2022

Performance Evaluation of an Edge Computing Implementation of Hyperledger Sawtooth for IoT Data Security

Sean Connolly
Technological University Dublin, Ireland

Recommended Citation

Connolly, S. (2022). Performance evaluation of an edge computing implementation of Hyperledger Sawtooth for IoT data security. [Technological University Dublin].

Performance evaluation of an edge computing implementation of Hyperledger Sawtooth for IoT data security

Sean Connolly

A dissertation submitted in partial fulfilment of the requirements of
Technological University Dublin for the degree of
M.Sc. in Computer Science (Advanced Software Development)

16 June 2022

Signed: [illegible]

Date: **12 June 2022**

ABSTRACT

Blockchain offers a potential solution to some of the security challenges faced by the internet-of-things (IoT) by using its practically immutable ledger to store data transactions. However, past applications of blockchain in IoT encountered limitations in the rate at which transactions were committed to the chain as new blocks. These limitations were often the result of the time-consuming and computationally expensive consensus mechanisms found in public blockchains.

Hyperledger Sawtooth is an open-source private blockchain platform that offers an efficient proof-of-elapsed-time (PoET) consensus mechanism. Sawtooth has performed well in benchmarks against other blockchains. However, a performance evaluation for a practical application of Sawtooth for IoT data security using real data was found to be lacking in the literature.

To address this gap, an experiment was designed to evaluate the performance of an edge computing implementation of Sawtooth to store temperature data from a physical IoT device. Experiments were then performed for a range of input transaction rates to evaluate performance under different workloads.

The results of the experiments indicate that Sawtooth can store transactions at a rate of at least 10 transactions per second in the edge computing implementation that was evaluated. The implementation was highly reliable in terms of transactions submitted versus transactions committed. The experiment also demonstrates that blockchain applications for IoT data security can be extended to any environment that has access to relatively low specification hardware and Wi-Fi internet connectivity.

Some limitations were encountered during the experiments, particularly in relation to the amount of variance in the rate at which transactions were committed to the blockchain. This could have implications for some use cases at the business solution layer that depend on stable and consistent performance.

Key words: *Blockchain, Internet-of-Things, IoT, Edge computing, Hyperledger Sawtooth, Data security*

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to my supervisor Dr Marisa Llorens Salvador for her support and guidance throughout my dissertation journey. Our weekly calls kept me calm and focused and were something I looked forward to.

I would also like to thank all my lecturers and the academic staff at TUDublin who made my learning experience as enjoyable as it has been. Their level of knowledge and experience is both exceptional and inspiring.

I would also like to thank the Hyperledger developer community, whose amazing work made my work possible, and to Dan Middleton for responding to my queries. I hope I can now contribute some of my learning and experience back to this community.

Finally, a special thanks to my partner Becca for her support and patience through those long days and nights when I was engrossed in my computer, and to my friends and family whom I look forward to seeing more of when I finally step away from said computer.

TABLE OF CONTENTS

TABLE OF FIGURES

TABLE OF TABLES

1. INTRODUCTION

This chapter will briefly introduce some of the important concepts and features of blockchain, the internet-of-things, and edge computing. It will then introduce the research problem, research objectives, research methodologies, and scope and limitations, before finally giving an outline of the remaining chapters.

1.1 Background

The Internet-of-Things (IoT) is the name given to the ubiquitous collection of internet connected devices and sensors that enable smart environments by producing, sharing, and consuming data in real-time or close to real-time. This includes many household and personal devices, everything from smart TVs to wearables and fitness trackers that are deeply embedded and integrated into our day-to-day lives.

These devices keep us informed, help us monitor our health and well-being, keep us entertained, and generally add to the convenience and comfort of modern life. On a larger scale, IoT devices monitor and control critical infrastructure like transport systems and electricity grids. Unfortunately, the vast and heterogeneous IoT devices have known issues relating to the security and privacy of data (Alfandi et al., 2021), and there are already too many examples of bad actors exploiting vulnerabilities (Alladi et al., 2020).

One of the proposed solutions to enhance IoT security is blockchain (Alfandi et al., 2021). Blockchain is a decentralised and distributed ledger that uses a consensus mechanism to add new blocks containing transactions. The consensus mechanism is designed such that it is very difficult for a would-be attacker to gain control and alter transactions (Nakamoto, 2008). Transactions may be financial transactions in the traditional sense of a ledger, but equally a transaction can be used to store other types of data and information.

Previous applications of blockchain for IoT have shown some successes but have also encountered limitations due the time taken for the consensus mechanism to add new blocks (Huh et al., 2017). Consensus mechanisms vary depending on the type of blockchain platform, and this is one of the differentiators of the various blockchain platforms that have emerged and evolved. Evaluating the different performance of consensus mechanisms for different applications is an active area of research.

Blockchains are often categorised as being public or private. Public blockchains, also called permissionless, have open memberships and security is maintained by robust consensus mechanisms. Private blockchains, also called permissionless, have restricted membership that allows more efficient consensus mechanisms as there is already a level of implicit trust in the network.

An example of a private blockchain is Hyperledger Sawtooth which was originally developed by IBM and is now supported by the Hyperledger Foundation[1]. Sawtooth features a proof-of-elapsed-time (PoET) consensus mechanism that has performed well when benchmarked against other blockchain platforms (Rasolroveicy & Fokaefs, 2020).

Edge computing has also been proposed alongside blockchain as a solution to IoT security (Hassija et al., 2019). Edge computing complements blockchain and IoT by providing the distributed computational and storage resources to enable their decentralisation.

1.2 Research problem

Blockchain has been proposed by several authors as a solution to some of the challenges faced within the domain of IoT security, by providing a secure and immutable distributed log of transactions (Alfandi et al., 2021; Hassija et al. 2019). This is supported by successful demonstrations of blockchain for IoT applications (Dorri et al., 2017).

However, previous studies on blockchain for IoT found limitations on the transaction throughput performance of proposed solutions, often due to the amount of time taken for the consensus mechanism to add new blocks to the chain (Huh et al., 2017). Transaction throughput limitations become an issue when the level of performance is not sufficient to support the real-time or close to real-time decision-making requirements of some IoT applications (Alfandi et al., 2021).

Hyperledger Sawtooth offers a potential solution to the transaction throughput limitation, based on its performance in benchmarking studies (Rasolroveicy & Fokaefs, 2020). However, a gap exists in the availability of data on the performance of practical applications of Sawtooth for IoT data security. Many studies on Sawtooth performance relied on test cases for benchmarking purposes (Ampel et al., 2019; Shi et al., 2019),

[1] https://www.hyperledger.org/about

while the practical applications focused on operational or other aspects (Baralla et al., 2019; Kromes et al., 2019; Malik et al., 2019). This is the primary research problem, to determine if a practical application of Hyperledger Sawtooth blockchain for IoT data security, can achieve a minimum level of transaction throughput performance.

A secondary research problem is the network design and solution architecture for blockchain applications in IoT security. Network design and architecture didn't get much attention in studies that focused on performance, again due in part to the reliance on test cases for benchmarking (Ampel et al., 2019; Shi et al., 2019). This is something that needs to be addressed, as it has been suggested that an effective architecture for blockchain applications in IoT is not yet available (Pavithran et al., 2020).

Some researchers have proposed the integration of blockchain, IoT, and edge computing, as complementary elements of an overall system to increase data security and deliver other benefits (Yang et al., 2019; Gadekallu et al., 2021). However, like the primary research problem, there is a gap in the research for demonstrations of practical applications that integrate Hyperledger Sawtooth blockchain with edge computing to increase IoT data security.

Bringing together the primary research problem and the secondary research problems leads to the overall research problem as defined below.

Research Question: Can an edge computing implementation of Hyperledger Sawtooth blockchain process and securely store data transactions from an IoT device, at a predefined minimum rate of transaction throughput?

1.3 Research Objectives

The overall objective of this dissertation is to answer the research question that has been formulated and defined. To do this, an edge computing architecture will be designed that integrates a Hyperledger Sawtooth blockchain and an IoT device. The edge computing architecture will consist of a physical edge server that will receive transactions of real data from a physical IoT device.

Experiments will then be carried out to collect data for analysis of performance under different workloads. Data will be analysed at the most granular level possible to understand the mean and variance, as these are key metrics to understanding the performance implications for real-world applications. Data from experiments will also be used to test a hypothesis derived from the research question to understand if the minimum level of performance is likely to be achieved.

The general and specific objectives can be summarised as:

- Review existing literature to identify the open challenges and understand the state-of-the-art in relation to the research question.
- Design the network. Experiments will require an edge computing network consisting of a physical edge server hosting multiple virtual machines to receive data from the physical IoT device.
- Develop custom code required to get sensor data, add the data to a transaction, then stream data from the IoT device to the Hyperledger Sawtooth blockchain via REST API. Develop a transaction processor for IoT data to save sensor data in the blockchain state.
- Install and configure Hyperledger Sawtooth blockchain. Use the built-in test functionality available in Sawtooth to confirm the network is operating as expected.
- Stream data from the IoT device to the Hyperledger Sawtooth blockchain, monitor performance in real-time, and gather data for analysis. Data will be streamed at multiple rates of input transaction to evaluate performance at different workloads.
- Assess and validate the quality of data. Analyse the results to gain insights on the mean and variance for each rate of input.
- Repeat experiments where initial findings lead to new insights requiring further investigation.
- Apply any post-processing and transformations to the data in preparation for the hypothesis test.
- Test the hypothesis, discuss the results and other findings, then draw conclusions on the findings and outcomes.
- Provide directions and recommendations for future work.

1.4 Research Hypothesis

To deliver on the research objectives and answer the research question, the null and alternate hypotheses are defined below. The minimum rate of transaction for the hypotheses has been defined as at least 10 transactions per second. The minimum rate of 10 transactions per second was selected on the basis that the implementation could support up to 10 devices sending an update every second.

H_0: If a Hyperledger Sawtooth blockchain distributed ledger is implemented on an edge computing server, then it will not process and securely store internet-of-things (IoT) data transactions, in the form of temperature readings from a sensor connected to a Raspberry Pi computer and transmitted via REST API, at a rate of at least 10 transactions per second.

H_1: If a Hyperledger Sawtooth blockchain distributed ledger is implemented on an edge computing server, then it will process and securely store internet-of-things (IoT) data transactions, in the form of temperature readings from a sensor connected to a Raspberry Pi computer and transmitted via REST API, at a rate of at least 10 transactions per second.

1.5 Research Methodologies

A deductive research methodology has been applied to formulate the null and alternate hypothesis at the outset, based on gaps identified in the existing literature. Quantitative methods will be applied to test the hypothesis using primary data for transaction throughput, that will be generated by conducting a series of experiments. The hypothesis test will provide an empirical result indicating if performance of the blockchain implementation is or is not within the defined minimum rate of transaction throughput required to reject the null hypothesis.

1.6 Scope, Assumptions, and Limitations

The scope of this study is the application of blockchain, implemented on an edge computing network, to provide security for IoT device data. It is assumed that IoT data is secure when it has been committed as a transaction on the practically immutable blockchain. A limitation of the research is that not every possible IoT device can feasibly be modelled as they are vast in quantity and heterogeneous in nature. This will be delimited by using a Raspberry Pi computer connected to a temperature sensor to represent the IoT device.

1.7 Document Outline

Chapter 2 will review the existing literature, starting with the challenges and threats facing IoT security. From there, it will explore some of the current research around blockchain and how it offers a potential solution for securing IoT data, with a specific focus on the performance and challenges of the different types of blockchain platforms. Finally, this chapter will investigate edge computing as a proposed solution architecture for blockchain applications in IoT.

Chapter 3 will provide an overview of the experiment design and methodology. It will start with the design of the network, then go on to outline the process to install and configure Hyperledger Sawtooth and other software and hardware components. It will also cover the design of the IoT client and the process to create and submit data transactions. The methodology for conducting the experiments and collecting and processing data will also be described.

Chapter 4 will present the results of the experiments and the outcomes and findings will be evaluated. The hypothesis will be tested to answer the research question. Finally, the results of the hypothesis test, as well as any other findings or insights, will be discussed thoroughly.

Chapter 5 will summarise and conclude the results and findings in the context of the research question and hypothesis test. It will address any limitations found in the results and offer direction and recommendations for future research.

2. LITERATURE REVIEW

This chapter will review the existing relevant research and explore the current state-of-the-art in relation to the research question. It will begin with the internet-of-things (IoT), to gain an initial understanding of the security challenges and the solutions that have been proposed. It will then outline some of the research on blockchain applications for IoT security and examine studies on performance aspects of different blockchain platforms. Finally, it will explore the research on how blockchain and edge computing can provide an integrated solution to the challenges of IoT data security.

2.1 Internet-of-things

The internet-of-things (IoT) is the name given to the ubiquitous collection of internet connected devices and sensors that enable smart environments by producing, sharing, and consuming data in real-time or close to real-time. Data exchanged by IoT devices is often sensitive or highly sensitive in nature, everything from the health information of individuals generated by fitness trackers to the real-time state of power grids and other critical infrastructure. Keeping such data secure and private is imperative and the consequences of a breach of data security could be devastating for individuals, businesses, and society in general.

IoT systems have different architectural layers, with the lowest layer, often referred to as the perception or sensing layer, containing the physical devices such as sensors and actuators (Hassija et al., 2019). The highest layer is the application layer that contains the business logic and use cases that encompass various smart environments.

Between the perception layer and the application layer is the network layer that carries data from the devices to a middleware layer containing databases and other data services to support the application layer. The middleware layer is particularly important when considering data security as it is the location for the repositories and services that exchange data (see Figure 1).

Threats to security are present throughout all layers of IoT systems with multiple attack modes described in the literature (Alfandi et al., 2021). Smartmeters, Fitbits, and Nest thermostats are just some of the everyday consumer IoT devices that have had vulnerabilities successfully exploited (Alladi et al., 2020). The challenge of securing IoT devices is compounded by the heterogeneity of devices that are low-powered and have limited computational resources (Alfandi et al., 2021).

Given the scale of the challenge, providing security for IoT systems will require an integrated and multi-layered approach that operates over the multiple layers of heterogeneous devices. Replacing the middleware layer with a blockchain layer and edge computing are among the solutions that have been proposed as a solution to IoT security (Alfandi et al., 2021; Hassija et al., 2019).

Other solutions based on deep learning, big data technology, and intrusion detection systems (IDS) have also been proposed to enhance security in IoT (Elrawy et al., 2018; Amanullah et al., 2020).

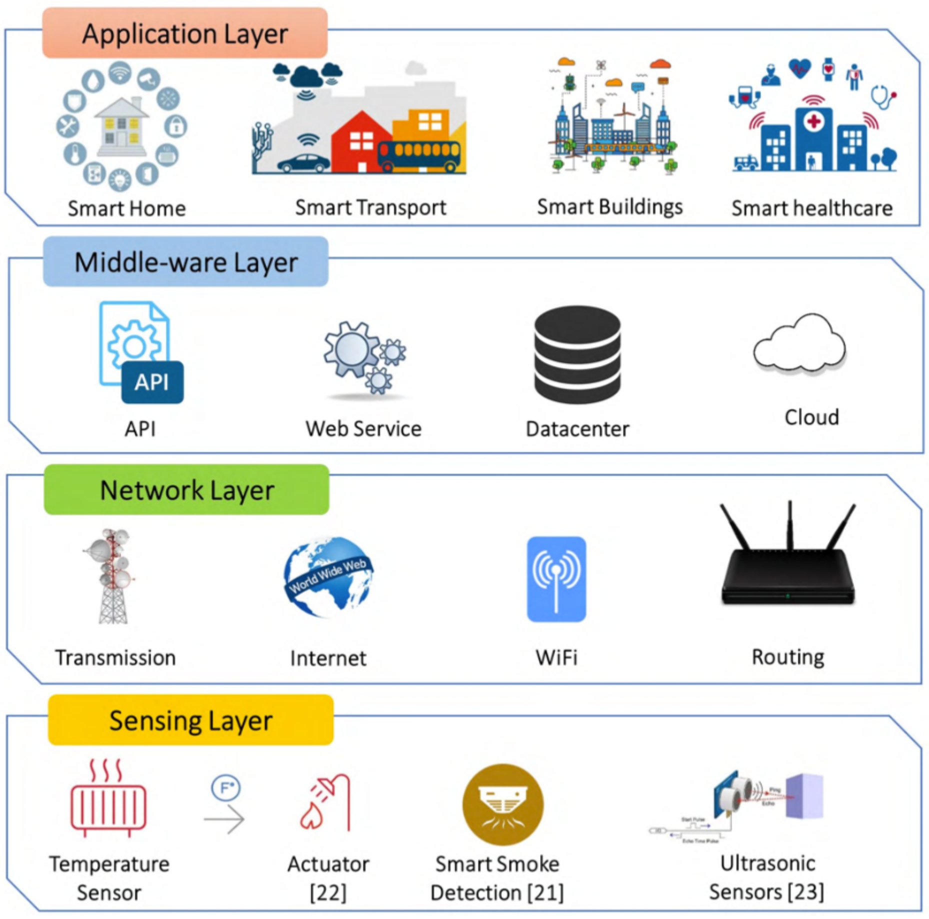


Figure 1: Architectural layers in an IoT system (Hassija et al., 2019)

2.1.1 Raspberry Pi

While discussing IoT devices, it is worth mentioning the humble Raspberry Pi as a device that is frequently used as a component for prototyping, testing, and evaluating proposed IoT solutions. First released in 2012, the Raspberry Pi is a fully functioning computer about the size of a credit card with a Linux based operating system that costs around €30 (see Figure 2). General purpose input/output (GPIO) pins support connection of sensors and other peripheral devices directly to the Raspberry Pi.

