UNIVERZA NA PRIMORSKEM
FAKULTETA ZA MATEMATIKO, NARAVOSLOVJE IN
INFORMACIJSKE TEHNOLOGIJE

Master's thesis
(Magistrsko delo)

The role of visualization and data analysis in blockchain networks

(Vloga vizualizacije in analize podatkov v tehnologiji veriženja blokov)

Ime in priimek: *Daniil Baldouski*
Študijski program: *Podatkovna znanost, 2. stopnja*
Mentor: *prof. dr. Michael Mrissa*
Somentor: *asist. Aleksandar Tošić*

Koper, september 2022

Ključna dokumentacijska informacija

Ime in PRIIMEK: Daniil BALDOUSKI
Naslov magistrskega dela: Vloga vizualizacije in analize podatkov v tehnologiji veriženja blokov

Kraj: Koper
Leto: 2022

Število listov: 55 Število slik: 28

Število prilog: 1 Število strani prilog: 4

Število referenc: 66

Mentor: prof. dr. Michael Mrissa

Somentor: asist. Aleksandar Tošić

UDK: 004.7(043.2)
Ključne besede: porazdeljeni sistemi, tehnologija veriženja blokov, vizualizacija
Izvleček:
V zadnjem desetletju smo priča hitremu razvoju tehnologije veriženja blokov. Tej sistemi s pomočjo protokolov tvorijo decentralizirana omrežja. Vozlišča hranijo verigo blokov, ki jo gradijo s pomočjo mehanizma za dosego konsenza. Tovrstnih mehanizmov je veliko. Največja je družina mehanizmov, ki temeljijo na glasovanju, kjer se na različne načine vozliščem dodelijo vloge na podlagi katerih glasujejo, preštevajo glasove, in gradijo verigo. Zaradi decentralizirane narave je take sisteme zelo težko razhroščvati. V tej magistrski nalogi si bomo ogledali obstoječe pristope za razhrošćevanje in analizo tovrstnih sistemov s pomočjo vizualizacije in analize podatkov.

Key document information

Name and SURNAME: Daniil BALDOUSKI
Title of the thesis: The role of visualisation and data analysis in blockchain networks

Place: Koper
Year: 2022

Number of pages: 55 Number of figures: 28

Number of appendices: 1 Number of appendix pages: 4

Number of references: 66

Mentor: Prof. Michael Mrissa, PhD

Co-Mentor: Assist. Aleksandar Tošić

UDC: 004.7(043.2)
Keywords: distributed systems, blockchain, visualisation
Abstract:
In the past decade, decentralized systems have been increasingly gaining more attention. Much of the attention arguably comes from both financial, and sociological acceptance, and adoption of blockchain technology. One of the frontiers has been the design of new consensus protocols, topology optimisation in these peer to peer(P2P) networks, and gossip protocol design. Analogue to agent based systems, transitioning from the design to implementation is a difficult task. This is due to the inherent nature of such systems, where nodes or actors within the system only have a local view of the system with very little guarantees on availability of data. Additionally, such systems often offer no guarantees of a system wide time synchronisation. This research offers insight into the importance of visualisation techniques in the implementation phase of vote based consensus algorithms, and P2P overlay network topology. We overview a selected set of state of technologies for visualising time series data, and present our architecture. We present our custom visualisations, and note their usefulness in debugging, and identifying potential issues in decentralized networks. Our use case is an implementation of an innovative blockchain protocol, which we briefly describe.

List of Contents

List of Figures

List of Abbreviations

i.e.	that is
e.g.	for example
DLT	distributed ledger technology
PoS	proof of stake
PoW	proof of work
CPU	central processing unit
P2P	peer-to-peer
F2F	friend-to-friend
MITM	man-in-the-middle
L1	layer 1
L2	layer 2
L3	layer 3
2PC	two phase commit
VDF	verifiable delay function
RNG	random number generator
CERN	Conseil Européen pour la Recherche Nucléaire
DHCP	dynamic host configuration protocol
DNS	domain name system
MIT	Massachusetts Institute of Technology
GUI	graphical user interface

Acknowledgments

I would like to thank my mentor prof. dr. Michael Mrissa and co-mentor Aleksandar Tošić for introducing me to this topic and their guidance for completing this work.

1 INTRODUCTION

Distributed systems are notoriously difficult to inspect and their problems difficult to identify. The difficulty stems from the fact that predominant issues can be stochastic and difficult to reproduce, and from the inability to easily observe, compare, and test multiple programs running on separate machines at the same time. Another important aspect in distributed systems is that they inherently make heavy use of the network. The use of various network protocols imposes additional complexity, which increases the search space in identifying bugs. In recent years, distributed systems have been gaining more attention both in academia and private sector. This increasing interest can be largely attributed to the rapid development of distributed ledger technology, and blockchain. In recent years, many new consensus mechanisms, blockchain protocols, network protocols, improvement in gossip protocols have been proposed. Many of them are transitioning from a theoretical framework to a practical implementation. However, public distributed ledgers (or distributed ledger technology or DLT) and blockchains secure their consensus mechanisms and provide spam resistance through the use of tokens representing value. The use of digital value within the protocol enables the protocol to enforce a level of security through economic incentives, and game theoretical aspects that make most attack vectors economically infeasible or impractical for the attacker. A good example of this is the Proof of Stake (PoS) consensus mechanism, where nodes in the decentralized protocol secure the consensus mechanism by requiring to stake and lock up a considerable amount of value, which can be deducted (usually refereed to as slashing) by the protocol in case the node misbehaves. The economic aspect of public blockchains poses a very high security risk. With such strong economic incentives to identify and exploit potential bugs, and system faults, it is of upmost importance for the developers to thoroughly test and examine potential problems. However, the aforementioned difficulties in debugging distributed and decentralized protocols require developers to be equipped with tools that supports their efforts.

In this work we plan to investigate the structure of distributed systems with the main focus on peer-to-peer (P2P) networks. Finding a use case of a blockhain network we want to consider possible challenges and issues related to the implementation of such network. We plan to create data visualization tools for inspecting decentralized networks and their voting based consensus mechanisms, to help developers with analysis and debugging of such systems. Such tools should be open-source and easily configurable to be able to comply with the needs of different implementations of the

considered distributed networks.

In Chapter 2, we present an overview of many known visualization techniques with examples of their usage. We mention several existing tools that are widely used for constructing various types of visualizations. We give an introduction to all the necessary definitions related to the distributed systems and, specifically, to the research project. In the section 2.4 we consider Bitcoin, being one of the most famous implementations of blockchain technology and mention issues, related to the energy consumption of its network.

An important part of the distributed system is its protocols. In Chapter 3 we present various existing visualization projects that are being used to visualize some of the most famous protocols of the distributed systems. We consider routing protocols, that are being used to send messages between nodes and consensus protocols, that manage a correct state of the network. All inherent network protocols should be tolerant to various kinds of faults in the system. In section 3.3 we address the most important considerations for the networks, which is the Byzantine behaviour of its actors.

In Chapter 4 we review different approaches in testing and debugging complex distributed systems and show the existing solutions for the considered problem. We note that those solutions do not address visualization of the voting based consensus mechanisms and underlying network topology of the distributed systems and develop a visualisation tools specifically designed for researchers and developers to help monitoring, analyze such networks and compare real-time observed data with the expected. We conclude that visualisation techniques can be complementary to traditional log based debugging, and testing techniques by applying them to a custom developed blockchain. Moreover, we provide our tools as open source software as plugins for popular visualisation platform Grafana [30]. The plugins can be configured via Grafana plugin configuration interface to fit the specifics of the protocol implementation and we provide an example of how to setup one of the plugins for one of such protocols.

2 THEORETICAL BACKGROUND

2.1 INTRODUCTION OF INFORMATION VISUALIZATION

As the age of Big Data is gaining momentum, information visualization is becoming an increasingly important tool for understanding the millions of rows of data generated every day. Data visualization helps interpret data by transforming it into a more understandable form, highlighting trends and outliers (and other useful information). Visualizations can save a lot of time and become a daily routine for thousands of people. An example of such visualization would be the London Underground map (see Figure 1.

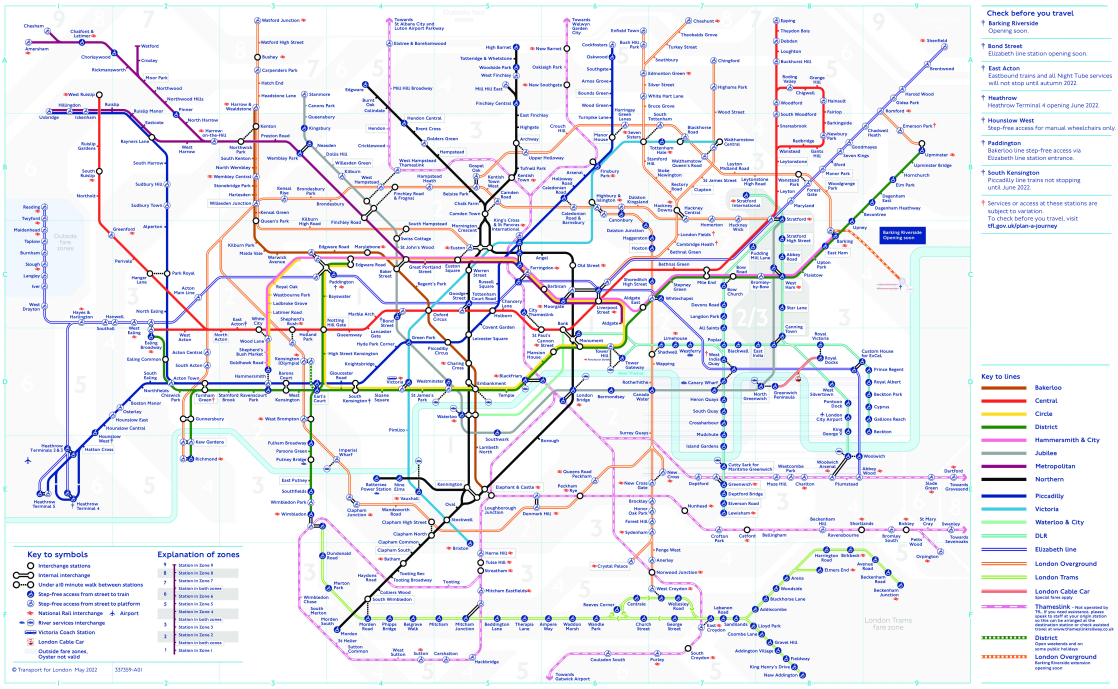


Figure 1: London Underground map, source: tubemaplondon.org.

What is the criteria for good visual representation? One approach would be to calculate the value of information visualization with quantifiable metrics [22]. Authors of the research summarize an attempt by Van Wijk [59] with his *Economic Model* of value and state that the recognition of value of information visualization (InfoVis) is extremely difficult. Another approach is proposed by Edward Tufte in [57], where he states the following principles of graphical integrity:

1. The representation of numbers, as physically measured on the surface of the graphic itself, should be directly proportional to the numerical quantities represented.

2. Clear, detailed, and thorough labeling should be used to defeat graphical distortion and ambiguity. Write out explanations of the data on the graphic itself. Label important events in the data.

3. Show data variation, not design variation.

4. In time-series displays of money, deflated and standardized units of monetary measurement are nearly always better than nominal units.

5. The number of information-carrying (variable) dimensions depicted should not exceed the number of dimensions in the data.

6. Graphics must not quote data out of context.

In [63] Wilkinson explains *The Grammar of Graphics* (or GOG), which is a system of seven graphical component sets, combination of elements of which is meaningful. The following dataflow diagram (see Figure 2), which is based on the similar visualization by Wilkinson, shows the order and examples of mappings required to create a visualization from a dataset. The detailed explanation of each GOG component is available throughout Wilkinson's article. One of the most famous tools, *ggplot2* [61], is an open source implementation of GOG for programming language R [56].

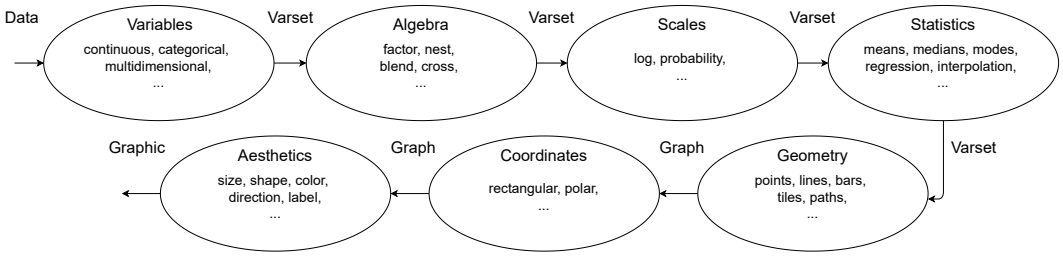


Figure 2: GOG dataflow diagram, based on the visualization by Wilkinson [63].

There are a lot of ways to split different visualization techniques into groups and it is challenging to derive a comprehensive taxonomy. A data-driven approach for choosing the right visualization technique is presented in [33]. Some examples were created by the authors using *Protoviz*, which is no longer under active development and exists in the form of a new tool called D3.js [42], with improved support for animation and interaction. For the purpose of simplicity, we consider the following types of visualizations:

- A graphical representation of quantitative information in a form of a diagram. For example, line charts, histograms, scatter plots and stacked area graphs. Examples of such graphs could be found in *The Data Visualisation Catalogue* [53], which is a library of different information visualisation types. Some of the graphs from there can be seen in Figure 3.

