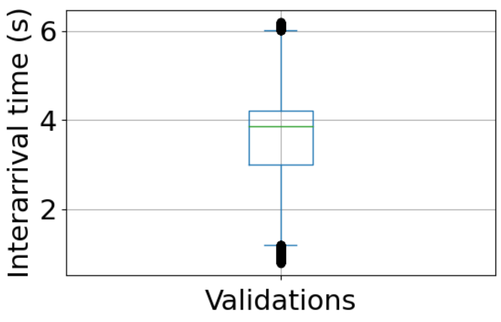
The evaluation results are summarized in Table 3.2.

Table 3.2: Experiments summary.

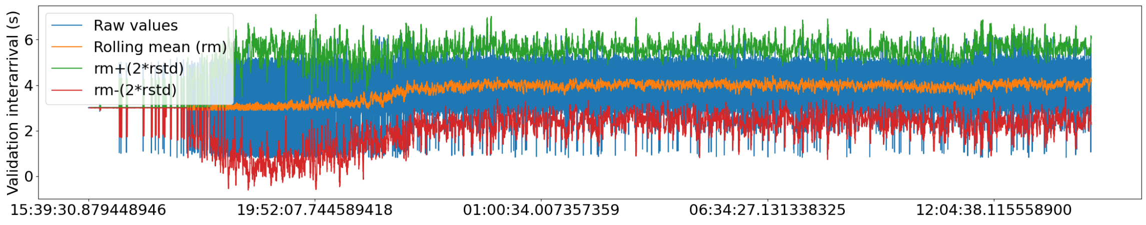
Model	**Topo**	**Val inter-arrival time**			**XRP node load**	**NIC load**		**Content Store (rates / min)**		
		q(0.25)	q(0.5)	q(0.75)	vals in+out/ledger	avg bitrate (5min)	pkt/10min	misses	hits	entries
Baseline	tri	3.00	3.00	3.00	7.34	59kbit/s	14420	N/A	N/A	N/A
Adv-req	tri	3.00	4.00	5.00	not collected	20kbit/s	11800	not collected		
Polling	tri	2.95	3.48	4.52	not collected					
Announce	star	3.00	3.86	4.21	not collected			170->790 (2h)	0	887->1520 (2h)
Pull	tri	3.86	4.07	4.84	not collected			900->1500 (2h)	0	190-785 (2h)
Piggyback	tri	3.00	3.00	3.00	3	80kbit/s	13700	785 (flat)	0	65 (flat)



(a) Pdf: validation interarrival time

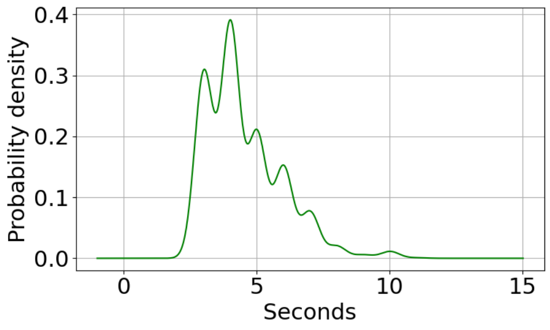


(b) Validation interarrival time

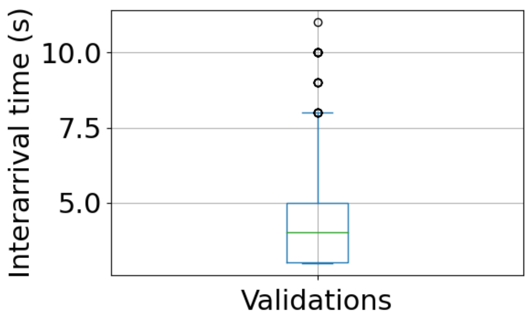


(c) Time series: validation interarrival time

Figure 3.16: Validation interarrival time: *Announce-Pull(star)*.



(a) Pdf: validation interarrival time



(b) Validation interarrival time

Figure 3.17: Validation interarrival time: *Advanced-request(tri)* model.

3.5 Discussion

According to the results, for the concrete case of XRPL validations, the most suitable solution is their encapsulation in Interest messages and dissemination with multicast. This approach uses few additional messages (the goal was to minimise the overall number of messages, and the ratio we obtained was 3 to 7 between our model and the baseline, respectively). This model improves significantly over the baseline as shown by comparing the interarrival times, while ensuring robust dissemination and low latency. Also, by retaining the data push model, the impact on security should be minimized.

The topic of blockchain message dissemination over NDN has also been studied in peer-reviewed work like [47] which proposes a design and implementation for propagating the ETH tx's and blocks over NDN. However, the design is focused on PoW blockchains, with a concrete case for ETH. Nevertheless, the needs of consensus-validation based DLTs are fairly different from those of PoW DLTs, to require separate consideration. For example the size of the consensus messages in XRPL (proposals and validations) is much smaller than the size of ETH blocks, and XRPL uses the concept of UNLs where, strictly from a consensus perspective, a validator needs only receive messages from those other validators in its defined UNL. Moreover on NDN, the data can be signed and dated by the producer, which in XRPL case is already known (UNL validators are known from a logical point of view), making some types of attacks discussed in this paper not applicable for our work on XRP-NDN Overlay. In [47], NDN data sync models are dismissed for various reasons, including security. While for XRPL's validations for example, the sync vector can be easily constructed, these models could indeed add unnecessary traffic hindering scalability, and also, they were not designed with Byzantine failures in mind.

The authors of [47] also propose an *announce-pull* model for both tx and block propagation, arguing that this can benefit from in-network caching and multicasting to avoid redundant traffic. For the case of XRPL, because a *Consumer* knows in advance the identity of the originating *Producer* (a validator on its UNL), and because the interval between new validations is somewhat predictable (3-5 seconds in real-life) this model can be simplified to consider the announce already made, and issue pull requests in advance. Moreover, the authors use the ETH P2P overlay to broadcast the creation of a new block and then NDN to pull the block after learning about it. The problem on XRPL is fundamentally different and consists of a very large number of messages - a result of the flood mechanism when the network is scaled up. For XRPL, this number needs to be minimised. As such in this work we have been searching for, and proposed, a paradigm suitable for the XRP Ledger.

XRPL *Proposal* messages share similar characteristics to *validations* and could use the same dissemination model. Because of specific use cases such as trading or high frequency trading which need that *transactions* propagate as fast as possible such that they are included in the earliest possible ledger, the transactions could also use the *piggybacking* on Interest model to propagate. This approach however, might be subject to poisoning attacks and thus require additional mitigation measures, such as in-flight transaction verification, auditing, or node scoring.

The XRPL consensus leverages the concept of UNLs where a validator may want to be interested to hear only validations from nodes on its own UNL. Currently in production, only two largely overlapping UNLs co-exist. Using more UNLs in production will not impact NDN traffic on any local NDN node since the local NDN nodes can be independently set to also relay any other intended traffic at NDN level.

The experiments were carried out on a real testbed deployed in our lab and on the live XRPL network. We used the original XRPL code which required a significant effort to integrate NDN. The code is open source and can be found on Github [4]. The experimentation was limited to the scenarios and topologies reported.

[4]https://github.com/FlavScheidt/sntrippled, valid in January 2023

4

Performance Monitoring and Evaluation

This chapter describes the work on network monitoring for the XRP Ledger, and on system-level non-intrusive monitoring and instrumentation of application performance.

In order to monitor an XRP Ledger node at networking level, several questions had to be answered:

- How can we build a Testbed?
- What are the assessment criteria? number of messages?
- What is the impact of different optimisation methods for message dissemination?

On the other hand, we have different implementations of the Interledger protocol specification, where for the same specification, different design decisions for different software architectures have been taken. The design decisions can potentially lead to differences in application performance. We wanted to see if a different performance can be noticed, where does it come from, and how could we achieve such insights. We chose eBPF because it enables non-intrusive monitoring and instrumentation at system level. This approach lets us see, in a detailed way, how the functions are implemented and how are they called (system calls).

4.1 Network monitoring for the XRP Ledger

The goal of this work was to monitor and evaluate the performance differences between two different approaches for the dissemination of XRPL consensus messages. An unmodified version of XRPL implementing message flooding was compared with a modified version implementing a message relaying reduction mechanism called *"squelching"*.

In the *Squelching approach*[1], based on defined criteria, each node selects five of its neighbors and then squelches the rest of them for a random time. The node sends Pause/Resume commands to some of its neighbours, to control the flow of messages. The nodes keep up to T seconds record concerning the status of their peers. By reducing the number of peers with which to share messages, the total number of messages circulating in the p2p XRPL network is reduced while maintaining the overall *"flooding"* approach and to some extent its advantages : guarantees for message dissemination and network robustness.

In order to do the evaluation, a real-life environment was sought, in order to simulate the real-life conditions where nodes are spread among different geographical locations.

Grid 5000[2] is a large-Scale HPC platform, a testbed with interconnected sites in France and Luxembourg. It features a large amount of resources: 15000 cores, 800 compute-nodes and 10 GB Ethernet links. Grid 5000 offers the possibility of reconfigurable and controllable experimentation with advanced monitoring and measurement features. It was chosen because of its geographically distributed nodes and the configurable testing environment.

To deploy the XRPL network on Grid 5000 and perform the evaluation, we used a previous work, the BlockZoom [114] Tool, which is a large-scale blockchain testbed developed to run on top of Grid 5000. It offers a reproducible environment for experimentation with DLT and smart contracts. The behavior of different blockchains and the performance of applications can be evaluated at a life-like scale for different configurable scenarios. The tool has two main sections: the framework for the blockchain orchestration and the module for configuration

[1]https://xrpl.org/blog/2021/rippled-1.7.0.html, valid in January 2023
[2]https://www.grid5000.fr/w/Grid5000:Home, valid in January 2023

of experiment parameters like: site location, number of nodes, duration, operating system, blockchain platform, smart contract and workload generator. The code for BlockZoom is available on Github[3].

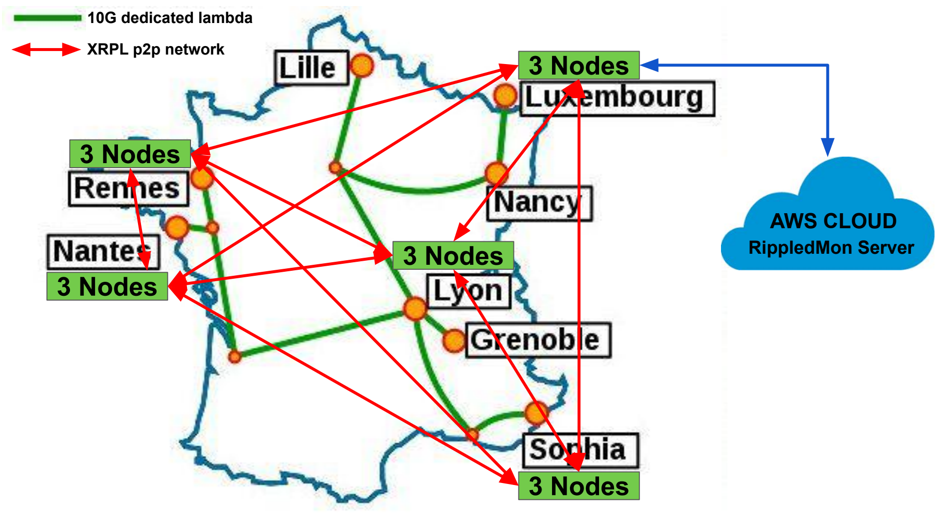


Figure 4.1: XRPL network monitoring testbed deployed on Grid 5000.

The Baseline used for the evaluation was an unmodified version of XRPL, which was compared with a modified XRPL[4] implementing the Squelching mechanism.

The evaluation steps were:

- Baseline evaluation (XRPL v1.6).
- Squelching evaluation (XRPL v1.7).
- Results analysis.

The deployment steps were:

- Integrate XRPL to our BlockZoom Tool for deployment.
- Deploy the testbed nodes on Grid 5000.
- configure the validator nodes and configure the network among peers.

[3]https://github.com/wshbair/BlockZoom, valid in January 2023
[4]https://github.com/XRPLF/rippled/pull/3412, valid in January 2023

• Deploy and configure the statistics tool *Rippled Monitor*.

The testbed comprised of 15 XRPL nodes evenly spread over five *Grid 5000* site locations, meaning three nodes per site. The consensus quorum was set to six (in real life not all nodes on the production XRPL network are validators). The nodes boot with no transactions being generated and after a preset time, a number of 1000 transactions per site was sent in parallel.

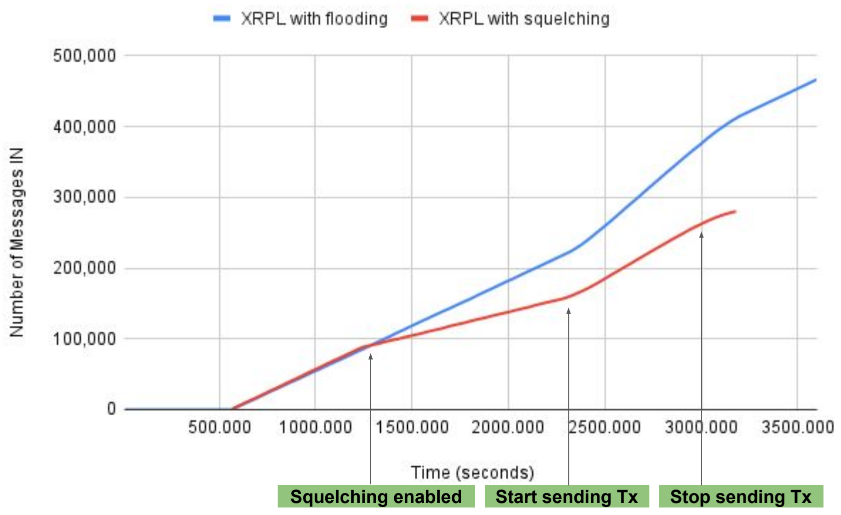


Figure 4.2: Unmodified XRPL vs XRPL with squelching: total number of Messages IN.

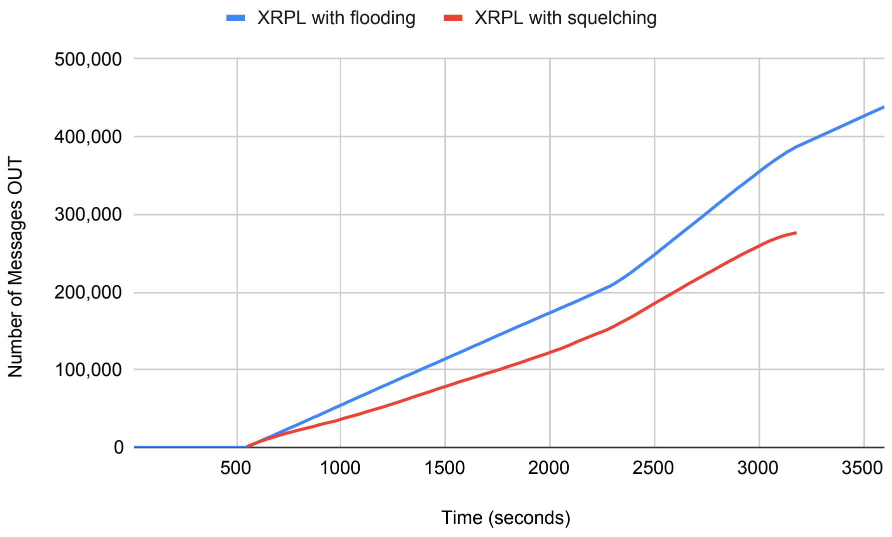


Figure 4.3: Unmodified XRPL vs XRPL with squelching: total number of Messages OUT.

The comparative results are presented in Figures 4.2, 4.3, 4.4, 4.5. The annotated Fig-

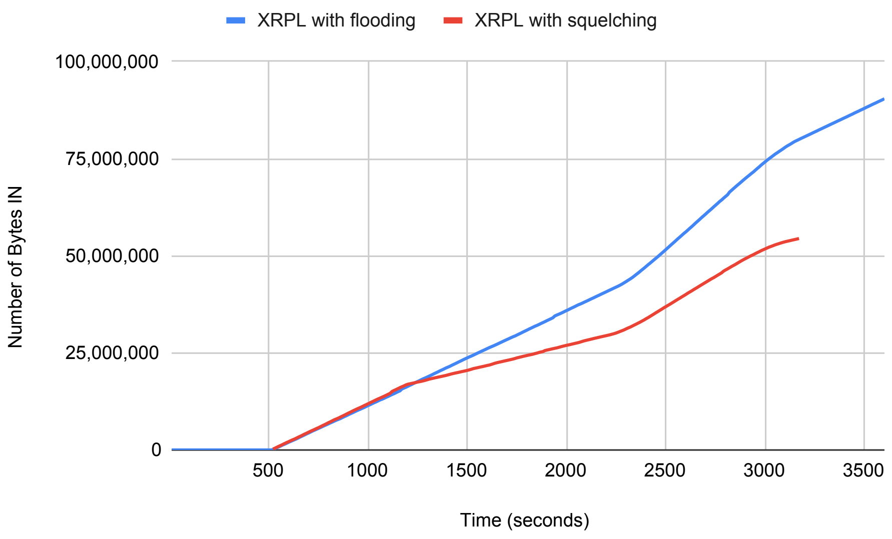


Figure 4.4: Unmodified XRPL vs XRPL with squelching: total number of Bytes IN.

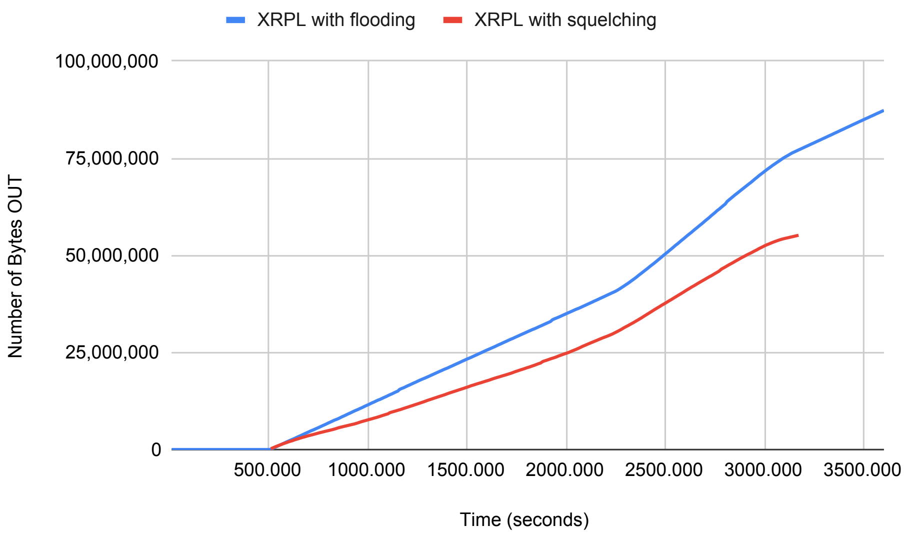


Figure 4.5: Unmodified XRPL vs XRPL with squelching: total number of Bytes OUT.

ure 4.2 facilitates the discussion of the results: both XRPL versions start similarly, until the squelching mechanism is activated on the modified version (annotated as "*squelching enabled*"). From this moment it is possible to see a sensible reduction in the total number of messages. The number of messages changes again when the transactions start to flow through the network - annotated as "*start sending Tx*" and "*stop sending Tx*".

Conclusion. The evaluation showed, as expected, that the messaging efficiency of the squelching approach is sensibly better than flooding. Nevertheless, the squelching technique raises additional questions concerning the security, resilience and robustness of the XRPL network.

4.2 Using eBPF for non-intrusive performance monitoring

The requirement to monitor computer software at different levels of the software stack appeared seamlessly with the introduction of computers in industry. Monitoring helps and is the only way to diagnose different types of problems or anomalies [115]. Nonetheless, this activity is not a new requirement or a new problem. Cloud service providers require to monitor further and at scale what is happening in their data center [116]. What has changed in the last few years is that new tools at the operating system level and new tools to package and deploy software appeared (e.g. micro-services). This has been made possible thanks to Linux kernel evolution (e.g. namespace, cgroups) [117]. However, the non-intrusive management of containers relies on the observability of the underlying operating system. For this reason, we explore the observability capability of the Linux kernel offered by eBPF. To do this, we analyze the potential of eBPF-based tools, which offer unique capability. Beyond networking functions, eBPF instruction set allows for monitoring in-kernel IO subsystems, i.e. tracing, analytics, and security functions. We specifically assess the added-value of the extended Berkeley packet filter tracing framework. Our contribution is the experimental study of eBPF-based tools in the context of non-intrusive profiling and diagnoses of a user-space application in production. Indeed, all parties hosting and/or operating an application cannot rely on predefined metrics externalized by developers. Moreover, to find root causes after an incident[5], cloud users may also want to monitor the internals of the system.

Therefore, we address here the challenge of monitoring a new generation of user-space applications at the deepest level without any support from the application.

The structure of this chapter is as follows. The Section 4.2.1 discusses the state of the art and the capability of eBPF for networking and beyond. Section 4.2.2 presents the eBPF

[5]https://landing.google.com/sre/sre-book/chapters/monitoring-distributed-systems, valid in January 2023

tracing tools against process isolation mechanisms. Section 4.2.3 presents and analyzes information extracted for Interledger Connectors. This Section explains also the rationale behind the design of our performance tool-chain. Section 4.3 concludes.

4.2.1 Background and related work

In the last few years, BPF has evolved with the extension of its instruction set. BPF becomes the extended Berkeley Packet Filter (eBPF), the Linux subsystem making possible the safe execution in the kernel of untrusted user eBPF programs [118]. Since, eBPF is completely redefining the scope of usage and interaction with the kernel. It offers the possibility to instrument most parts of the kernel. eBPF adds the ability to inject code at specific tracepoints. This goes from network tracing to process or I/O monitoring like proposed in the I/O Visor project[6]. eBPF does not require a new kernel module in comparison to other tools (e.g. LTTng or SystemTap). eBPF is by default in the Linux kernel. Metrics can be collected by attaching to the following possible points: kernel functions - kprobes; system calls - seccomp; userspace functions - uprobe, and tracepoints. Thanks to SECure COMPuting with filters (seccomp), eBPF changes also the way we can perform security monitoring in the system. Like explained in the Linux kernel documentation, seccomp enables processes to specify filters for the incoming system calls. The filter is expressed as a BPF program. For instance in [119], the authors use it to spot malicious system activities. Instead of using packets as traditional applications to intercept communications, they monitor network activities by exploiting trace points and Linux kernel probes. Here, we explore these capabilities to trace and profile user-land applications.