Having access to a filesystem for storage and the ability to execute higher level programming languages like python gives the Raspberry Pi some advantages compared to other microcontrollers like the Arduino. Pahl et al. (2016) found that the Raspberry Pi is a feasible, cost-effective, and robust solution for sensor integration and local data processing in an environment subject to power supply problems.

Raspberry Pi computers connected to temperature or humidity sensors have also been used to represent IoT devices in the development, test, and evaluation of blockchain applications for IoT (Kullig et al., 2020).

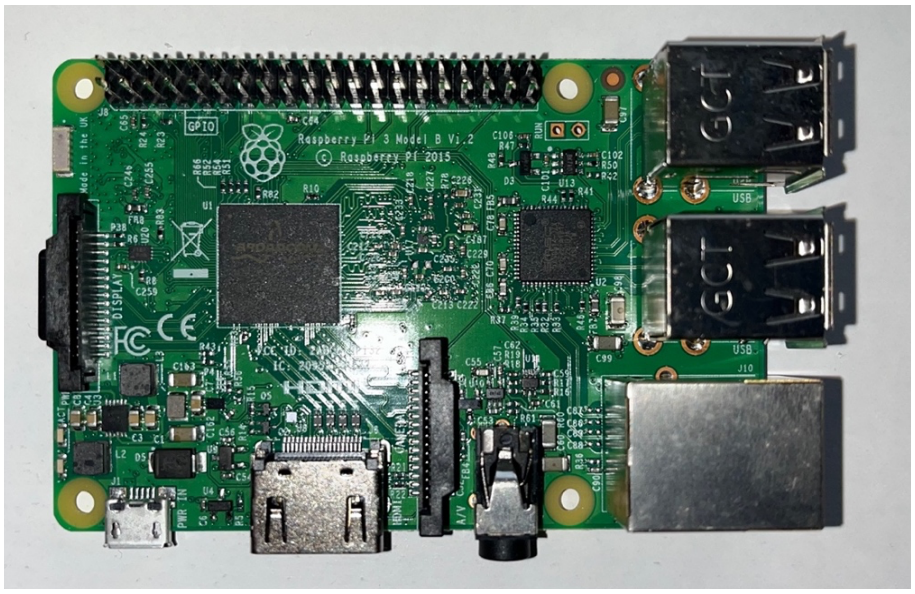


Figure 2: Raspberry Pi computer

2.2 Blockchain

Blockchain is a decentralised and distributed ledger that uses a consensus mechanism to verify and add blocks containing transactions. The consensus mechanism makes it impractical for bad actors to gain control of the blockchain or alter transactions, while privacy is preserved by using encryption with public and private keys (Nakamoto, 2008). Every block that is added contains a reference to the previous block, creating a verifiable chain that extends back to the first or *genesis* block. Each block also contains the root hash of a Merkle-Tree data structure consisting of the hashed transactions of all the child nodes in the tree (see Figure 3).

Some blockchains, such as Ethereum and Hyperledger Fabric, support on-chain execution of programmable scripts that contain transaction processing rules and other business logic (Wang et al., 2018). These scripts are known as smart contracts and have been a key enabler for blockchain applications in domains other than cryptocurrency, such as healthcare, supply chains, agriculture, energy, manufacturing, and IoT (Alladi et al., 2020; Wang et al., 2018).

Many blockchains use a consensus mechanism called proof-of-work (PoW) to add new blocks, where blockchain nodes compete to solve a cryptographically complex problem, for which they are then rewarded with cryptocurrency coins or other incentives. This process of adding new blocks to a blockchain is often referred to as mining. Mining blocks in a PoW blockchain is a time consuming and computationally expensive process that consumes a large amount of energy, with bitcoin alone using as much electricity as some entire countries (Sedlmeir et al., 2020).

However, PoW is just one of many consensus mechanisms that are now available across multiple blockchain platforms that have been developed for various use cases. Evaluating the performance of different blockchain platforms for different applications is an on-going and active area of research. Non-PoW consensus mechanisms have been proposed to mitigate the sustainability issues relating to blockchain energy consumption (Sedlmeir et al., 2020).

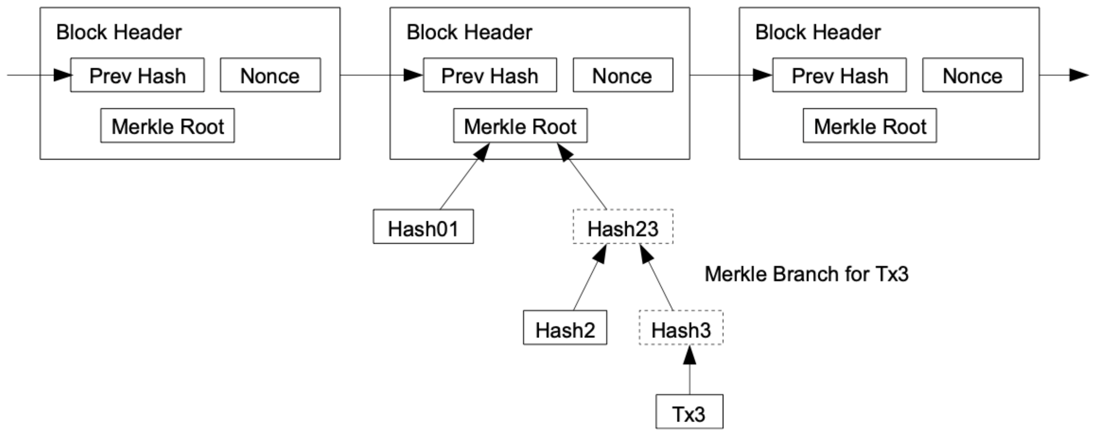


Figure 3: Outline of basic blockchain architecture (Nakamoto, 2008)

Another feature of blockchains is that they can be public, private, or consortium. Public (or permissionless) blockchains are fully decentralised and open to any new members, but often use computationally expensive consensus mechanisms like PoW. Bitcoin is perhaps the best known of the public blockchains.

Blockchains that use PoW consensus mechanisms have been shown not to support the required transaction throughput for IoT applications, with one experiment clocking up delays for transactions to be processed of up to 2.5 hours (Kullig et al., 2020). This is not surprising as PoW was designed for fully decentralised public blockchains where there is no trust between parties, requiring a robust consensus mechanism.

Private blockchains such as Hyperledger Fabric require permission to join, which makes them more centralised (Hassija et al., 2019), but they offer computationally more efficient consensus mechanisms that give better performance (Iftekhar et al., 2021). One of these more efficient consensus mechanisms is called Practical Byzantine Fault Tolerance (PBFT).

PBFT is a vote-based system that can tolerate up to a third of nodes being dishonest or faulty when validating a block. PBFT has been described as the most widely used blockchain consensus mechanism and is available on a blockchain platform called Hyperledger Fabric (Pavithran et al., 2020).

Consortium blockchains have multiple parties but membership is restricted with no single party in control. An example of a consortium blockchain would be a supply chain management system where multiple actors interact to exchange data on a particular product (Baralla et al., 2019).

2.2.1 Blockchain applications for IoT

Blockchain has several features that make it suitable for application to IoT security and privacy. These features include decentralisation, encryption of data, immutability of data on the blockchain, and management of identity (Alfandi et al., 2021; Hassija et al. 2019).

Other benefits include reduced costs by lowering cloud storage overheads, eliminating single points of failure through decentralisation, and building trust in the network (Pavithran et al., 2020). Replication of data through the distributed blockchain also provides a secure back-up of data and redundancy in the event of failure of individual nodes in the network.

Experiments on the application of blockchain for IoT have produced some positive results indicating that it does offer a potential solution. Some examples of these applications are summarised in Table 1. However, it is also clear from Table 1 that there remain some challenges to be overcome before blockchain can be widely adopted as a solution for IoT data security.

Authors	**Blockchain**	**Description, results, and limitations**
Dorri et al., 2017	Not specified	Smart home with a miner node to validate transactions and interact with the IoT devices, and local and cloud data storage Additional resource requirements to implement the blockchain were low and manageable and worth the benefits of increased security and privacy. However, the authors do not specify which blockchain platform was used in the study.
Huh et al., 2017	Ethereum	IoT devices in a smart home communicate with a blockchain containing energy management policies established in smart contracts, but the relatively long time taken for transactions to be approved would limit the feasibility of applications.
Kullig et al., 2020	Ethereum	Raspberry Pi computers connected to temperature and humidity sensors send data over the internet to a simulated Ethereum blockchain. The test results of the experiment indicate very long smart contract processing times that increase linearly with the number of blocks on the chain due to Ethereum's PoW consensus mechanism.
Iftekhar et al., 2021	Hyperledger Fabric	Access control system for IoT devices that uses Hyperledger Fabric, an open source private blockchain. Access is granted to trusted members, verified by a Certification Authority, based on rules that are established using Fabric’s smart contracts. Specially compiled software installations were required to operate the IoT device as a blockchain node potentially limiting practical application.
Baralla et al., 2019	Hyperledger Sawtooth	Farm-to-fork food traceability system for the agri-food sector. Data from both human and non-human (sensor) agents is exchanged over the Sawtooth REST API and accessed via web and mobile clients. The relatively high level of complexity in the system will require extensive testing involving many parties.
Kromes et al., 2019	Hyperledger Sawtooth	Blockchain used to send data to police and other relevant parties following a car accident. The analysis indicated that a significant amount of the execution time was consumed by the data hashing functions. The solution relied on non-standard software installation and modified hardware, potentially limiting scalability.

Table 1: Examples of blockchain applications for IoT

Among the challenges for blockchains in IoT is to find a blockchain platform with a consensus mechanism that can process transactions at a sufficient rate to support the real-time or close to real-time dependency on data to make decisions and perform actions (Alfandi et al., 2021).

Another challenge is to find a suitable architecture. Pavithran et al. (2020) conclude that a suitable architecture for blockchain applications in IoT is still not available and building blockchain networks that rely on cloud computing is a contradiction of the original decentralised objective of blockchain.

This establishes two important requirements for blockchain applications in IoT security, the first is to achieve a level of performance in terms of transaction throughput to support the data requirements at the application layer, the second is that the architecture of any proposed solution should be complementary to the distributed nature of both blockchain and IoT itself.

2.2.2 Performance metrics for blockchain

Determining if a blockchain platform is suitable for a particular IoT application, or comparing the performance of different blockchain platforms and consensus mechanisms, requires a common set of metrics that can be used to evaluate performance. Two widely used performance metrics for blockchain are latency and throughput.

Latency is the time taken for a transaction to be usable across a network, and throughput is the rate at which transactions are committed to a blockchain in a defined time-period (Monrat et al., 2020).

Throughput and latency are influenced by the number of input transactions and the number of nodes in the network (Monrat et al., 2020; Shi et al., 2019), and performance under different rates of input transactions can be non-linear (Ampel et al., 2019). For this reason, metrics for performance of a proposed blockchain application for IoT need to be evaluated over a range of input transactions to understand how performance varies under different workloads.

Hyperledger Caliper has been widely used in many studies to measure performance in terms of throughput and latency (Ampel et al., 2019; Monrat et al., 2020). However, Caliper is a benchmarking tool that measures performance using predefined use cases with generic transactions so is not best suited for monitoring performance in practical applications of blockchain for IoT.

InfluxDB, a time series database, has also been used to collect time-series data for monitoring and measuring the performance of blockchains (Shi et al., 2019). The InfluxDB approach is preferable as it enables the gathering of data for practical applications of blockchain and not just predefined use cases.

2.3 Hyperledger blockchains

Hyperledger Foundation is non-profit organisation, supported by the Linux Foundation, that aims to develop and promote open-source enterprise blockchain technology[2]. Hyperledger Fabric and Hyperledger Sawtooth are two blockchains from the Hyperledger family that have been widely proposed for applications in IoT. Some of the proposed applications were briefly outlined in Table 1.

2.3.1 Hyperledger Fabric

Hyperledger Fabric is a private blockchain with access granted to trusted members that are verified by a Certification Authority. Permissions within the blockchain are governed by policies to control who has access to which network resource, and the policies themselves are approved by consensus within the network. More specific access policies can be applied using an implementation of a smart contract called chain-code, which again requires consensus from the network (Iftekhar et al., 2021).

Fabric has been shown to outperform other permissioned blockchains for throughput and latency due to its simpler and efficient modular consensus approach (Monrat et al., 2020). A proposed access control application of Hyperledger Fabric achieved a transaction throughput of 200 transactions per second (Iftekhar et al., 2021).

2.3.2 Hyperledger Sawtooth

Hyperledger Sawtooth is another open-source permissioned blockchain platform from the Hyperledger family. Sawtooth offers both PBFT and Proof-of-Elapsed-Time (PoET) consensus mechanisms, alongside a devmode consensus mechanism used for development and testing. PoET is a lottery style consensus that uses a random timer to determine which node is selected as leader to propose a new block. The consensus

[2] https://www.hyperledger.org/about

mechanism on a Sawtooth blockchain is established with the genesis block but can be changed later by submitting a settings transaction.

The architecture of a Sawtooth node, as shown in Figure 4, features among its components, a validator to approve transactions, one or more transaction processors, a state store, and a REST API to interface to clients. Permissions and other settings are stored on-chain and can only be changed by consensus.

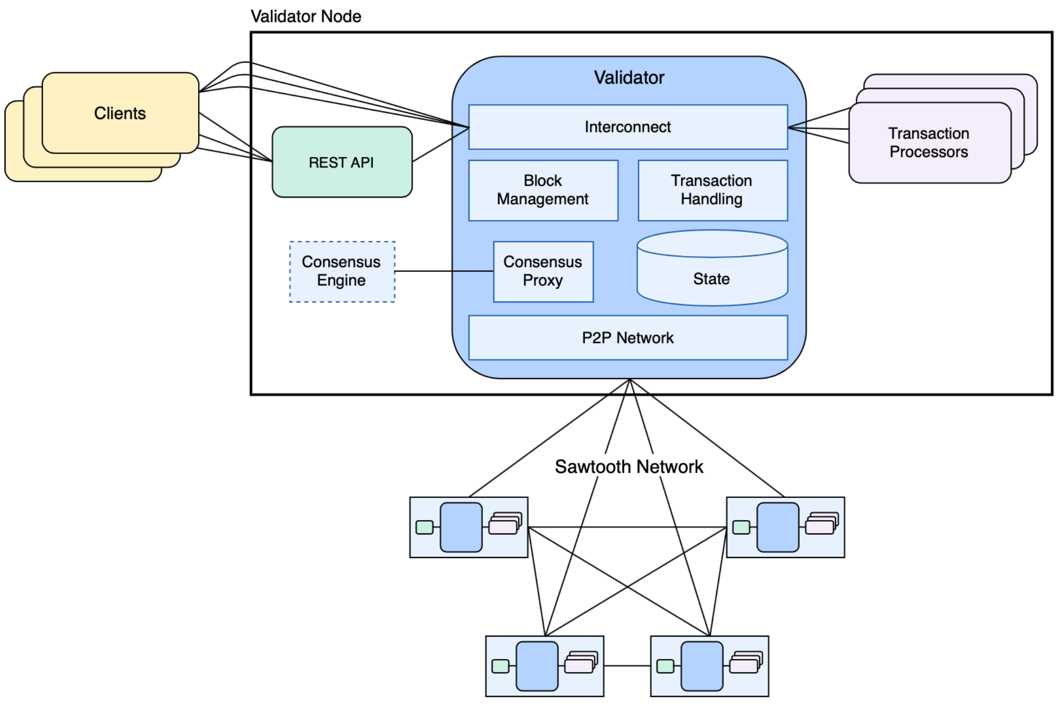


Figure 4: Hyperledger Sawtooth architecture[3]

Sawtooth processes transactions as transaction families, each having their own transaction processor to apply business logic and save data to the blockchain state. Transaction processors are effectively smart contracts that run as plugins rather than on the blockchain itself. A Sawtooth blockchain can support multiple transaction families and multiple transaction processors can run simultaneously on the same blockchain.