(a) Line graph.

(b) Stacked area graph.

(c) Scatter plot.

(d) Histogram.

Figure 3: Examples of various types of graphs from The Data Visualisation Catalogue.

When choosing a suitable chart, you can be guided by the following flowchart by Andrew Abela at [2] (see Figure 4) or use an interactive online tool, such as *ChartChooser* [37].

- In order to understand some type of information, it should be viewed throughout a certain period of time. There is a famous example of a timeline visualization made in 1769 by Joseph Priestley (see Figure 5). In the short description of the chart (can be found in the upper left part of the chart), Priestley wrote that "this Chart is intended to exhibit a picture of history, or to give a clear view of the rise, progress, extent and duration of every considerable empire or state". In this chart we can see two timelines in the top and bottom boundaries, several great empires, distinguished by colors and other smaller kingdoms, divided by different regions.

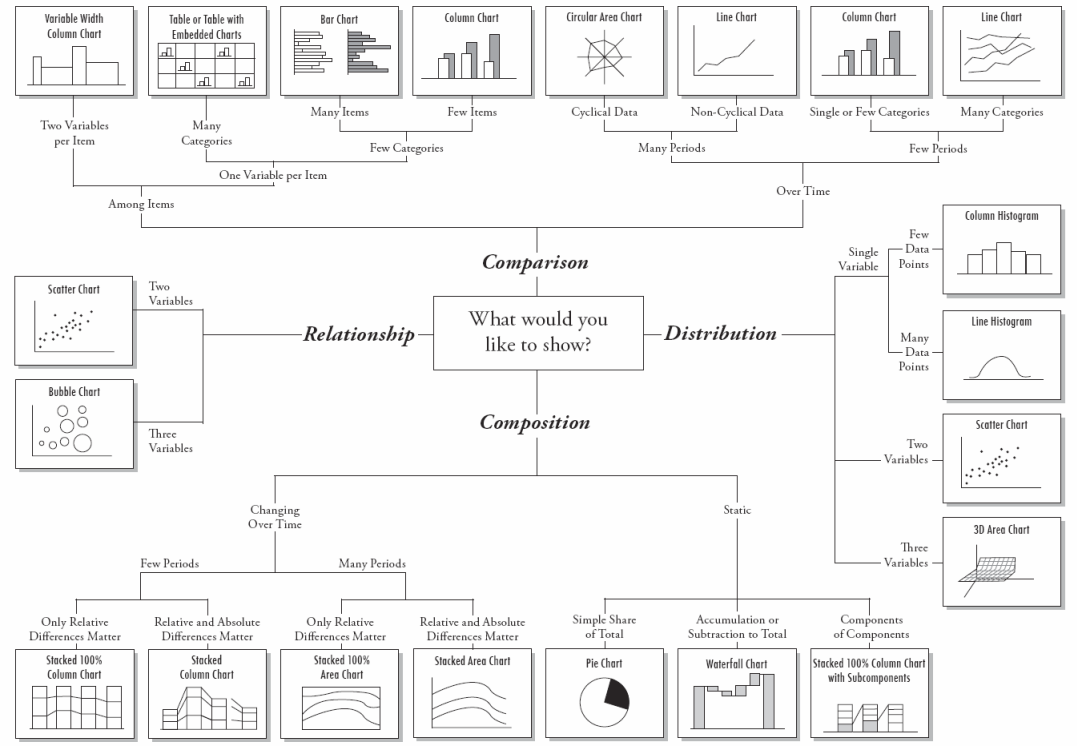


Figure 4: Chart selection diagram.

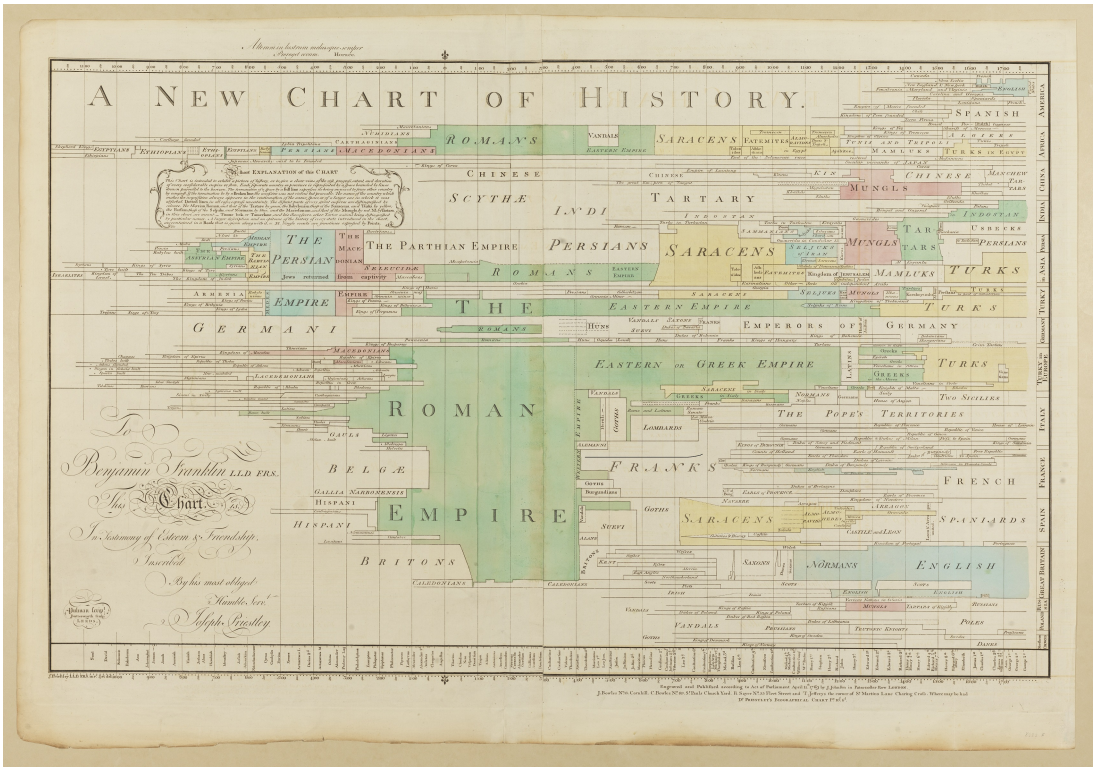


Figure 5: Joseph Priestley's A New Chart of History, source: wikipedia.org.

Another good example would be a map about the largest meteorites impacts throughout history (see Figure 6). This is also an interactive visualization [51],

where users can choose a specific historical interval and view detailed information about each meteorite in particular.

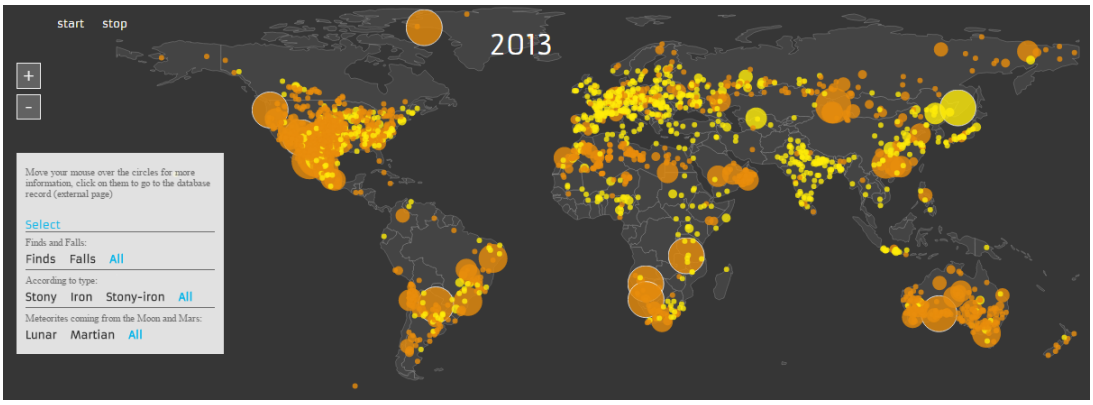


Figure 6: Meteorite impacts map by Roxana Torre.

- Concept visualization helps to develop complex ideas. For example Gantt charts, that are commonly used for project planning, help teams to plan work against deadlines and properly allocate resources (see Figure 7).

Figure 7: Roadmaps, made with Jira software, source: Atlassian.

- For the graph and network visualization at the input we usually just have a set of vertices and edges. We can calculate some statistical characteristics, graph metrics on it, but this will not give us a clear picture of what it is. A good visualization can show if there are clusters or bridges in the graph, if it is homogeneous or it merges into one main center of attraction. Tools, such as D3.js or Graphviz [55] can be used to visualize different types of graphs and networks (see Figure 8 from [43]).

  For large graphs, very often pictures are obtained that cannot be read nor understood, mostly due to the overlapping of the elements of the graph. In addition, graph algorithms are generally very slow, many have algorithmic complexity proportional to the square (sometimes cube) of the number of vertices and/or edges.

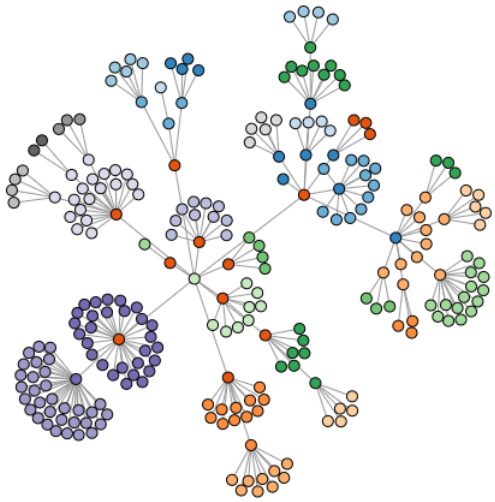


Figure 8: Force layout of the graph made with D3.js by Mike Bostock.

At [29] we can find what are the approaches to drawing large graphs and graphs in general. In this paper authors explain the principles of work of several layout algorithms, test and compare them, resulting in a very detailed comparison table. In the section 4.3.1 we present our tool for visualizing network topology, which is essentially, a graph visualization. For that we use the following principles of how to create good-looking graph and network visualizations:

- Minimum intersection of edges.
- Minimum overlap of vertices and edges.
- Distribute vertices and/or edges evenly.
- Adjacent vertices should be close to each other, non-adjacent ones should be far.
- If applicable, reflect the symmetrical nature of the graph by symmetrical layout.

Details on how those principles are applied can be found later in the plugin description section.

- The goal of infographics is to tell a story and allow users to quickly grasp the main idea of the covered topic. A good example of infographics would be the chart made by Charles Minard's in 1869, describing Napoleon's 1812 Russian campaign army, their movements and temperature they encountered on the return path (see Figure 9).

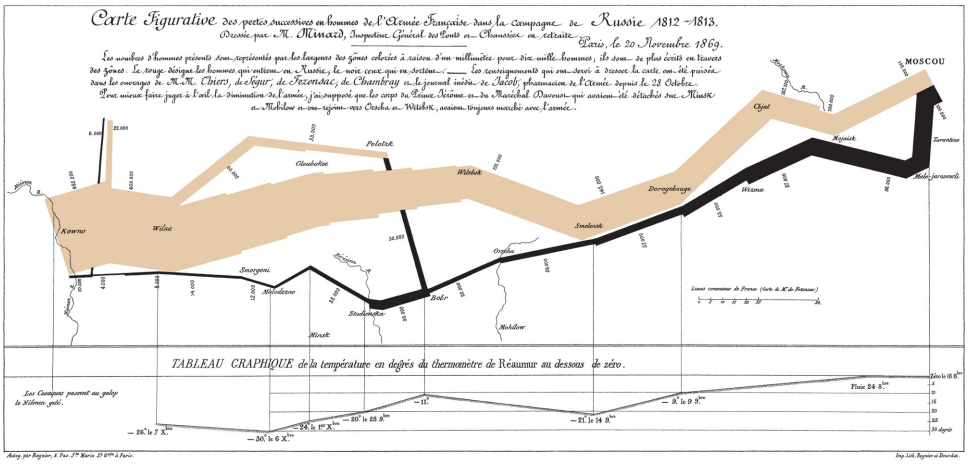


Figure 9: Charles Minard's 1869 chart, source: wikipedia.org.

For some types of information visualization works best when supported by animation. An interesting example of animated infographics would be the illustration of the wingbeats of five different flying species [20].

- Animation is commonly used in interactive visualizations. Interactive visualization is a type of graphical representation of information that allows users to interact with the information display system and observe its response. Interactivity is the key to enhancing user's ability to receive information from visualizations. For better understanding interaction, in [66] authors analyze several studies relevant to interactive visualizations and propose a comprehensive summary of existing taxonomies of interaction techniques, including taxonomies of low-level interaction techniques, taxonomical dimensions of interaction techniques, a taxonomy of interaction operations and taxonomies of user tasks. Interactive visualizations usually refer to the interfaces that support the following principles:

  1. Changes are made easily, quickly and with the ability to return to the previous state.
  2. Many charts and tables can be easily combined and displayed on the same screen at the same time.
  3. A set of visualizations can be linked so that operations on one display are reflected on others.