Because of its origin, eBPF is already used heavily for networking subsystems [120]. In this context, eBPF is used in conjunction with the eXpress data path (XDP) framework. XDP has been created to process packets before they enter the kernel, unlike the traffic control subsystem (TC). TC operates in the network stack, which means that the packet has been already processed by the kernel. Indeed, after a packet passes the XDP layer, the kernel allocates a buffer to parse the packet and store metadata about it. Then, to access the packet, the eBPF program uses a pointer to *sk_buff*, and not *xdp_buff*, which is not in

[6]www.iovisor.org/, valid in January 2023

the kernel. Nonetheless, the primary goal of these frameworks is to perform efficient switching, traffic classification [121], virtualized networks [122, 123], routing, traffic generation or communication optimization [124]. For instance, in [125] the main technical contribution of the authors is to show how this nascent technology can be used to not only build in-kernel programmable VNFs but also how to interconnect them on a single system. Indeed, the Linux kernel limits the size of eBPF programs to 4096 instructions or 32 kbytes, i.e. 4096 instructions of 64 bits. To get around this instruction limit imposed by Linux for security, Tail-calls are possible. This means that one program triggers another filter/program to create a chain of eBPF programs. In this area, Polycube is a framework for creation and deployment of custom virtual network functions [7].

Another new feature offered by this tool allows the authors of [126] to deal with the increase in encrypted traffic. To obtain access to the clear text payload in the case of applications with end to end encryption they propose to use the recently merged kernel TLS functionality. The method does not require any data decryption and re-encryption, and reduces latency and overhead. However eBPF/XDP is not the only system which facilitates programmable packet processing [127]. The Data Plane Development Kit (DPDK) is the main alternative to use accelerators (hardware or software) to manage transport protocols of the future and minimize the impact of the network service chain [128]. The approach of XDP is opposite to bypassing kernel. Next, like explained by the authors of [129], the monitoring of network traffic is traditionally using packet analysis. While many times useful, it does not help for detailed visibility in the case of virtual systems and containers. Like they suggest in their work, a merge between system and network monitoring is required. That is why, we explore the limits of eBPF to monitor system-wise a containerized user-space application. Indeed, most of the research work on eBPF is related to the primary goal of the tool, i.e. packet processing. Therefore, talking of eBPF without talking about packet processing was until recently not relevant. However, eBPF is now valuable for network filtering and security monitoring, but also program debugging, tracing or profiling.

With regards to the opportunity for Cloud infrastructures, cloud data centres of today have providers at the physical level and customers at virtual level. To see the load patterns

[7]https://github.com/polycube-network/polycube, valid in January 2023

and detect malfunctions [130], they monitor both at the hardware and at the software levels, which offers opportunities for alerting, resource allocation, and visualization [131]. The added-value of eBPF is performing those tasks for applications in production environment, even when micro-services are used. Indeed, in [132] the authors explain how micro-services are challenging the classical methodology and performance tools. Unlike traditional client-server applications, resolving performance issues requires to determine which micro-service is the root cause of performance degradation. The major contribution of the authors is their benchmarks, which unfortunately does not consider payment or micro-payment infrastructure, like Interledger [133].

4.2.1.1 Profiling and Tracing tools

The recommended front-end for using BPF tracing framework is the BPF Compiler Collection (BCC). BCC was created by Brenden Blanco in April 2015 but the work of Brendan D. Gregg is now more than influential [134]. Performance tools allow performing two major tasks:

- Tracing, to report when events occur over time.
- Profiling, to report the number of occurrences of each type of event tracked.

There are many ways to perform tracing in Linux [135], from the original Linux tracer *Ftrace* [8], to *perf* profiling tools. Over time, more complex kernel tools like BPF, eBPF, or LTTng [9], SystemTap [10] and Dtrace [11] appeared [136]. The most intrusive method uses only static instrumentation: tracepoints and user application statically defined tracepoints (USDT). However, this has several implications. First, this required access to the source code to add the user markers (USDT). Then, in any case, it requires recompilation to activate pre-defined marker. Second, user markers will have to be maintained over time based on the evolution of the code base. However, eBPF opens a new world of possibilities with kernel and user level dynamic tracing (kprobes and uprobes). For instance, in Java the dynamic instrumentation known as dynamic tracing enables tracing of functions in running binaries

[8]https://www.kernel.org/doc/html/v5.0/trace/ftrace.html, valid in January 2023
[9]https://lttng.org/, valid in January 2023
[10]https://sourceware.org/systemtap/, valid in January 2023
[11]http://dtrace.org/blogs/about/, valid in January 2023

without restart. Nonetheless, one downside of this approach is the continuity of service over time because instrumented functions can be removed or renamed depending of the maturity of the software interface observed.

eBPF being introduced, we present now the mainstream front-ends for using it, which have been disseminated in a conference organized by the Linux foundation [137]. Moreover, we specifically used them in the context of our experimentation.

First, the **BPF Compiler Collection** includes a BPF library and interfaces for writing programs based on Python, Lua and C++. BCC[12] front-end is a toolkit suitable for complex tools with a bound scope or for agents. In this respect, BCC provides a large set of predefined tools. Brendan just released most of his work in a book [138] where he presents all these tools. An interesting fact is the audience targeted by the book, i.e. system administrators and reliability engineers. This confirms the relevance of the tools for cloud providers or in-house critical IT infrastructure. BCC requires a program built in three parts:

1. Importing the dependencies to use the eBPF framework.
2. The eBPF program always written in C-like language and stored in a variable.
3. The processing script allowing to load, execute and retrieve data from the eBPF program injected in the kernel.

Second, the *Bpftrace* for quick instrumentation (e.g. for the detection of zero day vulnerabilities). Bpftrace[13] enables the creation of new metrics by decomposing them into distributions, or logs per event. Ultimately, the tool helps to uncover blind spots. Bpftrace was created by Alastair Robertson [139]. An LLVM based backend compiles the scripts into BPF bytecode. It interacts with Linux BPF system through BCC and also kprobes, uprobes, and tracepoints Linux tracing. The language is based on C, awk, and previous tracers like DTrace, SystemTap. Bottom line, Bpftrace is suitable for quick investigations and small scripts.

Third, the *Performance Co-Pilot* (PCP) provides a range of services that are used to monitor and manage system performance. PCP [14] is a system-level suite of tools for perfor-

[12]https://github.com/iovisor/bcc, valid in January 2023
[13]https://github.com/iovisor/bpftrace, valid in January 2023
[14]https://pcp.io, valid in January 2023

mance analysis. It has been field-tested in RedHat distributions and provides a framework. PCP uses a daemon and relies on PMDA (Performance Metric Domain Agent) [15] to collect metrics.

eBPF collects the data in the kernel and transfers it in the user-space thanks to BPF Maps. This is a key/value store inside the kernel. This allows to collect metrics at each iteration of the eBPF program. Currently, one BPF program can directly access up to 64 maps. As such, eBPF is changing drastically the capability of the Linux kernel [140, 141].

4.2.2 Non-intrusive monitoring and profiling

Today performance monitoring for Cloud service providers is complex; while they do not have control of the application layer, they must provide the technical means to align with new regulations such as data protection in Europe. Moreover, containers have become a commonly-used method for deploying services on many operating systems, providing short application start-up times, resource control, and ease of deployment. Consequently, Cloud service providers have to deal with Continuous integration (CI) and continuous delivery (CD), application elasticity, and distributed data processing workloads.

4.2.2.1 Container isolation

With the release of version 0.9, Docker.io has dropped LXC as the default execution environment, replacing it with their own libcontainer. Consequent investments around containers technology point out interesting opportunities for research to update the way containers are managed. The isolation boundaries of a container can vary. In the case of Linux, namespaces and cgroups are used to set those boundaries. The first meaning that we attach to the principle of isolation implies protecting each process from other processes within the operating system. In consequence, this segregate the memory space of process A from the memory space of process B. To enforce this, Docker container and other container technologies use a collective noun for a group of isolation and resource control primitives. Moreover, after the rise of Platform-as-a-Service and Software-as-a-Service, Function-as-a-service (FaaS) is the new model to run code in the Cloud. A factor pushing this model is the low running cost. Indeed,

[15]https://manpages.ubuntu.com/manpages/xenial/man3/PMDA.3.html, valid in January 2023

it is cheaper to fire-up and run specific application functions only when needed. Nonetheless, despite fancy names, all of this relies on the container, and it has become difficult for the service provider to gather performance indicators and guarantee security. For instance, the Google gVisor [16], the IBM Nabla [17] secure container systems, and others propose a promising approach to containment [142]. The authors propose X-Containers, a security solution to isolate cloud native, single containers. They explain that replacing some of the isolation primitives with local system call emulation sandboxes is not enough guarantee.

In this context, where isolation of software components is strengthening, we advocate for non intrusive performance monitoring with in-kernel facilities. Security and performance are the primary concern but gathering indicators for performance or to guarantee security becomes a challenge. On the one hand, user level tools are required to monitor the full stack of their application in an outsourced environment and, on the other hand, more advanced sysadmin tools are required to monitor black-box containers. Indeed, it is not clear yet how dynamic instrumentation will be able to safely penetrate the containment boundaries where the business code is. The added-value of non-intrusive monitoring has been proven in many other works [143], [144], [145], [146], [147] and [116]. For instance, Cilium[18] leverages eBPF in a cloud native and micro-services context. However, *Kubectl-trace* is a command line front end for Kubernetes which allows running bpftrace in Kubernetes cluster machines [148]. While currently, using Kubernetes or similar it is not possible to activate capturing of metrics during the deployment of containerized applications.

4.2.2.2 eBPF integration

Our goal is to dissect the behavior of Interledger connectors. This means that at minimum, we want to perform the full scope of classic system resources monitoring, e.g. CPU, memory, TCP connections. Then, two other requirements are the possibility to filter traffic at the XDP level and the possibility to filter the metrics gathered per container. For this, we use the PID or a port number for a container running a specific network service.

[16]https://github.com/google/gvisor, valid in January 2023
[17]https://nabla-containers.github.io/, valid in January 2023
[18]cilium.io, valid in January 2023

The *key features* that we consider for our tool chain are:

F1) The extensibility of the tools used.

F2) The possibility to store collected data. We want the ability to archive metrics to perform a posteriori analysis.

Like explained previously, BCC and bpftrace are meant to be used for the creation of higher level tools such as eBPF_exporter [19] or Performance Co-Pilot (PCP). At first, we experiment with eBPF_exporter, open sourced by Cloudflare, to extract metrics and feed the main Prometheus server [20]. The server collects and stores time series data. It is supported by the Linux foundation and uses special exporters for the eBPF_exporter. Prometheus has a data model that is multi dimensional. The time series data is identified by the name of the metric and pairs of key-value. Prometheus provides a query language named PromQL.

Then, we were not satisfied with the module provided by the eBPF_exporter. Indeed, we did not have the possibility to monitor the garbage collector or any facility to add IP filtering. Between modifying the parser of the YAML files used by the exporter written in Golang and switching to PCP, we decide to switch to PCP. Another argument in favor of PCP is that it provides support for the creation and management of archive logs.

Figure 4.6 shows the selected tool chain. PMDA modules are in charge of injecting eBPF bytecode in the Kernel. Then, PMDA modules gather and transmit collected metrics to a PCP web daemon. Next, a visualization layer is required. In this context, we try the following tool composition for the realization of our performance monitoring tool chain:

- eBPF Exporter: integration with Prometheus and Grafana.
- Vector and Performance Co-Pilot (PCP): for remote BPF monitoring.
- Grafana with PCP: for remote BPF monitoring.

Regarding data visualizations, we experiment with Grafana and Vector. Built on top of PCP, Vector enables real time visualisation and analysis of metrics like CPU, memory, storage and network at system and application level. Application profiling is possible through flamegraphs. PCP web daemon is connected to the metric collector daemon to provide Vector

[19]https://blog.cloudflare.com/introducing-ebpf_exporter/, valid in January 2023
[20]https://prometheus.io/, valid in January 2023

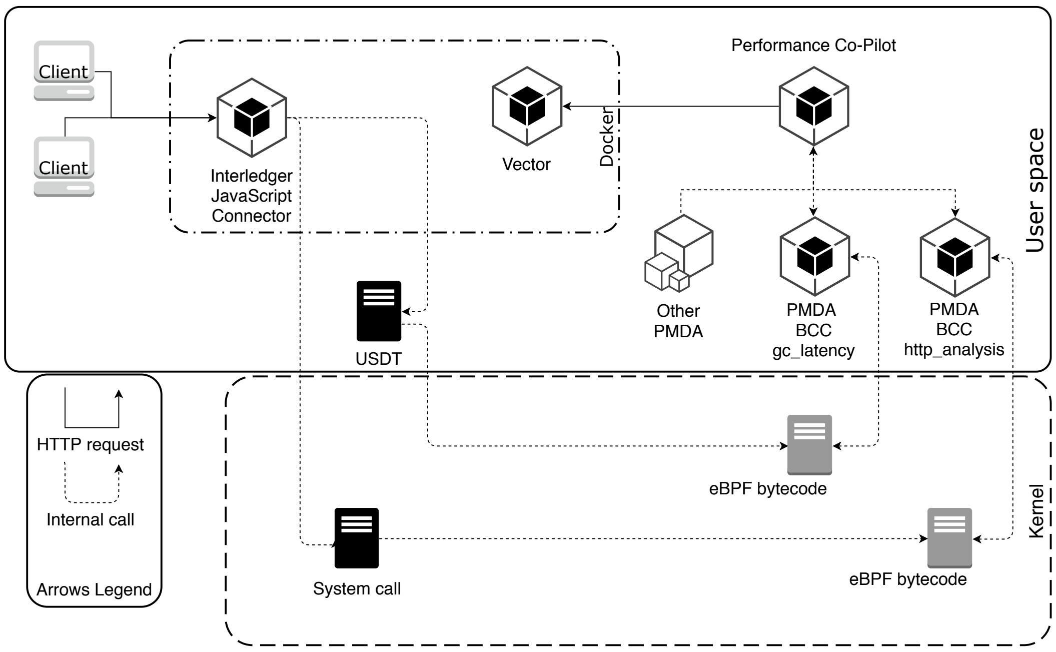


Figure 4.6: Overview of the performance monitoring setup.

with performance data. This is a lightweight model because data is not stored along browser sessions and no metrics are aggregated between hosts.

Finally, in spite of our critics on static program instrumentation, we try user application statically defined tracepoints as shown in Figure 4.6. This approach is simple, easy to parse but completely intrusive and lacks features such as typed arguments. This is only suitable for the development and debugging phase.

4.2.3 Profiling and tracing of Interledger

As explained before, on-line monitoring is required in our case. The default toolkits do not fulfill our needs. That is why we have to create new eBPF programs, which will be integrated in our monitoring solution based on Vector and PCP, like presented in Figure 4.6. We add dedicated modules for the purpose of monitoring the connector during its execution. For each module, parameters can be set through a common configuration file for the BCC PMDA of PCP. Therefore, we filter the monitoring to the connector itself, by setting the ports, IP address and PID of the network stream to be monitored. It is even possible in some cases to limit to a process selected by regular-expression matched name. Each module was systematically added in three steps. First, write the corresponding user-space program to inject and perform the measurement with the eBPF program. Second, gather and store data with PCP, by adding the corresponding module in the BCC PMDA. Third and last step, display the data in one or more widgets in Vector, to allow for on-line monitoring.

4.2.3.1 Interledger Connector

The goal of Interledger is to provide an architecture and a minimal set of protocols to enable interoperability for any value transfer system. The Interledger protocol is literally a protocol for Interledger payments. Interledger Connectors aim to realize the vision of an international friction-less payments routing system. In other words, a standard for bridging diverse financial systems [133].

The reference implementation of the connector specification is in JavaScript and so is the new Rafiki connector. However, other implementations are also written, in Java and Rust. Figure 4.7 shows how connectors are bridging all ledgers and their end-users, represented in the Figure by *"Alice"* and *"Bob"*.

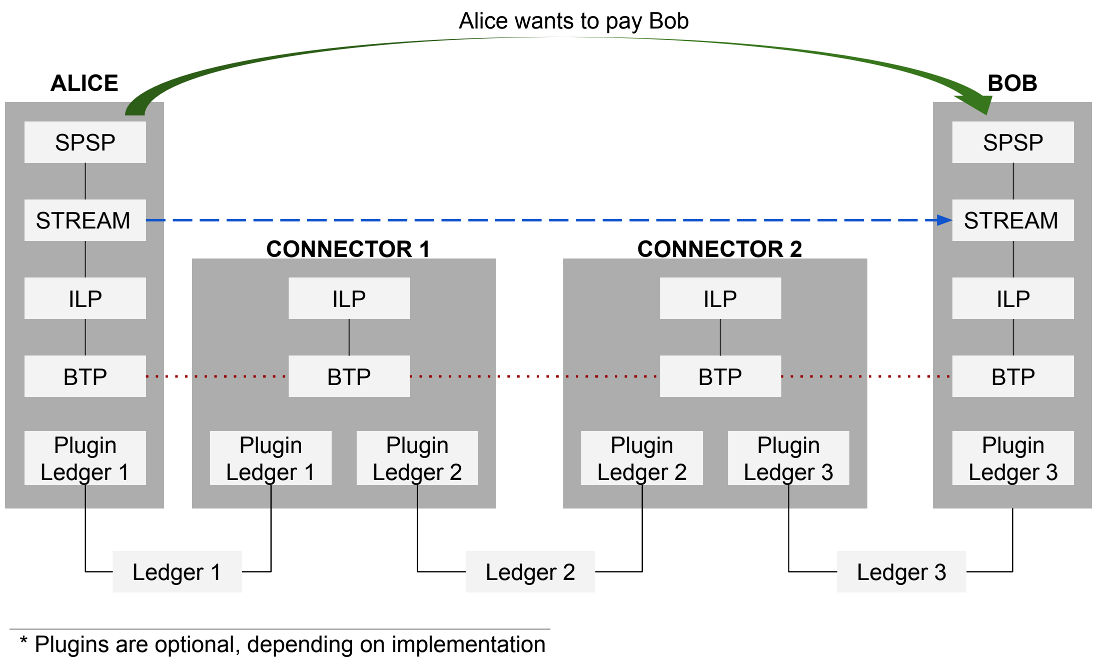


Figure 4.7: ILP payment chain.

Connectors are run by different entities and offer payment inter-operability across the payment platform to the ”customers” running a ”customer app”. The connectors are the ”service providers”, or ”market makers”, or ”liquidity providers”, because they provide end-users access to other payment networks, provide payment routing, exchange and liquidity. To do this, the Connectors use, among others, the Interledger Protocol (ILP), and ILP addresses.

In this Section, we use two full-fledged implementations of the Interledger protocol. Each of them runs over the last long-term support version of nodejs (v10) in a Docker container. The reference implementation of the protocols is in github [21]. This is the reference JavaScript implementation of an Interledger connector. The second implementation of an Interledger connector is also in JavaScript and all the source code is on github [22]. To facilitate the discussion about the performance and flamegraph analysis, the Bilateral Transfer protocol and the STREAM protocol which are also run by the Interledger connectors are presented below.

The Bilateral Transfer Protocol (BTP) emerged as a necessity, due to a combination of ILP goals (fast and cheap transactions) and the realities of some ledgers (expensive and/or slow settlements). With BTP, two parties can send funds directly to each other, up to a maximum amount they are willing to trust before settlement. BTP is used between connectors for transferring ILP packets and messages necessary to exchange payments, settlement, configuration and routing information.

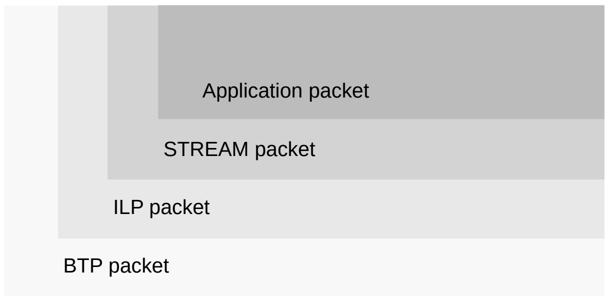


Figure 4.8: Interledger packets data structure. [88, 89]

[21]https://github.com/interledgerjs/ilp-connector, valid in January 2023
[22]https://github.com/interledger/rafiki, valid in January 2023

As shown in Figure 4.8, BTP is a "carrier" for ILP packets and as such, for other protocols like STREAM for example. BTP establishes the "link" between connectors, on top of which the ILP packets are being sent. When setting-up the connector plugins, one also generally sets-up a BTP connection. The data is sent over web socket connections. One of the peers acts as a server while the other is connected as a client. It implements a Bilateral Ledger, where the two peers keep track of their (yet) un-settled accounts and balances. The Bilateral Ledger, a micro-ledger kept by the two peers in-between them, is not to be confused with the Underlying Ledger - the main ledger where all accounts and transactions are stored, e.g. the XRP Ledger. The ILP protocol is a standalone protocol specification which can still work without BTP [149]. The BTP in its current form is a binary request/response and authentication protocol implemented over WebSockets and includes the "sub-protocol" naming [82].

The BTP protocol is illustrated in Figure 4.9: In order to connect to Interledger, each of Alice's and Bob's ILP modules establish a BTP connection over wss with the parent connector. As long as they are connected to Interledger, this connection will be live. The ILP packets will travel over BTP. While opening the BTP connection, both of them also negotiate a unique paychan with their direct peer, the connector.