Transaction processors can be written in Python, JavaScript, and Go, and there are several transaction processors for development, testing, and evaluation of networks available for installation with Sawtooth. This modularity provided by separating the application level from the core system is one of the distinct features offered by Sawtooth that has driven its adoption (Baralla et al., 2019). In the context of IoT, this could mean

[3] https://sawtooth.hyperledger.org/docs/1.2/architecture/

having transaction families for different types of sensors that could each have their own validation rules based on the type of data.

2.3.3 Performance of Hyperledger blockchains

A summary of the findings from several performance evaluations of both Hyperledger Fabric and Hyperledger Sawtooth is presented in Table 2. Hyperledger Fabric has shown strong results for throughput, outperforming other permissioned blockchains (Monrat et al., 2020). However, Sawtooth has in turn outperformed Fabric, due to its parallel processing of transactions, less transaction processing complexity, and its Proof-of-Elapsed-Time (PoET) consensus mechanism (Rasolroveicy & Fokaefs, 2020).

Authors	**Blockchain(s)**	**Findings**
Monrat et al. (2020)	Ethereum, Hyperledger Fabric, Quorum, Corda	Hyperledger Fabric performed better than the other permissioned blockchains undergoing evaluation due to its simpler and efficient modular consensus approach.
Rasolroveicy & Fokaefs (2020)	Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Burrow, BigChainDB	Hyperledger Sawtooth was the best performing blockchain in terms of writing latency and resource utilisation due to its Proof-of-Elapsed-Time (PoET) consensus mechanism, parallel processing of transactions, and less processing complexity.
Shi et al. (2019)	Hyperledger Sawtooth	Improving VM specs of a blockchain node has a significant positive effect on performance. Performance varies significantly when running a huge workload in a short time period. Parallel scheduling of transaction batches increased performance by around 30%.
Ampel et al. (2019)	Hyperledger Sawtooth	Performance bottlenecks exist at high input transaction rates (>1000 transactions per second) and at large transaction batch size. Latency and system resource usage increases exponentially with increasing input transaction rates.
Moschou et al. (2020)	Hyperledger Sawtooth	PoET consensus mechanism, parallel processing of transactions, and using the GoLang transaction processor, all had a positive (lowering) impact on the execution time for a transaction to be confirmed on the blockchain.

Table 2: Studies on the performance of blockchain platforms

The studies listed in Table 2 reported different rates of throughput for Sawtooth. Ampel et al. (2019) showed very high transaction throughput of up to 2,300 transactions per second (tps). However, another performance study of Sawtooth deployed on cloud based VMs put throughput in the range of up to around 15 tps (Shi et al., 2019). Shi et al. (2019) found that increasing the specification of virtual machines positively affects throughput performance. This may be why some studies using high specification machines hosted in the cloud obtained very high rates of throughput (Ampel et al., 2019; Monrat et al., 2020).

A limitation to the generalisability of some of the studies presented in Table 2 was the use of test transactions to evaluate performance (Ampel et al., 2019; Shi et al., 2019). Results obtained in benchmarking studies that rely on test transactions are useful for comparing performance of different blockchain platforms. However, the throughput rates achieved in such studies should not be generalised as being indicative of performance in real-world applications.

A performance evaluation of a practical application of Sawtooth for IoT security, using actual data transactions from a real device, will offer a better insight to the actual throughput that can be achieved. Lower specification machines should also be utilised as they are more representative of the resources that are likely to be available in the resource constrained IoT environments at the edge of the network.

2.4 Edge Computing

Cloud computing has transformed computer networks by providing practically unlimited scalability of storage and computational resources provided by remote data centres and available as-a-service. However, the physical separation and distance between the cloud, and the site where data is generated and consumed, presents challenges relating to higher latency and the loss of direct control over data security (Sittón-Candanedo et al., 2019). Security concerns also arise when personally and commercially sensitive data is being transferred and stored in the cloud.

To mitigate against the risks and overcome the challenges in cloud computing, edge computing emerged as a distributed computing paradigm that utilises computational and storage resources that are available closer to the physical devices that generate and consume data. Edge computing architectures consist of IoT devices and edge nodes or edge servers that cooperate to support low-latency, real-time processing and analysis of data, resulting in lower operating and management costs (Sittón-Candanedo et al., 2019).

Other benefits of edge computing include bandwidth savings from sending only summarised or aggregated data to the cloud, reduced risk of security breaches with less data-in-transit, and compliance with laws that restrict cross-border movement of data (Hassija et al., 2019).

However, edge computing nodes are also vulnerable to attack and hence the need for secure data storage at the edge, which is where blockchain offers a solution by providing an encrypted, decentralised, and immutable ledger of transactions, verified through consensus (Yang et al., 2019).

2.4.1 Edge computing for blockchain applications in IoT

Edge computing and blockchain have been described as complementary technologies, with blockchain enhancing the privacy and security of edge computing, and edge servers enabling the participation of low-powered devices in the blockchain (Yang et al., 2019). Another complementary feature of blockchain and edge computing is their decentralisation that enables scalability of IoT systems by avoiding centralised performance bottlenecks (Misra et al., 2020).

Blockchain is effectively the distributed platform for securing and providing trust in the network of interconnected IoT devices in an edge computing network. Blockchain also provides benefits to the application layer through the non-repudiable and secure exchange of data between entities using smart contracts (Sittón-Candanedo et al., 2019).

This creates opportunities for autonomous operation of smart systems through the decentralised exchange of data, with applications in smart transportation, smart grids, smart cities, smart homes, and smart healthcare (Gadekallu et al., 2021).

2.4.2 Edge computing architectures for blockchain applications in IoT

A prototype edge computing architecture with integrated blockchain for an agricultural use case is shown as Figure 5. In this model, the IoT layer is collecting data from sensors on a farm and adding it to the blockchain, then sending the encrypted data to an IoT edge gateway for pre-processing and analysis, before finally forwarding the lower volume data to the business solution layer for visualisation and decision making (Sittón-Candanedo et al., 2019). This model could also be modified to host the blockchain on the edge layer, if computational resources at the IoT layer were limited.

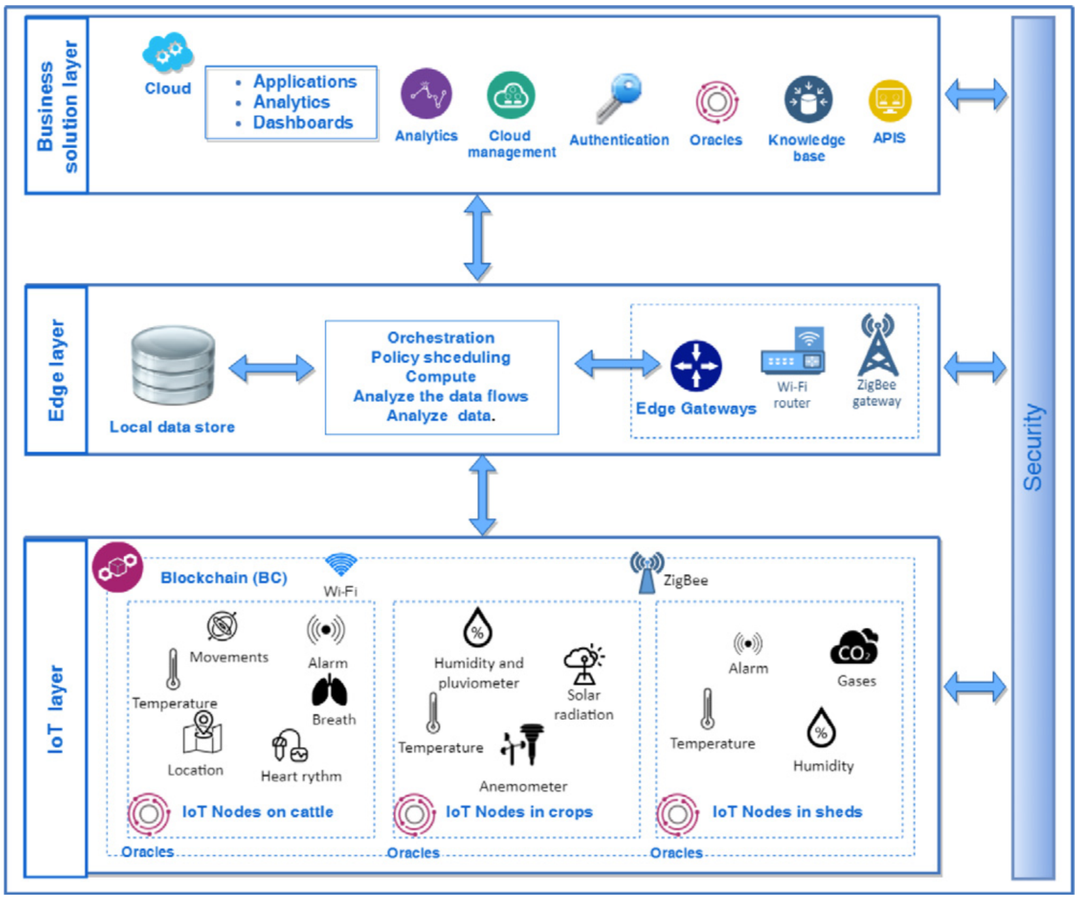


Figure 5: Edge computing architecture for IoT (Sittón-Candanedo et al., 2019)

Similar architectures have been applied on a smaller scale for smart homes with a centralised edge server as the blockchain node that communicates with IoT devices over Wi-Fi in the home network (Dorri et al., 2017; Kullig et al., 2020). In Dorri et al. (2017), the edge server is responsible for running the consensus mechanism and appending blocks to a local private blockchain, as well as performing other admin tasks and providing additional storage if required.

The centralised edge server approach could just as well be applied to any smart environment. Individual smart environments can then become a node in a larger area blockchain, even encompassing entire cities to form a smart-city (Malik et al., 2019).

An alternative to the centralised edge server approach, is to operate the IoT device itself as a node in the blockchain (Huh et al., 2017; Iftekhar et al., 2021). This approach avoids the use of any centralised edge server and functions as a fully distributed network with no compromise on the decentralisation of blockchain. However, the experiments performed by Huh et al. (2017) and Iftekhar et al. (2021) used a Raspberry Pi as the IoT device and blockchain node. The Raspberry Pi is a relatively powerful piece of hardware that is not representative of most IoT devices, which are typically power and computationally constrained (Alfandi et al., 2021). Furthermore, the performance capability of a Raspberry Pi as a blockchain node is itself in doubt due to its resource limitations compared to a more powerful server machine (Misra et al., 2020).

A further limitation in Iftekhar et al. (2021) was the specially compiled Hyperledger Fabric Docker image for the Raspberry Pi ARM processor. A non-standard installation cannot be considered a readily scalable solution. Notwithstanding the need to innovate and test different approaches, a blockchain-IoT network based on standard software installations and unmodified hardware will improve scalability by eliminating the complexity of customisation.

2.4.3 Communication in the blockchain-IoT-edge network

Performance of blockchain for IoT devices in an edge network is affected by network connectivity, with wired connections outperforming wireless (Misra et al., 2020). However, the lower performance of wireless connections can be offset by the advantages of mobility, provided performance achieves an acceptable level for a particular use case.

While it is acceptable to find compromises on performance in the network in lieu of other benefits, it should not be considered acceptable for security and privacy to be compromised under any circumstances. Sharing potentially sensitive data over a WiFi network exposes it to eavesdropping or man-in-the-middle attacks while the data is in transit.

Securing data against such attacks requires an encrypted communication channel between the IoT device and the server to prevent theft of personal network data (Alladi et al., 2020). The ultimate goal should be full end-to-end encryption of data (Hassija et al., 2019), but this can be a challenge for the computationally limited IoT devices (Kromes et al., 2019).

Communication between IoT devices and an Ethereum blockchain on an edge server using HTTP implemented in Python has been successfully demonstrated by Kullig et al. (2020). Exchanging data over HTTP opens the possibility of using APIs to support the development of modular clients and applications in higher level programming languages and web development frameworks. This functionality is a core feature of Hyperledger Sawtooth blockchains with its native REST API and SDK for many popular programming languages.

An implementation of Sawtooth to send data to police and other relevant parties following a car accident used a Raspberry Pi as an off-chain IoT node to send data to the blockchain via REST API (Kromes et al., 2019). The base case of their study found that the Raspberry Pi was able to deliver a 1 MB payload of data to the blockchain in under 5 seconds. A payload of 1 MB is a sizable amount of data for most IoT devices, and this baseline level would be an acceptable level of performance for many applications.

2.5 Summary

Security of IoT data is a challenge that requires an integrated solution throughout the various architectural layers of an IoT system. Blockchain offers several benefits to enhance IoT security, including a secure and practically immutable log of transactions (Alfandi et al., 2021; Hassija et al. 2019). However, previous studies on blockchain for IoT encountered some limitations due to the length of time the consensus mechanism took to approve and add transactions as new blocks (Huh et al., 2017).

Hyperledger Sawtooth offers an efficient consensus mechanism in an open-source private blockchain (Rasolroveicy & Fokaefs, 2020). Performance evaluations of Sawtooth have shown promising results but they were often obtained using high specification environments and test transactions (Ampel et al., 2019; Shi et al., 2019).

A practical application of Sawtooth for IoT security using actual data transactions and lower specification environments is required to evaluate performance in a way that is more representative of real-world scenarios.

Another requirement for integrating blockchain and IoT is finding a suitable solution architecture. To this end, edge computing was identified as a suitable approach as it provides the computational resources close to the resource constrained IoT devices.

3. DESIGN & METHODOLOGY

This chapter will describe the design and methodology for conducting an experiment to evaluate the application of Hyperledger Sawtooth as a blockchain platform to securely store data from an IoT device using an edge computing architecture. It will begin by describing the network design and architecture for the experiment. This will be followed by an outline of the steps to install and configure Hyperledger Sawtooth and related required software. After that, the python code developed for the experiment to handle and process the IoT data transactions will be discussed. Finally, it will detail the methodology for initiating multiple experiments and the process to collect performance data under different workloads and using different settings.

3.1 Network Design & architecture

The experimental network was set up on a home Wi-Fi network, with a MacBook Air laptop as the edge server that hosted three Ubuntu 18.04 virtual machines (VMs) on VirtualBox virtualisation software (see Figure 6). This was the edge layer of the network that hosted the Sawtooth blockchain.

The PoET consensus mechanism in Sawtooth requires at least three validator nodes to operate, hence the requirement for three VMs. Data transactions were sent randomly to the REST API on one of the three VMs to load balance, but in general *EDG10* VM was the primary VM and blockchain node.

Docker images are available from Sawtooth for testing, but VMs are preferred as they give better isolation for performance assessment (Shi et al., 2019) and greater control over the software and other system configurations. The VMs were set up with mostly the default VirtualBox settings but *network* was set to *bridged adaptor* to give the VM its own IP address on the Wi-Fi network. Having an IP address on the Wi-Fi network allowed remote login from the host to the VM for installing software, monitoring, configuration, changing settings, and other admin tasks. SSH was enabled on the VMs to expose port 22 for remote login.

The operating system installed on the VMs was Ubuntu 18.04 in line with the version specified in the Hyperledger Sawtooth documentation[4]. It was installed using

[4] https://sawtooth.hyperledger.org/docs/1.2/app_developers_guide/installing_sawtooth.html#using-ubuntu-for-a-single-sawtooth-node

the minimal install option to save the maximum possible amount of resources for the core system components. The full specifications of the VMs are listed in Table 3. The IoT device was a Raspberry Pi 3 Model B with its native Raspian OS and represents the IoT layer. SSH was also enabled on the Raspberry Pi to enable remote login.