  An example of interactive visualization would be an interactive map by Gapminder [25], where users can check information about most countries in the world and compare them to each other. Figure 10 shows the bubble chart, plotting *Income* vs *Life Expectancy* of the countries in the year 2021.

  Interactive visualizations are commonly used in the form of dashboards. For example, tools like Grafana can be used to create dashboards that can help to

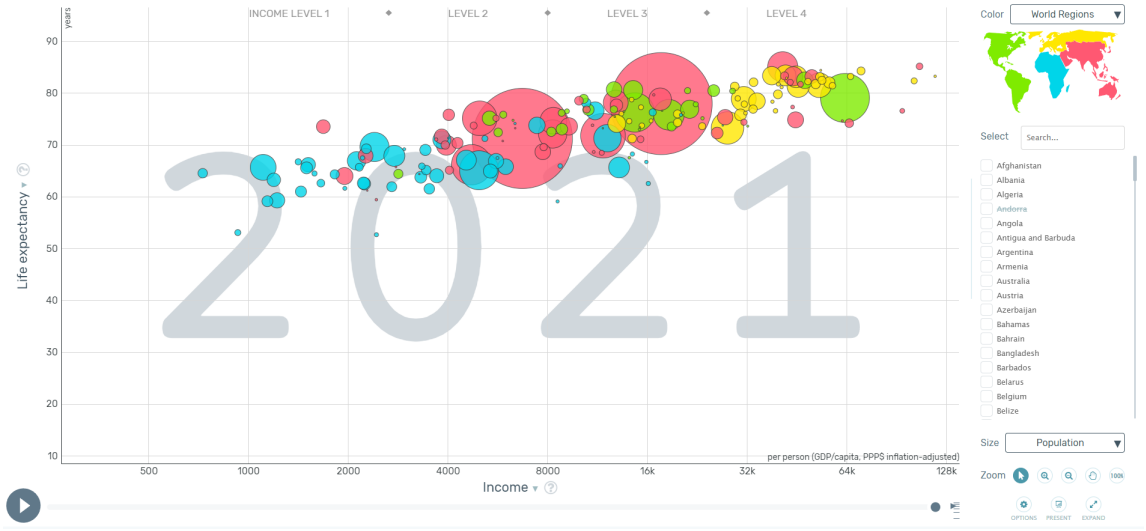


Figure 10: Gapminder's interactive map.

monitor server activity (see Figure 11). In the Chapter 4 we present two custom plugins that extend the functionality of Grafana and show the dashboards made with these plugins.

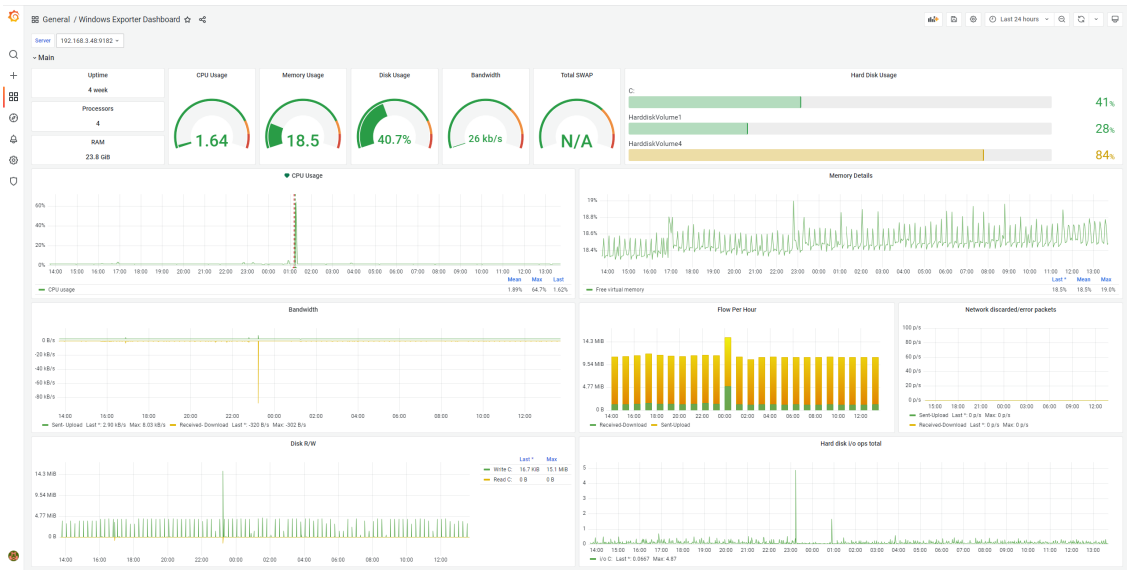


Figure 11: Example of a dashboard created with Grafana.

As mentioned above, we have developed two custom plugins for Grafana for building an interactive visualization for distributed systems, specifically for PoS based consensus mechanisms. In the next section, we give an introduction to distributed systems, their protocols and blockchain, and give an overview of the related issues of such systems with examples of existing visualizations.

2.2 DISTRIBUTED SYSTEMS

Distributed computing systems are physical computer systems, as well as software systems that implement in one way or another parallel data processing on many computing nodes [35]. Nodes coordinate their actions by sending messages with various types of data to each other. Unlike shared-memory systems, where all CPUs can access the same physical address space, nodes in the distributed system are independent. The main properties of the distributed systems are:

- Each node of a distributed system has its own time (lack of a global clock) [19].
- Communication between nodes of a distributed system is not instantaneous, but with a significant delay.
- Communication is unreliable, i.e. messages may be lost.
- Any node can be turned off or fail at any time.

An important role plays CAP theorem [10], that is a statement that the following three properties cannot be achieved simultaneously in distributed systems:

- **Consistency** (C) - all non-failed nodes have the same data.
- **Availability** (A) - requests to all non-failed nodes return a response.
- **Partition tolerance** (P) - even if the connection in the system has become unstable, but the nodes are working, the system as a whole continues to work.

We can create a Venn diagram (see Figure 12), explaining the relation between CAP theorem and protocols that are mentioned in section 3.

Usually, distributed systems are quite large, i.e. thousands of nodes are constantly exchanging messages with various types of information. Also, the ability to add new nodes to the system (*horizontal scalability*) plays an important role for distributed systems. There are other challenges on top of that and all together that makes such system complex and difficult to debug. That is why visualizations are necessary for better understanding of such systems and we consider this in more detail in Chapter 4 with relation to our plugins. We give examples of visualizations for some types of distributed systems, mentioned below in the following sections, as well as their protocols.

2.2.1 P2P networks

In a centralized system, all participants are connected to some central network authority (server). Such authority stores data and gives access to it to other users of

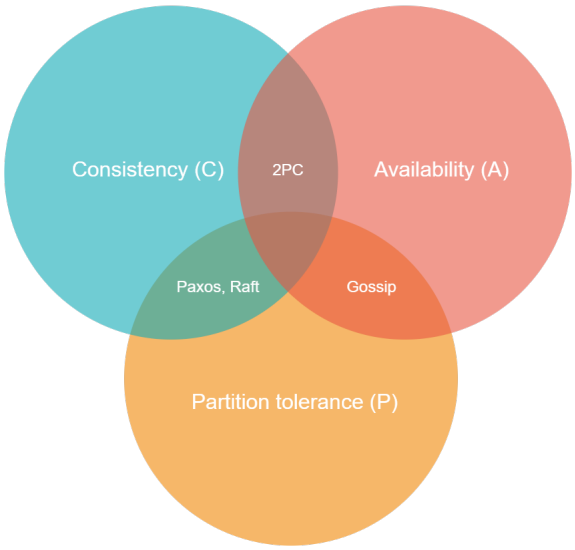


Figure 12: Venn diagram for CAP theorem.

the network. But this system has an important limitation. If the server goes down, the system stops working properly and users cannot access data. Because a centralized system needs a central authority to connect all other users and devices, network availability depends entirely on that authority. Such systems usually can be developed and deployed easier and faster than distributed systems. But the concerns with network stability and data confidentiality make centralized systems no longer the preferred choice for many organizations and distributed networks, such as peer-to-peer networks, are getting more popular.

Peer-to-peer(P2P) network is a type of a distributed system based on the idea of equal rights for all network participants. There are no dedicated servers on the network and unlike traditional centralized networks, each of the participants is both a client and a server. The main advantage of such a network is the almost complete independence of the network from its size. While building and maintaining a server for hundreds of thousands of users is not an easy task, P2P networks do an excellent job of it.

We can see an example of a simple visualization in Figure 13, made with "p2p-graph" package [23].

Figure 13: P2P network visualization made with *p2p-graph* package.

2.2.2 F2F networks

Friend-to-Friend(F2F) is a type of P2P network in which users only communicate with peers they trust. Authentication uses digital signatures and passwords. Unlike P2P, users cannot see the list of F2F network members except their friends. Among the advantages of such networks, it can be noted that F2F are much more protected from hacker attacks. The use of F2F networks allows you to avoid man-in-the-middle (MITM) attacks [15], that is, users can safely exchange secret data (for example, crypto-keys) with their friends.

2.3 BLOCKCHAIN

Blockchain is a distributed database, which is a chain of blocks, each of which has a list of transactions. The basic information that each block contains is:

- hash of the whole block,
- hash of the previous block in the chain,
- serial number of the current block in the chain,
- list of transactions.

The hash function is sha-256. A Merkle tree [6] is used to hash the list of transactions (see Figure 14).

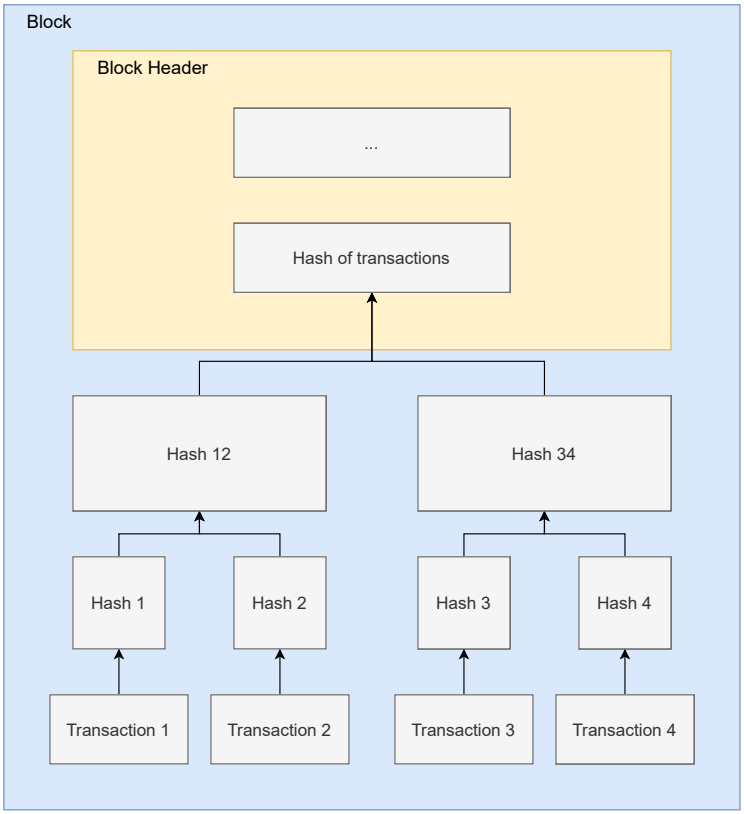


Figure 14: Transaction hash scheme.

Blockchains have a layered architecture, where Layer 0 (L0) includes the main components of blockchain technology. These components are hardware, protocols, connections, and so on. The layer is responsible for interchain compatibility, provides communication between blockchains and helps solve scalability issues. Layer 1 (L1) considers different consensus mechanisms, blockchain protocols (e.g. proof of work or proof of stake) and other issues specific to different blockchains. Layer 2 (L2) is built on top of Layer 1 as an integrated solution, managing transactions for Layer 1. Users mostly interact with applications built on Layer 3 (L3).

In the next section we consider one of the most famous implementations of blockchain technology - Bitcoin [45] and look at the visualization of its transactions.

2.4 BITCOIN

Bitcoin is a decentralized digital currency running on the Internet and is based on Blockchain technology. The idea of creating a distributed cryptocurrency is fraught with difficulties, the main one being the problem of *Double-spending* [50]. Unlike ordinary paper money, which cannot be simply taken and copied, it is easy to make a copy of the electronic file that contains information about the wallet and try to spend money at the same time in different parts of the system. Since the system is distributed, there is no such server from which it would be possible to request information about the

current account. On the one hand, this greatly complicates the principle of operation, but on the other hand, it gives huge advantages associated with distribution.

Since Blockchain is a P2P network, visualizations are similar to Figure 13. For example, *Daily Blockchain* [1] displays Bitcoin transactions using *VivaGraphJS* [4] library (see Figure 15).