To complete a payment, the Interledger Connectors run also the **STREAM protocol**. The STREAM module inside a Connector is able to break the payment into multiple packets, which will be sent over ILP using *prepare-fulfill-error* packets. The STREAM module at the *receiver's* end will finally reassemble the payment. The biggest Interledger connectors' risk is not to be able to fulfill an incoming transfer after the corresponding outgoing transfer was done [150]. This is called the *Fulfillment Failure.* To diminish as much as possible the odds of that occurring, some mitigation measures [23] have been proposed. For two of them, eBPF has a clear added-value:

- Packet filtering - White-listing or denial of service protection
- Redundant Instances - Difficulty to interfere with program instance(s)

Therefore, monitoring is a must in this case and in the case of an unexpected attack. Indeed,

[23]https://interledger.org/rfcs/0018-connector-risk-mitigations/, valid in January 2023

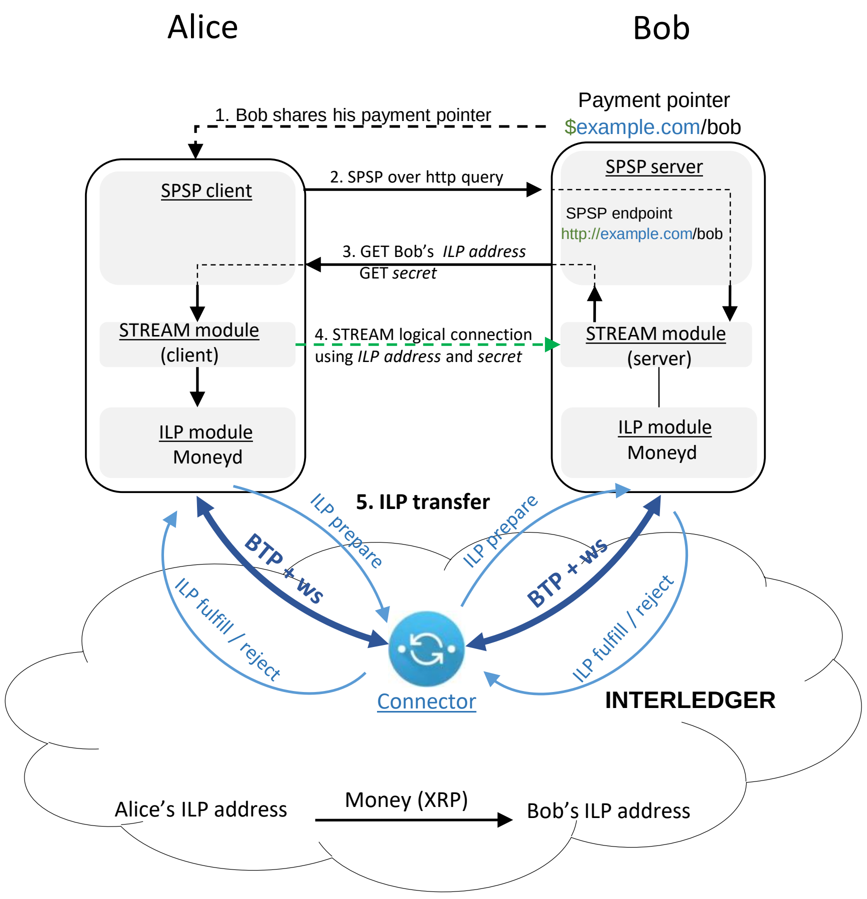


Figure 4.9: The BTP protocol in practice. [88, 89]

a worldwide payments routing system will certainly be a clear target.

4.2.3.2 Performance analysis and flamegraph analysis

This Subsection shows how we profiled both Interledger connectors to point out performance flaws. All the material and code produced in the context of the experimentation we carried out is available on a github repository[24].

In [151], we explain how to setup an Interledger test-bed connected to a private RippleNet and Ethereum PoA. We used a simplified version of our test-bed to first generate traffic only through two connectors based on the reference implementation. This simplified test-bed is

[24]https://github.com/Oliryc/monobpf, valid in January 2023

composed of a private RippleNet and three interconnected connectors, i.e. forming a triangle where one is only observer/idle. Then, using the same setup, we generate traffic through the Rafiki connector. To generate the workload, we trigger payments between two users connected to two different connectors and exchanging 50000 XRP at a rate of 1 XRP per ILP packet. This means that we create a proper payment channel and carry well formed ILP packet.

To perform the stack trace profiling, i.e. flamegraphs, we use a tool called *0x*[25]. The stack trace profiling can be resource intensive. Therefore, *0x* proposes a method to generate a flamegraph on a production server. By default *0x* uses the internal profiler of the JavaScript engine (V8). This means native frames are omitted.

Each sample is a coarse fixed-rate 1 ms "snapshot" of the current stack, which is an effective profiling method allowing to identify which code paths take more time to execute.

Flamegraphs help visualize two metrics: how much time a function spends on the CPU, and the time a function spends at the top of stack. In the given concrete case, each block in the Flamegraph represents a JavaScript function.

On the Y-axis, the last function to be called is at the top of the stack. High blocks signify a deep call stack. They also show the parent to child function relationships. If a function is at the top of the stack more often than others, this means it may be blocking the loop of events and it is called as *hot*.

The X-axis helps illustrate the *ticks* accumulated by the investigated function. The width of a block in the Flamegraph represents the amount of time a function appears in the same stack in ratio to the total samples. The longer the block, the more time the function, including children, spends on the CPU.

The block color represents the *heat*, meaning how much time the function appeared at the top of stack compared to the total samples.

The relative position of the blocks does not represent the sequence of execution, meaning the left blocks are not necessarily executed before the blocks on right side. The Flamegraph does not show the passage of time from left to right; the left to right ordering is in fact the alphabetical sort of frames, which maximizes frame merging [138]. Therefore, a more subtle

[25]https://github.com/davidmarkclements/0x, valid in January 2023

inspection is required to finding when a method is effectively called.

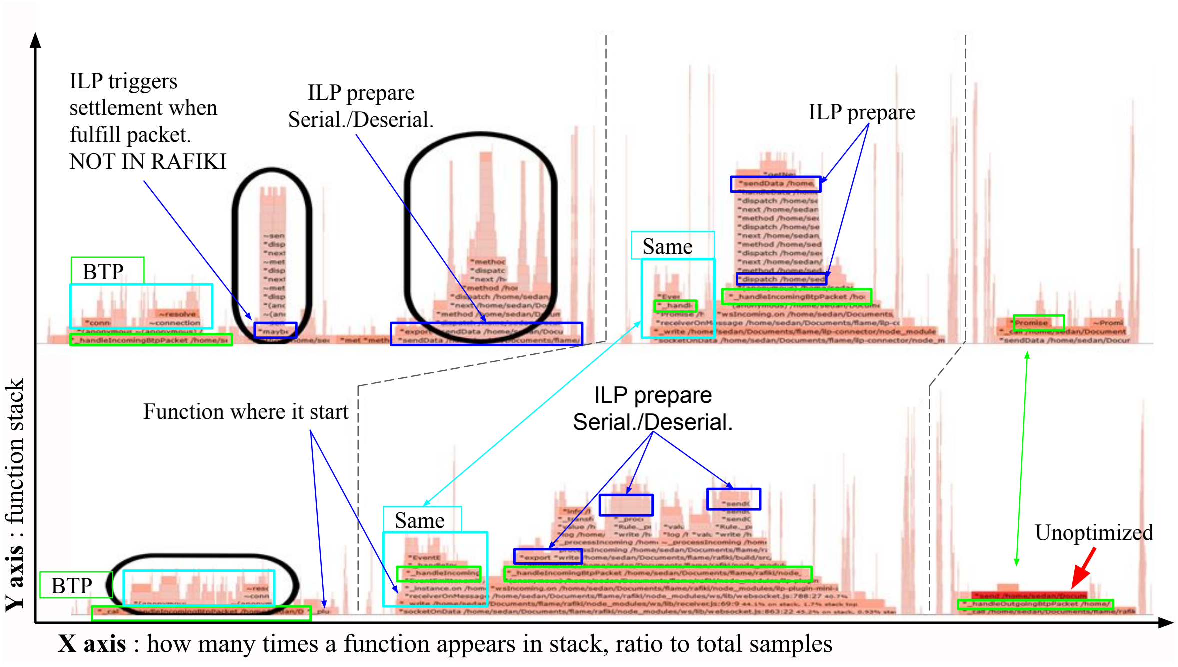


Figure 4.10: ILP implementations at WORK - full flamegraph : Reference connector (top) and the Rafiki connector (bottom).

Tracing presents how events occur over time, while *profiling* shows the number of occurrences per event type.

Figure 4.10 shows the flamegraph generated for the *Reference implementation* (top) and for *Rafiki implementation* (bottom), both under workload, i.e. sending ILP packets. These flamegraphs are presented to emphasize first the differences in terms of behavior. Conceptually, the two connectors do the same thing. Indeed, both implementations are based on the same specification and the same JavaScript libraries. The main thing that distinguishes Rafiki is its software architecture.

To compare the Interledger connectors, we also carry their stack trace profiling while in idle state, to have a *baseline*. Figure 4.11 shows the stack trace profiling for the Reference implementation while Figure 4.12 shows the same type of graph for Rafiki. Clearly, when IDLE, the Figures are colder, i.e. the flames are less red.

Besides, in our Vector dashboard we are also able to monitor application specific metrics: garbage collector, TCP sessions lifetime, HTTP traffic (HTTP verbs, code), Websocket ses-

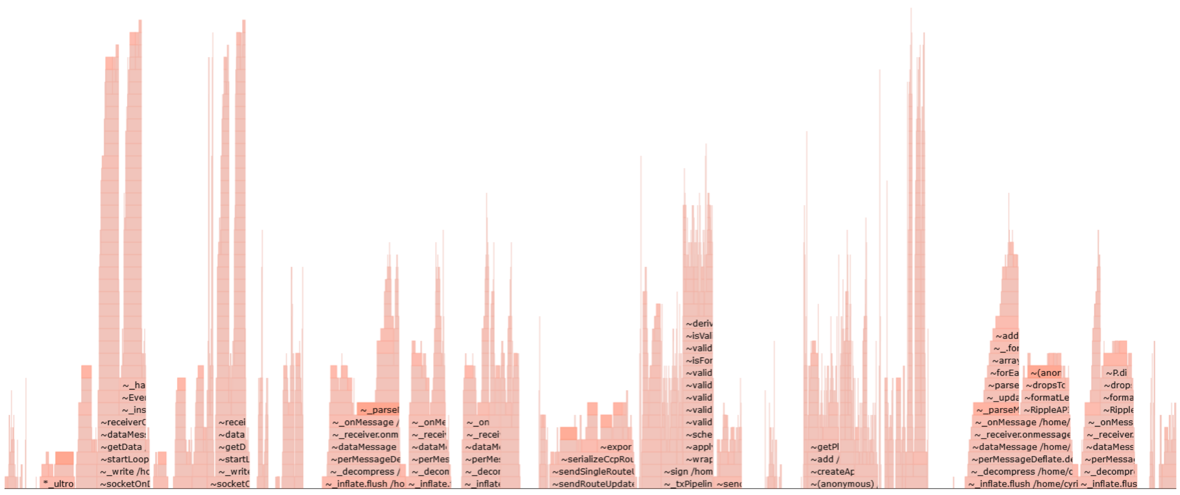


Figure 4.11: Reference implementation while IDLE - full flamegraph.

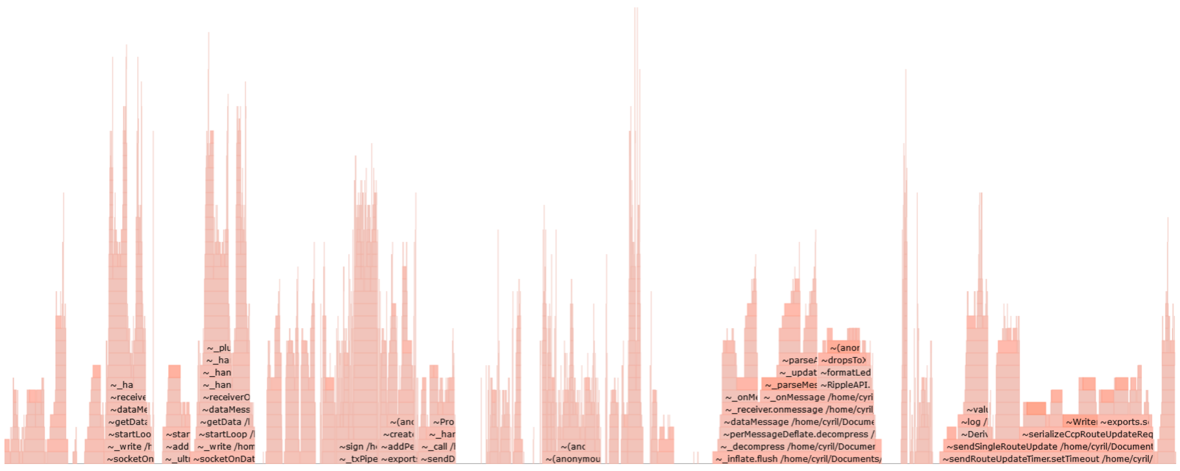


Figure 4.12: Rafiki implementation while IDLE - full flamegraph.

sions, network throughput and other classic metrics. At this scale, Figure 4.10 only allow us to point out major performance differences between the two implementations.

One of the major added-value of such Flamegraphs is the interactive nature of the plot that lets you drill-down and zoom on any function call. This is helpful when you do not know what you are looking for. That is why, we primarily used the Flamegraph as a map to find out where the Bilateral Transfer Protocol (BTP) and the Interledger Protocol v4 are active. It is also possible to zoom-in on key "business functionality" of the ILP implementation.

Flamegraph interpretation: At this scale, the Figures are especially suitable for two things:

- Observe the general shape of the flamegraph. Knowing that functions are ordered alphabetically.
- Observe the proportion at runtime of each function level, i.e. native, business, etc.

For the reference implementation we notice several prominent columns pointing out deep and long function calls at the application level. The software stack at issue is managing part of the ILP packets processing and part of the settlement process. Therefore, we investigated further how ILP packets are (de-)serialized and how the settlement process is triggered. It turns out that in Rafiki developers decouple the settlement engine from the packet processing logic. Therefore, results are consistent with their architectural modification. In Rafiki, the columns pattern disappears.

The decoupling of packet processing from settlement allows independent packet processing with potentially positive impact on packet latency. Thus, this design approach can allow for mitigation of ILP connectors' financial risks arising from unfulfilled ILP packets (fulfillment failure) [26].

4.2.3.3 New eBPF program created

Monitoring the garbage collector. BCC provides an example to count the number of executions of the Garbage Collector (GC) and time them. This was however not integrated into PCP and Vector. Therefore, we built on the example to create a PMDA module and a heatmap widget for Vector. The heatmap in Figure 4.13 presents the lifetime and the frequency of all the calls to the garbage collector.

HTTP traffic Identification. We get the packets in a raw form, quite close to the bytes circulating on the network. Even though some processing has been performed by the network card at this stage. For instance, on card TCP checksum validation leads to an incorrect value of the corresponding field when reading the packet at our stage. All the following process is performed in the kernel at TC level. Our goal was to show how far we can go in the packet

[26]https://interledger.org/rfcs/0018-connector-risk-mitigations/, valid in January 2023

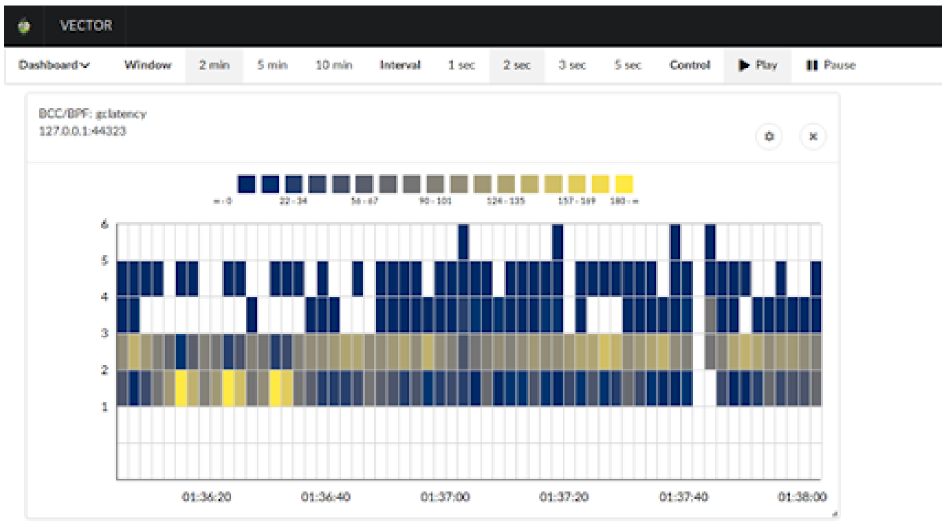


Figure 4.13: Latency of nodejs garbage collector.

analysis at this level to, for instance, filter packets as early as possible. Consequently, we process each network layer to locate the HTTP content. First, we read the ”type” field of the Ethernet layer header. If the value is the one associated with IP protocol, the packet is candidate, i.e. it could be an HTTP packet and we process it further. The next header is then from the Internet protocol. We know the position of its first byte because the Ethernet layer is fixed-size. The ”nextp” field of the IP header is checked for correspondence with TCP. If so, the packet is still candidate. To know where the first byte of this TCP header is, we use the ”ihl” field of the IP header. Similarly, to know where the first byte of the HTTP header is, the ”doff” field of the TCP header can be used to know the length of this last header. Finally, we can check the packet for typical HTTP content, like methods (GET, POST). If this matches, we send the packet to the user-space program, where a similar task is performed to locate the HTTP payload. From this payload, features of the HTTP protocol can be extracted, like methods, headers, return codes. Note that as soon as the packet is not candidate anymore, the treatment is interrupted.

IP Whitelisting and Denial of Service Attack. As a hardening measure, we leveraged eBPF and XDP to improve the resilience of the ILP connectors against denial of service (DoS) attacks. It has been observed in the industry that rejecting packets with iptables rules

Header	**How to know the size?**
Ethernet	Fixed size in spec.
IP	Size inferred from the ”ihl” field.
TCP	Size inferred from the ”doff” field.
HTTP	Sequence-delimited size.

Table 4.1: Protocol layers to decapsulate manually to locate the HTTP payload.

was not efficient enough to successfully handle medium-sized DoS attack.

The reason is that once iptable decides to drop a packet based on one of its rules, it is quite late already. Some copy and processing already went through the kernel stack and situations where all the CPU time is used to merely drop the packets arise. To solve this, a new good practice is to rely on XDP. However, this security layer is not perfect but clearly improve the capability of the kernel. With XDP, we can decide to drop a packet right away, on the networking card, thus without entering the kernel stack, as shown in Figure 4.14. Indeed, this technique becomes even more relevant for the new generation of network cards.

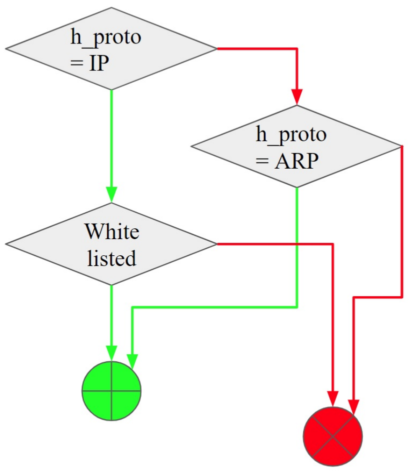


Figure 4.14: ILP DoS mitigation with XDP.

A key point of XDP compared to iptable is that there are no costs associated to the re-injection of the packet into the kernel when we want to keep it. This is paramount to avoid slowing down legitimate packets during an attack. Nonetheless, since it is based on IP white-listing, the risk for DoS attacks still exists if an attacker succeeds to find an IP in the white-list. Finally, the code of the eBPF program used for DoS protection is presented in Figure 4.15.

```c
#define WHITE4SIZE 6

static int ip4white[] = { 3137448128, 1644275904, 16885952, 2516691136, 2197924032,
↪  1140959424};

int xdp_prog1(struct CTXTYPE *ctx) {
    nh_off = sizeof(*eth);

    if (data + nh_off  > data_end) {
        return rc;
        }
    h_proto = eth->h_proto;
    if (h_proto == htons(ETH_P_IP)) {
        // Allow packet to pass if its IP is in the whitelist
        int ip = get_ipv4(data, nh_off, data_end);
        if (ip == NOIP) {
          return XDP_DROP;
        }
        #pragma unroll
        for (int i = 0; i < WHITE4SIZE; i++) {
          if (ip4white[i] == ip) {
            return XDP_PASS;
          }
        }
        return XDP_DROP;
    } else if (h_proto == htons(ETH_P_ARP)) {
      return XDP_PASS;
    } else {
      return XDP_DROP;
    }
}
```

Figure 4.15: Code snippet to prevent DDOS with XDP.

4.3 Conclusion

In this Chapter, we presented a part of our results. Indeed, to perform a precise analysis and diagnosis of the program under workload the cross-validation of all the different metrics collected is required. Our experimentation are twofold. We use eBPF to better understand Interledger connectors when treated as a black-box program. Then, we assess the potential of eBPF probes to monitor the full stack from operating system to application layer when the application is containerized and not modifiable.

This work was published in IEEE NOMS 2020 [152].

5

SPON: Enabling Resilient Inter-Ledgers Payments with an Intrusion-Tolerant Overlay

Lately, considerable attention from private, academic and governmental actors is focusing on improving the global payment systems. Besides directly customer related aspects like speed, cost, interconnectivity or transparency of payments, the ever growing cyber risks require renewed efforts towards increasing key requirements like resilience, reliability and security.

New technologies like Distributed Ledgers create new opportunities towards achieving these goals. For example a payment initiated on some ledger could cross different ledgers until reaching the final payee on another ledger.

But how could different DLTs be connected in a standardized way? Recent developments in protocols like the Interledger protocol, which can also accommodate FIAT currencies, enable transfers of value between different ledgers through means of Interledger Connectors having accounts on different ledgers and thus able to facilitate the transfer of value. They charge a small fee in return for their service.

The current version of Interledger, ILPv4, includes the STREAM protocol that works on top of it and facilitates splitting of a larger payment into small chunks, then sends them as a continuous STREAM of value and data. Thus, ILP mitigates risks associated with transfers of larger amounts, and accommodates micro-payments use cases.