Device	Operating System (OS)	OS Version	RAM	Python version
MacBook Air 2020	MacOS Big Sur	11.6	8 GB	N/A
Virtual Machines	Ubuntu	18.04	1 GB	3.6.9
Raspberry Pi 3, Model B	Raspian	9	1 GB	3.5.3

Table 3: Hardware and virtual machine (VM) specs

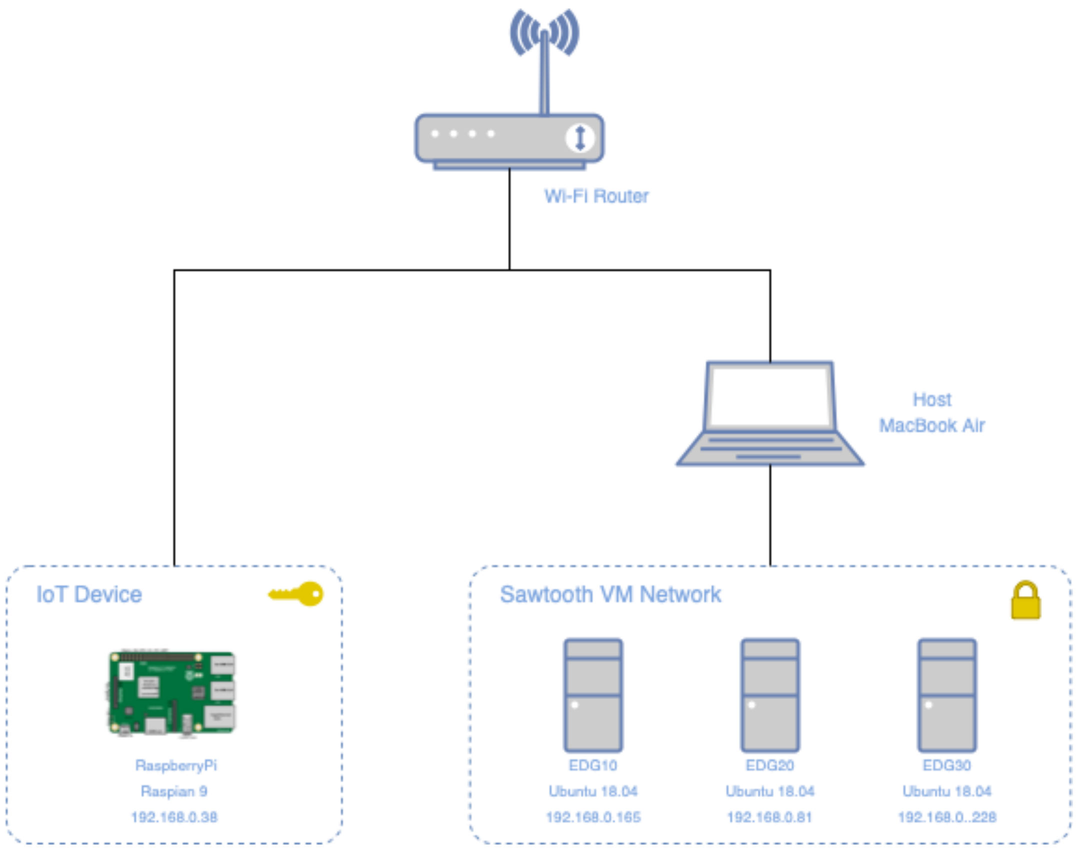


Figure 6: Network design for experiment

A temperature sensor to provide real-world data to store on the blockchain was connected to the Raspberry Pi but is omitted from the network diagram in Figure 6. The architecture is consistent with the model proposed by Sittón-Candanedo et al. (2019), having an edge layer and an IoT layer. There was no business solution layer in the experiment design which focused on performance of the blockchain at the edge and IoT layers.

3.2 Software installation

The main software component for the experiment was Hyperledger Sawtooth, which was selected for its high performance in previous studies that were summarised in Table 2. Hyperledger Sawtooth components were installed on all the VMs that comprise the blockchain network.

Some additional python packages, like *secp256k1* which is used for cryptographic signatures, had to be installed before all Sawtooth components could be successfully installed. These were not mentioned in the documentation and required some debugging that led to delays in the initial set up.

InfluxDB and Telegraf are additional third-party components that were installed on the VMs to collect and store performance metrics. Telegraf is an open-source tool to collect metrics on system performance and InfluxDB is an open-source time-series database used to store the metrics collected by Telegraf and Sawtooth.

Grafana, an open-source data visualisation tool, was installed on the host machine to present the performance metrics on a dashboard[5] for real-time monitoring. The version numbers for each of the major software components are listed in Table 4.

Device	Software	Version
MacBook Air	VirtualBox	6.1.30
Virtual machine	Hyperledger Sawtooth	1.2.6
Virtual machine	InfluxDB	1.8.10
Virtual machine	Telegraf	1.21.4
MacBook Air	Grafana	8.3.4

Table 4: Major software components installed

3.3 Sawtooth configuration

Sawtooth settings are stored both on-chain and off-chain. An example of an on-chain setting is the type of consensus mechanism. On-chain settings are changed by submitting transactions that are approved by the consensus mechanism. Off-chain settings, which are usually network related, are stored in TOML[6] configuration files. Network parameters for the various blockchain components can be passed as command line

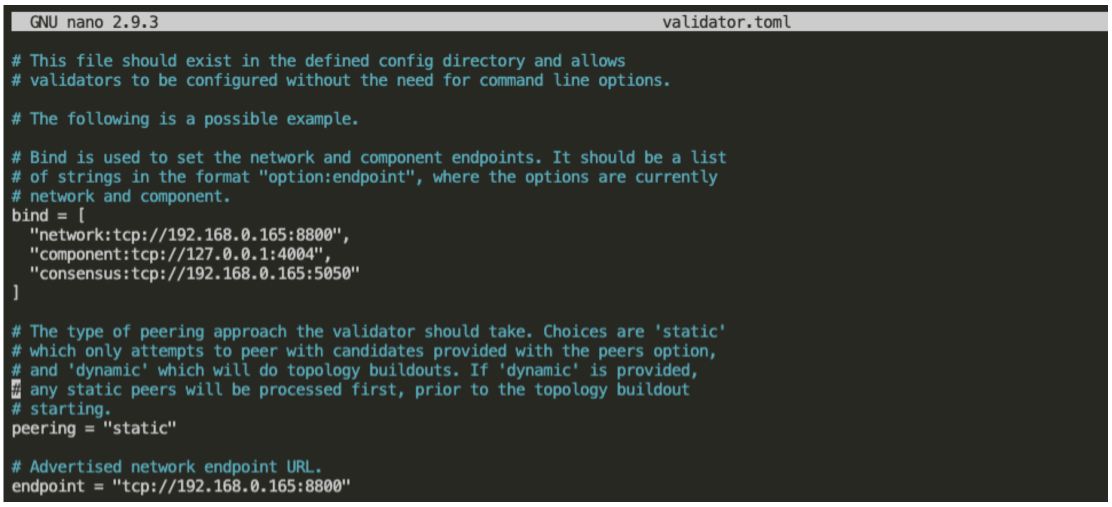
[5] https://raw.githubusercontent.com/hyperledger/sawtooth-core/1-0/docker/grafana/dashboards/sawtooth_performance.json

[6] https://en.wikipedia.org/wiki/TOML

arguments when they are started, but a more optimal solution is to hard code the parameters within the TOML configuration files.

The network parameters tell Sawtooth where to listen for the local components of the current node, like the validator and REST API, to set where other nodes should listen for the current node, and to set the peer nodes of the current node, as shown in Figure 7. Network settings default to localhost if they are not added to the TOML configuration file or not passed as command line arguments when starting the Sawtooth processes. The settings for the InfluxDB database and Telegraf are also stored in the TOML configuration files, including the database name, a database user, and the IP address and port number for the database.

A public and private keypair was also generated and added to the TOML configuration file to secure communication between the Sawtooth components. Communication between nodes and components defaults to unsecured without this key pair. The process for updating the TOML configuration files is outlined in the Sawtooth documentation[7].



```
  GNU nano 2.9.3                                                   validator.toml

# This file should exist in the defined config directory and allows
# validators to be configured without the need for command line options.

# The following is a possible example.

# Bind is used to set the network and component endpoints. It should be a list
# of strings in the format "option:endpoint", where the options are currently
# network and component.
bind = [
  "network:tcp://192.168.0.165:8800",
  "component:tcp://127.0.0.1:4004",
  "consensus:tcp://192.168.0.165:5050"
]

# The type of peering approach the validator should take. Choices are 'static'
# which only attempts to peer with candidates provided with the peers option,
# and 'dynamic' which will do topology buildouts. If 'dynamic' is provided,
# any static peers will be processed first, prior to the topology buildout
# starting.
peering = "static"

# Advertised network endpoint URL.
endpoint = "tcp://192.168.0.165:8800"
```

Figure 7: TOML configuration file

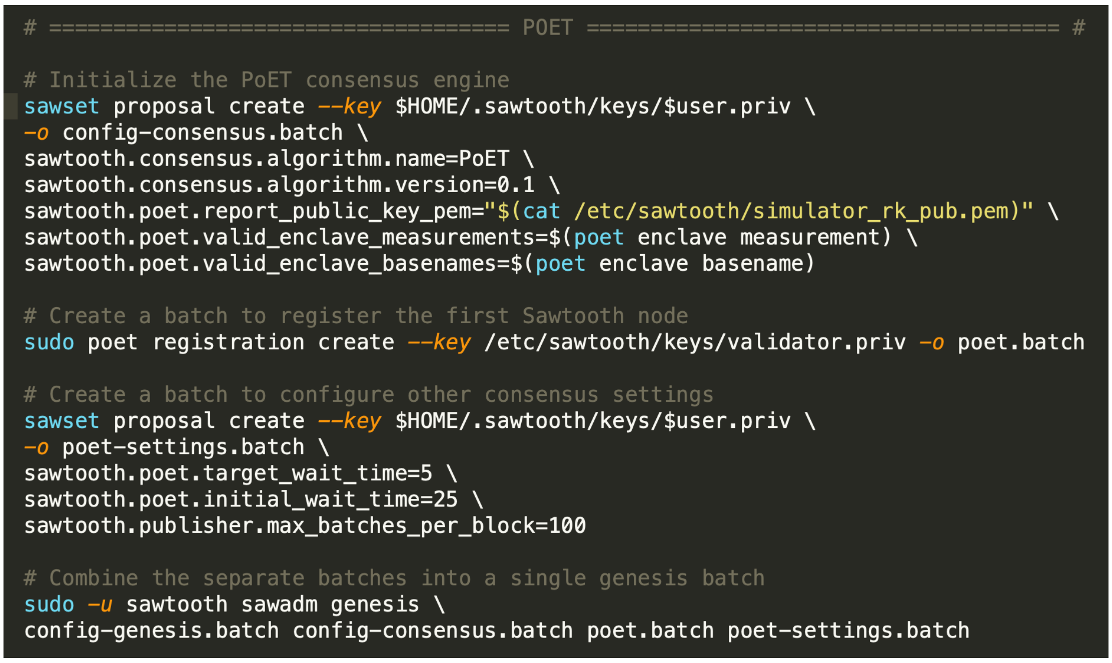
3.4 Initialising the blockchain

After installing the components and configuring the network, the next step was to create a genesis block to be the first block in the blockchain. The genesis block contains the initial settings, including the type of consensus mechanism, that are inherited by other peers when they join the network. Creating a genesis block was done once on the primary

[7] https://sawtooth.hyperledger.org/docs/1.2/sysadmin_guide/configuring_sawtooth.html

node and the entire blockchain from then on contains a chain of references back to that block, where each reference is the header signature of the previous block.

The commands to create the Genesis block, with the PoET consensus mechanism, is shown in Figure 8, and was executed on the EDG10 VM. It contains the type and version of the consensus mechanism and some other settings.



```
# ==================================== POET ===================================== #

# Initialize the PoET consensus engine
sawset proposal create --key $HOME/.sawtooth/keys/$user.priv \
-o config-consensus.batch \
sawtooth.consensus.algorithm.name=PoET \
sawtooth.consensus.algorithm.version=0.1 \
sawtooth.poet.report_public_key_pem="$(cat /etc/sawtooth/simulator_rk_pub.pem)" \
sawtooth.poet.valid_enclave_measurements=$(poet enclave measurement) \
sawtooth.poet.valid_enclave_basenames=$(poet enclave basename)

# Create a batch to register the first Sawtooth node
sudo poet registration create --key /etc/sawtooth/keys/validator.priv -o poet.batch

# Create a batch to configure other consensus settings
sawset proposal create --key $HOME/.sawtooth/keys/$user.priv \
-o poet-settings.batch \
sawtooth.poet.target_wait_time=5 \
sawtooth.poet.initial_wait_time=25 \
sawtooth.publisher.max_batches_per_block=100

# Combine the separate batches into a single genesis batch
sudo -u sawtooth sawadm genesis \
config-genesis.batch config-consensus.batch poet.batch poet-settings.batch
```

Figure 8: Commands to create genesis block for PoET consensus engine

The consensus mechanism is a setting that can be changed at any time with the consensus of the other approval nodes on the network. This dynamic consensus is a core feature of Sawtooth, with three consensus mechanisms available for installation – devmode, PBFT, and PoET.

The documentation advises that devmode is for testing single node environments and is not suitable for multi-node test and production networks, though it has been used in other studies to evaluate performance (Moschou et al., 2020). PBFT on the other hand is recommended for small networks that do not require open membership. PoET is recommended for large production networks with open membership[8]. For this experiment, the PoET consensus mechanism is selected for its high performance (Rasolroveicy & Fokaefs, 2020; Moschou et al., 2020).

[8] https://sawtooth.hyperledger.org/docs/1.2/sysadmin_guide/about_dynamic_consensus.html

3.5 IoT device configuration

There was less configuration required on the Raspberry Pi IoT device compared to the VMs, as it was a client only and did not run any blockchain components. The only software required for the Raspberry Pi to operate as an IoT client was python, which comes pre-installed as standard. The IoT client was developed in python specifically for this experiment and will be discussed in detail later.

The only other configuration step for the IoT device was the creation of a pair of public and private keys, as all components and devices need a public and private key to transact with the Sawtooth Network. The key pair for the Raspberry Pi was created on the EDG10 VM and transferred to the Raspberry Pi using an SCP command, by following the steps shown in Figure 9.

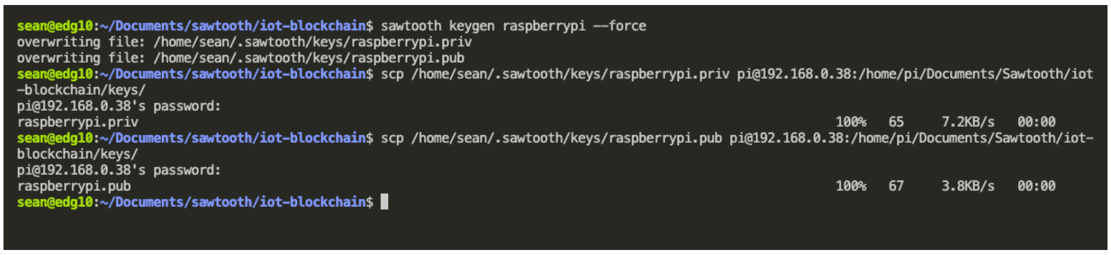

```
sean@edg10:~/Documents/sawtooth/iot-blockchain$ sawtooth keygen raspberrypi --force
overwriting file: /home/sean/.sawtooth/keys/raspberrypi.priv
overwriting file: /home/sean/.sawtooth/keys/raspberrypi.pub
sean@edg10:~/Documents/sawtooth/iot-blockchain$ scp /home/sean/.sawtooth/keys/raspberrypi.priv pi@192.168.0.38:/home/pi/Documents/Sawtooth/iot
-blockchain/keys/
pi@192.168.0.38's password:
raspberrypi.priv                                                                                   100%   65     7.2KB/s   00:00
sean@edg10:~/Documents/sawtooth/iot-blockchain$ scp /home/sean/.sawtooth/keys/raspberrypi.pub pi@192.168.0.38:/home/pi/Documents/Sawtooth/iot-
blockchain/keys/
pi@192.168.0.38's password:
raspberrypi.pub                                                                                    100%   67     3.8KB/s   00:00
sean@edg10:~/Documents/sawtooth/iot-blockchain$ █
```

Figure 9: Distributing a public and private key pair to the IoT device

The public key for the Raspberry Pi, along with the public key of the current node, was then added as a *PERMIT_KEY* in an IoT policy and assigned to the Sawtooth transactor role as shown in Figure 10. The IoT policy gets wrapped in a transaction that must be submitted and validated before being committed to the blockchain, just like any other transaction.

When the IoT policy is committed, transactions and batches can only be signed by the private keys corresponding to the public keys listed in the IoT policy. Any transaction or batch that is signed by any other private key not listed in the policy will be rejected by the *DENY_KEY* asterisk symbol (*) that denies all other keys.