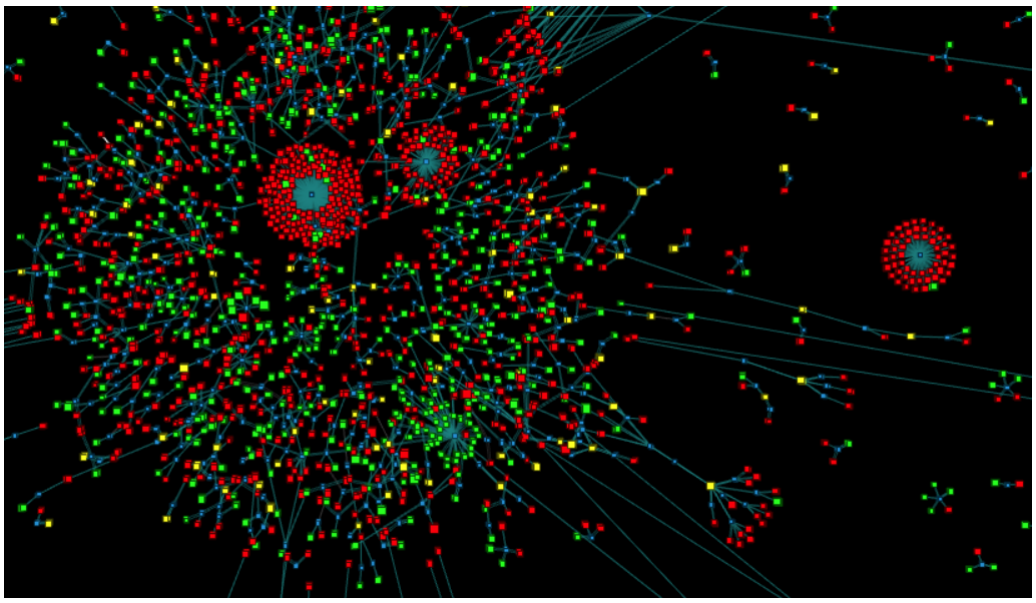


Figure 15: Visualization of Bitcoin transactions by DailyBlockchain.

Bitcoin uses Proof of Work (PoW) consensus mechanism, which requires miners to generate a valid hash to create a new block for the chain. Hash generation requires solving a mathematically complex problem, which means it requires computing resources. In order for people to mine, each newly found block not only records fresh transactions, but also gives the miner some bitcoins as a reward. The better computer performance a miner has, the more chances he gets for generating a correct hash first, thus getting a reward.

When one of the participants wants to transfer money to another, he sends information about this transaction to all network participants. The miner, having received the next transaction, checks that it is correct. That is, that the sender has enough money in the account. Since each miner keeps the entire history of transactions, this is easy to do. If everything is fine, then the transaction is added to the block, otherwise it is rejected.

The complexity of the mathematical problem for creating a block is constantly adjusted and maintained so that a new block is created on average once every 10 minutes.

The downside of the PoW approach is the amount of work required to keep the system running, which is energy-intensive. The Cambridge Bitcoin Electricity Consumption Index (CBECI) [13] estimates electricity demand of the Bitcoin network.

Figure 16 shows the historical Bitcoin network power demand.

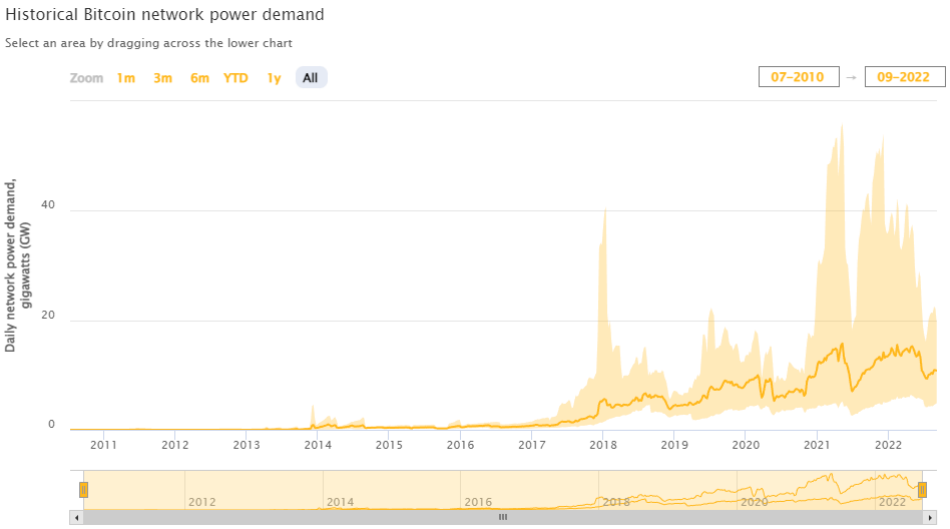


Figure 16: Historical Bitcoin network power demand from CBECI [13].

Proof of Stake consensus protocol addresses this problem and uses a different algorithm to ensure the security of the network. One of the most famous blockchains, Ethereum, is in the process of transitioning from PoW to PoS consensus. Ethereum claims that this transition (called *The Merge*) will reduce its network energy consumption by 99.95% [21]. According to CryptoSlate at [17], at the time of writing there are 260 known networks that use PoS consensus protocol for their systems. Finding a better ways to visualize different aspects of such networks could help developers to monitor them, detect possible bugs and anomalies, thus improving the security of the system.

3 DISTRIBUTED SYSTEMS PROTOCOLS

In P2P networks to maintain a list of active peers, each server sends a *heartbeat* to other servers, which is a message that one server sends to another to tell it if it is alive. Accordingly, if heartbeat does not come for a long time, then this server can be removed from the list of active peers. Constantly exchanging heartbeat with a large number of servers is time consuming. Because of that and other communication challenges, P2P protocols heavily rely on their routing algorithms, thus requiring more complex visualizations. In the next section, we consider one of such protocols - *Gossip protocol*.

3.1 GOSSIP PROTOCOLS

Gossip protocols are a family of protocols that allow you to provide eventual consistency in a distributed system. Nodes spread information about changes to each other as much as possible, and not only self-added changes, but also what they heard from other nodes (rumors, gossip). Then, in the absence of failures, sooner or later all nodes will learn about everything.

A good visualization of the Gossip protocols would be the "Gossip Simulator" [24]. User can choose an initial set of "infected" nodes (the ones who send messages (gossips)) and control two main parameters of the protocol, such as the number of nodes and the fanout, that determines how many messages each "infected" node sends to other nodes in the network. In Figure 17, red nodes are the "infected" ones and the rest are colored at random.

Since all the nodes in distributed systems are independent, each node within the system has perceived relative truth from a top level view. In the next section we talk about *consensus protocols*, through execution of which nodes need to reach an absolute truth from relative truths.

3.2 CONSENSUS PROTOCOL, AND CONSENSUS ROLES

The aim of consensus protocols is to prevent a single entity in the system to change the global state. In vote based consensus protocols, nodes vote for proposed blocks, a block is accepted by the protocol if it gathers sufficient amount of votes. However

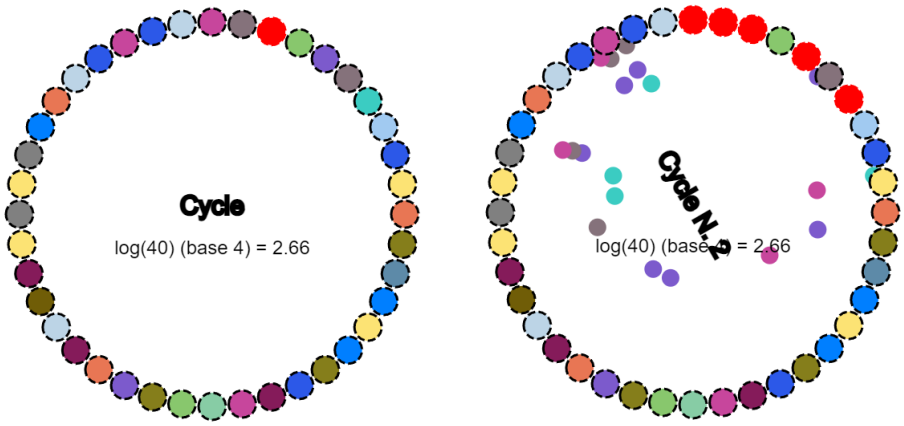


Figure 17: Gossip protocol with 40 nodes and a fanout of 4 in the initial state (left) and in the middle of the 2nd cycle (right).

in public permission-less networks, it is difficult to have all nodes participate in the protocol for each block. The issues are scalability related, where congestion in P2P networks is hardly avoidable since votes need to be broadcast through the network. To circumvent this, state of the art blockchain protocols [12] regularly shuffle nodes, and select a subset of them to vote. As long as the selected subset of nodes is large enough (representative sample), and nodes were chosen at random, the rest of the network can accept the block provided it passed the vote.

3.2.1 Two-phase commit

Two phase commit (2PC) lays the ground floor all voting consensus mechanisms share. In a 2PC scheme, nodes assume two roles, namely coordinator, and participant. The protocol is simple, the coordinator asks the participants to commit to a new state, and gathers votes. The replicas either reject or accept the state transition. In the second phase, once the vote is successfully passed, the coordinator informs participants to commit to the change.

3.2.2 Paxos

Paxos, first introduced in [40] by Leslie Lamport shares the same fundamentals as 2PC. However, unlike 2PC does not require unanimous voting. Instead, a quorum is created, and a majority vote for a proposal is accepted. A node assumes the role of *proposer* suggesting a state transition to *acceptor* nodes, which send their votes. If the majority votes to transition, the *proposer* sends a commit back. Nodes not participating are called *learners*, which accept commit to the change provided the majority voted in favor.

A good example of visualization of the Paxos algorithm would be the following animated demo [32]. It is implemented in Javascript and demonstrates a simple case with two different clients that propose new values to random nodes in the system (Figure 18).

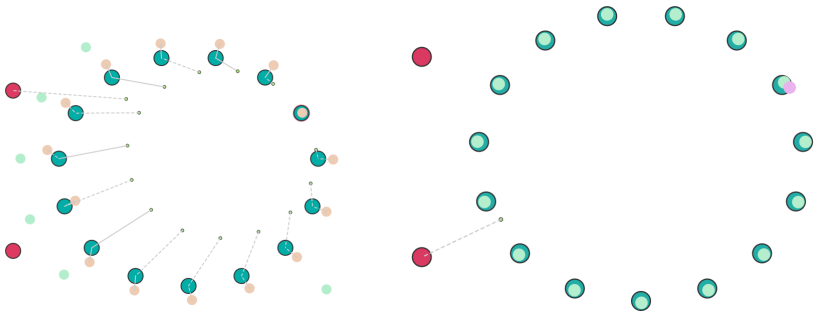
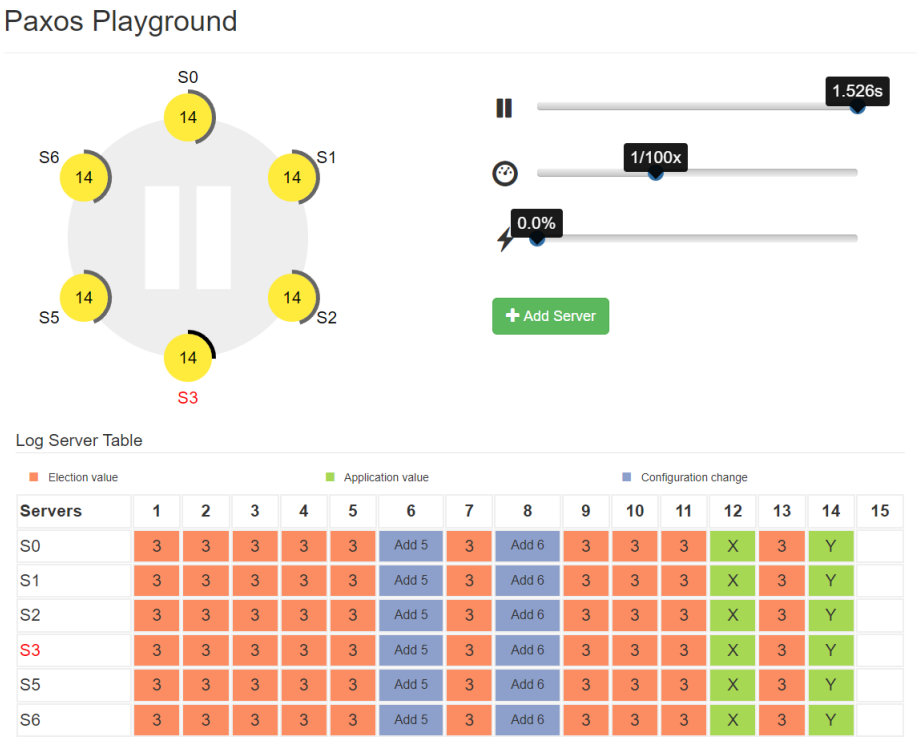


Figure 18: Paxos algorithm with 2 cases for the proposed node either to be accepted (left) or declined (right) by the acceptor nodes.

Another example is the Paxos Playground [36] (Figure 19), available on GitHub that simulates a replicated state machine using Paxos algorithm. It is based on the RaftScope [18] visualization and supports different scenarios (configurations) that users can try on their own.



Servers	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
S0	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	
S1	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	
S2	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	
S3	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	
S5	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	
S6	3	3	3	3	3	Add 5	3	Add 6	3	3	3	X	3	Y	

Figure 19: Paxos Playground with "Master Optimized + Config changes Strategy" configuration.