However because it works over the Internet, the Interledger protocol can be subject to Internet specific vulnerabilities like lossy paths, path failures and network partitions, or even BGP hijacking attacks. Moreover, due to the way the ILP connectors can form peering relationships, the ILP network is not necessarily constructed on latency or attack resilience criteria. This, combined with the current ILP payment routing mechanism, makes it possible that at a network level, a payment initiated in San Francisco could cross the world a few times until reaching a final payee in Frankfurt, thus increasing even more the vulnerability to Internet network path degradations or attacks.

We argue that while ILP is not meant to optimize at network level, for payment systems the desirable levels of resiliency and security are similar to cyber-physical systems or SCADA networks.

An overlay of relay nodes can help achieve the desired goals by leveraging redundancy in the IP network and deploying customized protocols. To answer these challenges, we present "Secure Payments with Overlay Networks", an overlay based design for global payments across different ledgers.

In this work, we demonstrate the advantages of SPON:

- Improved real-life performance by adding resilience to lossy paths.
- Increased service availability by providing resilience to network path failures.
- Increased security guarantees, including resilience to BGP hijacking attacks.

To achieve the stated goals, SPON architecture introduces an overlay of relay nodes strategically positioned on top of the underlying internet to leverage the redundancy in the IP infrastructure.

5.1 Introduction

As already stated, one approach to address the performance, resilience, and security issues is to use an overlay of relay nodes. These relay nodes are not part of the distributed ledger's nodes and their only goal is to relay communication between ledgers. Such an overlay of relays can leverage redundancy in the IP network and deploy customized protocols to provide desired security, latency performance, and resilience to failure and attacks.

This work shows how a global payment system enabling payments between different ledgers can be designed and deployed over the public Internet using ILP and Spines [153] intrusion-tolerant overlay network. ILP facilitates the interoperability of any payment systems across different ledgers, while Spines serves as secure and trusted transport backbone for ILP communication. We assume that clients conducting payments within the same ledger are handled by internal ledger-specific protocols (e.g. BTC), and *we focus on inter-ledgers communication.* While intra-ledger protocols typically consider that any ledger node can be compromised (e.g. BTC nodes), previous work using relays to connect ledgers did not assume that relay nodes between ledgers can also be compromised and not forward payments or that the relay network itself can be subject to BGP routing attacks.

We implemented SPON and investigated how well it achieves its goals.

We consider three network topologies:

- The first is a synthetic topology allowing to investigate different capabilities of SPON;

- The second topology is based on a real-life deployment [153] with nodes spread over North America, Europe and East Asia which allows to evaluate SPON's performance in a more realistic scenario;

- Finally, the third was used in [154] to show the impact of eclipse attacks conducted by partitioning the network using BGP hijacking, and we use it to show how SPON can be deployed to address such attacks.

The findings can be summarized as follows:

- SPON improves the payments latency over a *baseline* system not using the overlay. Benefits become higher as network loss increases, because the customized overlay protocols recover the lost packets from nodes closer to the recipient instead of recovering it from the sender.

- Even under extreme scenarios such as a network meltdown SPON was able to continue forwarding payments by rerouting around the failures, while the baseline system could not complete the payments.
- We used the network topology in [154] to show the impact of eclipse attacks conducted by partitioning the network using BGP hijacking, as a demonstrative example on how SPON should be deployed to address such attacks.

This Chapter is structured as follows: Section 5.2 discusses challenges for global payment systems and how to overcome them by using overlays. Section 5.3 presents the SPON design and implementation, while Section 5.4 presents the experimental results. The Chapter ends with a final discussion in Section 5.5.

5.2 Motivation

What makes the Interledger protocol really different is its ability to break a payment into many, arbitrary small packets and sending of that payment as a continuous STREAM of money. However it is desirable that data streams benefit from a good quality connection, and this opens the discussion about some of the limitations of ILP.

5.2.1 Limitations of ILP Payment Systems over the Internet

To facilitate the discussion about some of the limitations of current payment systems designs we present an example in Figure 5.1. The lower left thumbnail shows a possible example of an ILP network, where the nodes are ILP connectors. As ILP nodes may freely form links on the ILP network, according to reasons like regulatory, legal, business and trust relationships, the network is not constructed based on latency or attack resilience criteria. So, according to current ILP payment routing algorithm, a payment from San Francisco (SFO) to Frankfurt (FRA) could be routed along the green path in the thumbnail also including Hong Kong (HKG). The physical locations of these ILP nodes along the payment path highlighted in green could be spread all around the world, resulting in high end-to-end latency and increased vulnerability of the payment system to lossy data paths, faults and attacks.

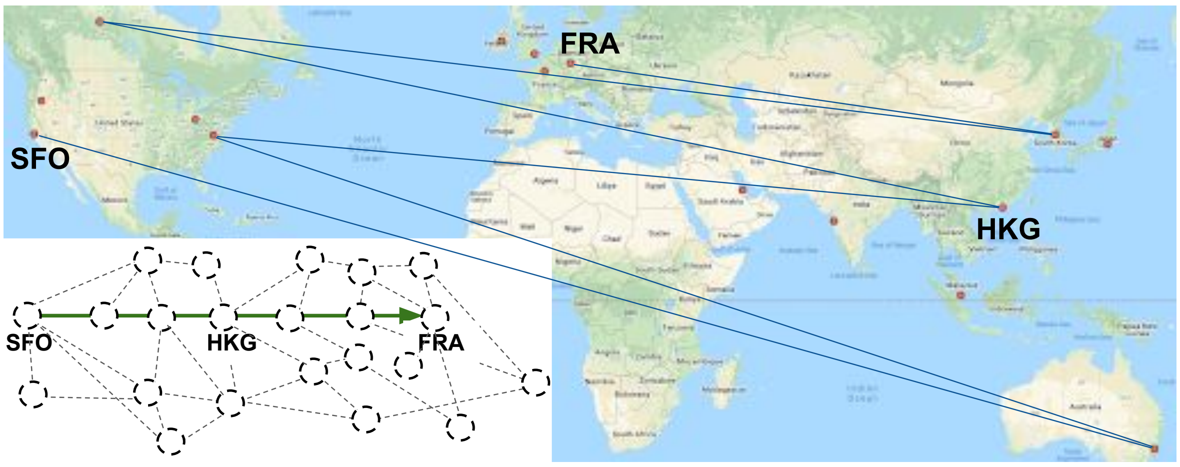


Figure 5.1: Example ILP payment routing (lower left thumbnail) and actual geographical location of corresponding ILP nodes.

This work focuses on network level limitations of ILP payment systems. We identify three such limitations: (1) resilience to lossy paths, (2) resilience to network faults and partitions, (3) resilience to DoS such as route hijacking.

Lossy paths can be problematic especially in the case of streaming payments, in which one single payment can be spawned over multiple smaller payments. This is encountered in pay-as-you go for torrent like distribution services [155], which can not afford packet losses even if the per packet level payment amount is tiny. Many underbanked communities [156] experience the downsides of digital, financial divides and even in developed economies some rural communities have to face mediocre Internet connectivity. Therefore, tolerance with respect to poor network connectivity is an essential feature of the payment system.

Path failures and network partitions. Network resilience is an important factor to consider since network enabled systems can be partitioned by intentional actions (censorship) or non-intentional (faults) accidents. The consequences for both are the same: outage, delays and degraded performance which impact the availability of the service. Payment systems should be capable to rapidly detect failures and react accordingly.

BGP hijacking attacks. BGP routing attacks against ILP could have a serious impact such as: partition the payment network and create a situation similar to a DoS, which can result in revenue loss for ILP nodes and their customers (open attack), delay all/chosen packets, while attacker's packets would be forwarded at normal rate (covert attack), hairpin drop packets from/to a certain ILP node/endpoint (covert attack), or at will, attacker can be the only one able to send/receive ILP transactions in/from both partitions. The attacker can also divert, store, map and analyse the traffic: get geo-location information of ILP providers/customers, gather/infer information about the volumes of payments per ILP node (average value carried by an ILPv4 packet at the attack moment is x XRP).

5.3 SPON Design and Implementation

In this section we describe SPON, our proposal for resilient global payment systems over Internet. We first describe the design goals for our system, then describe the attacker model, and give a description of the design and implementation.

5.3.1 Design Goals and High-level Approach

Our main goal is to design a global payment system that supports payments across different ledgers *while achieving:* improved performance (latency), improved service availability (fault-tolerance), and security guarantees, including resilience to routing attacks. We assume clients conducting payments within the same ledger are handled by ledger-specific protocols. While these internal protocols can also benefit from additional improvements, *our focus is on connecting different ledgers and not on services within a ledger.* We use ILP to facilitate the exchanges across different currencies and ledgers. However, ILP is not meant to optimize network communication and address fault-tolerance to network failures or BGP attacks. With our goals in mind, we would like our service connecting multiple ledgers to have the following properties:

G1 Improved payment latency: Our design should leverage the redundancy in the underlying IP network to take advantage of links offering better connectivity, by using customized routing protocols.

G2 Resilience to lossy paths: Our design should be resilient to lossy communication links across ledgers and as such improve the client network's resilience to lossy links.

G3 Resilience to path failure and node crashes: The design should increase payment service availability by increasing data flow availability through providing a system resilient to network path failures and relay node crashes.

G4 Resilience to BGP routing attacks: Our design, also deployment dependent, should provide resilience to routing attacks like *Coremelt* and *Crossfire* [157, 158].

Approach. These goals can be achieved by changing an existing payment-exchange protocol like ILP to add the desired performance, fault-tolerance, and attack resilience. However, we argue that a separation between the payment-exchange and the communication functionalities provides more flexibility in ILP node placement and modularized development. For example, the ledger pre-post processing functionality is better placed closer to the ledger; also, because they manipulate user value and data, the placement of ILP nodes in different geographical areas may involve different legal restrictions, licensing, regulations. A compromised ILP node is more dangerous than a compromised overlay node performing a simple forwarding because the forwarding nodes do not need visibility into the payments to perform network-level forwarding. Thus, our approach is to separate ledger processing from the forwarding functionality, to maximize performance and resilience to attacks, while accommodating legal restrictions. The data forwarding layer can be an overlay of *relay nodes* that implement customized routing algorithms for better latency, routing around failures and with BGP attack resilience. The ILP payment exchange connectors use the overlay of relays to communicate with each other.

Figure 5.2 shows how communication flows between ILP nodes Alice and Bob, through ILP and the overlay of relay nodes (Alice and Bob are not end-users but full ILP nodes): Each ILP node is connected to at least one overlay relay node. Each overlay relay node is connected to multiple Internet Service Providers (ISP) / Internet Exchange Points (IXP) / Autonomous Systems (AS). At ILP level, a payment originated from Alice for Bob, is routed through the "ILP connector" in the middle. However at data packet level, the two hops (Alice <-> Connector and Connector <-> Bob, are routed through redundant paths on the overlay network (thick arrows on the middle layer of Figure 5.2). Further, each overlay link benefits from disjoint, redundant paths at the Internet level below.

Need for intrusion-tolerant overlays. Overlay networks can improve latency because they can reduce re-transmissions [159, 160] and can provide resilience to benign faults by routing around them. However, the introduction of the overlay of relay nodes in the system design changes the trust model. First, the overlay itself risks being compromised since a software node is easier to compromise than a hardware router. Compromised overlay nodes can significantly impact the system performance as a whole, or target specific connectors or ledgers and discriminate against some clients conducting payments. Second, the nature of the overlay requires different payment streams to share the same logical structure which can allow some clients to create denial of service against competitor clients conducting payments through the same link(s) on the relay network. Such overlays need to be centrally managed to prevent topology related attacks. We set the following goals for our overlay of relays:

O1 **Resilience to attacks from compromised forwarding relays**: We want to prevent compromised relay nodes from being able to divert or stop traffic.

O2 **Resilience to denial-of-service from malicious clients**: In the presence of the overlay, payment flows from different competitor clients can potentially compete to each other at networking level to the point where one can generate a targeted denial of service for the other by saturating the link(s). We would like all payment flows to be treated fairly by the relay nodes, i.e. all payment streams receive the same share of available network bandwidth.

5.3.2 Threat Model

We assume that the overlay of relay nodes is centrally managed and the communication between the relay nodes is authenticated with Public Key Infrastructure (PKI): the administrator of the system has a public/private key pair with each of the overlay nodes, and they know all the other public keys. All overlay nodes are aware of the topology, that can be changed only by the administrator.

We also assume that overlay relay nodes can be compromised. A compromised node can exhibit Byzantine behavior such as arbitrary dropping, delaying, or incorrect forwarding of packets. We assume that overlay nodes have enough processing resources to handle all incoming messages in due time, but their buffers for storing the messages are limited.

We do not assume a specific bound on the number of compromised relays in the overlay network. Instead we assume that the adversary cannot partition the sender from the receiver, i.e. there is a path from the sender to the receiver where all relays are not controlled by the adversary.

We assume the attacker has amounts of bandwidth and processing power large enough for DDoS attacks like [157, 158].

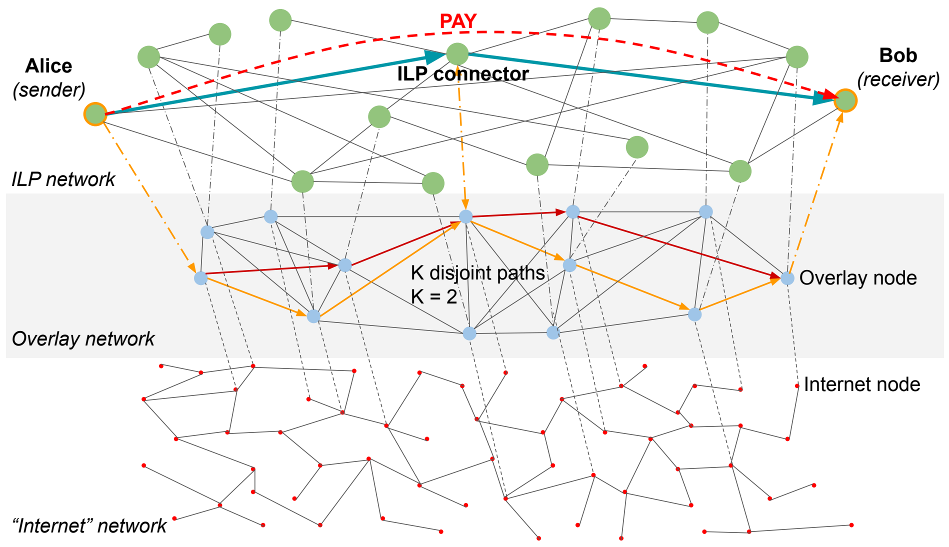


Figure 5.2: Communication mapping for Ledgers, Overlay, and Internet.

5.3.3 SPON Design and Implementation

We implemented SPON using ILP and the Spines overlay. Below, we first give a description of aspects of ILP and Spines relevant to our design, then describe our system, SPON.

To remind, the ILP environment consists of the below stack of main protocols, which are listed in order top-to-bottom:

- The *Simple Payment Setup Protocol (SPSP)*, ensuring the exchange of credentials required to establish a STREAM payment, which for specific reasons works over HTTP.

- The *Streaming Transport for the Realtime Exchange of Assets and Messages (STREAM)* protocol, implementing the concept of streaming value (money) and data over ILP (encapsulated in ILP packets). This concept offers a series of advantages over sending a transaction in full.

- The *Interledger Protocol (ILP)* itself, ensuring the value transfer across ledgers. The ILP packet offers a *data* field in size of 32k, where different information and sub-protocols can be encapsulated.

- *Bilateral Transfer Protocol* (BTP), responsible of establishing a link between two peers.

Spines is an open source overlay network [160, 5] that provides availability, resiliency, and timed-delivery, achieved by making use of multi-homing at multiple ISPs and deploying the nodes in strategically located datacenters (connectivity). The nodes are centrally managed and resilient overlay routing such as multiple disjoint paths and flooding [153]-p6 are used to ensure resilience to forwarding attacks. Buffer management like round robin is used to ensure that each node evenly processes packets per sender in case of priority sending, or per flow (sender-receiver pairs) in case of reliable sending.

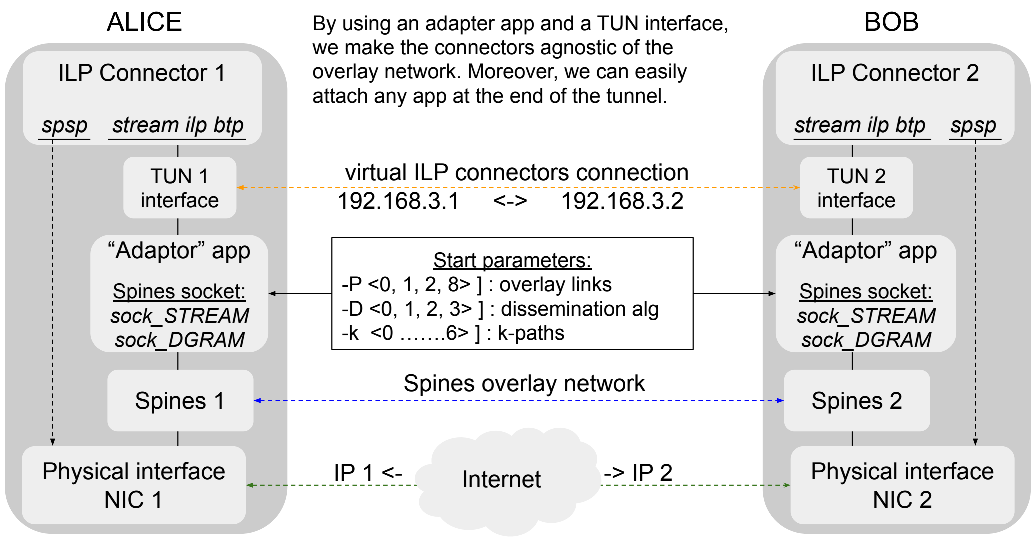


Figure 5.3: SPON Architecture.

We show the architecture of SPON in Figure 5.3. There are 3 network layers: the base internet layer, the Spines overlay, and the ILP network, each featuring their own addressing schemes and protocols. Each ILP node connects to a Spines node using the stack illustrated in Figure 5.3. The connector applications connect through a tunnel, agnostic of the overlay below. An *adapter* application makes the connection to the *spines_socket* exposed by the Spines node, and sends it the different parameters to use in order to forward data. We use the *Priority Messaging (PRI)* and *Reliable Messaging (REL)* communication services, shown and explained in Table 5.1.

Table 5.1: SPON services (via Spines).

Service	Details
PRIORITY (PRI)	Source-based routing with timeliness guarantees, i.e. packets are sent based on their priority, each node forwards packets fairly across all sources.
RELIABLE (REL)	Source-based routing with reliability guarantees, i.e. packets are sent with end-to-end reliably, each node forwards packets fairly across all sender-receiver pairs.

One advantage of SPON is that the service can be selected per ILP packet, because Spines provides its reliable or priority services on a per packet basis. Our design exposes this functionality to ILP payments and other ILP tools such as ILP-ping. As such, for example, the risk of *fulfillment failure* specific to ILP, could now be alleviated by prioritizing the *fulfilling* over the *prepare* packets[1]. As needed, any ILP related flow can be *prioritized* or sent *reliably*, for example routing updates or SPSP data could use the reliable protocol.

Because the connectors are agnostic of the overlay below, our design also allows for a *partial deployment*, where some connectors choose to join the network and others do not. This involves the existence of some *bridge* connectors, having connections both outside and inside SPON.

5.4 Experimental Results

In this section we describe the evaluation of SPON. We seek to answer the following questions:

[1]https://interledger.org/rfcs/0018-connector-risk-mitigations/, valid in January 2023

Q1 What are the latency improvements of SPON when compared with an approach that does not use relays?

Q2 How does SPON react to more severe network events such as network meltdowns?

Q3 How does SPON handle denial of service attacks where some clients try to overload the links with payments?

Q4 How does SPON react to severe network events such as route misdirections and BGP hijacking attacks?

5.4.1 Methodology

We conduct our experiments using *Mininet* [161] to better control the network topology, links and their properties. The Mininet testbed implemented and used is illustrated in Figure 5.4: each SPON node is implemented in Mininet on machines $\mathrm{h_1...h_n}$, and runs a Spines instance shown on the Figure in red ($\mathrm{S_1...S_n}$). They communicate via $\mathrm{s_1...s_n}$ represented in blue on the same Figure. To ensure the connectors are able to perform their ILP settlement, external access to XRPL is provided via $\mathrm{s_6}$ through the main host machine NIC. The SPON architecture shown in Figure 5.3 is represented here on Mininet hosts 1 and 5, which also run ILP connector instances in order to provide the ILP service required for the experiments.

We used the "reference" ILP connector[2] and a private XRP ledger deployed in our lab.

Topologies. We used 3 topologies for our evaluations, and a fourth to demonstrate BGP resilience. The first, referred as *Chain Topology* (Figure 5.5) is a demonstrative topology allowing to investigate different path capabilities of our overlay. The second, referred as *Global Topology* (Figure 5.6) is a real-life topology spanning the Internet and obtained from [10] which allows to demonstrate the performance and resilience of SPON in a more realistic scenario. Link latencies were obtained from specialized websites[3]. Third setting, shown in Figure 5.12 helps answer Q3, while Q4 is discussed using Figure 5.14.

Systems. We compare the following configurations:

- *Baseline:* payments are sent via the ILP nodes, without SPON.