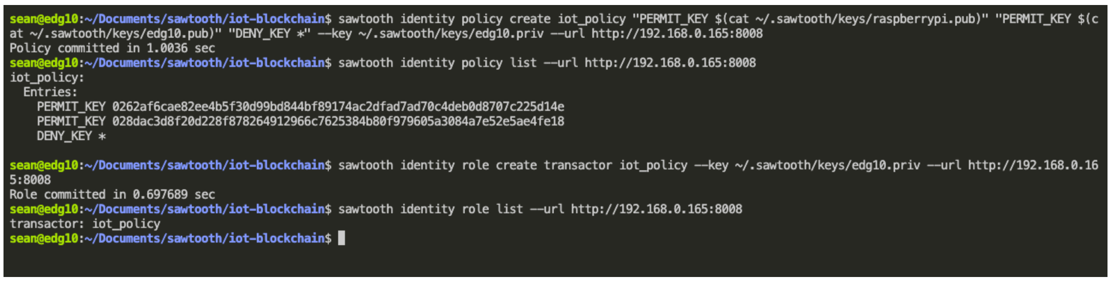
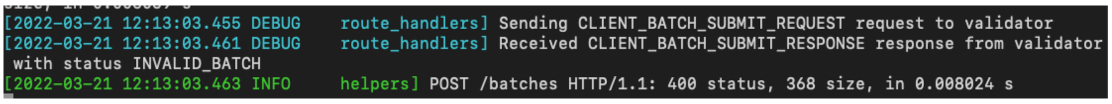

```
sean@edg10:~/Documents/sawtooth/iot-blockchain$ sawtooth identity policy create iot_policy "PERMIT_KEY $(cat ~/.sawtooth/keys/raspberrypi.pub)" "PERMIT_KEY $(c
at ~/.sawtooth/keys/edg10.pub)" "DENY_KEY *" --key ~/.sawtooth/keys/edg10.priv --url http://192.168.0.165:8008
Policy committed in 1.0036 sec
sean@edg10:~/Documents/sawtooth/iot-blockchain$ sawtooth identity policy list --url http://192.168.0.165:8008
iot_policy:
  Entries:
    PERMIT_KEY 0262af6cae82ee4b5f30d99bd844bf89174ac2dfad7ad70c4deb0d8707c225d14e
    PERMIT_KEY 028dac3d8f20d228f878264912966c7625384b80f979605a3084a7e52e5ae4fe18
    DENY_KEY *

sean@edg10:~/Documents/sawtooth/iot-blockchain$ sawtooth identity role create transactor iot_policy --key ~/.sawtooth/keys/edg10.priv --url http://192.168.0.16
5:8008
Role committed in 0.697689 sec
sean@edg10:~/Documents/sawtooth/iot-blockchain$ sawtooth identity role list --url http://192.168.0.165:8008
transactor: iot_policy
sean@edg10:~/Documents/sawtooth/iot-blockchain$ █
```

Figure 10: Setting transactor IoT policy

A test was performed by submitting a transaction from the IoT device that was signed by a randomly generated, but technically valid private key. The test resulted in the batch being rejected for being invalid and a HTTP 400 *bad request* response code was sent to the IoT client, as shown in the console log messages in Figure 11. This level of control improves security by requiring transactors to have explicit permission that is hard coded into the Sawtooth blockchain itself.

Only approved admin users that are also hard coded into the blockchain can change or update the policy. Similar controls on membership can be applied for validator node permissions. This is an example of what makes Sawtooth a private and permissioned blockchain. Public blockchains do not enforce policy or settings to restrict membership or transactions.



```
[2022-03-21 12:13:03.455 DEBUG    route_handlers] Sending CLIENT_BATCH_SUBMIT_REQUEST request to validator
[2022-03-21 12:13:03.461 DEBUG    route_handlers] Received CLIENT_BATCH_SUBMIT_RESPONSE response from validator
 with status INVALID_BATCH
[2022-03-21 12:13:03.463 INFO     helpers] POST /batches HTTP/1.1: 400 status, 368 size, in 0.008024 s
```

Figure 11: Batch rejected for unapproved private key

3.6 Sawtooth application for IoT

Sawtooth doesn't natively support IoT applications as such, but it does support the development of transaction processors and clients for specific use cases (Baralla et al., 2019; Kromes et al., 2019). Development of transaction processors is supported for several popular programming languages, including python, using the available SDK and interfaces. To evaluate the application of Sawtooth for securely storing IoT data using an edge computing architecture, an IoT transaction processor was developed in python as part of the experiment design.

Before transactions can be processed, a client-side application is required to read data from a sensor, add this data to a transaction, then sign, batch and submit the transactions to the Sawtooth REST API. Sawtooth also provides the necessary SDK and interfaces to support the development of client-side applications. An IoT client application was also developed in python as part of the experiment design.

Before discussing the IoT transaction processor and the IoT client-side code in any more detail, it is helpful to explore the structure of a Sawtooth transaction, and how transactions are batched for processing.

3.6.1 IoT transactions

Transactions in Sawtooth belong to families, for example, some on-chain settings are members of the *settings* transaction family and are processed by the settings transaction processor. The structure of a transaction is shown in Figure 12; it consists of a transaction header, a header signature, and a payload. The payload contains the data that will be handled by the transaction processor and can be stored in state. For the IoT application, the payload contained temperature sensor data.

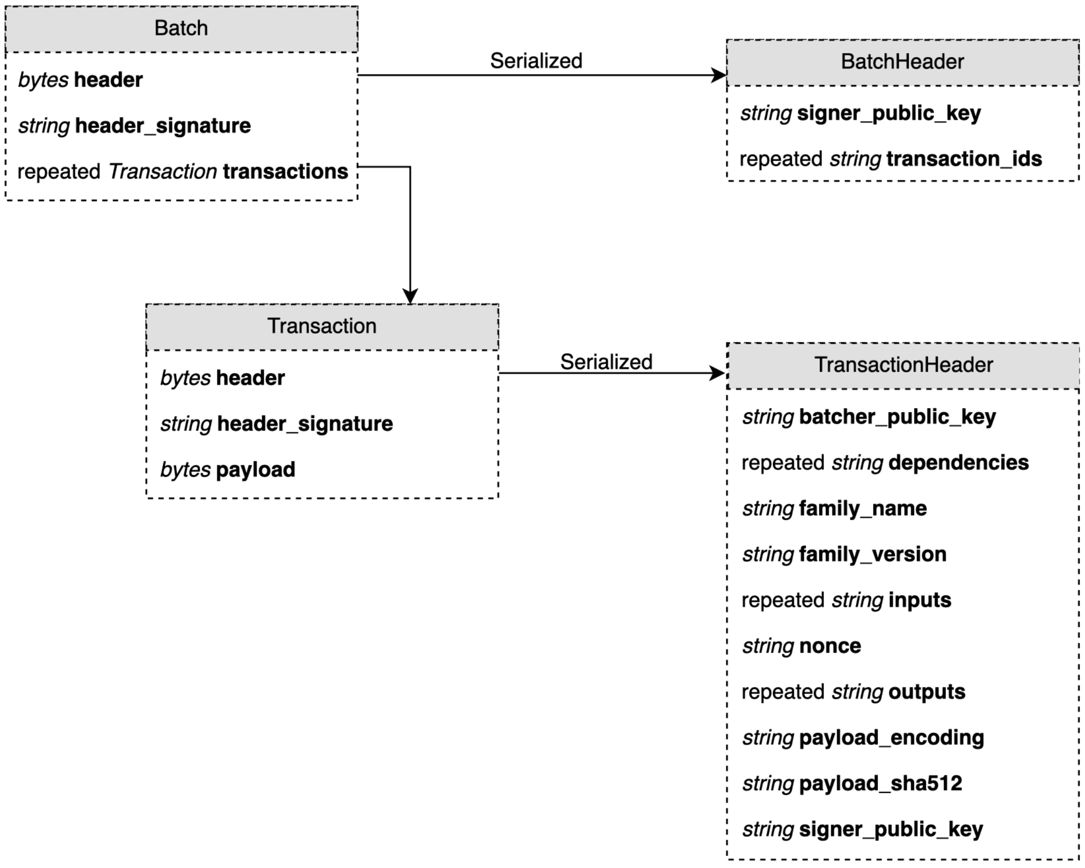


Figure 12: Sawtooth transactions and batches[9]

The transaction header contains multiple fields which are described in Table 5. A transaction header is signed with a private key, like the one that was created and authorised for the Raspberry Pi. Transactions are wrapped in a batch before being submitted to the Sawtooth REST API. A batch can contain one or more transactions in a list, with a batch header and a batch header signature, and is either committed or rejected as an atomic unit. The number of transactions per batch has been shown to have an impact on Sawtooth performance (Ampel et al., 2019).

[9] https://sawtooth.hyperledger.org/docs/1.2/architecture/transactions_and_batches.html

Field	Description
Batcher public key	The public key used to sign the batch containing this transaction
Dependencies	Specify dependencies for transaction processing order
Family name	The transaction family, e.g., IoT-Data
Family version	A version number for the transaction family
Inputs	Address of input state
Nonce	Random number to provide uniqueness of like transactions
Outputs	Address of output state
Payload SHA512	A SHA-512 hash of the transaction payload
Signer public key	Public key used to sign the transaction header

Table 5: Transaction header fields[10]

3.6.2 IoT Device namespace addressing

Sawtooth stores state on a Merkle-Radix tree data structure that has a namespace addressing scheme composed of 35 bytes or 70 hexadecimal characters, giving each leaf node its own unique namespace for storing state data, as shown in Figure 13. The addressing scheme must be unique and deterministic to avoid name collisions and allow retrieval or updating of state data.

A typical approach is to take the first three bytes of the address from a hash of the transaction family name and the remaining 32 bytes from a hash of the public key[11]. Another example in the documentation calculates the 35 bytes from the hash of various attributes belonging to some widget[12]. Another proposed supply chain use case used family name, a resource type, and hashed unique identifiers as the namespace addressing scheme (Baralla et al., 2019).

The approach adopted for the IoT application was to take the first 3 bytes from a hash of the transaction family name, the next 2 bytes from a hash of the device ID, and the remaining 30 bytes from a hash of the public key belonging to the IoT device. Including the device ID creates a unique address in the Merkle-Radix tree for that device, and the values stored in state are only contextual to parties with knowledge of the

[10] https://sawtooth.hyperledger.org/docs/1.2/architecture/transactions_and_batches.html

[11] https://github.com/danintel/sawtooth-cookiejar/blob/master/pyclient/cookiejar_client.py

[12] https://sawtooth.hyperledger.org/docs/1.2/app_developers_guide/address_and_namespace.html

physical device associated with that ID and who have knowledge of the public key for that device. This provides an additional level of privacy through anonymity.

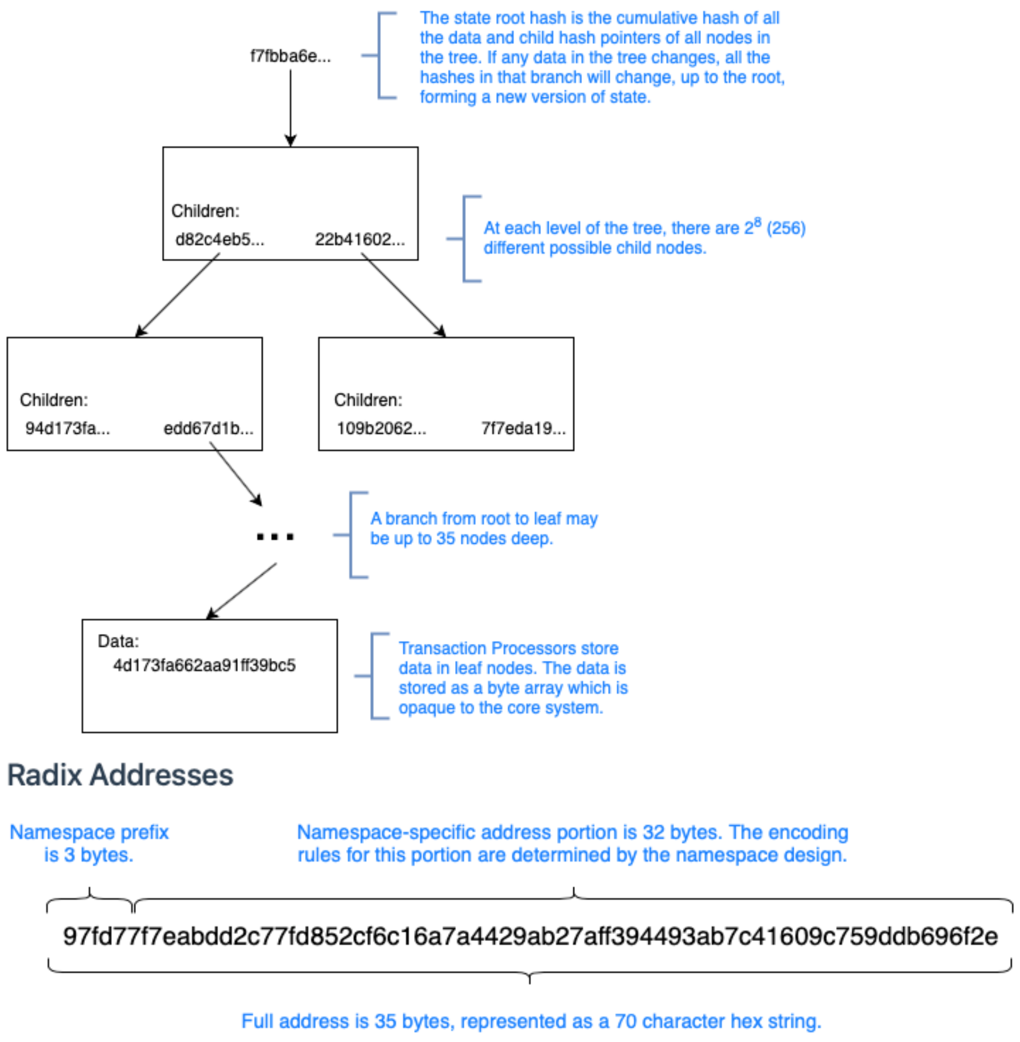

f7fbba6e...
The state root hash is the cumulative hash of all the data and child hash pointers of all nodes in the tree. If any data in the tree changes, all the hashes in that branch will change, up to the root, forming a new version of state.
Children: d82c4eb5... 22b41602...
At each level of the tree, there are 2^8 (256) different possible child nodes.
Children: 94d173fa... edd67d1b...
Children: 109b2062... 7f7eda19...
...
A branch from root to leaf may be up to 35 nodes deep.
Data: 4d173fa662aa91ff39bc5
Transaction Processors store data in leaf nodes. The data is stored as a byte array which is opaque to the core system.
Radix Addresses
Namespace prefix is 3 bytes.
Namespace-specific address portion is 32 bytes. The encoding rules for this portion are determined by the namespace design.
97fd77f7eabdd2c77fd852cf6c16a7a4429ab27aff394493ab7c41609c759ddb696f2e
Full address is 35 bytes, represented as a 70 character hex string.

Figure 13: Global state and namespace addressing[13]

3.6.3 IoT Transaction Processor

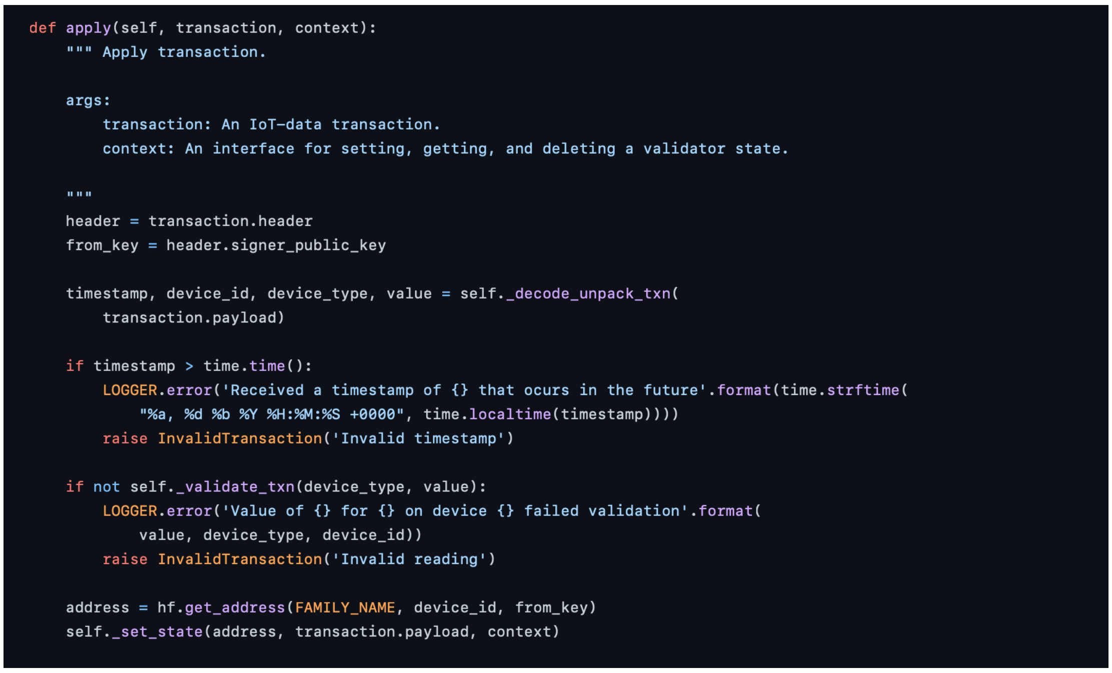
Sawtooth transaction processors are like smart contracts in other blockchains. It is here that business logic and other validation rules can be implemented through the *apply* function, and state data can be set or retrieved (see Figure 14). An IoT transaction processor for an IoT application was developed in python as part of the experiment design and assigned a family called *IoT-data*.