3.2.3 Raft

Raft was introduced in 2014 [47] as an easier to understand version of Paxos. It is a leader election mechanism in which nodes elect the leader to propose a state change. The leader is responsible for sending a periodic heartbeat to voting nodes. If no heartbeat is received, voting nodes wait a random amount of time and nominate themselves as leaders after. In case multiple nodes attempt to win the election which results in a tie, they repeat the process by waiting a random amount of time.

Educational visualization "Raft: Understandable Distributed Consensus" [7] is an animated guide through all of the main concepts of Raft, such as "Distributed Consensus", "Leader Election", "Log Replication" and so on. One of the many slides, showed in Figure 20 explains how Raft is achieving consistency with network partition that separated nodes A and B from nodes C, D and E.

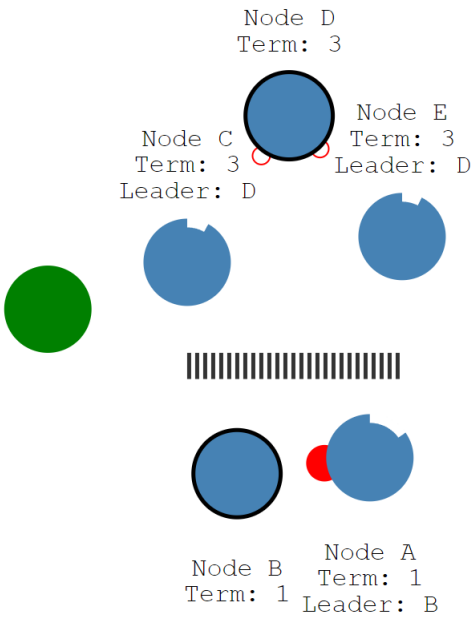


Figure 20: One slide from *Raft: Understandable Distributed Consensus.*

3.2.4 Practical Byzantine Fault Tolerance (PBFT)

PBFT introduces much needed byzantine fault tolerance. Like Raft, it has a leader election protocol, while state transitions are made in three phases. Additionally, messages are equipped with digital signatures to prevent vote falsification, and provable message authenticity. The voting occurs in three phases, namely *pre-prepare*, *prepare*, *commit*. Quorum is achieved if a proposed state change acquires $\frac{2}{3}$ votes.

More recent blockchain based consensus protocols draw parallels with the aforementioned mechanisms. Whichever the specific implementation, the protocols all share

common ideas in which nodes assume roles, form quorums, and vote for to accept a state transition or reject it. In blockchain voting based consensus protocols, nodes vote for proposed blocks, a block is accepted by the protocol if it gathers sufficient amount of votes. However in public permission-less networks, it is difficult to have all nodes participate in the protocol for each block. The issues are scalability related, where congestion in P2P networks is hardly avoidable since votes need to be broadcast through the network. To circumvent this, state of the art blockchain protocols [12] regularly shuffle nodes, and select a subset of them to vote. As long as the selected subset of nodes is large enough (representative sample), and nodes were chosen at random, the rest of the network can accept the block provided it passed the vote.

This research was inspired from implementation issues of a voting based consensus mechanism similar to aforementioned protocols [38]. The main difference is that randomness is provided with verifiable delay functions(VDF) [9]. The proof from a VDF is used as a seed for a random number generator(RNG). Nodes are then able to shuffle the set of nodes participating in the consensus refereed to as the validator set. Each block, nodes are given random roles and execute sub-protocols depending on their role.

- Validators are generic roles for all nodes participating in the consensus protocol. Their role is to validate the block by verifying the signatures, and transactions within the block.

- Committee members are a subset of nodes that are entrusted to vote for the proposed block. Their votes are BLS12-381 [60] signatures that get aggregated into a single signature while maintaining variability.

- The role of the block producer is to prepare a candidate block, broadcast it to the committee members and prepare to receive aggregated votes, verify them and broadcast the final block with the aggregated votes to validators.

For each round, these roles get shuffled in a random fashion. However, the protocol also has fail saves to achieve liveliness through skip blocks in case nodes fail to perform their roles. More details are provided in the next section 3.3.

3.3 FAULT TOLERANCE

One of the most important considerations in the design of distributed, and decentralized systems is the Byzantine behaviour. The consensus mechanisms, and all inherent network protocols must tolerate potential byzantine faults. In completely permissionless systems, such as public blockhains, this is even more important due to the frequency, and lack of control over nodes [11]. Nodes are not only exhibiting accidental byzantine faults but special consideration must be given to malicious actors. Generally,

blockchain consensus protocols are fault tolerant with some protocols tolerating as high as 50% fault tolerance meaning that if 50% of nodes experience byzantine faults, the network would still reach consensus, and be able to finalize blocks. However, moving from theory to practice it is important to simulate, test, analyze, and monitor the system and its performance when experiencing such faults. Here we consider the following faults:

- Network congestion Network congestion can result into nodes not being able to propagate messages in time. Identifying why network congestion occurs offers insight, and aids in implementing more efficient network protocols. Avoiding network congestion is of even greater importance in time synchronous protocols, where nodes are expected to deliver information within a predefined time interval. In our protocol, committee members are expected to deliver attestations within slot time $t_s = \frac{slot\_time*2}{3}$.

- Missing attestations Every slot, a pseudo-randomly chosen node is chosen to produce a block. Committee members must then attest to the proposed block in order for the validators to accept it. In order for the block to be accepted, more than 50% of attestations must be gathered, and aggregated into a single signature. In case of missing attestations, the block will not be accepted, and a skip block protocol will be initiated. The aforementioned network congestion is one among many reasons why attestations might be missing. Attestations must be aggregated using the BLS12-381 scheme, which is done by the members in a binary tree fashion. If a node fails to aggregate signatures of it's children, the entire sub-tree of signatures is lost. Additionally, malicious nodes can attempt to force a chain halt by deliberately preventing attestation aggregation.

- Block producer faults The randomly chosen block producer node can experience faults or a malicious attempt to halt the chain by not producing a candidate block for the committee to attest to. In case the slot time window for proposing the block is missed, nodes will initiate a skip block protocol, where the committee members must produce, and attest to an empty block that simply continues the chain to guarantee liveliness.

- Race conditions Network protocols are inherently concurrent, and must handle operations and access to data structures, and methods concurrently. A blocking message queue is in place to mitigate most race conditions. However, time synchronous protocols require some messages to avoid queue congestion, and hence have to either utilize an efficient message type based priority queue or handle all potential perils of parallel to avoid live-locks, dead-locks, concurrent modification

exceptions, etc. These types of faults are extremely difficult to debug as they are often hard to reproduce.

- Blockchain forks Blockchain protocols can experience forks in the chain itself that need to be resolved by the fork choice rule. Even if the protocol handles such forks, it is desirable that their occurrence is sporadic. There are a number of possible reasons why a blockchain can fork. Forks as a result of malicious behaviour or simply latency in information propagation through the overlay network. It is important to identify when, and why a fork occurs in order to minimize accidental forks thereby easily identifying malicious attempts.

4 GRAFANA PLUGINS FOR VISUALISING VOTE BASED CONSENSUS MECHANISMS AND P2P OVERLAY NETWORKS

4.1 RESEARCH OBJECTIVES

The main goal of this research is to build visualisation tools that offer more insight into a running distributed system using the time series log collection data. The targeted system is a custom proof of stake based blockchain. Such tools should visualize if nodes contributing to the consensus learned about their correct roles, and if they perform their roles accordingly. In the consensus algorithms this is done by sending messages, so the tools should visualise messages exchanged between nodes.

In the structured P2P networks information spreads using gossip protocols and network topology changes every time slot. Our tools should visualize such changes in the network topology by drawing nodes and their cluster representatives, while at the same time indicating the consensus roles for each node.

In our implementation time series data comes from InfluxDB [34], but we want our tools to have no assumption on the data storage implementation and there are other popular databases, such as kdb+ [39] and Prometheus [48], that work well with time series data. Because of that we choose Grafana as a platform for visualizations, which supports all of the aforementioned databases and many more at the time of writing. Grafana is also open-source and supports an easy to use GUI configurable query language. It comes with many prebuild visualisations such as line charts, bar charts and so on. For example, part of the Grafana dashboard from 21 is made with only the default Grafana plugins. However, to this day, despite its wide use, Grafana lacks visualisation plugins such as Network graph visualisation and monitoring a consensus protocol in a way described above is only possible in the aggregated logs format. This is not a huge drawback since the platform supports development of custom visualisation plugins as React components [31]. Additionally, the community developed plugins can be included in their plugin database and made available to the general public. In this work we implement two Grafana plugins built to visualize PoS based blockchains, and

decentralized network topology.

Our tools are designed with generality in mind, and are hence applicable to other PoS based blockchains and other distributed ledger implementations. The two main objectives are to identify potential errors in the consensus protocol, and network topology. Due to the semi-structured nature of the network topology, it is important to visualise whether nodes assume their network positions in a coherent way. The efficacy of message propagation greatly depends on the topology. The second objective is to observe the relative states of each node pertaining to the consensus protocol, and finally the global view. Nodes are self-elected into roles, and must execute the sub protocols depending on their role. We aim at identifying the possible discrepancy between the expected, and observed roles, the actions performed by nodes depending on their roles, and finally the fault tolerance of the protocol in cases where nodes do not perform as expected. We evaluate our tools by applying them to the custom developed blockchain and note their usefulness in debugging and identifying potential issues in decentralized networks. In section 4.4 we explain and provide an example of what is required to setup and configure our plugins in order to apply them to visualize other voting based consensus mechanisms. In the next section 4.2 we consider various existing approaches and solutions related to the debugging of distributed systems, note their limitations and compare some of them with our developed plugins.

4.2 THE ROLE OF VISUALIZATIONS IN DEBUGGING COMPLEX DISTRIBUTED SYSTEMS

Distributed and decentralized systems are difficult to debug as developers are working on the third layer. Which includes L1 (code level bugs), issues with concurrency on L2 (individual run-time), and finally the third dimension for potential bugs arising from the message exchange between nodes. In general, it is often hard to capture the state in a distributed system as debuggers cannot be attached to all nodes' run-times. Additionally, it is often difficult to reproduce errors when they are inherently stochastic. We consider several methods, such as *Logging, Remote debugging, Simulations* and *Visualisations*.

- Logging is the most common debugging method for all three layers. However, in distributed systems it is important to aggregate logs, and analyze them as a time series. Additionally, aggregating distributed logs assumes the system has some method of clock synchronization protocol. Log collection has been proven to be effective in detecting performance issues for systems such as *Hadoop* [64] and *Darkstar* [65]. The aggregation can be done with specific tools for log collection

such as *InfluxDB* [46], *Logstash* [52], etc. Aggregated logs then can be viewed in a form of a dashboard using tools like Grafana (see Figure 21).

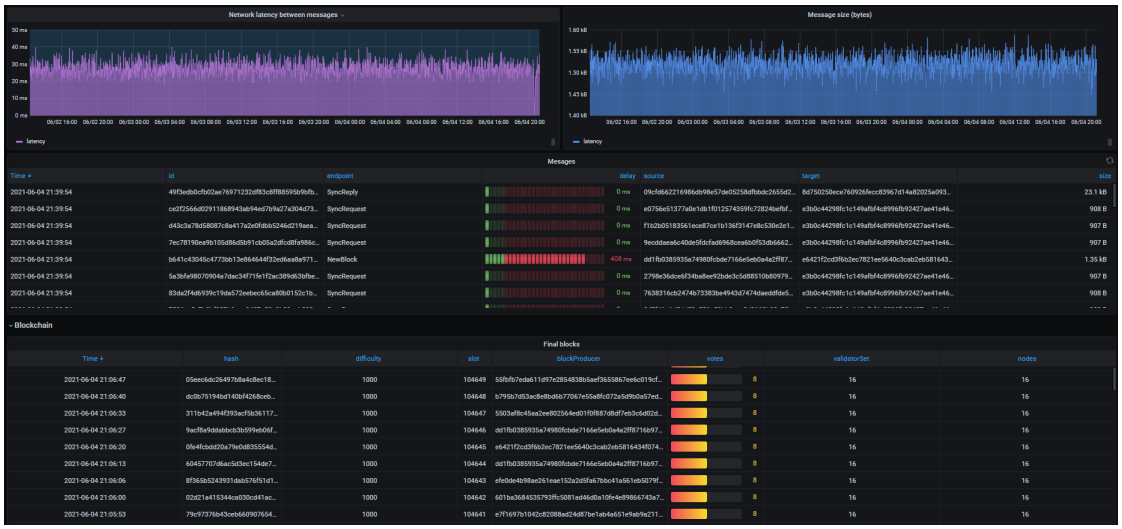


Figure 21: Part of the Grafana dashboard used by developers to gain insight into a running PoS based blockchain network.

- Remote debugging is a technique where a locally running debugger is connected to a remote node in the distributed system. This allows developers to use the same features as if they were debugging locally. However, it is difficult to determine which remote node should be debugged. Additionally, in case of Byzantine behaviour due to network faults connecting the debugger could fail.

- Distributed deterministic simulation and replay is a technique that attempts to address the issues of reproducibility in distributed systems. Tools like *Friday* [27] and *liblog* [28] can be used to record the specific state of the network to use and analyze it later. The technique suggests implementing an additional layer that abstracts the underlying hardware and the network interfaces to allow for an exact replay of all the state changes and messages exchanged between nodes. Tools such as FoundationDB [5] or even custom systems are built on containerisation software.