[2]https://github.com/interledgerjs/ilp-connector, valid in January 2023
[3]https://ipnetwork.windstream.net/, https://wondernetwork.com/pings, valid in January 2023

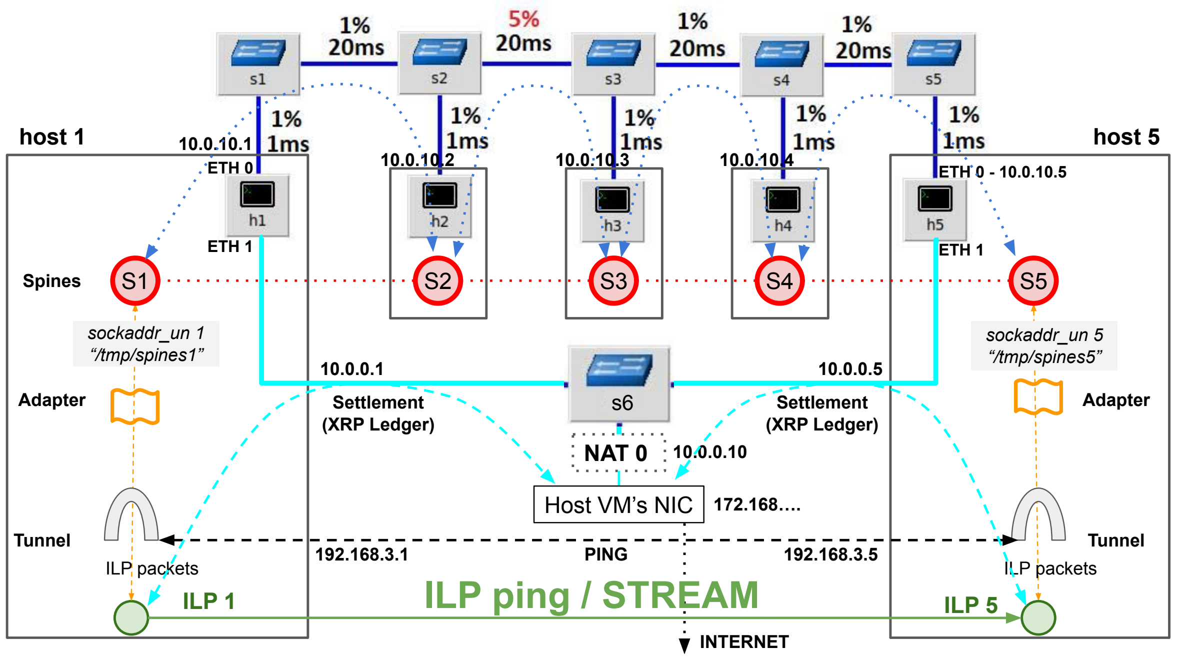


Figure 5.4: General diagram of the Mininet testbed.

- *Priority (PRI):* payments use SPON configured with source-based routing and timeliness delivery [153].
- *Reliable (REL):* payments use SPON configured with source-based routing and reliable delivery [153].

For both *Priority* and *Reliable* settings, we evaluated *Flooding (FLD)* and *k-path* as communication mechanisms. **Q1** and **Q2** are answered by comparing the *Baseline* with SPON's behavior in *PRI* and *REL* mode.

Metrics. We use *Round Trip Time on ILP* (RTT_{ILP}), as reported by the ILP Ping tool [4] to evaluate the communication between ledgers via SPON. For larger payments which are broken into a number of ILP packets and sent via STREAM, we use *Payment Latency* as the total time to complete a payment.

[4]https://github.com/martinlowinski/ilp-ping, valid in January 2023

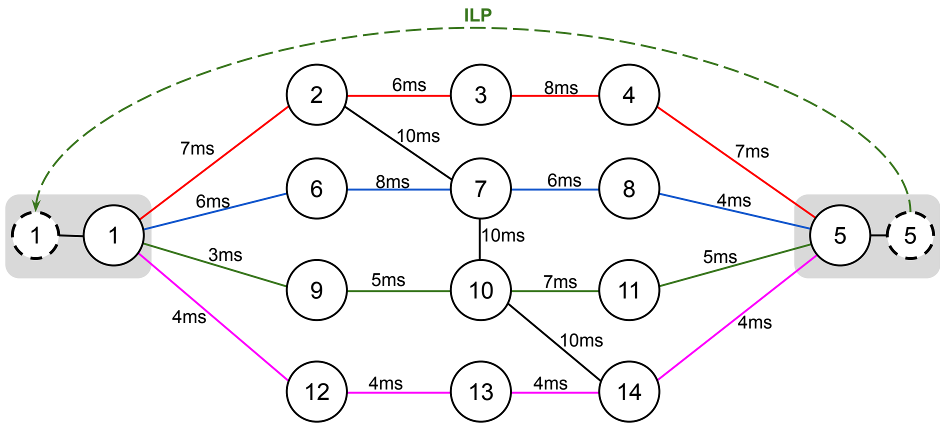


Figure 5.5: Chain Topology.

5.4.2 Performance

Here we investigate how the latency over the overlay compares with the latency over the baseline topologies defined below under different levels of network loss. In each setting we also include for reference the case with no loss on links, for which the results are illustrated in Figures 5.7a, 5.8a, 5.9a.

5.4.2.1 *Chain topology*

As illustrated in Figure 5.5, we use two ILP nodes (*5* and *1*) acting as sender and receiver, to send 100 ILP ping packets at a rate of 1 packet/s, using the ILP-PING tool. The baseline (RTT_{ILP}) is 32ms and equivalates the two *connectors* paired directly on the fastest path from the figure.

ILP RTT. To evaluate latency under loss, we introduce variable loss of 2, 5, and 10% on link S12-S13, chosen because it's on the fastest topology path, so it has high chances to have a visible impact on results, illustrated in Figures 5.7b, 5.7c, 5.7d. Solid grey bars represent baseline averages, grey striped bars represent *Priority* messaging with flooding (FLD), 1 or 2-paths [153], and dark grey bars represent *Reliable* messaging with FLD, 1 or 2-paths. While not shown experimentally, we appreciate that introducing loss on slower paths (9-10, 6-7, 2-3) would advantage SPON by enabling it to use the fastest path at full capability. We isolate

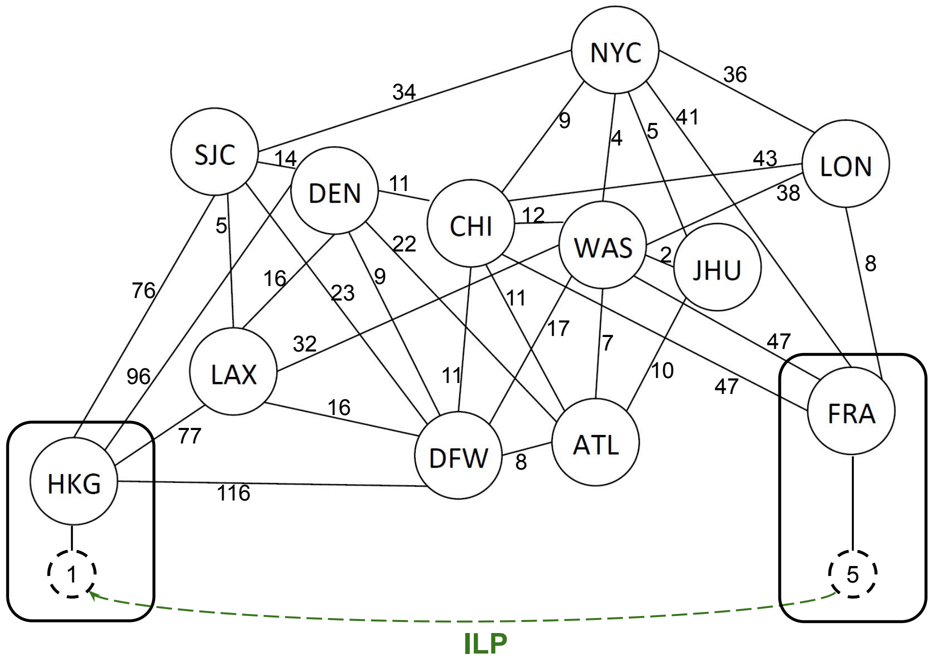
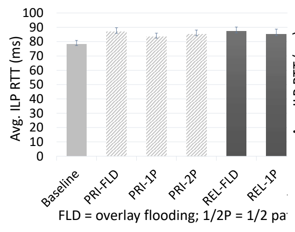
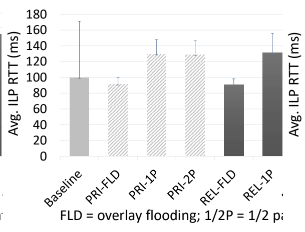


Figure 5.6: Global Topology.

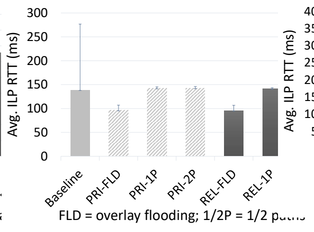
Spines' processing overhead by setting the loss to 0%; as shown in Figure 5.7a, SPON does fare a little bit worse than the baseline (5% or 6s in our case). This overhead however is small and does not prevent SPON from performing better than the baseline in realistic situations with loss: at 2% loss, Figure 5.7b shows that SPON already offers an advantage of 10% latency over the baseline when working in FLD mode. As loss increases, SPON's advantage increases, and at 5% loss the gain over same baseline is 33%, as depicted in Figure 5.7c. The error bars also point out that the service is more stable under loss, if using SPON.



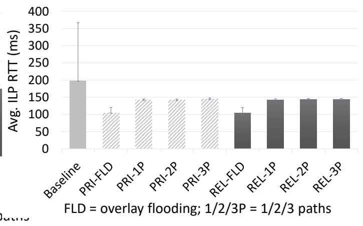
(a) Loss 0%



(b) Loss 2%

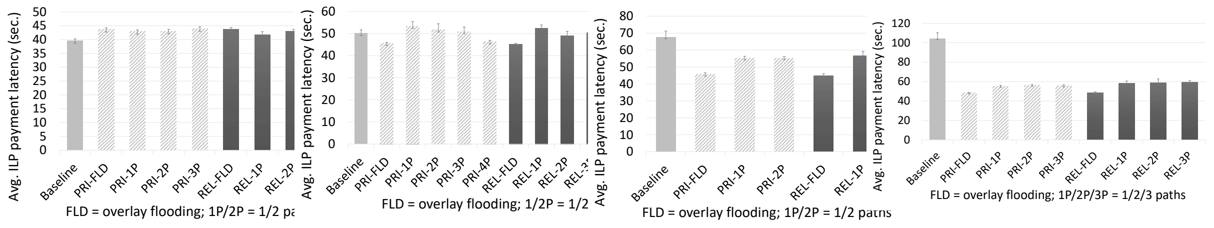


(c) Loss 5%



(d) Loss 10%

Figure 5.7: Average ILP ping RTT on the Chain topology in a network loss scenario, Priority (PRI) or Reliable (REL) messaging.



(a) Loss 0% (b) Loss 2% (c) Loss 5% (d) Loss 10%

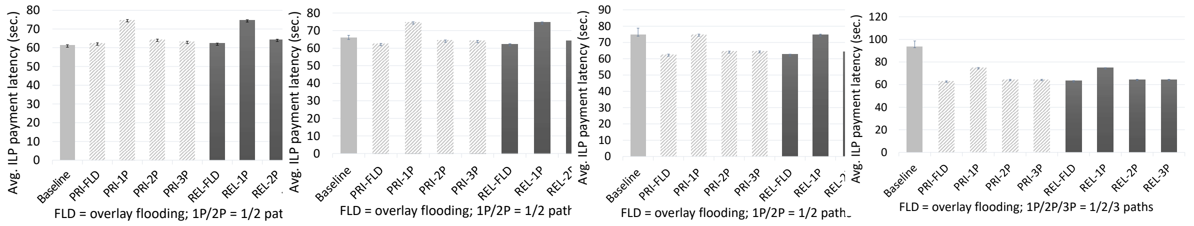
Figure 5.8: Payment latency on the Chain topology in a network loss scenario, Priority (PRI) or Reliable (REL) messaging.

Payment latency. We evaluate the latency of ILP payments under similar scenarios with network loss. On the topology in Figure 5.5 we sent 20 ILP *STREAM* payments. The amount per ILP payment was 100000 drops (1 drop = 0.000001 XRP)[5]; each STREAM packet was 100 drops. Thus, for each payment we sent 1000 ILP STREAM micro-transactions. We used *Priority* and *Reliable* messaging with FLD (k=0), 1 and 2-paths (k=1,2). The loss was set again on link S12-S13. In Figures 5.8a,5.8b,5.8c,5.8d we compare the time taken to complete the transactions over SPON, with the baseline: under ideal conditions (no loss on the links), the payment latency over SPON is a little bit larger than over the baseline (under 5%, or 2s in this case), while at 2% loss, SPON already offers a gain of 10% (5s) in FLD mode. At 5% loss, all SPON modes show 15-33% gains.

5.4.2.2 *The Global topology*

To demonstrate the behavior in a more realistic scenario, we repeat the experiments above on the Global topology; inspired from [10], it offers increased link redundancy while using well-chosen real-world, global locations spanning US, EU and Asia. Each circle represents an overlay node deployed on our Mininet testbed. As baseline, we sent STREAM ILP payments between two *connectors* paired directly over a single link with delay 148ms - equivalent to the fastest path from Figure 5.6. On the global topology, the *connectors* were attached to the overlay nodes FRA and HKG, and sent a total of 16 ILP payments directly through the STREAM protocol (no SPSP). The total transaction amount was 100000 drops per ILP payment, and each STREAM packet was 500 drops (200 STREAM micro-transactions). The

[5]https://xrpl.org/xrp.html, valid in January 2023



(a) Loss 0% (b) Loss 2% (c) Loss 5% (d) Loss 10%

Figure 5.9: Payment latency on the Global topology in a network loss scenario, Priority (PRI) or Reliable (REL) messaging.

loss was introduced between HKG and SJC because the link belongs to multiple low latency (possible) paths, and as such, with chances to impact multiple possible flows.

The results in Figure 5.9a,5.9b,5.9c,5.9d show that in ideal conditions, except for sending on 1-path, SPON adds only 1.5% to the total payment duration, compared to baseline; at 2% loss, SPON offers a gain of 5%; while at 5%, the gain is 16%.

In summary, in all scenarios we experimented with, the additional processing introduced by SPON and identified at loss 0 was small, and the payment system offered better performance under a link loss of 2, 5, 10%.

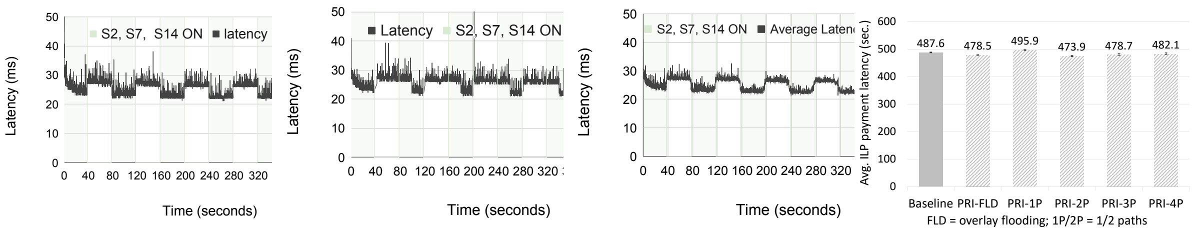
5.4.3 Resilience to Network Melting

Here we investigate how individual ILP packet latencies and the total duration of payments sent over the baseline versus SPON compare in more severe situations like node crashes. At least one path should remain available between the sender and receiver, such that there is still a way for the payment to physically go through.

5.4.3.1 The Chain topology

We want to see how an ILP payment sent over SPON behaves when all paths but one, fail. We set all links to loss zero. Because the baseline would obviously fail in this scenario, we can only assess how SPON's performance would compare with a functional baseline. As such, on the baseline, we send a payment between two *connectors* paired over a link of 20ms latency - equivalent to the remaining path 1-9-10-11-5 from Figure 5.5, if all other paths fail.

We send an ILP payment of 100000 drops, and packet size 10 drops. Thus, for each



(a) Flooding (b) 1-path (c) 2-paths (d) E2E payment latency.

Figure 5.10: Payment latency on the Chain topology in a network meltdown scenario, Priority messaging (PRI).

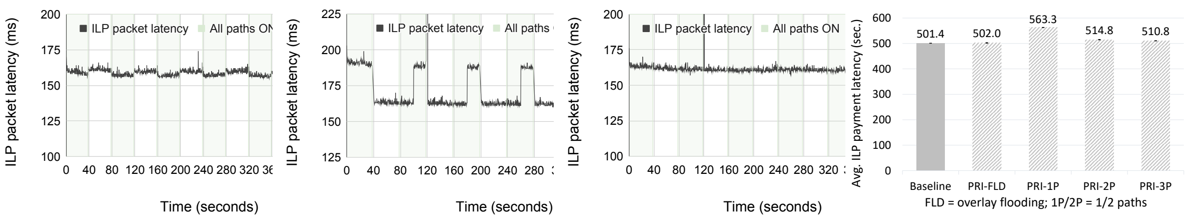
payment we sent 10000 ILP micro-transactions, for a total STREAM duration of 480s. While the STREAM is sent, we take down the communication of the overlay nodes *2*, *7*, *14* using *IPtables* on the respective machines, at a 40s interval, in a five-count cycle. This procedure completely melts and brings back every 40s, all the possible paths but the green one (nodes 1-9-10-11-5) from Figure 5.5.

In Figures 5.10a, 5.10b, 5.10c we plot individual ILP packet latencies. We observe that, if one of the currently active transmission paths is the actual path to remain unaffected by the network melt, then the system can offer optimal protection against the melt starting even from 2-paths; on 1-path, the minimal drawback comes due to the re-routing time to a better path after the network becomes available again. Even when all paths but one vanish, SPON's service continues reliably, with no packets lost during the experiment.

With respect to the total duration of payments sent over the baseline versus SPON, even when the latter was subjected to the severe path flipping above, it still performed slightly better than the baseline (3%), as shown in Figure 5.10d. This is because the baseline is able to send only on the 20ms link, while at times, SPON can also use the fastest path of 16ms.

5.4.3.2 The Global topology

Through our two *connectors* attached to the Spines nodes FRA and HKG, we sent a payment of 80000 drops, and packet size 50 drops (1600 ILP micro-payments), during a total time of 500s. While the STREAM is sent, we cut the communication of nodes SJC, NYC, LON,



(a) Flooding (b) 1-path (c) 2-paths (d) E2E payment latency.

Figure 5.11: Payment latency on the Global topology in a network meltdown scenario, Priority messaging (PRI).

WAS, JHU, DFW, ATL using *IPtables* on the respective machines, at a 40s interval, in a five-count cycle. This procedure completely melts and brings back every 40s, all the possible paths but FRA-CHI-DEN-LAX-HKG from Figure 5.6. The baseline is two ILP *connectors* paired over a single link with delay 151ms - equivalent to the remaining path (FRA-CHI-DEN-LAX-HKG) from Figure 5.6, after all other paths go down. To compare the time taken to complete the transactions over the overlay versus baseline, we repeat the experiment five times, average the results for each case, and finally represent them in Figure 5.11d. The individual ILP packet latencies are obtained after unique, single runs of the experiment with Priority messaging over 1, 2, 3-paths or FLD (Figures 5.11a, 5.11b, 5.11c). Results for 3-paths were similar to flooding and are not illustrated. We notice that in the case of a complete network melt up to 1-path, SPON's service continues, while the baseline completely fails. The end-to-end payment latency over SPON, illustrated in Figure 5.11d, is similar to the baseline (502 vs 501s).

5.4.4 Resilience to Denial of Service from Malicious Clients

With the aim to assess how an ILP flow sent over the overlay at maximum link capacity behaves in the presence of a second malicious flow trying to take over the channel Bandwidth (BW), we attach four ILP *Connectors* (1, 2, 5 and 6) to the overlay nodes 1, 2, 5 and 6 respectively (from the topology illustrated in Figure 5.12), and then we create two ILP flows.

Connector 5 is paired with, and sends an "honest" flow to Connector 2 while Connector 6 is paired with, and sends a "malicious" flow to Connector 2. To each connector we can

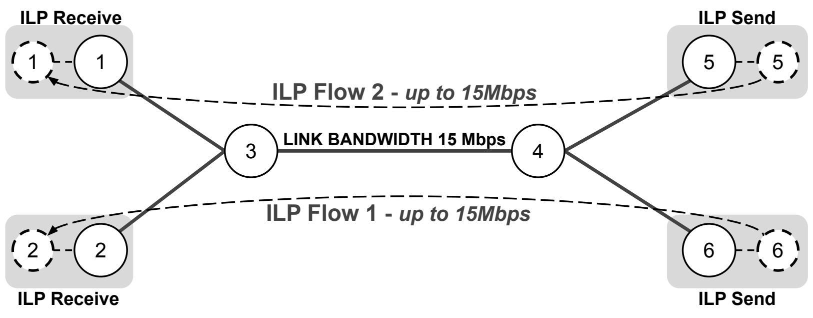


Figure 5.12: Network topology for the flow fairness.

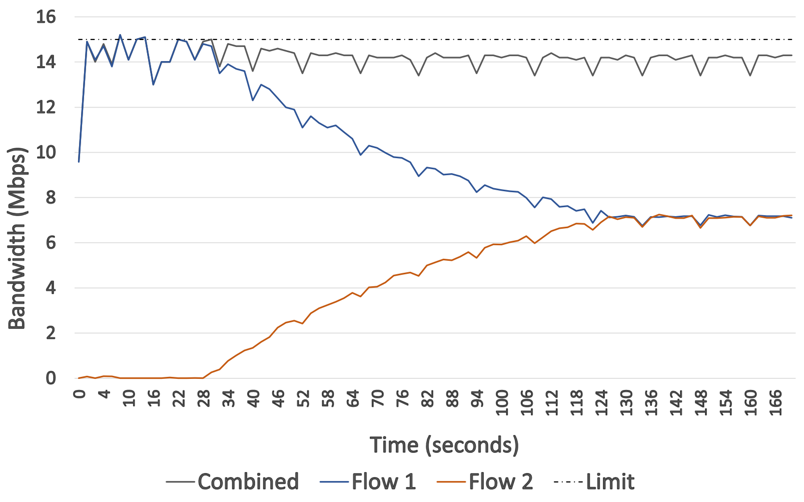


Figure 5.13: Legitimate and malicious flows contending for BW.

attach progressively, at 1s interval, up to 100 clients each sending over eight streams. We are thus able to generate for each flow a maximum traffic of 15Mbps, and as such, on our topology, we set maximum link capacity to 15Mbps. For this experiment we set all links to loss zero and as a metric, we used the flow size in Mbps. The experiment is carried as follows. While the first, legitimate flow (C5 to C1) is sent at maximum capacity, we progressively increase the malicious, contending flow, trying to fill the available channel BW up to the maximum capacity. Both flows were sent with Priority messaging over 1-path. As illustrated in Figure 5.13, the legitimate flow decreases progressively, but only up to its fair share of 1/2 channel capacity. Although the malicious flow tried to increase its flow and send at its maximum capacity of 15Mbps, it was not able to do so beyond its fair share of BW and hence, it could not take over the channel or stop the legitimate flow. While for the particular case of ILP we experimented with only two sources, related experiments which demonstrate the behavior of Spines in the case of multiple sources can be found in [153].