The IoT transaction processor registers with a Sawtooth node, then processes all IoT-data transactions submitted by the IoT client through the Sawtooth REST API. The

[13] https://sawtooth.hyperledger.org/docs/1.2/architecture/global_state.html

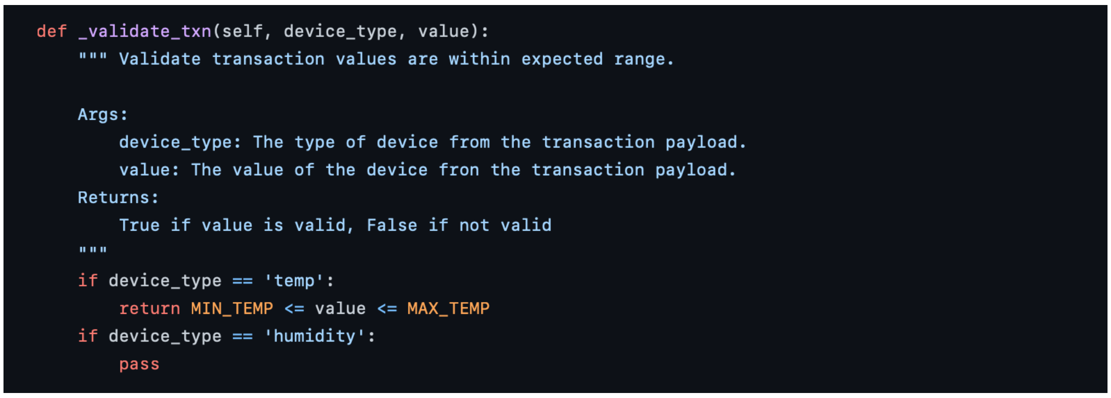
main task of the IoT transaction processor is to save the transaction payload to state. As an additional proof-of-concept, two validation rules were included in the IoT transaction processor, as can be seen in Figure 14 and Figure 15.

One of the validation rules checked that the timestamp was not greater than the current time, as a safeguard against future timestamping of transactions. The other validation rule checked that values for temperature sensors were within a realistic range of between -10 and +35 on the basis that temperatures outside of this range could be erroneous or even malicious readings.



```
def apply(self, transaction, context):
    """ Apply transaction.

    args:
        transaction: An IoT-data transaction.
        context: An interface for setting, getting, and deleting a validator state.

    """
    header = transaction.header
    from_key = header.signer_public_key

    timestamp, device_id, device_type, value = self._decode_unpack_txn(
        transaction.payload)

    if timestamp > time.time():
        LOGGER.error('Received a timestamp of {} that ocurs in the future'.format(time.strftime(
            "%a, %d %b %Y %H:%M:%S +0000", time.localtime(timestamp))))
        raise InvalidTransaction('Invalid timestamp')

    if not self._validate_txn(device_type, value):
        LOGGER.error('Value of {} for {} on device {} failed validation'.format(
            value, device_type, device_id))
        raise InvalidTransaction('Invalid reading')

    address = hf.get_address(FAMILY_NAME, device_id, from_key)
    self._set_state(address, transaction.payload, context)
```

Figure 14: IoT transaction processor apply method



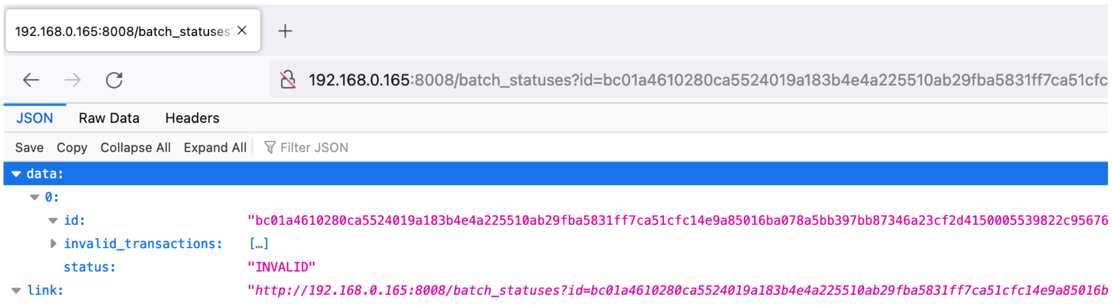
```
def _validate_txn(self, device_type, value):
    """ Validate transaction values are within expected range.

    Args:
        device_type: The type of device from the transaction payload.
        value: The value of the device fron the transaction payload.
    Returns:
        True if value is valid, False if not valid
    """
    if device_type == 'temp':
        return MIN_TEMP <= value <= MAX_TEMP
    if device_type == 'humidity':
        pass
```

Figure 15: IoT transaction processor validation rule

If the validation rules are violated, an invalid transaction exception is raised, and the payload will not be saved to state. The REST API response of a test transaction that contained an injected temperature value outside of the valid range is shown in Figure

16. The response shows the transaction with a status of *INVALID*, meaning it was not committed to the blockchain. This is a somewhat trivial example, but it serves to demonstrate the functionality that can be applied to increase security through enhanced validation rules in the transaction processor.

Applying validation rules helps to secure the blockchain against malicious attempts to inject data values outside of a specified range and can also help protect against spurious or erroneous readings from a defective or faulty sensor.


192.168.0.165:8008/batch_statuses
192.168.0.165:8008/batch_statuses?id=bc01a4610280ca5524019a183b4e4a225510ab29fba5831ff7ca51cfc
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
data:
  0:
    id:                      "bc01a4610280ca5524019a183b4e4a225510ab29fba5831ff7ca51cfc14e9a85016ba078a5bb397bb87346a23cf2d41500055398 22c95676
    invalid_transactions:    […]
    status:                  "INVALID"
link:                        "http://192.168.0.165:8008/batch_statuses?id=bc01a4610280ca5524019a183b4e4a225510ab29fba5831ff7ca51cfc14e9a85016b
```

Figure 16: REST API response showing invalid transaction

3.6.4 IoT Client Application

It was the job of the IoT client application to read sensor values, create and batch transactions, sign the transactions, then submit the batched transactions to the Sawtooth REST API on one of the VM nodes. The IoT client consisted of multiple python scripts which were developed and pushed to a GitHub repository[14]. A *requirements.txt* file with a list of the python packages and the specific version required was also bundled in the repository. To perform the experiments, the python scripts were cloned from their GitHub repository and run in a python virtual environment on the Raspberry Pi IoT device.

Some compatibility issues occurred initially with python packages that had been updated since the latest Sawtooth release (installed version). There was no explanation for this in the Sawtooth documentation and debugging was required to overcome some errors. The requirements.txt file lists the versions of the packages that are compatible with the installed version of Sawtooth, as discovered through the debugging process. Running in the virtual environment with the versions specified in the requirements.txt file overcame the compatibility issues.

[14] https://github.com/SeanConnolly82/iot-blockchain

The process to set-up and activate the virtual environment, including installation of the required packages, was initiated by running a start-up bash script that was included in the GitHub repository. Command line arguments to initialise a specific IoT device and its operation were passed to the IoT client by adding them to the start-up bash script. Logging functionality was also included in the IoT client to generate log files for performance monitoring and debugging in the event of any issues or errors occurring.

An important component of the IoT client was the *sensor* class to represent the physical sensor. The sensor class is shown as Figure 17. All attributes of the sensor were wrapped inside the single *payload* dictionary that became the payload of the Sawtooth transaction. The specific type of sensor was abstracted away from the class definition, so that a sensor object could represent any possible sensor as indicated by the *device_type* attribute.

For the current experimental implementation, two types of sensors were possible, temperature and humidity, consistent with the type of sensors used for similar studies (Kullig et al., 2020), though only temperature was implemented in the final experiment design. To avoid runtime errors, a control was applied to the command line argument parser to limit the selection of device type to either *temp* or *humidity*. The selection of either temp or humidity determines the validation rules that were applied when the transaction was being processed (see Figure 15).

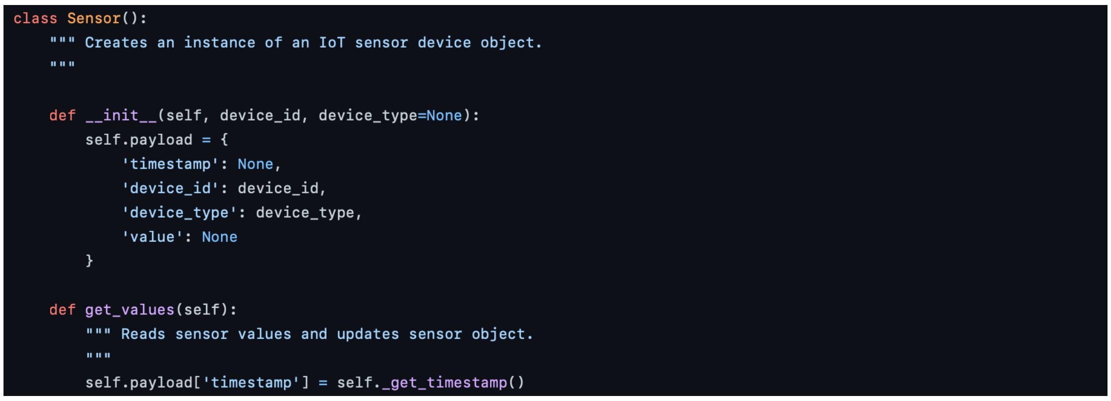

```
class Sensor():
    """ Creates an instance of an IoT sensor device object.
    """

    def __init__(self, device_id, device_type=None):
        self.payload = {
            'timestamp': None,
            'device_id': device_id,
            'device_type': device_type,
            'value': None
        }

    def get_values(self):
        """ Reads sensor values and updates sensor object.
        """
        self.payload['timestamp'] = self._get_timestamp()
```

Figure 17: IoT client sensor class

Each sensor object also had a *device_id* attribute to provide a unique identifier for a particular sensor. Recall that device id was also one of the inputs used to determine the namespace address for storing the sensor state. The device id only needs to be unique within the scope of a public key to avoid collisions for namespace addressing, but good practice would be to keep it unique among all sensor instances. The other attributes of the sensor were the *timestamp* in Unix time and the *value* which is the latest reading from the sensor.

After getting the sensor values and adding them to transactions and batches, the other major function of the IoT client was to send the batched transactions to the REST API of a Sawtooth node. Communication between the IoT client and the Sawtooth node was implemented using the python *requests* library. The batched transactions were sent in a HTTP *POST* request payload to the Sawtooth node IP address on port number 8008.

State values at a particular namespace address could also be retrieved using a *GET* request without submitting a transaction. This is a useful feature that enables authorised parties with knowledge of a public key and associated device ids to use the data stored in state for applications and use cases at the business solution layer.

For the current implementation, the physical temperature sensor circuit was assembled on a breadboard and connected to the Raspberry Pi GPIO pins using a starter kit available from Freenove[15]. The temperature sensor circuitry can be seen in Figure 18.

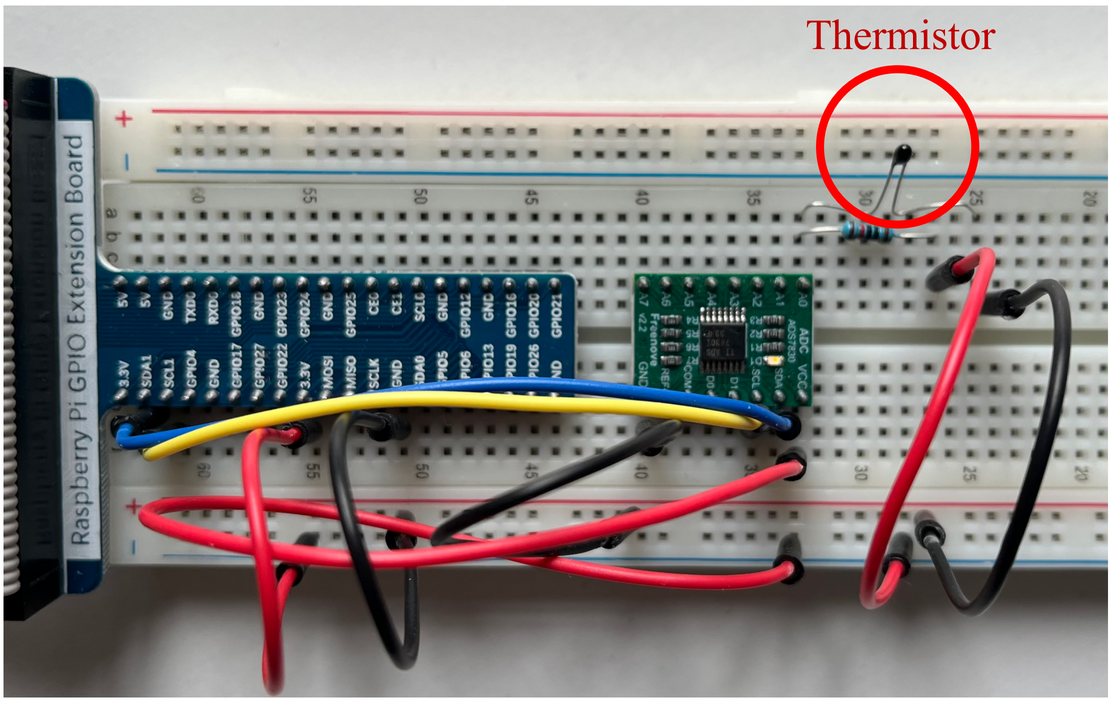


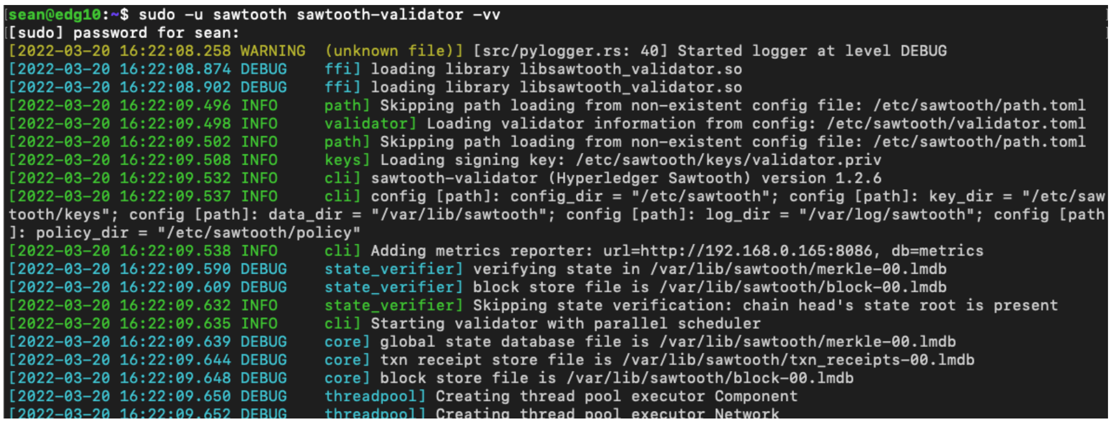
Figure 18: Temperature sensor breadboard circuit

[15] https://github.com/Freenove/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi

3.7 Starting the network

When all software and other components were installed, and the network was configured, the blockchain nodes and the IoT client were started. Sawtooth processes were started first, beginning with the EDG10 VM as the primary node that contained the genesis block. Processes were started manually from the command line of the VM via a remote SSH session from the host machine. The minimum processes that must be started on Sawtooth are the validator, the REST API, the settings transaction processor, the PoET consensus engine, and the IoT transaction processor.

Each Sawtooth process was started separately and immediately started logging output to the console window at the level of verbosity specified from *-v* (less) to *-vvv* (more). An example of the logging output can be seen in Figure 19. Higher levels of logging verbosity are better for monitoring during the initial testing and evaluation of the network. Log files are also written to disk if required for later analysis.