- Visualisation and time series analysis attempts at capturing the state of the system, and all the nodes by visualising the collected logs. Tools like Prometheus [58] and Grafana [14] are used extensively. Tools like *Theia* [26] and *Artemis* [16] are designed for monitoring and analyzing performance problems in distributed systems and support built-in visualization tools for data exploration. However, such tools provide logs aggregated based summaries of the distributed systems and are not capable of observing underlying low-level network properties, e.g. monitoring network communication, especially in real-time while the system is running.

ShiViz [8] on the other hand displays distributed system executions as an interactive timespace diagram. With this tool all the necessary events and interactions can be viewed in an orderly manner and inspected in detail. ShiViz visualization is based on logical ordering, meaning that unlike our tools, it is not capable of running in real-time, together with the considered distributed network. ShiViz also works with aggregated logs about various types of events of the distributed system and unlike our tools does not support direct database connections. ShiViz is generalized and works with all kind of distributed systems, while our tools are created specifically for monitoring PoS voting based consensus mechanisms and underlying network topology of the distributed system.

4.3 GRAFANA PLATFORM

Grafana is a multi-platform open-source software for interactive visualisations. The project targets time series databases, and provides charts, graphs, and alerts that automatically update, and render once the data-source changes. Grafana is being widely used by organizations such as Wikipedia [62] and CERN [3].

We have developed two plugins that extend the functionality of Grafana. Figure 22 outlines the architecture used in production. A server running a database instance (preferably time series i.e. InfluxDB), and the Grafana platform. Depending on the underlying blockchain implementation, nodes can insert their telemetry directly to the database, or if possible have an archive node gather telemetry from nodes, and report them. In this example, a cluster was used to run multiple nodes. A coordinating node is responsible for maintaining an overlay network and serving the nodes within the overlay with a DHCP, DNS, and routing. Nodes are packed within docker containers and submitted to the coordinator, which uses built in load balancing and distributes them to other cluster nodes.

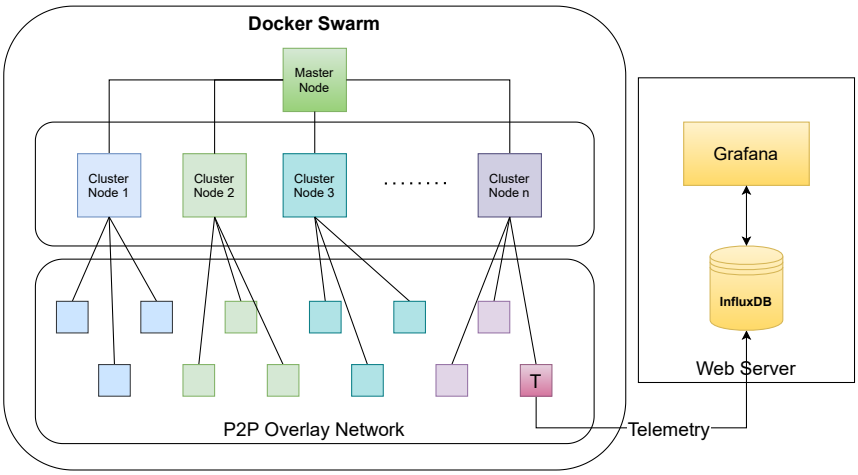


Figure 22: System architecture.

The telemetry inserted is timestamped to create a time series stream of data that is consumed by Grafana. Figure 21 shows a small part of the dashboard created within Grafana using the built-in plugins for typical visualisations. These visualisations are time series data of a running blockchain showing telemetry reported by the nodes. However, rendering telemetry from hundreds of nodes as factors is hardly informative.

Both plugins were developed as React [41] components, using a well-known D3.js [42] JavaScript library for animations. Life-cycle of the plugins is managed by Grafana and Figure 23 shows their architecture.

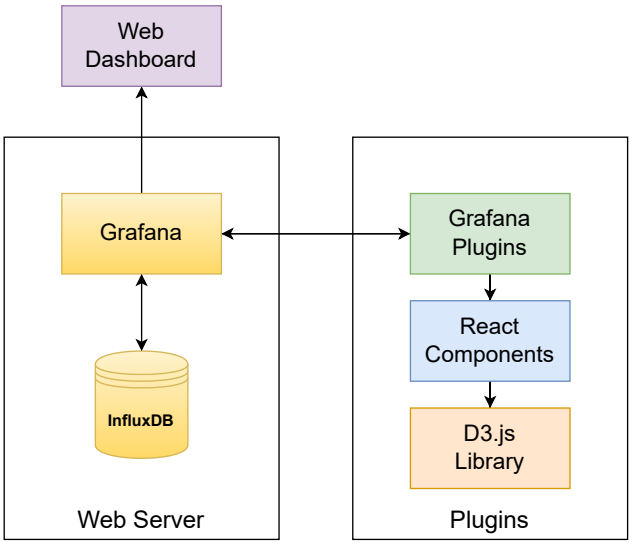


Figure 23: Grafana plugins architecture.

4.3.1 Network Plugin

P2P networks propagate information using gossip protocols. There are many variations of the general and implementation specifics but in general the family of protocols aims at gossiping the fact that new information is available in the network. Should a node hear about the gossip, and require the information it will contact neighbouring nodes asking for the data. In general, gossip protocols make no assumptions about the topology of the overlay network. However, with structured networks, the information exchange can be made much more efficient. The observed blockchain implementation utilized a semi structured network topology for propagating consensus based information. This is made possible by using a seed string shared between nodes that is used for pseudo-random role election every block. Using the seed, nodes self-elect into roles without the need to communicate. However, when performing roles, committee members must attest to the candidate block produced. The seeded random is therefore also used to cluster the network using a k-means algorithm. The clustering is again

performed by each node locally. The shared seed guarantees that nodes will produce the same topology, which is then used to efficiently propagate attestations to the block producer.

The network topology hence changes every slot. The plugin aims to visualize the changes in the network topology by drawing nodes, and their cluster representatives. Additionally, the consensus roles for each node are indicated with the vertex color. Figure 24 shows the network plugin rendering a test network of 30 nodes in real-time. The node in the center colored green is the elected block producer for the current slot, nodes surrounded by the red stroke are cluster representatives, the rest of the nodes are colored based on their role in the current slot.

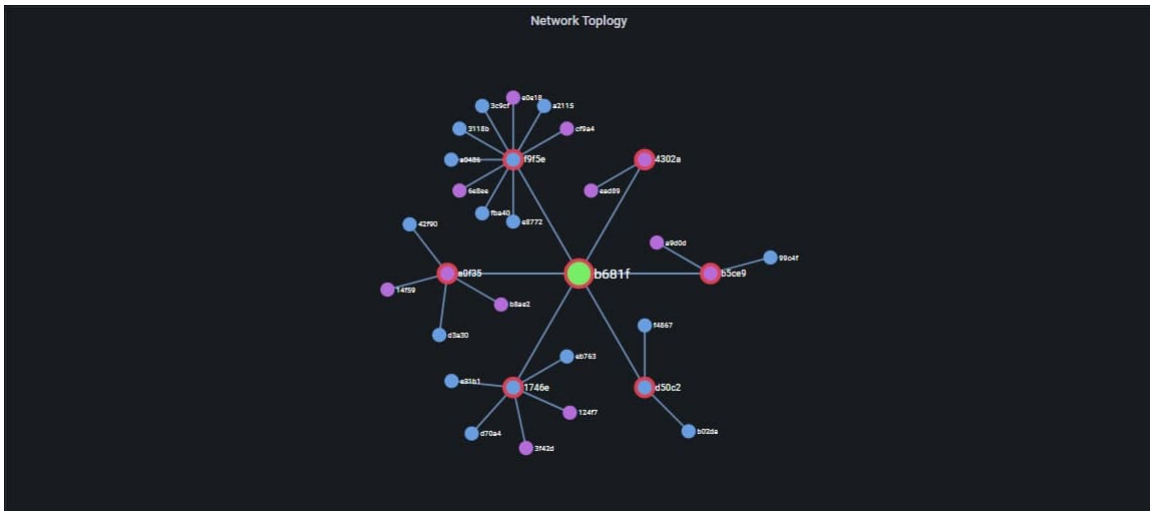


Figure 24: Network topology plugin visualising a test network of 30 nodes in real time.

In order to create a good-looking network visualization, we use the aforementioned principles for constructing graphs from 2.1. By construction, the network topology has a symmetrical layout, as can be seen in the Figure 24. First, we spread the cluster representatives evenly in a circle around the central node, which is a block producer node. Then inside each cluster, nodes should be distributed evenly around their cluster representative, so we place them in the circular pattern as well. We calculate the lengths of the edges in a way that ensures no overlapping between vertices and/or edges. It can happen that non-adjacent nodes can be closer to each other than the adjacent ones, but the symmetrical nature of the visualization makes a clear distinction between such nodes. Names of the nodes can overlap in the large networks, that is why it is recommended to turn off the labeling in the plugin options menu in such cases.

4.3.2 Consensus Plugin

The aim of visualising the consensus mechanism is to quickly evaluate if nodes contributing to the consensus learned about their correct roles, and if they perform their

roles accordingly. In order to have a scalable visualisation, nodes are placed around a circle, and scaled according to the size of the network. Roles are visualized with a color map. Each slot, nodes change their roles, and execute the protocol accordingly. To visualise the execution, the plugin visualises messages exchanged between nodes in a form of animated lines flying from an origin node to the destination node. The animations are time synchronous, and transfer times, and latencies are taken into account. Additionally, every message is logged with a type, indicating the sub protocol within which it was created. As an example, messages being sent from committee members to the block producer are attestations for the current block. The animated lines are colored indicating the message type.

The thickness of the animated lines indicates the size of the payload transferred between nodes. Figure 25 shows the consensus plugin running live visualising a test network of 30 nodes. The green coloured node indicates the block producer role for the current slot, nodes coloured violet are part of the committee, and blue nodes are validators.

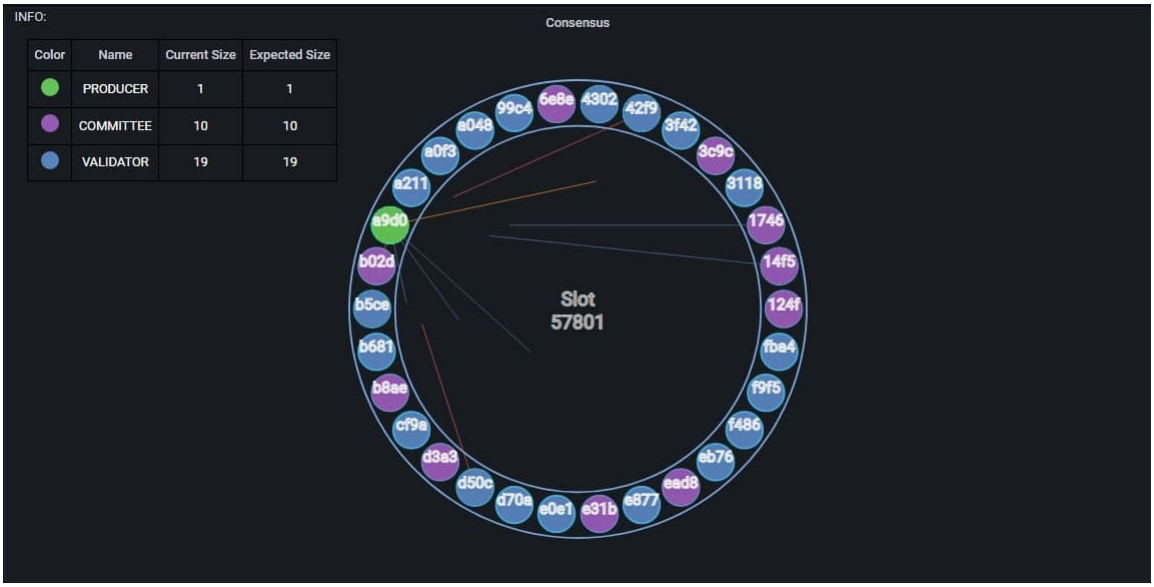


Figure 25: Consensus plugin (with legend) visualising a test network of 30 nodes in real time.

4.4 GENERALITY

In order to use the above plugins, users have to provide certain data to the Grafana dashboard and this can be done through Grafana GUI. For the plugins to work all of the data should follow a specific naming policy. For example, for the Consensus plugin there is one necessary query to visualize data about the nodes of the network. It can be provided using SQL [54] or Grafana GUI (see Figure 26):

```
SELECT "slot", "node", "duty" FROM "<table-name>" WHERE $timeFilter
```

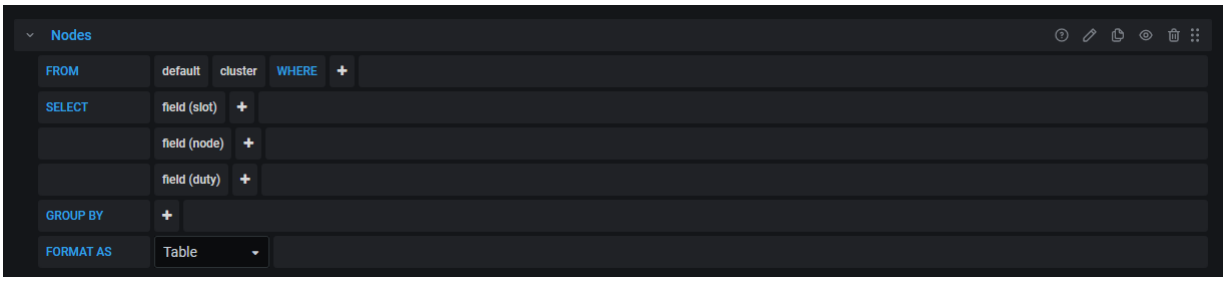


Figure 26: Nodes data query made with Grafana GUI.