5.4.5 BGP Hijacking Attacks and Benign Route Misdirections

BGP is a core Internet protocol which handles inter-AS routing. Its vulnerabilities stem from its very design, born at the "beginnings" of Internet: there is no mechanism to protect the integrity and authenticity of p2p BGP messaging, no way to check an AS' authority to announce prefixes and relay route information, or to verify the authenticity of path attributes in a prefix announcement [162]. Causes of BGP routing misdirections range from accidental BGP advertisement leaks to full blown attacks, and while generally they last from seconds to hours, there have been cases where the abnormal situation lasted for months or even years [163, 164, 165, 166].

BGP routing attacks have been widely explored in literature. Hijacking attacks followed by double spending on Ethereum have been discussed by [154] for private, consortium or public deployments. A successful attack on ILP would enable the attacker to control the ILP packets flow inside the ILP network. As such, the attacker could:

- Partition the payment network and create a situation similar to a DoS, which can result in revenue loss for ILP nodes and their customers (open attack).
- Delay all/chosen packets, while attacker's packets would be forwarded at normal rate (covert attack).
- Hairpin drop packets from/to a certain ILP node/endpoint (covert attack).
- At will, the attacker can be the only one able to send/receive ILP transactions in/from both partitions.
- **Impact on Privacy** The attacker can divert, store, map and analyse the traffic: get geo-location information of ILP providers/customers, gather/infer information about payments volumes per ILP node (average value carried by an ILPv4 packet at the

attack moment is x XRP). This data can further be used to focus an attack towards the most relevant ILP infrastructure.

An experimental topology for public networks has been illustrated in [154], and we use it as a working example to show how on the same topology, SPON can defend against AS-level BGP routing attacks, through a careful design of the network.

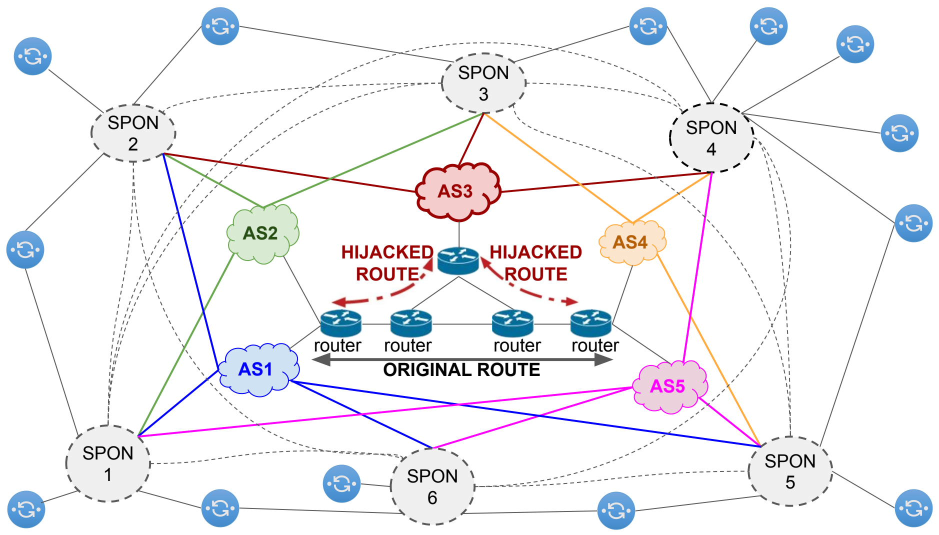


Figure 5.14: BGP attack mitigation with SPON.
ILP nodes in light blue; overlay links between SPON nodes in dashed curvy lines; SPON connections to different ASes/ISPs in straight colored lines. *Part of figure from [154].*

As represented in Figure 5.14, by deploying the SPON nodes in IXPs (Internet Exchange Points) and thus benefiting from access to say two or three ASes of interest, SPON nodes are able to ensure connectivity in spite of BGP attacks. For example, while the route between AS2 and AS4 is controlled by the adversary AS3 who partitioned AS2 from AS4, AS4 can still be reached from AS2 through SPON nodes placed appropriately in IXPs, with reduntant connections to multiple ASes, and thus still being able to relay traffic for their ILP clients located in AS2 and AS4, regardless of the hijacked route.

5.4.6 Results summary.

Synthesizing the results, in the scenarios and topologies we experimented with, SPON demonstrated:

1) Reasonable additional processing, measured as variation in latency of payments sent in ideal conditions (over links with no loss), as shown in Table 5.2:

Table 5.2: SPON additional processing.[a]

Payment latency variation	Best config.	Worst config.
Global topology (%)	+1.63	+5
Chain topology (%)	+5	+10

[a]"**config.**" means e.g. PRI-1P, REL-FLD,.., and lower values are better.

2) Performance gains in real-life conditions, measured as variation of the latency of payments sent over lossy links, as shown in Table 5.3:

Table 5.3: SPON gains in real-life conditions.[a]

Link loss	**2%**		**5%**		**10%**	
Payment latency variation	Best config.	Worst config.	Best config.	Worst config.	Best config.	Worst config.
Global topology (%)	-5.53	-2.36	-16.42	-13.74	-32.7	-31.16
Chain Topology (%)	-10.07	+3.21	-33.6	-18.28	-53.7	-43.3

[a]"config." means e.g. PRI-1P, REL-FLD,.., and lower values are better.

3) Fault tolerance (resilience to node and path failure): The payment latency variations observed in the experimental conditions described in section 5.4.3 were situated between -2.59 to +3%.

5.5 Discussion

Overlay Construction. The construction and deployment of overlay nodes significantly impact the performance and resilience of the application using them. The overlay allows the operator to deploy nodes to avoid some geographical areas, or to improve performance. However, in order to provide the resilience properties discussed in Section 5.3 the overlay needs to be carefully constructed as described in [153]. For example several properties require the overlay to provide multiple disjoint paths to ensure message delivery: in case some paths would fail, the remaining ones could still be able to deliver the messages. The overlay should be constructed such as the disjointness in the overlay topology is backed also by a disjointness at the lower level of the physical network infrastructure. If several overlay paths overlap at the underlying internet level there is an increased risk that a failure in an internet link will affect several overlay paths. To avoid this risk, it is recommended that nodes are deployed in strategic, carefully chosen locations. The usage of multiple network providers should increase resilience to ISP complete outages. Each SPON overlay node can connect to multiple ISPs to further improve the resilience of the SPON system. If, moreover, the same ISP is used at both ends of a link, the respective link will not be affected by an eventual BGP routing misdirection or attack.

Incremental Deployment. We do not expect the transition towards an overlay based system to occur in one single phase. In an operational environment, payment nodes will gradually join the overlay and thus incremental deployment is required. Because connections to an overlay node can be mapped to virtual tunnels and virtual interfaces [167], this multi-phased process can be implemented using standard tools for instrumenting a system level network stack.

Incentivization. The incentivization and monetization of nodes taking part in the overlay network is of paramount importance. Several business models are possible and technically viable. A service provider could run the nodes in several and well selected disjoint clouds and be monetized in a traditional pay-as-you-go or subscription based price model. Moreover, this deployment can be very cost-efficient with server-less implementations that do not require the permanent availability of a virtual machine. The dependence on a service provider might either require a trust based relationship or additional system level trusted execution environments [168]. Pay-as-you-go schemes can be easily implemented on top of ILP itself whereby client's data is admitted in the overlay as long as the former is paying to his overlay account. A service provider is though not required. ILP nodes and Spines nodes can be piggybacked and thus ILP connectors operate also as overlay nodes. In this scenario, the ILP routing stack coincides with the Spines overlay routing stack and incentivization and monetization is based on the ILP fee mechanism.

Different from solutions like Falcon, Fibre, bloXroute or SABRE to name a few, which act on the networking level supporting PoW DLTs - namely BTC for the aforementioned solutions, to improve the flooding speed or security of block propagation, and while it *could* technically do this also, SPON though acts on the networking level supporting global payment

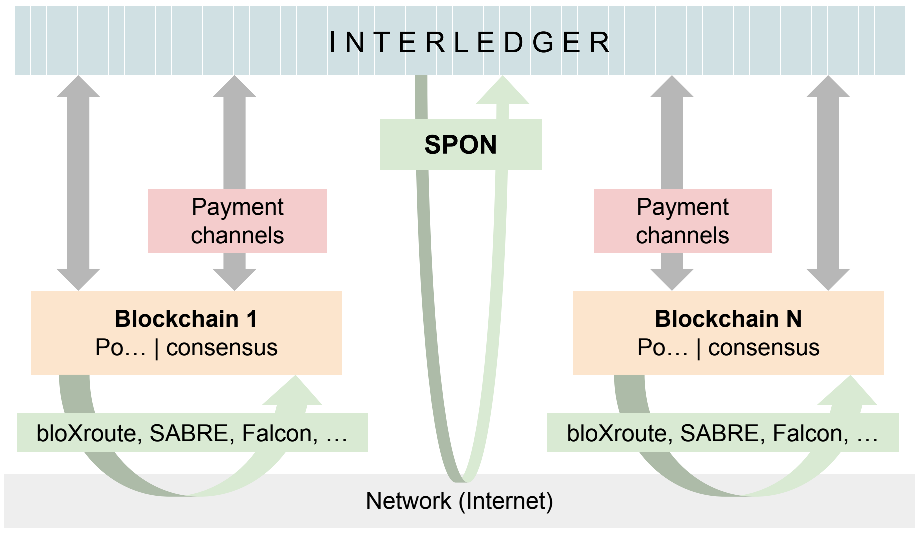


Figure 5.15: Positioning of SPON.

flows (e.g. Interledger, which is (sometimes) on top of payment channels, which are on top of possibly different DLTs) to improve the speed, resilience and security of data flows supporting global payments, like ILP STREAM. For more clarity, the positioning of SPON in this landscape is illustrated in Figure 5.15.

Moreover, all the above solutions but SPON are vulnerable to BGP failures.

To conclude, SPON Enables Resilient Inter-Ledgers Payments, by offering improved i) real-life performance, ii) service availability, and iii) resilience to BGP hijacking attacks through means of overlay relay nodes strategically deployed in chosen data center locations.

6

Discussion and Perspectives

6.1 Discussion

The research work presented here is focused on improving the security, resilience and efficiency of intra- and inter-ledger communication, with the concrete working case being the XRP ecosystem, which is payments-oriented. The payment systems are complex systems, with a network and a system component, and both of them have been investigated and discussed here.

6.1.1 The network component

The protocols and the architecture used in the network component play a major role, and they have been addressed in this thesis. More specifically, it was showed how specific advantages offered by overlay networks can be leveraged to improve the security, efficiency and resilience of network traffic for consensus-validation based blockchains and for blockchain inter-operability.

The consensus-validation blockchains have a flooding mechanism that lacked peer-reviewed research, with most related work having focused on other aspects like for example the consensus mechanism.

Therefore, this work investigated and showed how the dissemination of messages can be optimized using an overlay based on Named Data Networking with a minimal impact on security (a thorough security analysis was not performed and is future work). NDN is a promising overlay candidate because of its well-researched and optimized caching and flooding mechanisms. This work i) showed how the blockchain consensus messaging can be ported to use Named Data Networking message propagation and how can it be used, by proposing multiple mapping models for the transmission of the consensus-related messages, and ii) investigated the advantages and disadvantages of each of these models according to the specific needs of the blockchains using a consensus-validation system. The performance of the proposed solution, called *"XRP-NDN overlay"*, was evaluated in Chapter 3.

While the solution showed good results, a lot of engineering was required to divert the consensus messages to the overlay. Also, it was not evident how the mapping to NDN could be done and this involved design solutions and decisions. The solution proposed suits consensus-validation based blockchains because it leverages the small size of the consensus messages involved to *piggyback* them over the *interest* messages. However the size of the NDN interest message payload is limited, so for PoW-based blockchains where the block size is much larger, NDN-based pull solutions could be more suitable. *"XRP-NDN overlay"* [169] was accepted at *"IEEE/IFIP Network Operations and Management Symposium"* (NOMS), 8-12 May 2023.

The topic of distributed ledger interoperability was also investigated to see how the security, resilience and performance of communication can be improved at networking level, in the special context of inter-ledger connectors implementing the Interledger protocol. Aiming to see to what extent an overlay could improve the security at the networking level, *"**S**ecure **P**ayments with **O**verlay **N**etworks"* was proposed as an architecture for a global payment system that uses a reliable, intrusion-tolerant overlay network based on *Spines*, a concept well-known and studied in literature [1]. Interledger was investigated and mapped to

[1]https://www.cnds.jhu.edu/publications.html, valid in January 2023

Spines, then evaluated. The performance of the overlay architecture was evaluated and the advantages and disadvantages have been discussed. SPON provides (1) improved payment latency, (2) fault tolerance to benign failures such as node failures and network partitions, (3) resilience to routing attacks, while only incurring a small overhead.

The experimental results show that overlay networks are a viable solution for making global payment systems a reality by increasing their service availability and improving latency.

SPON was presented at the *"IEEE Conference on Communications and Network Security"* (CNS), 2021 [170].

6.1.2 The system component

The system component was also interesting to study, and the contribution of this thesis centers around methodologies to evaluate the system performance of a node and increase the security from the system level. The complexity of undertaking performance evaluations for containerized user-space applications, even when using Linux based profiling and tracing tools, was explained. The rationale behind the aim of performing non-intrusive performance monitoring was also discussed, and the added value of eBPF-based tools for performing non-intrusive performance monitoring of containerized user-space applications was demonstrated.

It was shown how eBPF can be used to achieve this on two Interledger implementations: Rafiki and the Reference Connector. The approach helped reveal for example how the processing of settlement was decoupled from the packet processing in Rafiki. As such, this thesis additionally explored the topic of blockchain interconnectivity at the system level: different architectures of Interledger implementations have been studied, with the goal to find performance bottlenecks in the different architectures and a solution for non-intrusive instrumentation at the eBPF level was proposed and implemented. Security wise, eBPF's added value also resides in its capability to enable the ILP connectors to better mitigate the *Fulfillment Risk* by *packet filtering* through white-listing or denial of service protection, and *redundant instances* - the difficulty to interfere with the program instance(s).

As researchers, we are always interested to perform precise measurements for our experimentation and this can be achieved with a better observability of the kernel. For system engineers, this offers the possibility to better point out the critical behavior of the applications, and for administrators this is the opportunity to better secure infrastructure and profile third-party applications.

This work involved experimentation with a Linux subsystem to monitor containerized user-space applications, and was published and presented at *"IEEE Symposium on Network Operations and Management" (NOMS) 2020* [152]. The experiments were carried in the context of the work on Interledger where the capability to monitor the software stack (i.e. ILP connectors) in production is a must-have. The tool landscape created to support eBPF was explored and assessed, and the contribution encompasses:

- The use of eBPF/XDP programs with one of the industry standard tools, Performance co-Pilot.
- Experimentation of the tools on Interledger connectors.

A monitoring and evaluation of two XRPL versions featuring different message dissemination capabilities were also performed on a real testbed deployed on *Grid 5000*. This confirmed the feasibility of the *squelching* solution performance-wise. Nevertheless the robustness and security of the approach should also be confirmed.

6.1.3 Conclusion

The scalability of the consensus-validation based blockchains is challenged by the flooding mechanism currently used in intra-ledger communication. *XRP-NDN overlay* showed how the efficiency of the message dissemination can be improved using an NDN-based overlay.

At system level, there is a need for live performance investigation and added security, in the case of running containerized applications treated as black-boxes. Taking as a working case the Interledger connectors, it was shown how these can be achieved by leveraging eBPF capabilities. This work contributes to improving the security and resilience of inter-ledger communication from the system level.

Blockchain interoperativity solutions, in particular the Interledger protocol enabling world-wide inter-ledger payments, can suffer from lossy links and vulnerability to routing hijacks or benign routing mis-directions. The proposed solution, SPON, uses a carefully designed overlay to solve these challenges.

6.2 Future work

With respect to the *XRPL-NDN Overlay*, larger and more life-like topologies could be investigated to also assess security aspects like for example resilience to poisoning attacks and the robustness of the chosen solution versus flooding. A cost analysis would be interesting to perform as well, and the analysis of different behaviors of the validators on the default UNL on the live XRP network with machine learning tools is also on the roadmap.

SPON-related experiments can be extended to include larger scenarios for the *fairness* of bandwidth allocation per overlay client (ILP connector), when more than two sources (overlay clients) are contending for bandwidth. Experiments could also include a *throughput* assessment.

The audit control packets sent in bloXroute could be implemented in SPON at the ILP level using STREAM. The *incentivization* of overlay operators proposed by *bloXroute* and *Falcon* could also potentially be implemented in *SPON* as future work. However, the monetization of SPON is still an open question and possible solutions to evaluate could be :

- Generating tokens that should be bought.
- Classic payment.
- A central operator of SPON, which raises the question of consensus between the interested parties.

During *the work with eBPF*, only the reference ILP connector and the Rafiki implementations were tested, however, multiple ILP connector architectures are available at this moment, and it would be interesting to see what the related results would be. Nowadays, in complex n-tier architectures, the challenge is to trace a request all along its journey. Future work could also assess how this type of tools can create an end-to-end view of a distributed system.

With respect to governance, on the other hand, it could also be argued that if the control of a network is placed in the hands of one or more entities, then the network is indirectly put in the hands of the government where those entities reside, and as such it could be, in extremis, controlled and censored. Hence, it may be desirable that, like blockchain p2p networks, overlays are operated and controlled by the entire respective community (decentralized). This might be a stronger foundation for an overlay, because any (small) group controlling it could at any time decide for various reasons to withdraw its support and shut it down. For example, Falcon domain www.falcon-net.org appears at the time of writing having been sold to a logistics-shipping company.

This work was supported by the Luxembourg National Research Fund through grant PRIDE15/10621687/SPsquared. In addition, we thankfully acknowledge the support from the RIPPLE University Blockchain Research Initiative (UBRI)[2] for our research.

[2]https://ubri.ripple.com/, valid in January 2023

List of Publications, Tutorials and Achievements during PhD Thesis Work

Publications. The work done in [171] and [172] was awarded the Grand Prize at the *UBRI 2020 Hackathon* and it was also presented at *UBRI Connect 2020*. The event spanned over a two-weeks period, with top US teams taking part. The work done in [173] was awarded an *XRPL grant* in 2021, and was also presented at *UBRI Connect 2022* in London, UK.

[173] Lucian Trestioreanu, Wazen M. Shbair, Flaviene Scheidt de Cristo, and Radu State. "Blockly2Hooks: Smart Contracts for Everyone with the XRP Ledger and Google Blockly". In: *(Accepted at:) 2023 IEEE/DAPPS International Conference on Decentralized Applications and Infrastructures.* 2023. URL: http://hdl.handle.net/10993/54819 (visited on 03/2023).

[174] Lucian Trestioreanu, Wazen M. Shbair, Flaviene Scheidt de Cristo, and Radu State. "XRP-NDN Overlay: Improving the Communication Efficiency of Consensus-Validation based Blockchains with an NDN Overlay". In: *NOMS 2023 - 2023 IEEE/IFIP Network Operations and Management Symposium.* 2023. arXiv: 2301.10209 [cs.NI]. (Visited on 03/2023).

[175] Lucian Trestioreanu, Cristina Nita-Rotaru, Aanchal Malhotra, and Radu State. "SPON: Enabling Resilient Inter-Ledgers Payments with an Intrusion-Tolerant Overlay". In: *2021 IEEE Conference on Communications and Network Security (CNS).* 2021, pp. 92–100. DOI: 10.1109/CNS53000.2021.9705048.

[176] Cyril Cassagnes, Lucian Trestioreanu, Clement Joly, and Radu State. "The rise of eBPF for non-intrusive performance monitoring". In: *NOMS 2020 - 2020 IEEE/I-*

FIP Network Operations and Management Symposium. 2020, pp. 1–7. DOI: 10.1109/NOMS47738.2020.9110434.

[177] Lucian Andrei Trestioreanu, Cyril Cassagnes, and Radu State. *Deep dive into Interledger: Understanding the Interledger ecosystem*. Tech. rep. University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust (SnT), 2019.

[171] Flaviene Scheidt de Cristo, Wazen M. Shbair, Lucian Trestioreanu, Aanchal Malhotra, and Radu State. "Privacy-Preserving PayString Service". In: *2021 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2021, pp. 1–3. DOI: 10.1109/ICBC51069.2021.9461076.

[172] Flaviene Scheidt De Cristo, Wazen M. Shbair, Lucian Trestioreanu, Radu State, and Aanchal Malhotra. "Self-Sovereign Identity for the Financial Sector: A Case Study of PayString Service". In: *2021 IEEE International Conference on Blockchain (Blockchain)*. 2021, pp. 213–220. DOI: 10.1109/Blockchain53845.2021.00036.

[178] Flaviene Scheidt De Cristo, Wazen M. Shbair, Lucian Trestioreanu, and Radu State. "Pub/sub Dissemination on the XRP Ledger". In: *(Under review at:) 2023 IEEE Global Communications Conference: Communication QoS, Reliability and Modeling*. 2023.

Tutorials. The tutorial [179] follows from the *Technical Report* [177].

[179] Lucian Trestioreanu, Cyril Cassagnes, and Radu State. "Deep Dive into Interledger: Understanding the Interledger Ecosystem". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2020. URL: https://icbc2020.ieee-icbc.org/tutorial-6.html (visited on 01/2023).

[180] Lucian Trestioreanu, Shbair Wazen, Cyril Cassagnes, and Radu State. "Ripple XRP Ledger: from Theory to Practice". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2022. URL: https://icbc2022.ieee-icbc.org/program/tutorials.html (visited on 01/2023).