```
sean@edg10:~$ sudo -u sawtooth sawtooth-validator -vv
[sudo] password for sean:
[2022-03-20 16:22:08.258 WARNING  (unknown file)] [src/pylogger.rs: 40] Started logger at level DEBUG
[2022-03-20 16:22:08.874 DEBUG    ffi] loading library libsawtooth_validator.so
[2022-03-20 16:22:08.902 DEBUG    ffi] loading library libsawtooth_validator.so
[2022-03-20 16:22:09.496 INFO     path] Skipping path loading from non-existent config file: /etc/sawtooth/path.toml
[2022-03-20 16:22:09.498 INFO     validator] Loading validator information from config: /etc/sawtooth/validator.toml
[2022-03-20 16:22:09.502 INFO     path] Skipping path loading from non-existent config file: /etc/sawtooth/path.toml
[2022-03-20 16:22:09.508 INFO     keys] Loading signing key: /etc/sawtooth/keys/validator.priv
[2022-03-20 16:22:09.532 INFO     cli] sawtooth-validator (Hyperledger Sawtooth) version 1.2.6
[2022-03-20 16:22:09.537 INFO     cli] config [path]: config_dir = "/etc/sawtooth"; config [path]: key_dir = "/etc/saw
tooth/keys"; config [path]: data_dir = "/var/lib/sawtooth"; config [path]: log_dir = "/var/log/sawtooth"; config [path
]: policy_dir = "/etc/sawtooth/policy"
[2022-03-20 16:22:09.538 INFO     cli] Adding metrics reporter: url=http://192.168.0.165:8086, db=metrics
[2022-03-20 16:22:09.590 DEBUG    state_verifier] verifying state in /var/lib/sawtooth/merkle-00.lmdb
[2022-03-20 16:22:09.609 DEBUG    state_verifier] block store file is /var/lib/sawtooth/block-00.lmdb
[2022-03-20 16:22:09.632 INFO     state_verifier] Skipping state verification: chain head's state root is present
[2022-03-20 16:22:09.635 INFO     cli] Starting validator with parallel scheduler
[2022-03-20 16:22:09.639 DEBUG    core] global state database file is /var/lib/sawtooth/merkle-00.lmdb
[2022-03-20 16:22:09.644 DEBUG    core] txn receipt store file is /var/lib/sawtooth/txn_receipts-00.lmdb
[2022-03-20 16:22:09.648 DEBUG    core] block store file is /var/lib/sawtooth/block-00.lmdb
[2022-03-20 16:22:09.650 DEBUG    threadpool] Creating thread pool executor Component
[2022-03-20 16:22:09.652 DEBUG    threadpool] Creating thread pool executor Network
```

Figure 19: Logging output of the Sawtooth validator

After starting the components for the first time, an approved node inherits the genesis block and associated settings from a node already active on the network. Sawtooth nodes must be fully peered before any transactions can be approved by the consensus engine. Prior to starting the experiments, peering was confirmed by reviewing the logging output and by checking through a web browser connection to the REST API.

As soon as peering had occurred, the operation of the network was confirmed by submitting a transaction to the *IntegerKey* built-in transaction processor. IntegerKey provides shell commands for submitting test transactions to confirm the network is functional. Some studies have based their entire performance evaluation of Sawtooth on the IntegerKey transaction processor (Shi et al., 2019) but this limits the validity of the experiment to be generalised for real-world applications.

3.7.1 Starting the IoT Client and sending transactions

When the Sawtooth network was confirmed to be fully peered and operational, the IoT client application was started on the IoT device. The Raspberry Pi IoT device was interfaced via an SSH session from the host machine and the IoT client was started by calling a bash script.

The bash script included the commands to set up and activate a python virtual environment and start the python processes with the required command line arguments reflecting the parameters for the experiment. This included the time interval for reading the sensor data to enable the application to send transactions at different rates of input to evaluate performance across varying workloads.

When a time interval argument was included, the IoT client read data and sent transactions, every number or fraction of seconds specified, in a continuous loop until interrupted using a keyboard interrupt. If the interval was omitted, the IoT client sent a single transaction to Sawtooth.

Table 6 outlines the specification for the IoT client command line arguments. The first command line argument is either *GET* or *POST*, indicating the HTTP method to be called. POST requires two further command line arguments to specify the type of device, and an ID for the device, with a further optional argument to specify the interval for sending data to Sawtooth.

If the first command line argument is GET, then only one additional argument is required – the device ID, which is sufficient to retrieve the current state value from the namespace address for that device. GET requests are retrieved from state using a REST API endpoint and do not require a transaction to be created.

Argument	**Description**	**Allowed values**
action	HTTP method that will be invoked	POST, GET
device_id	An ID for the individual sensor	Any string
device_type	The type of sensor	temp, humidity
interval	Interval in seconds for sending transactions	A valid floating-point number

Table 6: IoT client command line arguments

When the network was fully peered, and the IoT client was started, an initial batch of 200 transactions was submitted from the IoT device to the REST API to test that the network was fully operational. Operation could be monitored in real-time through the Grafana dashboard on the host machine (see Figure 20). When the 200 transactions were observable in the count of committed transactions on the dashboard, the experiment was initiated by executing the IoT client start-up script with an interval in seconds to give the required rate of input transactions.

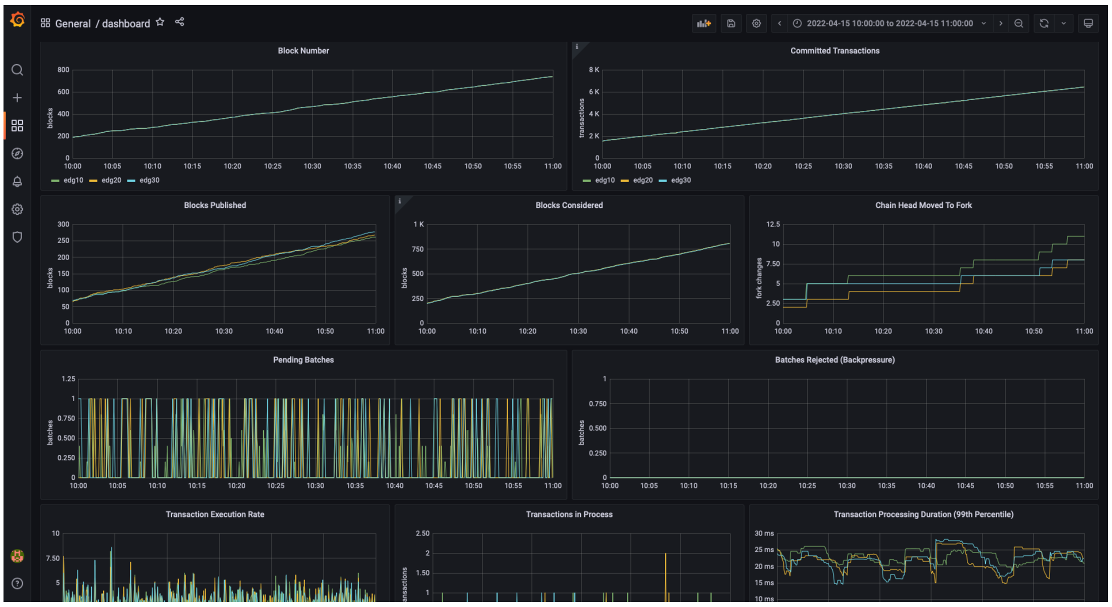


Figure 20: Monitoring performance on the Grafana dashboard

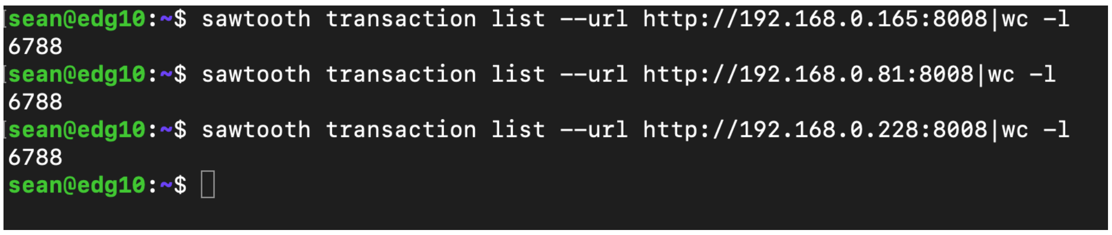
3.8 Experiment process

Multiple experiments were carried out to evaluate performance of different configurations under different input transaction workloads. Transactions were submitted as either single transaction batches or multiple transaction batches as the number of transactions per batch has been shown to have an impact on Sawtooth performance (Ampel et al., 2019).

Single transaction batches were submitted with a target tps of between 2 and 12, incrementing in steps of 2 transactions. Multiple transaction batches were submitted with a target tps of 8, 10, and 12, enabling direct comparison with the upper end of the single batch transactions target tps. A further target tps of 20 was also submitted as a multiple transaction batch, to measure performance at a higher throughput workload. The number of transactions in the multiple transaction batches was set to 10, in line with the number of transactions per second specified in the hypothesis test.

Each experiment was allowed to run for just over one hour to ensure that there was a 60-minute window of stable operating data available for each rate of input transaction. Following each experiment, a counter of the number of transactions sent by the IoT device was used to confirm that all transactions had been committed to the blockchain. The metric for how many transactions were committed versus how many were sent is known as the success rate, and the expected performance is 100% (Ampel et al., 2019).

The number of committed transactions must also be consistent across each of the blockchain nodes, indicating that all nodes have received and committed all transactions that were sent by the IoT device. Figure 21 shows an example of a consistent count of transactions across blockchain nodes, using the *show transaction list* command piped into a count of line endings. This data is also available through the Grafana dashboard and can be accessed from the InfluxDB database for more detailed analysis.



```
sean@edg10:~$ sawtooth transaction list --url http://192.168.0.165:8008|wc -l
6788
sean@edg10:~$ sawtooth transaction list --url http://192.168.0.81:8008|wc -l
6788
sean@edg10:~$ sawtooth transaction list --url http://192.168.0.228:8008|wc -l
6788
sean@edg10:~$ 
```

Figure 21: Checking the number of committed transactions

Data was collected for each experiment and stored in the InfluxDB instance running on EDG10. InfluxDB is a time-series database that stores data in table-like structures that it calls measurements. It has been used in other performance evaluations of Hyperledger Sawtooth (Shi et al., 2019).

InfluxDB was the only source of data used for the analysis and evaluation of Sawtooth performance. No external or third-party data was required for any part of the analysis or results. A small change was required to the Sawtooth core files to lower the granularity of the time-series from 10 seconds to one second, to capture data at the rate required to test the hypothesis. This change to the core Sawtooth files was the only non-standard feature of the entire installation.

Analysis of the results dataset to evaluate performance and test the hypothesis was performed in the R statistical analysis application. A package called *influxdbr* enabled connection from R to the InfluxDB instance on EDG10 to import data. InfluxDB implements a SQL-like InfluxQL query language and this was one of the options available to perform filtering, aggregation, and other data transformations at the database level. However, the approach adopted was to use influxdbr to load the entire dataset from its InfluxDB source into an R dataframe. In this way, all the analysis, transformation and visualisation of data could be performed in one application.

Data on throughput was sourced from an InfluxDB measurement that contains a value for the count of transactions stored on the blockchain. The value was in absolute terms and required some additional processing to calculate the relative change in the number of transactions from one time interval to the next. This step was necessary to report the number of transactions per second. The same process was followed for data on the number of blocks added, which was available in another InfluxDB measurement.

Mean tps was the primary measure of throughput and the key metric for measuring overall performance. Throughput is the rate at which transactions are added to the blockchain. A secondary measure of performance was the standard deviation, to evaluate the amount of variance in the rate of transaction throughput. A low standard deviation relative to its corresponding mean tps, is indicative of stable and consistent performance, whereas higher values are indicative of unstable and intermittent performance.

3.9 Summary

An experiment to evaluate performance of a practical application of Sawtooth for IoT data security was designed with an architecture consisting of an edge layer and an IoT layer, similar to that proposed by (Sittón-Candanedo et al., 2019). The edge layer consisted of a MacBook Air laptop that hosted three virtual machines. The IoT layer consisted of a Raspberry Pi connected to a temperature sensor.

Hyperledger Sawtooth was installed on each of the three VMs as this is the minimum number of nodes required for the PoET consensus mechanism to operate. Other software was also installed on the edge layer to collect and display data and metrics on the blockchain performance.

A transaction processor was developed in python for the experiment, to receive transactions and apply some proof-of-concept validation rules before saving the temperature data to the blockchain state. An IoT client was also developed in python to read temperature data from the sensor and then add it to a transaction within a batch that was then submitted to the REST API.

The physical sensor device, which was assembled on a breadboard and connected to the Raspberry Pi, was represented in the IoT client as an object of a sensor class, and its attributes would become the payload of the IoT data transaction.

After the IoT client and Sawtooth components were started, some testing was applied to confirm that the blockchain nodes had peered and the network was operational. A series of experiments were then performed with a range of input transactions in both single and multiple batch transactions. This ensured performance was evaluated under a range of workloads.

Each experiment submitted transactions for a 60-minute period and data on blockchain performance was collected in an InfluxDB time-series database at one-second intervals. Data from InfluxDB was then imported into R for analysis, visualisation and to perform the hypothesis test.

Mean tps was the primary measure of performance in terms of throughput, with standard deviation as a secondary measure to evaluate the variance. Measuring variance is important to understand the overall stability or intermittency in the performance of transaction throughput.

4. RESULTS, EVALUATION, AND DISCUSSION

In this chapter, the results of the experiments described in chapter 3 will be presented alongside an initial discussion of the key metrics relating to transaction throughput. This will be followed by a detailed analysis and evaluation of the data, leading up to the hypothesis test. The chapter will conclude with a discussion of the main outcomes and findings from the experiments and hypothesis test.

4.1 Real-time observations of experiment performance

Performance during the experiments was monitored in real-time using the Grafana dashboard. This provided an initial subjective evaluation before results were available for analysis. Under ideal conditions, a linear rate of input transactions would have an approximately corresponding linear rate of transactions being committed to the blockchain. This ideal scenario would provide a stable platform with predictable performance to support applications and use cases at the business solution layer.

Figure 22 shows an example of the Grafana dashboard displaying the increase in the number of committed transactions during one of the experiments. At a macro level, the rate of increase of committed transactions appears approximately linear, as would be expected for a given constant rate of input transactions. However, while the rate of committed transactions does have an approximate linear form, zooming in reveals occasional stepped increases after periods of no change.

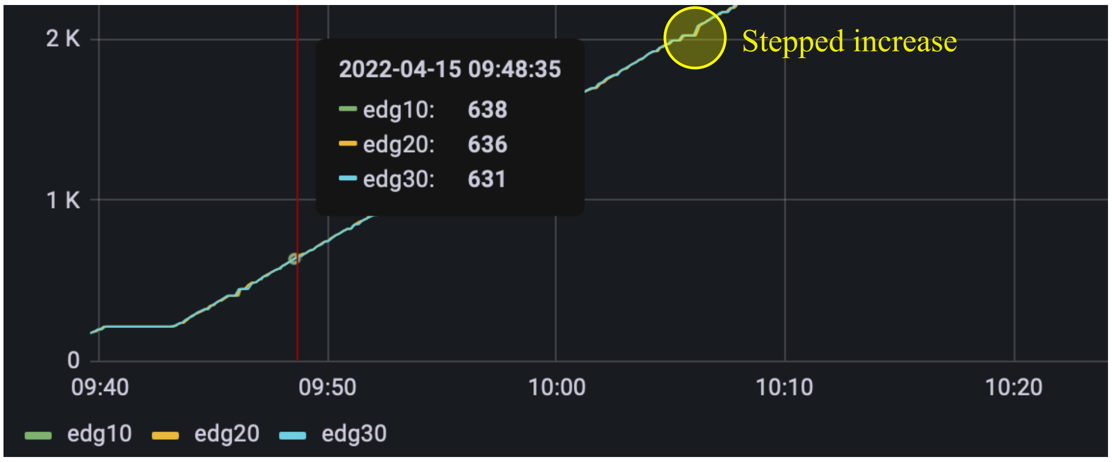
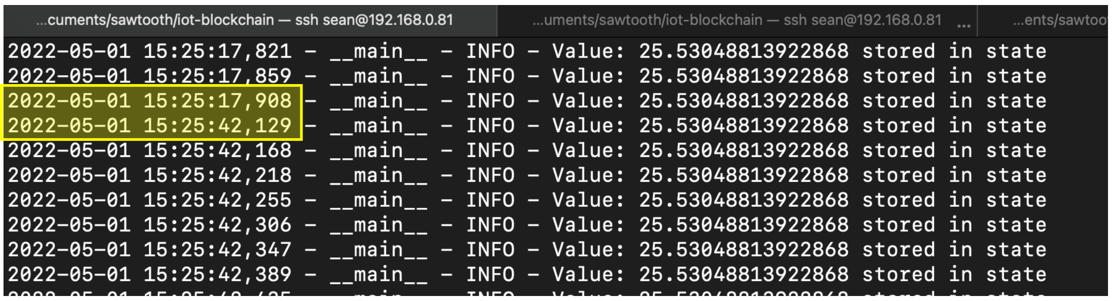


Figure 22: Grafana committed transactions dashboard, showing stepped increase

The pattern of stepped increments in Figure 22 is consistent with some intermittency in the console logging output of the transaction processor, which was also monitored during the experiment. The logging messages often froze temporarily, followed by a short burst of activity, before settling back to a steadier rate of processing activity.

An example of intermittency in the logging output is highlighted in Figure 23, which shows an approximately 25 second period when no transactions were processed by the IoT transaction processor. These periods of inactivity, followed by bursts of activity, were probably the result of the internal operation of the Sawtooth components and the consensus mechanism. Another possibility is that some of this behaviour may have been due to bottlenecks and latencies in the network.


...cuments/sawtooth/iot-blockchain — ssh sean@192.168.0.81
...uments/sawtooth/iot-blockchain — ssh sean@192.168.0.81
...ents/sawtoo