Other queries, such as *Messages, Containers* or *Migrations* are optional and can be added in order to visualize additional parameters of the network. Within each query, some parameters are necessary (e.g. *id* for *Messages* or parameters *from* and *to* for *Migrations*) and some are optional (e.g. *delay* for *Messages* or *duration* for Migrations). We can see an example from the source code of how *Nodes* and *Messages* queries are being parsed in A.1 and A.2 respectively (where *data* variable comes from Grafana, see A.7).

Grafana auto updates data points with time query and plugins require a complete state of the network for the particular slot in order to visualize it. In the consensus plugin we are also looking at the behavior of the nodes between different (consecutive) states in order to understand how the roles change. For that, for each Grafana update, for the chosen time period, we have to remember two latest states. That is, if slot t is the latest in the given time period, we would remember states s_{t-1} and s_t and visualize the transition between them, thus leaving the state s_t as the final image in the Grafana dashboard window. In order to keep the track of the currently visualized slot, its number is shown in the center of the image of the Consensus plugin and it is recommended to use both plugins together, positioning then side-by-side in the Grafana dashboard.

Grafana can handle both relative and absolute time periods (see Figure 27) and since our plugins are supposed to show the latest (current) state, the field *From* in the Grafana dashboard refresh options should stay relative, thus enabling the transitions. And, in this case, when Grafana updates and the next slot $t+1$ becomes the latest in the given time period, new states s'_{t+1} and s'_t are chosen to be drawn. Since, s'_t and s_t are the same states, we can keep the current image as the starting point and it is left to visualize the transition from $s_t = s'_t$ to s'_{t+1}.

It is recommended to set your Grafana dashboard refresh time equal to a *slot time* variable to get the best usage of the plugins. Otherwise, using the previous example, states s'_t and s_t would be different, resulting in the flickering change between images of those states.

In the case that all the recommendations above are followed, a complete transition

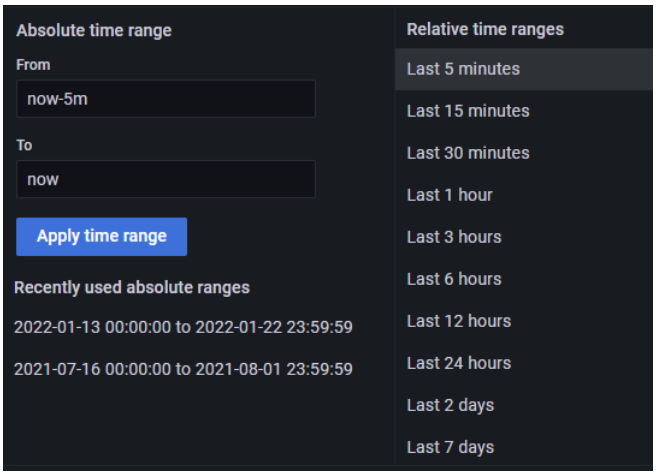


Figure 27: Grafana dashboard refresh options window.

between two consecutive states takes *slot time* seconds. At first, messages exchanged between nodes are animated, if provided to the plugin, using an interesting technique from [49]. The parameter *size* of the message is taken into account, if provided to the plugin, and is reflected by the line thickness in order to make a clear difference between high load messages and the others, since such messages usually contain critical information. If parameter *delay* is provided, it is scaled pseudo logarithmically and converted into the duration of the line animation (see A.6). After all the messages pass, transitioning nodes change their parameters accordingly (see A.3), new nodes are being added (see A.4), while nodes that stop working are being removed (see A.5). All the animated transitions above are implemented using the D3.js library (see [44]).

For the network plugin we are not interested in observing the transition between two different states, that is why just the latest state is drawn at each update. Even though it is not required for the network plugin to have Grafana refresh time to be equal to the *slot time* of the network, it is still recommended to keep it so in order to use both plugins well simultaneously.

Both plugins can be customized from the Grafana options menu. For example, users can add new roles, name and color them. Figure 28 shows the consensus plugin options menu, where users can additionally turn on or off display of messages, nodes or containers labels and so on. For both plugins, users have to manually provide the *slot time* of the network in the plugins options menus.

By using our tools we can visualize other protocols. For example with the consensus plugin we can visualize the aforementioned Paxos algorithm (see section 3.2.2). For that, we should provide the plugin with the *Nodes* and *Messages* queries. For the *Nodes* query, parameters *slot*, *node* and *duty* should be provided, which represent the slot number, node id and the role of the node respectively. From the point of nodes and slots, for this visualization Paxos works in the same way as the example of the PoS

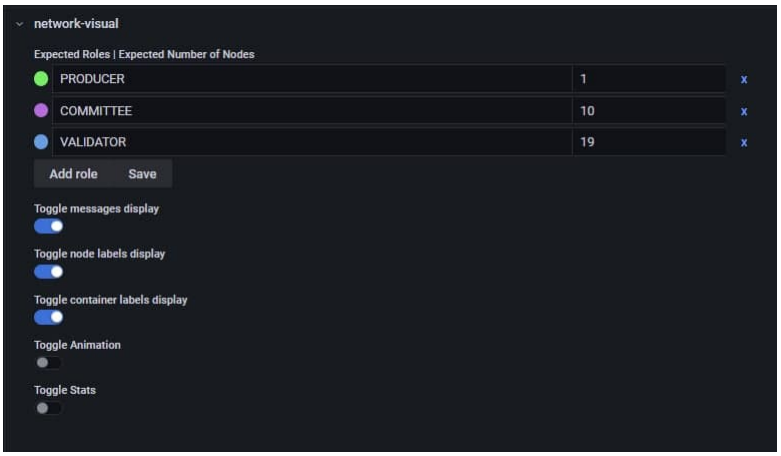


Figure 28: Consensus plugin options menu.

based consensus we mentioned before. For the *duty* parameter, nodes can have one of the three roles: *proposer*, *acceptor* or *learner*. That is why in the options menu of the plugin we should create 3 roles and name them according to the names from the data table.

We should specify *slot time* (in seconds) in the plugin options menu and at this point we can set the Grafana dashboard refresh time and see the results, since all the necessary conditions are fulfilled. But in order to gain more information from the plugin, we should add the *Messages* query. For the data we should have the following parameters: *id*, *source*, *target* and *endpoint*, which represent the message id, node id that sends the messages, node id that receives the message and the type of the message. For the additional information we can specify parameters *delay* (in seconds) and *size* of the message.

If we know the expected amount of nodes for some role, we can put it in the in plugin options menu to see this information in the plugin legend. In a similar way we should be able to visualize other consensus protocols, for example 2PC or Raft.

Source code for both plugins is open source, licensed under the MIT license and available on GitLab, where users can find the installation procedure of the plugins:

- Network plugin - https://gitlab.com/rentalker/topology-visualization-plugin,
- Consensus plugin - https://gitlab.com/rentalker/consensus-visualization-plugin.

5 CONCLUSION

We developed two Grafana plugins for visualising PoS based blockchains, and the underlying overlay network topology. The plugins were used to identify critical bugs, and faults in the protocol. With the help of visualisations, we were able to detect two problems when running test-nets.

- **Network congestion:** for every slot, validators must report their statistics to the block producer. Prompt delivery is desired but not critical. However, as the network grew in size, reporting statistics to a single node (block producer) became increasingly latent as all nodes attempted to propagate messages in tandem, and even more importantly, the network topology required a lot of routing for messages to arrive to the block producer. The network plugin helped us identify what the problem was by looking at the topology.
- **State synchronisation:** at random, nodes failed to perform their roles. This resulted in missing votes even on small test-nets, and sometimes a chain halt where no blocks were produced for the slot. We observed the likelihood of this happening grows in correlation with network size. However, it was infeasible to debug the state of all nodes in a large network. Visualising the state of nodes at a given slot we observed that states were not always synchronized and hence, some nodes did not learn about their consensus role.

We conclude that visualisation is an important tool in design and implementation of decentralized, and distributed systems. The methods serve a complementary role to existing debugging methods, and are very powerful at observing unexpected behaviour of the system as a whole. Visualisation techniques are specifically important in detecting stochastic faults that are non-trivial to reproduce. Our tools are open-source and available for researchers and engineers to use. They are suitable for testing any kind of voting-based consensus protocol with little effort.

For future work we would like to further develop our tools to accommodate other consensus protocols and help developers visualize and debug other types of issues related to distributed systems. Also, we would like to explore other types of visualizations and other existing tools that can help developers as well. Since Grafana is rapidly evolving, our developed plugins can be updated and new technologies can be integrated with our tools to improve their performance.

6 DALJŠI POVZETEK V SLOVENSKEM JEZIKU

Znano je, da je porazdeljene sisteme težko razhroščevati. Težava izhaja iz dejstva, da so prevladujoče težave lahko stohastične in jih je težko reproducirati, ter iz nezmožnosti enostavnega opazovanja, primerjave in testiranja več programov, ki se izvajajo na ločenih računalnikih hkrati. Drugi pomemben vidik pri porazdeljenih sistemih je, da sami po sebi močno uporabljajo omrežje. Uporaba različnih omrežnih protokolov nalaga dodatno kompleksnost, kar povečuje iskalni prostor pri prepoznavanju hroščev. V zadnjih letih porazdeljeni sistemi pridobivajo večjo pozornost tako v akademskem kot zasebnem sektorju. To vse večje zanimanje je mogoče v veliki meri pripisati hitremu razvoju tehnologije veriženja blokov. V zadnjih letih je bilo predlaganih veliko novih mehanizmov soglasja, omrežnih protokolov, izboljšav protokolov za ogovarjanje. Mnogi od njih prehajajo iz teoretičnega okvira v praktično izvedbo. Vendar pa javne porazdeljene knjige (ali tehnologija porazdeljene knjige ali DLT) in verige blokov varujejo svoje mehanizme soglasja in zagotavljajo odpornost proti neželenim napadom z uporabo žetonov, ki predstavljajo vrednost. Uporaba digitalne vrednosti v protokolu omogoča, da protokol uveljavi raven varnosti z ekonomskimi spodbudami in teoretičnimi vidiki teorije iger, zaradi katerih je večina vektorjev za napad ekonomsko neizvedljiva ali nepraktična za napadalca. Dober primer tega je mehanizem soglasja Proof of Stake (PoS), kjer vozlišča v decentraliziranem protokolu zavarujejo mehanizem soglasja tako, da od vozlišč zahtevajo, da zastavijo in zaklenejo precejšnjo količino žetonov, ki jo je mogoče odšteti (običajno imenovano rezanje). po protokolu v primeru, da se vozlišče ni pošteno. Ekonomski vidik javnih verig blokov predstavlja zelo veliko varnostno tveganje. S tako močnimi ekonomskimi spodbudami je prepoznavanje morebitnih hroščev in sistemskih napak je zelo pomembno. Rrazvijalci morajo temeljito testirati in preučiti morebitne težave. Vendar zgoraj omenjene težave pri odpravljanju napak v porazdeljenih in decentraliziranih protokolih zahtevajo, da so razvijalci opremljeni z orodji, ki podpirajo njihova prizadevanja.

V magisterskem delu predstavljamo orodja za vizualizacijo, ki ponujajo boljši vpogled v delujoč porazdeljen sistem z uporabo podatkov. Dva glavna cilja sta bila odkriti morebitne napake v protokolu soglasja PoS in topologiji omrežja. Zaradi polstrukturirane narave omrežne topologije je bilo pomembno vizualizirati, ali vozlišča prevzemajo svoje omrežne položaje na skladen način. Učinkovitost širjenja sporočila je močno

odvisna od topologije. Drugi cilj je bil opazovati relativna stanja vsakega vozlišča, ki se nanaša na protokol soglasja, in končno globalni pogled. Vozlišča so samoizvoljena v vloge in morajo izvajati podprotokole glede na svojo vlogo. Orodje pomaga ugotoviti morebitno neskladje med pričakovanimi in opazovanimi vlogami, dejanji, ki jih izvajajo vozlišča glede na njihove vloge, in končno toleranco protokola na napake v primerih, ko vozlišča ne delujejo po pričakovanjih.

7 REFERENCES

[1] Visualization of bitcoin transactions. `http://dailyblockchain.github.io/`. [Online; accessed 30-August-2022].

[2] Andrew Abela. *Advanced Presentations by Design: Creating Communication that Drives Action, 2nd Edition.* Pfeiffer, paperback edition, 4 2013.

[3] Alberto Aimar, Asier Aguado Corman, Pedro Andrade, Javier Delgado Fernandez, Borja Garrido Bear, Edward Karavakis, Dominik Marek Kulikowski, and Luca Magnoni. Monit: Monitoring the cern data centres and the wlcg infrastructure. In *EPJ Web of Conferences*, volume 214, page 08031. EDP Sciences, 2019.