Acronyms

PoA Proof of Authority. 9–11

PoS Proof of Stake. 9, 10

PoW Proof of Work. 9, 10, 14

SPSP Simple Payment Setup Protocol. 26, 27, 29, 97

STREAM Streaming Transport for the Realtime Exchange of Assets and Messages. 22, 26–29, 98, 101

UNL Unique Node List. 12

XRPL XRP Ledger. 2, 11–13, 15

Glossary

AS An autonomous system is a large network (or group) with a unified routing scheme. 110

CCC The exchange of information between one or more blockchains. 16

dApp Applications residing and running on a distributed network (e.g. blockchain). 18, 20

DeFi Financial technologies using distributed ledger technology. 21

Ethereum The decentralized blockchain which forms and uses a peer to peer network to verify and securely execute smart contract code. 109

IXP The Internet Exchange Points enable participant Internet Service Providers to exchange data between their networks. The Internet Exchange Points are usually placed in locations (datacenters) benefiting from connections to multiple networks, and operate physical infrastructure to interconnect the participants (ISPs). 110

Moneyd An ILP end-point, allowing all applications on an end-user computer to use funds on the live ILP network. 22, 28

Switch API A SDK for cross-chain trading between BTC, ETH, DAI and XRP with Interledger Streaming. 22

XRP XRPL's digital payment asset which is used for Interledger payments. 22

References

[1] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* (July 1982), pp. 382–401. URL: https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/ (visited on 03/2023).

[2] David Schatsky and Craig Muraskin. *Beyond bitcoin: Blockchain is coming to disrupt your industry.* Deloitte University Press. 2022. URL: https://www2.deloitte.com/us/en/insights/focus/signals-for-strategists/trends-blockchain-bitcoin-security-transparency.html (visited on 01/2023).

[3] Stefan Thomas and Evan Schwartz. *A Protocol for Interledger Payments.* 2016. URL: https://interledger.org/interledger.pdf (visited on 03/2023).

[4] Alexander Afanasyev et al. "A Brief Introduction to Named Data Networking". In: *MILCOM 2018 - 2018 IEEE Military Communications Conference (MILCOM)* (2018), pp. 1–6.

[5] Yair Amir et al. *Spines.* Mar. 2020. URL: http://spines.org (visited on 01/2023).

[6] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. *GossipSub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networks.* July 2020. URL: https://arxiv.org/abs/2007.02754 (visited on 03/2023).

[7] Nate Brawn and Ryan Huber. *Nebula.* 2019. URL: https://github.com/slackhq/nebula (visited on 01/2023).

[8] A. Rodriguez-Natal et al. “Programmable Overlays via OpenOverlayRouter”. In: *IEEE Communications Magazine* 55.6 (June 2017), pp. 32–38. ISSN: 1558-1896. DOI: 10.1109/MCOM.2017.1601056.

[9] Y. Rekhter, T. Li, and S. Hares. *A Border Gateway Protocol 4 (BGP-4)*. 2006. URL: https://www.rfc-editor.org/rfc/rfc4271 (visited on 03/2023).

[10] A. Babay, E. Wagner, M. Dinitz, and Y. Amir. “Timely, Reliable, and Cost-Effective Internet Transport Service Using Dissemination Graphs”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. June 2017, pp. 1–12. DOI: 10.1109/ICDCS.2017.63.

[11] Bengt Ahlgren, Christian Dannewitz, Claudio Imbrenda, Dirk Kutscher, and Borje Ohlman. “A survey of information-centric networking”. In: *IEEE Communications Magazine* 50.7 (2012), pp. 26–36. DOI: 10.1109/MCOM.2012.6231276.

[12] Interledger Foundation. *Interledger Protocol V4*. 2019. URL: https://interledger.org/rfcs/0027-interledger-protocol-4/ (visited on 03/2023).

[13] M.T. Ozsu and P. Valduriez. “Distributed database systems: where are we now?” In: *Computer* 24.8 (1991), pp. 68–78. DOI: 10.1109/2.84879.

[14] Ali Sunyaev. “Distributed Ledger Technology”. In: *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer International Publishing, 2020, pp. 265–299. ISBN: 978-3-030-34957-8. DOI: 10.1007/978-3-030-34957-8_9. URL: https://doi.org/10.1007/978-3-030-34957-8_9.

[15] CCSF Luxembourg. *Distributed Ledger Technologies (DLT) and blockchain*. Whitepaper. 2022. URL: https://www.cssf.lu/en/Document/white-paper-distributed-ledger-technologies-dlt-and-blockchain/ (visited on 03/2023).

[16] Santeri Paavolainen, Tommi Elo, and Pekka Nikander. *Interledger: Theory and practice*. IEEE ICBC 2019 tutorial. 2019. URL: http://site.ieee.org/icbc-2019/files/2019/05/ICBC-2019-Tutorial-4-Interledger.pdf (visited on 01/2023).

[17] Satoshi Nakamoto. “Bitcoin: A Peer-to-Peer Electronic Cash System”. In: *Cryptography Mailing list at https://metzdowd.com* (Mar. 2009). URL: https://bitcoin.org/bitcoin.pdf (visited on 01/2023).

[18] Fahad Saleh. “Blockchain without Waste: Proof-of-Stake”. In: *The Review of Financial Studies* 34 (July 2020). DOI: 10.1093/rfs/hhaa075.

[19] Janno Siim and Michał Zajac. *Proof of Stake.* Research Seminar in Cryptography. 2017. URL: https://courses.cs.ut.ee/MTAT.07.022/2017_fall/uploads/Main/janno-report-f17.pdf (visited on 03/2023).

[20] Stefano De Angelis, Leonardo Aniello, Federico Lombardi, Andrea Margheri, and V. Sassone. “PBFT vs proof-of-authority: applying the CAP theorem to permissioned blockchain”. In: *Italian Conference on Cybersecurity.* Jan. 2017.

[21] Caixiang Fan, Changyuan Lin, Hamzeh Khazaei, and Petr Musilek. “Performance Analysis of Hyperledger Besu in Private Blockchain”. In: *2022 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS).* IEEE. 2022, pp. 64–73.

[22] Stefano Dalla Palma, Remo Pareschi, and Federico Zappone. “What is your Distributed (Hyper)Ledger?” In: *2021 IEEE/ACM 4th International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB).* 2021, pp. 27–33. DOI: 10.1109/WETSEB52558.2021.00011.

[23] Xodex. *Flawless, Decentralized Innovation.* 2022. URL: https://www.xo-dex.com/assets/images/whitepaper.pdf (visited on 01/2023).

[24] Miguel Castro and Barbara Liskov. “Practical Byzantine Fault Tolerance”. In: *3rd Symposium on Operating Systems Design and Implementation (OSDI 99).* New Orleans, LA: USENIX Association, Feb. 1999. URL: https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance (visited on 01/2023).

[25] Ignacio Amores-Sesar, Christian Cachin, and Jovana Mićić. *Security Analysis of Ripple Consensus.* 2020. DOI: 10.48550/ARXIV.2011.14816. URL: https://arxiv.org/abs/2011.14816 (visited on 01/2023).

[26] David Schwartz, Noah Youngs, and Arthur Britto. *The Ripple Protocol Consensus Algorithm*. 2014.

[27] Gabriel Antonio F. Rebello, Gustavo F. Camilo, Lucas C. B. Guimarães, Lucas Airam C. de Souza, and Otto Carlos M. B. Duarte. "Security and Performance Analysis of Quorum-based Blockchain Consensus Protocols". In: *2022 6th Cyber Security in Networking Conference (CSNet)*. 2022, pp. 1–7. DOI: 10.1109/CSNet56116.2022.9955597.

[28] Frederik Armknecht, Ghassan O. Karame, Avikarsha Mandal, Franck Youssef, and Erik Zenner. "Ripple: Overview and Outlook". In: *Trust and Trustworthy Computing*. Ed. by Mauro Conti, Matthias Schunter, and Ioannis Askoxylakis. Cham: Springer International Publishing, 2015, pp. 163–180. ISBN: 978-3-319-22846-4.

[29] Lara Mauri, Stelvio Cimato, and Ernesto Damiani. "A Formal Approach for the Analysis of the XRP Ledger Consensus Protocol". In: Feb. 2020. DOI: 10.5220/0008954200520063.

[30] Chuanwang Ma et al. "Ripple+: An Improved Scheme of Ripple Consensus Protocol in Deployability, Liveness and Timing Assumption". In: *Computer Modeling in Engineering & Sciences* 130.1 (2022), pp. 463–481. ISSN: 1526-1506. DOI: 10.32604/cmes.2022.016838. URL: http://www.techscience.com/CMES/v130n1/45706 (visited on 03/2023).

[31] Brad Chase and Ethan MacBrough. "Analysis of the XRP Ledger Consensus Protocol". In: *CoRR* abs/1802.07242 (2018). arXiv: 1802.07242. URL: http://arxiv.org/abs/1802.07242 (visited on 01/2023).

[32] Sebastián Facundo D'Agostino and Juan Pablo Timpanaro. "Ripple Protocol performance improvement: Small world theory applied to cross border payments". In: *XIX Simposio Argentino de Ingeniería de Software (ASSE)* (Sept. 2018), pp. 143–154.

[33] Rameez Yousuf, Zubair Jeelani, Dawood Khan, Owais Bhat, and Tawseef Teli. "Consensus Algorithms in Blockchain-Based Cryptocurrencies". In: *International Conference on Advances in Electrical, Computing, Communication and Sustainable Technologies (ICAECT)*. Feb. 2021, pp. 1–6. DOI: 10.1109/ICAECT49130.2021.9392489.

[34] Marijn Roelvink, Mitchell Olsthoorn, and Annibale Panichela. *Log inference on the Ripple Protocol: testing the system with an empirical approach*. Delft University of Technology. June 2020. URL: http://resolver.tudelft.nl/uuid:ee55a433-e514-4507-8912-4196f0a9ba1c (visited on 03/2023).

[35] Crystal Andrea Roma and M. Anwar Hasan. "Energy Consumption Analysis of XRP Validator". In: *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2020, pp. 1–3. DOI: 10.1109/ICBC48266.2020.9169427.

[36] Hideaki Aoyama, Yoshi Fujiwara, Yoshimasa Hidaka, and Yuichi Ikeda. "Cryptoasset networks: Flows and regular players in Bitcoin and XRP". In: *PLOS ONE* 17 (Aug. 2022). DOI: 10.1371/journal.pone.0273068.

[37] Pedro Moreno-Sanchez, Navin Modi, Raghuvir Songhela, Aniket Kate, and Sonia Fahmy. "Mind Your Credit: Assessing the Health of the Ripple Credit Network". In: *WWW '18: Proceedings of the 2018 World Wide Web Conference*. Apr. 2018, pp. 329–338. ISBN: 978-1-4503-5639-8. DOI: 10.1145/3178876.3186099.

[38] Marios Touloupou, Klitos Christodoulou, Antonios Inglezakis, Elias Iosif, and Marinos Themistocleous. "Benchmarking Blockchains: The case of XRP Ledger and Beyond". In: *Hawaii International Conference on System Sciences*. Jan. 2022. DOI: 10.24251/HICSS.2022.730.

[39] Vytautas Tumas, Sean Rivera, Damien Magoni, and Radu State. "Centralized or not Centralized? Topology Analysis of the XRP Ledger". In: *The 38th ACM/SIGAPP Symposium On Applied Computing*. Mar. 2023.

[40] Wolf Bubberman and S. Roos. *TLS MITM attack on the Ripple XRP Ledger*. online, TU Delft. 2020. URL: http://resolver.tudelft.nl/uuid:393083dc-a364-477a-afc8-faaca0a244c6 (visited on 03/2023).

[41] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. "Erlay: Efficient Transaction Relay for Bitcoin". In: *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. CCS '19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 817–831. ISBN:

9781450367479. DOI: 10.1145/3319535.3354237. URL: https://doi.org/10.1145/3319535.3354237 (visited on 03/2023).

[42] Yifan Mao, Soubhik Deb, Shaileshh Bojja Venkatakrishnan, Sreeram Kannan, and Kannan Srinivasan. “Perigee: Efficient Peer-to-Peer Network Design for Blockchains”. In: *Proceedings of the 39th Symposium on Principles of Distributed Computing*. PODC ’20. Virtual Event, Italy: Association for Computing Machinery, 2020, pp. 428–437. ISBN: 9781450375825. DOI: 10.1145/3382734.3405704. URL: https://doi.org/10.1145/3382734.3405704 (visited on 03/2023).

[43] Joao Leitao, Jose Pereira, and Luis Rodrigues. “Epidemic Broadcast Trees”. In: *2007 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*. 2007, pp. 301–310. DOI: 10.1109/SRDS.2007.27.

[44] Miguel Castro et al. “SplitStream: High-Bandwidth Content Distribution in Cooperative Environments”. In: *Peer-to-Peer Systems II*. Ed. by M. Frans Kaashoek and Ion Stoica. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 292–303. ISBN: 978-3-540-45172-3.

[45] Divya Saxena. “Named Data Networking: A Survey”. In: *Elsevier Computer Science Review* 19 (Jan. 2016). DOI: 10.1016/j.cosrev.2016.01.001.

[46] Jiang Guo et al. “Enabling Blockchain Applications Over Named Data Networking”. In: *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*. 2019, pp. 1–6. DOI: 10.1109/ICC.2019.8761919.

[47] Quang Tung Thai, Namseok Ko, Sung Hyuk Byun, and Sun-Me Kim. “Design and implementation of NDN-based Ethereum blockchain”. In: *Journal of Network and Computer Applications* 200 (2022), p. 103329. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2021.103329. URL: https://www.sciencedirect.com/science/article/pii/S1084804521003143 (visited on 03/2023).

[48] George Sedky and Amr El Mougy. “BCXP: Blockchain-Centric Network Layer for Efficient Transaction and Block Exchange over Named Data Networking”. In: *2018 IEEE 43rd Conference on Local Computer Networks (LCN)*. 2018, pp. 449–452. DOI: 10.1109/LCN.2018.8638229.

[49] Tong Jin, Xiang Zhang, Yirui Liu, and Kai Lei. "BlockNDN: A bitcoin blockchain decentralized system over named data networking". In: *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. 2017, pp. 75–80. DOI: 10.1109/ICUFN.2017.7993751.

[50] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. "VectorSync: Distributed Dataset Synchronization over Named Data Networking". In: *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ICN '17. Berlin, Germany: Association for Computing Machinery, 2017, pp. 192–193. ISBN: 9781450351225. DOI: 10.1145/3125719.3132106. URL: https://doi.org/10.1145/3125719.3132106 (visited on 03/2023).

[51] Zhenkai Zhu and Alexander Afanasyev. "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking". In: *2013 21st IEEE International Conference on Network Protocols (ICNP)*. 2013, pp. 1–10. DOI: 10.1109/ICNP.2013.6733578.

[52] Minsheng Zhang, Vince Lehman, and Lan Wang. "Scalable name-based data synchronization for named data networking". In: *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*. 2017, pp. 1–9. DOI: 10.1109/INFOCOM.2017.8057193.

[53] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. *A brief introduction to state vector sync*. URL: https://named-data.net/wp-content/uploads/2021/07/ndn-0073-r2-SVS.pdf (visited on 03/2023).

[54] Gavin Wood. "Polkadot: Vision for a heterogeneous multi-chain framework". In: *White paper* 21.2327 (2016), p. 4662.

[55] Jae Kwon and Ethan Buchman. "Cosmos whitepaper". In: *A Netw. Distrib. Ledgers* (2019), p. 27.

[56] Kürsat Aydinli. "Performance Assessment of Cardano". In: *Independent Study Communication Systems Group* (2019), pp. 1–39.

[57] Guido Barbian and Florian Mellentin. *The Cardano Proof-of-Stake Protocol "Ouroboros"*. URL: https://www.researchgate.net/profile/Florian-Mellentin/publication/

358677239_Ouroboros_Cardano's_Proof-of-Stake_Consensus_Protocol/links/620edf6af02286737ca82524/Ouroboros-Cardanos-Proof-of-Stake-Consensus-Protocol.pdf (visited on 03/2023).

[58] Joseph Poon and Vitalik Buterin. "Plasma: Scalable autonomous smart contracts". In: *White paper* (2017), pp. 1–47.

[59] Gilbert Verdian, Paolo Tasca, Colin Paterson, and Gaetano Mondelli. "Quant overledger whitepaper". In: *Release V0* 1 (2018), p. 31.

[60] Vitalik Buterin. "Chain interoperability". In: *R3 Research Paper* (2016).

[61] T. Koens and E. Poll. "Assessing interoperability solutions for distributed ledgers". In: *Pervasive and Mobile Computing* 59 (2019), p. 101079. ISSN: 1574-1192. DOI: https://doi.org/10.1016/j.pmcj.2019.101079. URL: https://www.sciencedirect.com/science/article/pii/S1574119218306266 (visited on 01/2023).

[62] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. "A Survey on Blockchain Interoperability: Past, Present, and Future Trends". In: *ACM Comput. Surv.* 54.8 (Oct. 2021). ISSN: 0360-0300. DOI: 10.1145/3471140. URL: https://doi.org/10.1145/3471140 (visited on 03/2023).

[63] Vasilios A Siris, Dimitrios Dimopoulos, Nikos Fotiou, Spyros Voulgaris, and George C Polyzos. "Interledger smart contracts for decentralized authorization to constrained things". In: *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE. 2019, pp. 336–341.

[64] Joseph Poon and Thaddeus Dryja. *The bitcoin lightning network: Scalable off-chain instant payments*. 2016. URL: https://nakamotoinstitute.org/research/lightning-network/ (visited on 03/2023).

[65] Alexei Zamyatin et al. "SoK: Communication Across Distributed Ledgers". In: *IACR Cryptology ePrint Archive*. 2019.

[66] Jonas Nick, Andrew Poelstra, and Gregory Sanders. "Liquid: A bitcoin sidechain". In: *Liquid white paper.* (2020). URL: https://blockstream.com/assets/downloads/pdf/liquid-whitepaper.pdf (visited on 03/2023).

[67] Paolo Bellavista, Christian Esposito, Luca Foschini, Carlo Giannelli, and Nicola Mazzocca. "Interoperable Blockchains for Highly-Integrated Supply Chains in Collaborative Manufacturing". In: *Sensors* 21 (July 2021), p. 4955. DOI: 10.3390/s21154955.

[68] World Economic Forum. *Bridging the Governance Gap: Interoperability for blockchain and legacy systems*. Whitepaper. 2020. URL: https://www.weforum.org/whitepapers/bridging-the-governance-gap-interoperability-for-blockchain-and-legacy-systems/ (visited on 03/2023).

[69] Ghareeb Falazi et al. "Smart Contract Invocation Protocol (SCIP): A Protocol for the Uniform Integration of Heterogeneous Blockchain Smart Contracts". In: *Advanced Information Systems Engineering*. Ed. by Schahram Dustdar, Eric Yu, Camille Salinesi, Dominique Rieu, and Vik Pant. Cham: Springer International Publishing, 2020, pp. 134–149. ISBN: 978-3-030-49435-3.

[70] E. Fynn, A. Bessani, and F. Pedone. "Smart Contracts on the Move". In: *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. Los Alamitos, CA, USA: IEEE Computer Society, July 2020, pp. 233–244. DOI: 10.1109/DSN48063.2020.00040. URL: https://doi.ieeecomputersociety.org/10.1109/DSN48063.2020.00040.

[71] H. Abbas, M. Caprolu, and R. Di Pietro. "Analysis of Polkadot: Architecture, Internals, and Contradictions". In: *2022 IEEE International Conference on Blockchain (Blockchain)*. Los Alamitos, CA, USA: IEEE Computer Society, Aug. 2022, pp. 61–70. DOI: 10.1109/Blockchain55522.2022.00019. URL: https://doi.ieeecomputersociety.org/10.1109/Blockchain55522.2022.00019.

[72] Alfonso Cevallos and Alistair Stewart. "Validator election in nominated proof-of-stake". In: *CoRR* abs/2004.12990 (2020). arXiv: 2004.12990. URL: https://arxiv.org/abs/2004.12990 (visited on 03/2023).

[73] Hanaa Abbas, Maurantonio Caprolu, and Roberto Di Pietro. *Analysis of Polkadot: Architecture, Internals, and Contradictions*. 2022. DOI: 10.48550/ARXIV.2207.14128. URL: https://arxiv.org/abs/2207.14128 (visited on 03/2023).

[74] Polkadot. *Open Source Polkadot Stack*. 2022. URL: https://wiki.polkadot.network/docs/build-open-source (visited on 01/2023).

[75] Carlos Toro, Wei Wang, and Humza Akhtar. *Implementing Industry 4.0 The Model Factory as the Key Enabler for the Future of Manufacturing*. eng. 1st ed. 2021. Intelligent Systems Reference Library, 202. Cham: Springer International Publishing, 2021. ISBN: 9783030672706.

[76] XRPL Foundation. *Payment Channels*. 2019. URL: https://xrpl.org/payment-channels.html (visited on 03/2023).

[77] Kincaid O'Neil. *Settlement Architecture*. 2019. URL: https://github.com/interledger/rfcs/blob/master/deprecated/0024-ledger-plugin-interface-2/0024-ledger-plugin-interface-2.md (visited on 01/2023).

[78] Interledger Foundation. *Settlement Architecture*. 2019. URL: https://interledger.org/rfcs/0038-settlement-engines/ (visited on 01/2023).

[79] Ben Sharafian. *Moneyd - payment channels explanation*. 2019. URL: https://forum.interledger.org/t/moneyd-payment-channels-explanation/374/3 (visited on 01/2023).

[80] Evan Schwartz. *Protocol Stack Deep Dive - Boston Interledger Meetup*. URL: https://www.slideshare.net/Interledger/interledger-protocol-stack-deep-dive-boston-interledger-meetup (visited on 01/2023).

[81] Adam Langley et al. "The QUIC Transport Protocol: Design and Internet-Scale Deployment". In: *SIGCOMM '17: Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 2017.

[82] Evan Schwartz. *Thoughts on Scaling Interledger Connectors*. 2019. URL: https://medium.com/interledger-blog/thoughts-on-scaling-interledger-connectors-7e3cad0dab7f (visited on 01/2023).

[83] Interledger Foundation. *Simple Payment Setup Protocol (SPSP)*. 2019. URL: https://github.com/interledger/rfcs/blob/master/0009-simple-payment-setup-protocol/0009-simple-payment-setup-protocol.md (visited on 01/2023).

[84] Interledger Foundation. *Payment Pointers and Payment Setup Protocols*. 2019. URL: https://github.com/interledger/rfcs/blob/master/0026-payment-pointers/0026-payment-pointers.md (visited on 01/2023).

[85] Evan Schwartz. *STREAMing Money and Data Over ILP*. 2019. URL: https://medium.com/interledger-blog/streaming-money-and-data-over-ilp-fabd76fc991e (visited on 01/2023).

[86] Interledger Foundation. *STREAM: A Multiplexed Money and Data Transport for ILP*. 2019. URL: https://interledger.org/rfcs/0029-stream/ (visited on 01/2023).

[87] The Interledger Foundation. *Case Study: COIL*. 2022. URL: https://interledger.org/case-studies/coil/ (visited on 01/2023).

[88] Interledger Foundation. *Relationship between Protocols*. 2019. URL: https://interledger.org/rfcs/0033-relationship-between-protocols/ (visited on 01/2023).

[89] Interledger Foundation. *Interledger Architecture*. 2019. URL: https://interledger.org/rfcs/0001-interledger-architecture/#protocol-layers (visited on 01/2023).

[90] M. Apostolaki, A. Zohar, and L. Vanbever. "Hijacking Bitcoin: Routing Attacks on Cryptocurrencies". In: *2017 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 375–392. DOI: 10.1109/SP.2017.29.

[91] Maria Apostolaki, Gian Marti, Jan Müller, and Laurent Vanbever. "SABRE: Protecting Bitcoin against Routing Attacks". In: *26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019*. The Internet Society, 2019. URL: https://www.ndss-symposium.org/ndss-paper/sabre-protecting-bitcoin-against-routing-attacks/ (visited on 01/2023).

[92] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. *Decentralization in Bitcoin and Ethereum Networks*. Springer Berlin Heidelberg, 2018. DOI: https://doi.org/10.1007/978-3-662-58387-6_24.

[93] Matt Corallo. *FIBRE Fast Internet Bitcoin Relay Engine*. URL: https://bitcoinfibre.org/ (visited on 01/2023).

[94] Uri Klarman, Soumya Basu, Aleksandar Kuzmanovic, and Emin Gün Sirer. "bloXroute: A Scalable Trustless Blockchain Distribution Network WHITEPAPER". In: *IEEE Internet of Things Journal.* 2018.

[95] Wei Bi, Huawei Yang, and Maolin Zheng. "An Accelerated Method for Message Propagation in Blockchain Networks". In: *ArXiv* abs/1809.00455 (2018).

[96] Y. Amir, C. Danilov, S. Goose, D. Hedqvist, and A. Terzis. "An Overlay Architecture for High-Quality VoIP Streams". In: *IEEE Transactions on Multimedia* 8.6 (Dec. 2006), pp. 1250–1262. ISSN: 1941-0077. DOI: 10.1109/TMM.2006.884609.

[97] Paolo Bellavista, Christian Esposito, Luca Foschini, Carlo Giannelli, and Nicola Mazzocca. "Interoperable Blockchains for Highly-Integrated Supply Chains in Collaborative Manufacturing". In: *Sensors* 21 (July 2021), p. 4955. DOI: 10.3390/s21154955.

[98] CryptoEQ Analysts Team. *Polkadot core report.* online, Houston, TX. 2022. URL: https://www.cryptoeq.io/corereports/polkadot-abridged (visited on 01/2023).

[99] Gavin Wood. *Gov2: Polkadot's Next Generation of Decentralised Governance.* online. 2022. URL: https://polkadot.network/blog/gov2-polkadots-next-generation-of-decentralised-governance/ (visited on 03/2023).

[100] Peter Somogyvari and Azahara C. *Cactus Governance.* online. 2021. URL: https://github.com/hyperledger/cactus/blob/main/GOVERNANCE.md (visited on 01/2023).

[101] Lihao Zhang, Taotao Wang, and Soung Chang Liew. "Speeding up Block Propagation in Bitcoin Network: Uncoded and Coded Designs". In: *Comput. Netw.* 206.C (Apr. 2022). ISSN: 1389-1286. DOI: 10.1016/j.comnet.2022.108791. URL: https://doi.org/10.1016/j.comnet.2022.108791.

[102] Kai Otsuki, Yusuke Aoki, Ryohei Banno, and Kazuyuki Shudo. "Effects of a Simple Relay Network on the Bitcoin Network". In: Aug. 2019, pp. 41–46. ISBN: 978-1-4503-6849-0. DOI: 10.1145/3340422.3343640.

[103] Sen Yang et al. *SoK: MEV Countermeasures: Theory and Practice.* 2022. DOI: 10.48550/ARXIV.2212.05111. URL: https://arxiv.org/abs/2212.05111 (visited on 01/2023).

[104] Kaihua Qin and Arthur Gervais. *An overview of blockchain scalability, interoperability and sustainability.* URL: https://www.eublockchainforum.eu/sites/default/files/research-paper/an_overview_of_blockchain_scalability_interoperability_and_sustainability.pdf (visited on 01/2023).

[105] Gianmaria Del Monte, Diego Pennino, and Maurizio Pizzonia. “Scaling Blockchains without Giving up Decentralization and Security: A Solution to the Blockchain Scalability Trilemma”. In: *Proceedings of the 3rd Workshop on Cryptocurrencies and Blockchains for Distributed Systems.* CryBlock ’20. London, United Kingdom: Association for Computing Machinery, 2020, pp. 71–76. ISBN: 9781450380799. DOI: 10.1145/3410699.3413800. URL: https://doi.org/10.1145/3410699.3413800.

[106] Damien Magoni. “Network Topology Analysis and Internet Modelling with Nem”. In: *International Journal of Computers and Applications* 27.4 (2005), pp. 252–259. DOI: 10.2316/Journal.202.2005.4.202-1540. URL: https://hal.science/hal-00344484.

[107] G. Tsipenyuk and N. D. Bougalis. *Message routing optimizations, pt. 1: Proposal & validation relaying.* 2021. URL: https://xrpl.org/blog/2021/%20message-routing-optimizations-pt-1-proposal-validation-relaying.html (visited on 03/2023).

[108] Saurab Dulal and Lan Wang. “Adaptive Duplicate Suppression for Multicasting in a Multi-Access NDN Network”. In: *Proceedings of the 9th ACM Conference on Information-Centric Networking.* ICN ’22. Osaka, Japan: Association for Computing Machinery, 2022, pp. 156–158. ISBN: 9781450392570. DOI: 10.1145/3517212.3559480. URL: https://doi.org/10.1145/3517212.3559480.

[109] Van Jacobson et al. “Networking Named Content”. In: *Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies.* CoNEXT ’09. Rome, Italy: Association for Computing Machinery, 2009, pp. 1–12. ISBN: 9781605586366. DOI: 10.1145/1658939.1658941. URL: https://doi.org/10.1145/1658939.1658941.

[110] Junxiao Shi. *NDNts: Named Data Networking libraries for the Modern Web - codebase*. https://github.com/yoursunny/NDNts. (Visited on 01/2023).

[111] Junxiao Shi. *NDNts: Named Data Networking libraries for the Modern Web - homepage*. https://yoursunny.com/p/NDNts/. (Visited on 01/2023).

[112] Teemu Toivola. *Vnstat*. online. URL: https://humdi.net/vnstat/ (visited on 01/2023).

[113] The Wireshark Foundation. *tshark*. online. URL: https://www.wireshark.org/docs/man-pages/tshark.html (visited on 01/2023).

[114] Wazen M. Shbair, Mathis Steichen, Jérôme François, and Radu State. "BlockZoom: Large-Scale Blockchain Testbed". In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. 2019, pp. 5–6. DOI: 10.1109/BLOC.2019.8751230.

[115] Betsy Beyer and Rob Ewaschuk. *Monitoring Distributed Systems*. Ed. by O'Reilly. O'Reilly Media, Inc., 2016.

[116] G. Liu and T. Wood. "Cloud-Scale Application Performance Monitoring with SDN and NFV". In: *Proceedings of the IEEE International Conference on Cloud Engineering (IC2E)*. Mar. 2015, pp. 440–445. DOI: 10.1109/IC2E.2015.45.

[117] Dirk Merkel. "Docker: Lightweight Linux Containers for Consistent Development and Deployment". In: *Linux Journal* 2014.239 (Mar. 2014). ISSN: 1075-3583.

[118] Elazar Gershuni et al. "Simple and Precise Static Analysis of Untrusted Linux Kernel Extensions". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI 2019. Phoenix, AZ, USA: ACM, 2019, pp. 1069–1084. ISBN: 978-1-4503-6712-7. DOI: 10.1145/3314221.3314590.

[119] Luca Deri, Samuele Sabella, and Simone Mainardi. "Combining System Visibility and Security Using eBPF". In: *Proceedings of the Third Italian Conference on Cyber Security (ITASEC)*. Vol. Vol-2315. ITASEC'19. 2019, pp. 50–62.

[120] Cilium Authors community. *BPF and XDP Reference Guide*. https://docs.cilium.io/en/v1.6/bpf. 2019. (Visited on 03/2023).

[121] Dominik Scholz et al. “Performance Implications of Packet Filtering with Linux eBPF”. In: *2018 30th International Teletraffic Congress (ITC 30)*. Vol. 01. Sept. 2018, pp. 209–217. DOI: 10.1109/ITC30.2018.00039.

[122] T. Nam and J. Kim. “Open-source IO visor eBPF-based packet tracing on multiple network interfaces of Linux boxes”. In: *2017 International Conference on Information and Communication Technology Convergence (ICTC)*. Oct. 2017, pp. 324–326. DOI: 10.1109/ICTC.2017.8190996.

[123] Kun Suo, Yong Zhao, Wei Chen, and Jia Rao. “vNetTracer: Efficient and Programmable Packet Tracing in Virtualized Networks”. In: *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*. July 2018, pp. 165–175. DOI: 10.1109/ICDCS.2018.00026.

[124] S. Baidya, Y. Chen, and M. Levorato. “eBPF-based content and computation-aware communication for real-time edge computing”. In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. Apr. 2018, pp. 865–870. DOI: 10.1109/INFCOMW.2018.8407006.

[125] Zaafar Ahmed, Muhammad Hamad Alizai, and Affan A. Syed. “InKeV: In-kernel Distributed Network Virtualization for DCN”. In: *SIGCOMM Comput. Commun. Rev.* 46.3 (July 2018), 4:1–4:6. ISSN: 0146-4833. DOI: 10.1145/3243157.3243161.

[126] Thomas Graf. “Accelerating Envoy with the Linux Kernel”. In: *CloudNativeCon Europe and KubeCon Europe*. 2018.

[127] Toke Høiland-Jørgensen et al. “The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel”. In: *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*. CoNEXT ’18. Heraklion, Greece: ACM, 2018, pp. 54–66. ISBN: 978-1-4503-6080-7. DOI: 10.1145/3281411.3281443.

[128] Simon Jouet and Dimitrios P. Pezaros. “BPFabric: Data Plane Programmability for Software Defined Networks”. In: *2017 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. May 2017, pp. 38–48. DOI: 10.1109/ANCS.2017.14.

[129] Luca Deri and Samuele Sabella. "Merging System and Network Monitoring with BPF". In: *Open Source Developers' European Meeting (FOSDEM)*. 2019.

[130] Jibum Hong, Seyeon Jeong, Jae-Hyoung Yoo, and James W. Hong. "Design and Implementation of eBPF-based Virtual TAP for Inter-VM Traffic Monitoring". In: *2018 14th International Conference on Network and Service Management (CNSM)*. Nov. 2018, pp. 402–407.

[131] C. B. Hauser and S. Wesner. "Reviewing Cloud Monitoring: Towards Cloud Resource Profiling". In: *Proceedings of the IEEE 11th International Conference on Cloud Computing (CLOUD)*. July 2018, pp. 678–685. DOI: 10.1109/CLOUD.2018.00093.

[132] Yu Gan et al. "An Open-Source Benchmark Suite for Microservices and Their Hardware-Software Implications for Cloud & Edge Systems". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS '19. Providence, RI, USA: ACM, 2019, pp. 3–18. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304013.

[133] Evan Schwartz. "A Payment Protocol of the Web, for the Web: Or, Finally Enabling Web Micropayments with the Interledger Protocol". In: *Proceedings of the 25th International Conference Companion on World Wide Web*. WWW '16 Companion. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 279–280. ISBN: 978-1-4503-4144-8. DOI: 10.1145/2872518.2889305.

[134] Brendan Gregg. "Performance Superpowers with Enhanced BPF". In: *Proceedings of USENIX Annual Technical Conference (ATC)*. Santa Clara, CA: USENIX Association, July 2017.

[135] Mohamad Gebai and Michel R. Dagenais. "Survey and Analysis of Kernel and Userspace Tracers on Linux: Design, Implementation, and Overhead". In: *ACM Comput. Survey* 51.2 (Mar. 2018), 26:1–26:33. ISSN: 0360-0300. DOI: 10.1145/3158644.

[136] Matheus Marchini. "Enhancing User Defined Tracepoints". In: *Linux Plumbers Conference (LPC)*. 2018.

[137] Brendan Gregg. “System observability with BPF”. In: *Linux Storage, Filesystem, and Memory-Management Summit (LSFMM)*. 2019.

[138] Brendan Gregg. *BPF Performance Tools*. Addison-Wesley Professional, 2019. DOI: 9780136588870.

[139] Alastair Robertson. *bpftrace*. 2023. URL: https://github.com/iovisor/bpftrace (visited on 01/2023).

[140] Toke Høiland-Jørgensen and Jesper Dangaard Brouer. “XDP - Challenges and Future Work”. In: *Linux Plumbers Conference (LPC)*. 2018.

[141] Viet-Hoang Tran and Olivier Bonaventure. “Making the Linux TCP stack more extensible with eBPF”. In: *Netdev 0x13*. 2019.

[142] Zhiming Shen et al. “X-Containers: Breaking Down Barriers to Improve Performance and Isolation of Cloud-Native Containers”. In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ASPLOS ’19. Providence, RI, USA: ACM, 2019, pp. 121–135. ISBN: 978-1-4503-6240-5. DOI: 10.1145/3297858.3304016.

[143] Jianbin Wei and Cheng-Zhong Xu. “sMonitor: A Non-intrusive Client-perceived End-to-end Performance Monitor of Secured Internet Services”. In: *Proceedings of the Annual Conference on USENIX ’06 Annual Technical Conference*. ATEC ’06. Boston, MA: USENIX Association, 2006, pp. 21–21.

[144] M. Wagner, J. Doleschal, A. Knüpfer, and W. E. Nagel. “Selective runtime monitoring: Non-intrusive elimination of high-frequency functions”. In: *Proceedings of the International Conference on High Performance Computing Simulation (HPCS)*. July 2014, pp. 295–302. DOI: 10.1109/HPCSim.2014.6903698.

[145] Tianwei Sheng et al. “RACEZ: A Lightweight and Non-invasive Race Detection Tool for Production Applications”. In: *Proceedings of the 33rd International Conference on Software Engineering*. ICSE ’11. Waikiki, Honolulu, HI, USA: ACM, 2011, pp. 401–410. ISBN: 978-1-4503-0445-0. DOI: 10.1145/1985793.1985848.

[146] B. Sengupta, N. Banerjee, A. Anandkumar, and C. Bisdikian. “Non-intrusive transaction monitoring using system logs”. In: *Proceedings of the IEEE Network Operations and Management Symposium (NOMS)*. Apr. 2008, pp. 879–882. DOI: 10.1109/NOMS.2008.4575237.

[147] C. E. T. de Oliveira and R. F. Junior. “A transparent and centralized performance management service for CORBA based applications”. In: *Proceedings of the IEEE Network Operations and Management Symposium NOMS (IEEE Cat. No.04CH37507)*. Vol. 1. Apr. 2004, 439–452 Vol.1. DOI: 10.1109/NOMS.2004.1317684.

[148] Alban Crequy. “bpftrace meets Kubernetes with kubectl-trace”. In: *Open Source Developers’ European Meeting (FOSDEM)*. 2019.

[149] Interledger Foundation. *The Bilateral Transfer Protocol*. 2019. URL: https://interledger.org/rfcs/0023-bilateral-transfer-protocol/ (visited on 01/2023).

[150] Adrian Hope-Bailie and Stefan Thomas. “Interledger: Creating a Standard for Payments”. In: *Proceedings of the 25th International Conference Companion on World Wide Web*. WWW ’16 Companion. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, 2016, pp. 281–282. ISBN: 978-1-4503-4144-8. DOI: 10.1145/2872518.2889307.

[151] Lucian Andrei Trestioreanu, Cyril Cassagnes, and Radu State. *Deep dive into Interledger: Understanding the Interledger ecosystem*. Tech. rep. University of Luxembourg, Interdisciplinary Centre for Security, Reliability and Trust (SnT), 2019.

[152] Cyril Cassagnes, Lucian Trestioreanu, Clement Joly, and Radu State. “The rise of eBPF for non-intrusive performance monitoring”. In: *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*. 2020, pp. 1–7. DOI: 10.1109/NOMS47738.2020.9110434.

[153] D. Obenshain et al. “Practical Intrusion-Tolerant Networks”. In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. June 2016, pp. 45–56. DOI: 10.1109/ICDCS.2016.99.

[154] P. Ekparinya, V. Gramoli, and G. Jourjon. “Impact of Man-In-The-Middle Attacks on Ethereum”. In: *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)*. 2018, pp. 11–20. DOI: 10.1109/SRDS.2018.00012.

[155] *ILP Torrent - The technical deep dive*. https://coil.com/p/sabinebertram/ILP-Torrent-The-technical-deep-dive/S2cdTMKby/. (Visited on 01/2023).

[156] Terri Friedline, Sruthi Naraharisetti, and Addie Weaver. “Digital Redlining: Poor Rural Communities’ Access to Fintech and Implications for Financial Inclusion”. In: *Journal of Poverty* 24.5-6 (2020), pp. 517–541. DOI: 10.1080/10875549.2019.1695162. eprint: https://doi.org/10.1080/10875549.2019.1695162. URL: https://doi.org/10.1080/10875549.2019.1695162.

[157] Ahren Studer and Adrian Perrig. “The Coremelt Attack”. In: *Computer Security – ESORICS 2009*. Ed. by Michael Backes and Peng Ning. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 37–52. ISBN: 978-3-642-04444-1.

[158] Min Suk Kang, Soo Bum Lee, and Virgil D. Gligor. “The Crossfire Attack”. In: *2013 IEEE Symposium on Security and Privacy*. 2013, pp. 127–141. DOI: 10.1109/SP.2013.19.

[159] Claudiu Danilov. “Performance and Functionality in Overlay Networks”. PhD thesis. Baltimore: The Johns Hopkins University, Sept. 2004. URL: http://www.dsn.jhu.edu/~yairamir/Claudiu_thesis.pdf (visited on 03/2023).

[160] A. Babay et al. “Structured Overlay Networks for a New Generation of Internet Services”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. June 2017, pp. 1771–1779. DOI: 10.1109/ICDCS.2017.119.

[161] B. Heller, N. McKeown, C. Kozyrakis, G.M. Parulkar, and Stanford University. Computer Science Department. *Reproducible Network Research with High-fidelity Emulation*. Stanford University, 2013. URL: https://books.google.lu/books?id=_EFzAQAACAAJ.

[162] Aggelos Koukounas, Eleni Vytogianni, and Marnix Dekker. *7 Steps to shore up the Border Gateway Protocol (BGP)*. https://www.enisa.europa.eu/publications/7-steps-to-shore-up-bgp. Accessed: January 2023.

[163] C. Demchak and Y. Shavitt. "China's Maxim – Leave No Access Point Unexploited: The Hidden Story of China Telecom's BGP Hijacking". In: *Military Cyber Affairs*. Vol. 3. 1. Oct. 2018.

[164] Pablo Moriano, Raquel Hill, and L. Jean Camp. "Using bursty announcements for detecting BGP routing anomalies". In: *Computer Networks* 188 (2021), p. 107835. ISSN: 1389-1286. DOI: https://doi.org/10.1016/j.comnet.2021.107835. URL: https://www.sciencedirect.com/science/article/pii/S1389128621000207.

[165] Rob Portman and Tom Carper. "THREATS TO U.S. NETWORKS: OVERSIGHT OF CHINESE GOVERNMENT-OWNED CARRIERS". In: *United States Senate, Committee on Homeland Security and Governmental Affairs*. June 2020, p. 31.

[166] Doug Madory. *Oracle Confirms Research: China Telecom Misdirected U.S. Internet traffic thru China*. https://techblog.comsoc.org/2018/11/07/oracle-confirms-research-china-telecom-misdirected-u-s-internet-traffic-thru-china/. Accessed: January 2023.

[167] Rami Rosen. *Linux Kernel Networking: Implementation and Theory*. 1st. USA: Apress, 2013. ISBN: 143026196X.

[168] Frank McKeen et al. "Intel® Software Guard Extensions (Intel® SGX) Support for Dynamic Memory Management Inside an Enclave". In: *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*. HASP 2016. Seoul, Republic of Korea: Association for Computing Machinery, 2016. ISBN: 9781450347693. DOI: 10.1145/2948618.2954331. URL: https://doi.org/10.1145/2948618.2954331.

[169] Lucian Trestioreanu, Wazen M. Shbair, Flaviene Scheidt de Cristo, and Radu State. *XRP-NDN Overlay: Improving the Communication Efficiency of Consensus-Validation based Blockchains with an NDN Overlay*. 2023. arXiv: 2301.10209 [cs.NI]. (Visited on 03/2023).

[170] Lucian Trestioreanu, Cristina Nita-Rotaru, Aanchal Malhotra, and Radu State. "SPON: Enabling Resilient Inter-Ledgers Payments with an Intrusion-Tolerant Overlay". In: *2021 IEEE Conference on Communications and Network Security (CNS)*. 2021, pp. 92–100. DOI: 10.1109/CNS53000.2021.9705048.