[4] Andrei Kashcha. Vivagraph. `https://github.com/anvaka/VivaGraphJS`. [Online; accessed 30-August-2022].

[5] Apple. Simulation and testing. `https://apple.github.io/foundationdb/testing.html`. [Online; accessed 30-August-2022].

[6] Georg Becker. Merkle signature schemes, merkle trees and their cryptanalysis. *Ruhr-University Bochum, Tech. Rep*, 12:19, 2008.

[7] Ben Johnson. Raft: Understandable distributed consensus. `http://thesecretlivesofdata.com/raft/`. [Online; accessed 30-August-2022].

[8] Ivan Beschastnikh, Patty Wang, Yuriy Brun, and Michael D. Ernst. Debugging distributed systems. *Commun. ACM*, 59(8):32–37, jul 2016.

[9] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[10] Eric Brewer. Cap twelve years later: How the "rules" have changed. *Computer*, 45(2):23–29, 2012.

[11] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains.* PhD thesis, 2016.

[12] Vitalik Buterin and Virgil Griffith. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437*, 2017.

[13] Cambridge Centre for Alternative Finance. Cambridge bitcoin electricity consumption index. https://ccaf.io/cbeci/index. [Online; accessed 30-August-2022].

[14] Mainak Chakraborty and Ajit Pratap Kundan. Grafana. In *Monitoring Cloud-Native Applications*, pages 187–240. Springer, 2021.

[15] Mauro Conti, Nicola Dragoni, and Viktor Lesyk. A survey of man in the middle attacks. *IEEE Communications Surveys & Tutorials*, 18(3):2027–2051, 2016.

[16] Gabriela F. Creţu-Ciocârlie, Mihai Budiu, and Moises Goldszmidt. Hunting for problems with artemis. In *Proceedings of the First USENIX Conference on Analysis of System Logs*, WASL'08, page 2, USA, 2008. USENIX Association.

[17] CryptoSlate. Proof of stake (pos) cryptocurrencies — cryptoslate. https://cryptoslate.com/cryptos/proof-of-stake/. [Online; accessed 30-August-2022].

[18] Diego Ongaro. Raftscope. https://github.com/ongardie/raftscope. [Online; accessed 30-August-2022].

[19] Danny Dolev, Joe Halpern, and H. Raymond Strong. On the possibility and impossibility of achieving clock synchronization. In *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*, STOC '84, page 504–511, New York, NY, USA, 1984. Association for Computing Machinery.

[20] Eleanor Lutz. Flight videos deconstructed. Flightvideosdeconstructed. [Online; accessed 30-August-2022].

[21] Ethereum Foundation. The merge. https://ethereum.org/en/upgrades/merge/. [Online; accessed 30-August-2022].

[22] Jean-Daniel Fekete, Jarke van Wijk, John T. Stasko, and Chris North. The Value of Information Visualization. In Springer, editor, *Information Visualization: Human-Centered Issues and Perspectives*, Lecture Notes in Computer Science, pages 1–18. Springer, 2008.

[23] Feross Aboukhadijeh. P2p-graph. https://github.com/feross/p2p-graph. [Online; accessed 30-August-2022].

[24] Félix López Luis. Gossip simulator. https://flopezluis.github.io/gossip-simulator/. [Online; accessed 30-August-2022].

[25] Gapminder. Gapminder tools. https://www.gapminder.org/tools/. [Online; accessed 30-August-2022].

[26] Elmer Garduno, Soila P. Kavulya, Jiaqi Tan, Rajeev Gandhi, and Priya Narasimhan. Theia: Visual signatures for problem diagnosis in large hadoop clusters. In *Proceedings of the 26th International Conference on Large Installation System Administration: Strategies, Tools, and Techniques*, lisa'12, page 33–42, USA, 2012. USENIX Association.

[27] Dennis Geels, Gautam Altekar, Petros Maniatis, Timothy Roscoe, and Ion Stoica. Friday: Global comprehension for distributed replay. volume 7, 01 2007.

[28] Dennis Geels, Gautam Altekar, Scott Shenker, and Ion Stoica. Replay debugging for distributed applications. In *2006 USENIX Annual Technical Conference (USENIX ATC 06)*, Boston, MA, May 2006. USENIX Association.

[29] Helen Gibson, Joe Faith, and Paul Vickers. A survey of two-dimensional graph layout techniques for information visualisation. *Information Visualization*, 12(3-4):324–357, 2013.

[30] Grafana Labs. About grafana. `https://grafana.com/oss/grafana/`. [Online; accessed 30-August-2022].

[31] Grafana Labs. Build a plugin. `https://grafana.com/docs/grafana/latest/developers/plugins/`. [Online; accessed 30-August-2022].

[32] Harry Brundage. Neat algorithms - paxos. `http://harry.me/blog/2014/12/27/neat-algorithms-paxos/`. [Online; accessed 30-August-2022].

[33] Jeffrey Heer, Michael Bostock, and Vadim Ogievetsky. A tour through the visualization zoo: A survey of powerful visualization techniques, from the obvious to the obscure. *Queue*, 8(5):20–30, may 2010.

[34] InfluxData. Influxdb. `https://www.influxdata.com/`. [Online; accessed 30-August-2022].

[35] Jeffrey Joyce, Greg Lomow, Konrad Slind, and Brian Unger. Monitoring distributed systems. *ACM Trans. Comput. Syst.*, 5(2):121–150, mar 1987.

[36] Juan Ignacio Vimberg. Paxos playground. `https://github.com/jivimberg/paxos-playground`. [Online; accessed 30-August-2022].

[37] Juice. Chart chooser - juice analytics. `https://www.juiceanalytics.com/chartchooser`. [Online; accessed 30-August-2022].

[38] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[39] Kx Systems. Developing with kdb+ and the q language. `https://code.kx.com/q/`. [Online; accessed 30-August-2022].

[40] Leslie Lamport. The part-time parliament. *ACM Transactions on Computer Systems 16, 2 (May 1998), 133-169. Also appeared as SRC Research Report 49. This paper was first submitted in 1990, setting a personal record for publication delay that has since been broken by [60].*, May 1998. ACM SIGOPS Hall of Fame Award in 2012.

[41] Meta Platforms. React - a javascript library for building user interfaces. `https://reactjs.org/`. [Online; accessed 30-August-2022].

[42] Mike Bostock. D3.js - data-driven documents. `https://d3js.org/`. [Online; accessed 30-August-2022].

[43] Mike Bostock. Mike bostock - d3 - data visualization meetup. `https://vimeo.com/29458354`. [Online; accessed 30-August-2022].

[44] Mike Bostock. Working with transitions. `https://bost.ocks.org/mike/transition/`. [Online; accessed 30-August-2022].

[45] Satoshi Nakamoto. Bitcoin whitepaper. *URL: https://bitcoin.org/bitcoin*, 2008.

[46] Syeda Noor Zehra Naqvi, Sofia Yfantidou, and Esteban Zimányi. Time series databases and influxdb. *Studienarbeit, Université Libre de Bruxelles*, 12, 2017.

[47] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association.

[48] Prometheus Authors. Prometheus - monitoring system & time series database. `https://prometheus.io/`. [Online; accessed 30-August-2022].

[49] Roman Kalyakin. Animating maps with d3 and topojson. `https://source.opennews.org/articles/animating-maps-d3-and-topojson/`. [Online; accessed 30-August-2022].

[50] Meni Rosenfeld. Analysis of hashrate-based double spending, 2014.

[51] Roxana Torre. Macrometeorites: the largest meteorites throughout history. `https://www.torre.nl/macrometeorites/`. [Online; accessed 30-August-2022].

[52] Sushma Sanjappa and Muzameel Ahmed. Analysis of logs by using logstash. In Suresh Chandra Satapathy, Vikrant Bhateja, Siba K. Udgata, and Prasant Kumar Pattnaik, editors, *Proceedings of the 5th International Conference on Frontiers in Intelligent Computing: Theory and Applications*, pages 579–585, Singapore, 2017. Springer Singapore.

[53] Severino Ribecca. The data visualisation catalogue. `https://datavizcatalogue.com/`. [Online; accessed 30-August-2022].

[54] Yasin Silva, Isadora Almeida, and Michell Queiroz. Sql: From traditional databases to big data. pages 413–418, 02 2016.

[55] The Graphviz Authors. What is graphviz? `https://graphviz.org/`. [Online; accessed 30-August-2022].

[56] The R Foundation. The r project for statistical computing. `https://www.r-project.org/`. [Online; accessed 30-August-2022].

[57] E.R. Tufte. *The Visual Display of Quantitative Information.* Graphics Press, 2001.

[58] James Turnbull. *Monitoring with Prometheus.* Turnbull Press, 2018.

[59] J.J. van Wijk. The value of visualization. In *VIS 05. IEEE Visualization, 2005.*, pages 79–86, 2005.

[60] Riad S Wahby and Dan Boneh. Fast and simple constant-time hashing to the bls12-381 elliptic curve. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 154–179, 2019.

[61] Hadley Wickham. A layered grammar of graphics. *Journal of Computational and Graphical Statistics*, 19(1):3–28, 2010.

[62] Wikimedia Foundation. grafana.wikimedia.org. `https://wikitech.wikimedia.org/wiki/Grafana.wikimedia.org`. [Online; accessed 30-August-2022].

[63] Leland Wilkinson. *The Grammar of Graphics*, pages 375–414. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[64] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael Jordan. Online system problem detection by mining patterns of console logs. In *2009 Ninth IEEE International Conference on Data Mining*, pages 588–597, 2009.

[65] Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I. Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles*, SOSP '09, page 117–132, New York, NY, USA, 2009. Association for Computing Machinery.

[66] Ji Soo Yi, Youn ah Kang, John Stasko, and J.A. Jacko. Toward a deeper understanding of the role of interaction in information visualization. *IEEE Transactions on Visualization and Computer Graphics*, 13(6):1224–1231, 2007.

Appendices

APPENDIX A Source code of the plugins

```
let frame_nodes = data.series.find(
  (value) => value.refId === 'Nodes');
  ...
const node_times = frame_nodes.fields.find(
  (field) => field.name === 'Time');
const node_slots = frame_nodes.fields.find(
  (field) => field.name === 'slot');
const node_nodes = frame_nodes.fields.find(
  (field) => field.name === 'node');
const node_roles = frame_nodes.fields.find(
  (field) => field.name === 'duty');
```

Listing A.1: Nodes parser

```
let frame_messages = data.series.find(
  (value) => value.refId === 'Messages');
  ...
const message_times = frame_messages.fields.find(
  (field) => field.name === 'Time');
const message_ids = frame_messages.fields.find(
  (field) => field.name === 'id');
const message_from = frame_messages.fields.find(
  (field) => field.name === 'source');
const message_to = frame_messages.fields.find(
  (field) => field.name === 'target');
const message_durations = frame_messages.fields.find(
  (field) => field.name === 'delay');
const message_types = frame_messages.fields.find(
  (field) => field.name === 'endpoint');
const message_sizes = frame_messages.fields.find(
  (field) => field.name === 'size');
```

Listing A.2: Messages parser

```
d3.select(cId)
  .transition()
  .attr('r', node.graphics.shape.radious)
  .attr('fill', node.graphics.color)
  .attr('stroke', node.graphics.color)
  .delay(nodesDelay)
```

```
    .duration(nodesDuration);
```

Listing A.3: Nodes animation

```
  d3.select(gId)
    .attr('opacity', 0)
    .transition()
    .delay(rolesDelay)
    .duration(removeDuration)
    .attr('opacity', 1);
```

Listing A.4: Nodes fade in animation

```
  d3.select(gId)
    .attr('opacity', 1)
    .transition()
    .delay(rolesDelay)
    .duration(removeDuration)
    .attr('opacity', 0);
```

Listing A.5: Nodes fade out animation

```
  let messageDurationPart = messageDuration / 10;

  if (message.duration < 10) {
    duration = messageDurationPart * 2;
  } else if (message.duration < 100) {
    duration = messageDurationPart * 4;
  } else {
    duration = messageDurationPart * 7;
  }

  line
    .attr('stroke-dasharray', length + ' ' + length)
    .attr('stroke-dashoffset', length)
    .attr('stroke-opacity', 0.8)
    .transition()
    .delay(messagesDelay + delay)
    .duration(duration)
    .attr('stroke-dashoffset', -length);
```

Listing A.6: Messages animation

```
export interface PanelData {
    /** State of the data (loading, done, error, streaming) */
    state: LoadingState;
    /** Contains data frames with field overrides applied */
    series: DataFrame[];
    /**
```

```
7       * This is a key that will change when the DataFrame[] structure
    changes.
8       * The revision is a useful way to know if only data has changed
    or data+structure
9       */
10     structureRev?: number;
11     /** A list of annotation items */
12     annotations?: DataFrame[];
13     /**
14      * @internal
15      * @deprecated alertState is deprecated and will be removed when
    the next generation
16      * Alerting is in place
17      */
18     alertState?: AlertStateInfo;
19     /** Request contains the queries and properties sent to the
    datasource */
20     request?: DataQueryRequest;
21     /** Timing measurements */
22     timings?: DataQueryTimings;
23     /** Any query errors */
24     error?: DataQueryError;
25     /** Contains the range from the request or a shifted time range
26      * if a request uses relative time
27      */
28     timeRange: TimeRange;
29 }
```

Listing A.7: PanelData interface by Grafana