```
2022-05-01 15:25:17,821 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:17,859 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:17,908 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,129 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,168 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,218 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,255 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,306 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,347 - __main__ - INFO - Value: 25.53048813922868 stored in state
2022-05-01 15:25:42,389 - __main__ - INFO - Value: 25.53048813922868 stored in state
```

Figure 23: IoT transaction processor log messages showing inactivity

The freeze in activity often coincided with spikes in pending batches that could be observed on another Grafana display. Recall that transactions are added and processed by the validator in batches. An example of a spike in pending batches is visible on Figure 24 between 20:34 to 20:35. There is no clear explanation for spikes like this as conditions were kept constant throughout the experiment. If pending batches are not processed within a certain time-period, it can lead to rejection of the batches and all transactions contained therein due to queue timeout (Ampel et al., 2019).

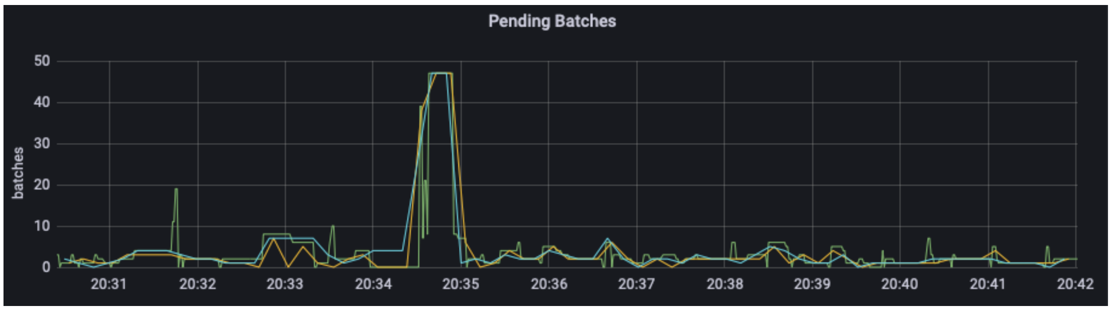


Figure 24: Grafana pending batches dashboard

Another observation of the data that is visible on Figure 22 is that the number of committed transactions on each node can be slightly different in a given time interval. The number of committed transactions on each of the three nodes varies between 631 and 638 in the time interval highlighted by the tooltip display on Figure 22. This is a point-in-time phenomenon, as the different nodes appended blocks slightly earlier or later than other nodes.

Different nodes alternated between leading and lagging, in terms of the number of committed transactions, at different times. While this does not necessarily represent an issue, it is a feature of the blockchain that could have implications for applications at the business solution layer that have time-sensitive functionality.

Ultimately, each of the nodes had the same block head and the same number of committed transactions when the experiments concluded, though the blockchain had forked several times during each of the experiments. Forking is the process by which the blockchain manages divergences in the blocks comprising the chain. This tendency to fork, especially at higher rates of input has also been observed in other performance evaluations of Hyperledger Sawtooth (Shi et al., 2019).

A full analysis and discussion of the consequences of forking are beyond the scope of this study but would warrant further analysis in future work. Forking in the blockchain is not necessarily an issue but a greater understanding of the causes and implications would be required before the experiment could be scaled for real-world use cases.

4.2 Experiment Results

A summary of the results obtained for each iteration of the experiment can be found in Table 7. The results include the mean number of transactions committed to the blockchain per second as *mean tps*, with its corresponding standard deviation, for each *target tps*. The target tps is the target rate of input transactions submitted by the IoT client via the REST API. Mean tps is the primary measure of performance in terms of transaction throughput and will ultimately be used to test the research hypothesis.

The number of *transactions per batch* are also included in the results table. The transactions per batch were either one or ten, but these will generally be referred to as either single transaction batches or multiple transactions batches.

The first observation on the results in Table 7 is that the rate of committed transactions, as measured by mean tps, did not achieve the target tps for any of the experiments, for either single transaction batches or multiple transaction batches. The mean tps was less than half the target tps in most instances. It is important to interpret this correctly, as the results are not indicating that, for example, 10 transactions were submitted and only 5 were committed to the blockchain, but rather that the actual input rate from the IoT client was constrained by the performance of the network to a level close to the mean tps.

Target tps	Transactions per batch	Mean tps	Standard deviation
2	1	1.367	4.498
4	1	1.917	6.619
6	1	2.484	8.752
8	1	3.814	13.046
10	1	4.518	13.996
12	1	5.006	16.064
8	10	5.953	20.077
10	10	6.783	23.541
12	10	7.634	26.911
20	10	10.680	37.116

Table 7: Results showing mean tps and standard deviations

Another observation on the results is the high variability in the number of committed transactions per second, as indicated by the large standard deviations relative to their respective means. This high variability is consistent with the findings of Shi et al. (2019) whose results also showed high variance that increased with increasing rates of input transactions. Variability in the results is an indicator of unstable performance which may impact on how data from the blockchain is used and consumed at the business solution layer.

The results in Table 7 also show that multiple transaction batches consistently achieved a higher mean tps compared to single transaction batches, though variance was higher for the multiple transaction batches. This observation is based on the target tps of 8, 10 and 12, where a direct comparison is possible between single and multiple transaction batches. Figure 25 shows the comparison where the higher throughput of multiple transaction batches is strongly evident. Ampel et al. (2019) also reported batch size as a factor affecting performance in their study of Sawtooth.

At a target rate of 20 tps, the mean tps achieved was more than 10 tps, offering some initial support for the hypothesis that Sawtooth can process IoT data transactions at a rate of at least 10 transactions per second.

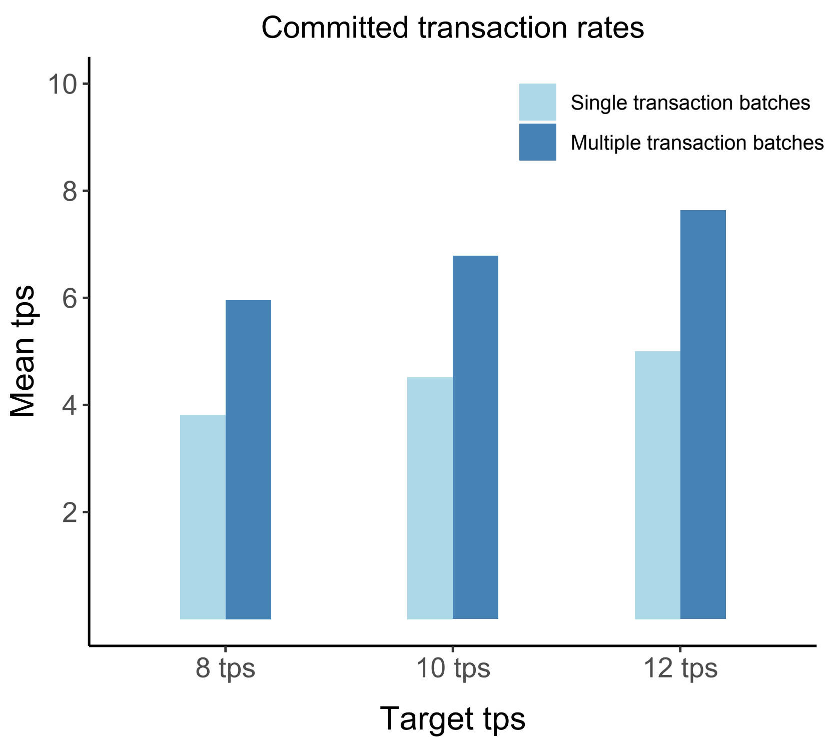


Figure 25: Mean tps for single and multiple transaction batches

4.2.1 Transaction throughput

InfluxDB records the second-by-second time-series of the number of transactions committed to the blockchain. Taking a closer look at the data at this level of granularity gives a greater insight into how transactions were committed to the blockchain. Figure 26 shows a two-minute time slice from one of the experiments. The first observation on the data when viewed at this level of granularity is the many one-second time intervals when no transactions at all were committed to the blockchain.

What Figure 26 shows is that transactions were committed intermittently, in volumes that were proportional to the amount of time that had elapsed since the last transactions were committed. This intermittency creates latency in the system, as a transaction is not available to applications at the business solution layer until it has been committed to the blockchain. Intermittency of committing transactions will have contributed to the high level of variance observed in the results data.

Another observation on the data was negative numbers of transactions in some one-second intervals, for some iterations of the experiment. The data has been thoroughly checked and it was confirmed that this is not a data quality issue. The negative values are most likely due to removal of duplicate transactions by Sawtooth to keep the blockchain consistent across nodes. Although this occurred infrequently, it warrants further investigation to obtain a fuller understanding in future studies.

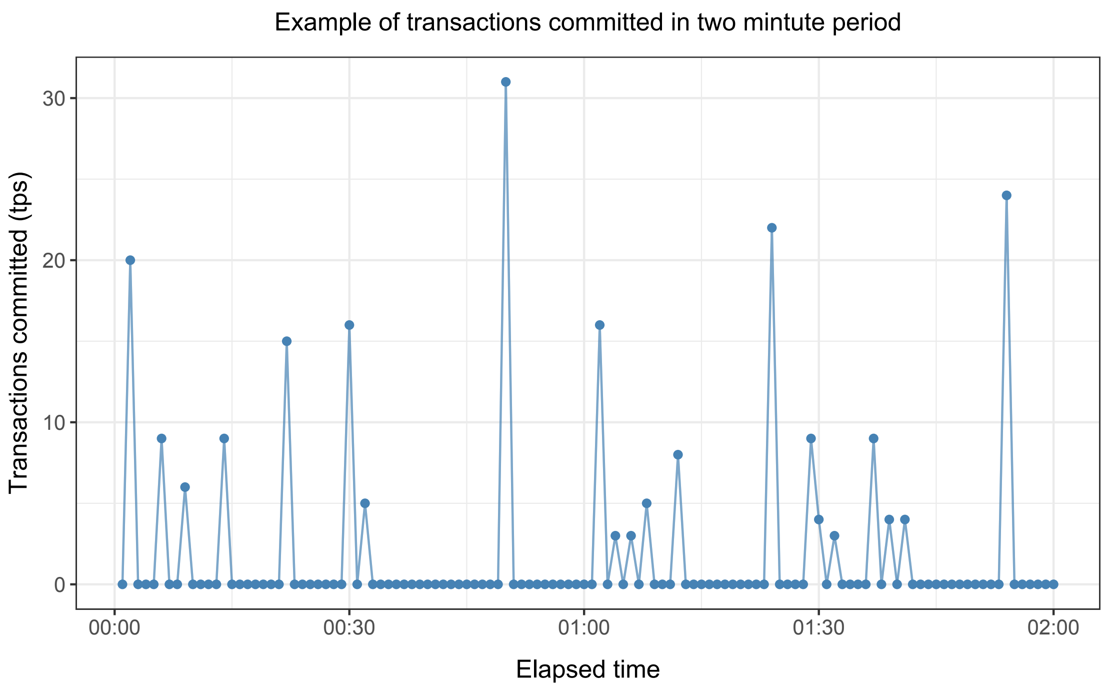


Figure 26: Example of transactions committed in two-minute period (target tps: 4)

Variability in the rate of committed transactions is also apparent in the histograms showing the number of transactions committed per second for each target tps (see Figure 27 and Figure 28). For readability, intervals with negative or zero number of transactions are not shown on the histograms.

Figure 27 shows that while the mean for single transaction batches ranges from between approximately 1.4 and 5.0 tps, there were up to 100 transactions committed to the blockchain in some one-second periods. This number coincides with the default maximum number of batches per block. The default setting for maximum batches per block was left unchanged in the initial configuration, though it can be changed by submitting a *settings* transaction.

For single transaction batches, the maximum number of transactions per block is equivalent to the maximum number of batches per block, as there exists a one-to-one relationship. Therefore, it's reasonable to assume from the histograms shown in Figure 27 that transactions submitted in batches of one were added to blocks containing up to the maximum of 100 batches allowed by the default setting.

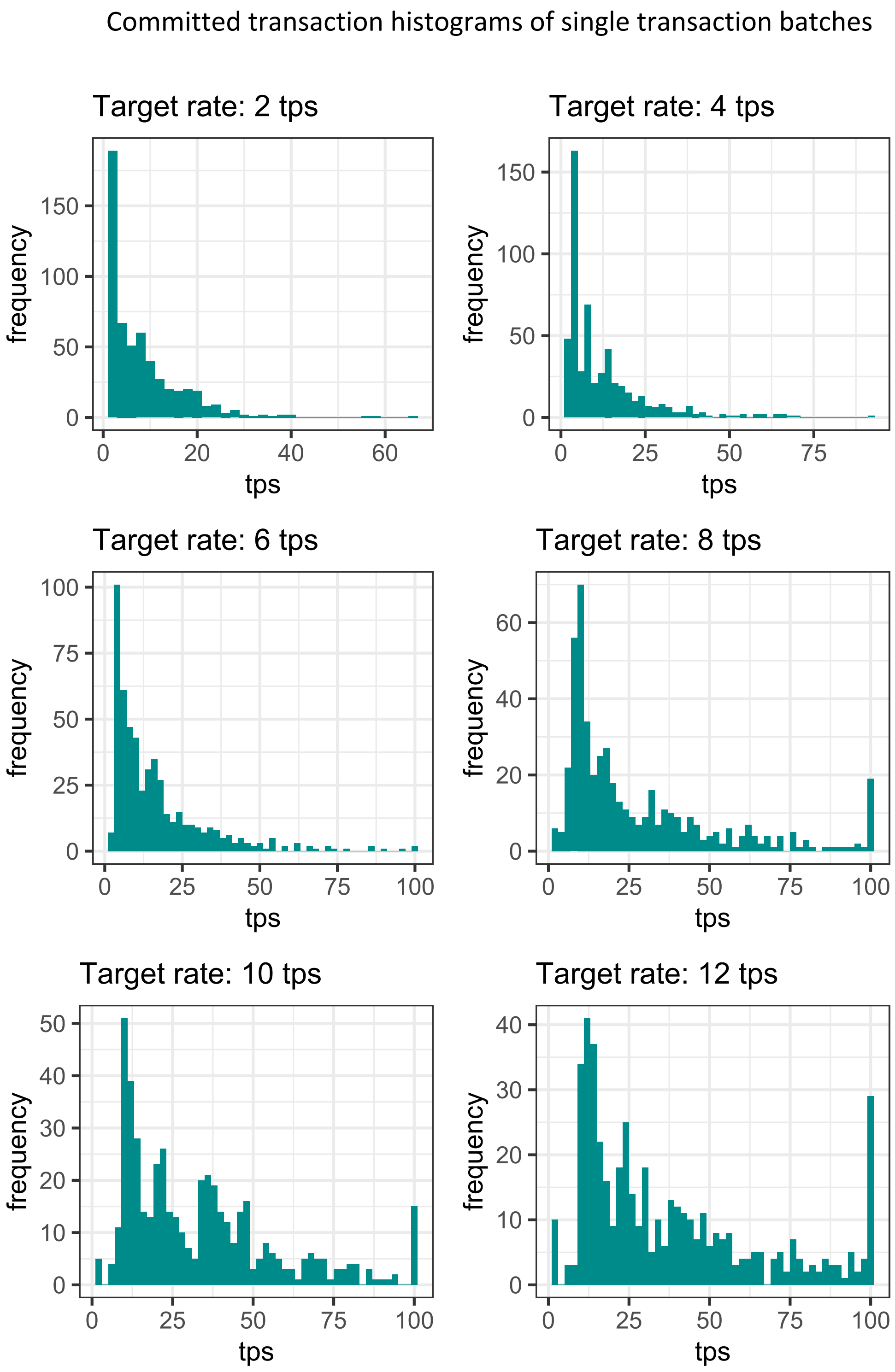


Figure 27: Committed transaction rates for single transaction batches

For multiple transaction batches, up to 520 transactions were committed in some one second intervals. Multiple transaction batches, with 10 transactions per batch, can have up to 1,000 transactions per block with the default setting for the maximum number of batches per block. The higher number of transactions per block for multiple transaction batches will have contributed to the higher variance for multiple transaction batches observed in the results in Table 7.

Both sets of histograms also display a shift from higher frequencies of low rates of tps, to lower frequencies of more varied rates of tps, as the target tps increased. The pattern can be observed in both single transaction batch inputs and multiple transaction batch inputs and will also have contributed to the increased variance as target tps increased.

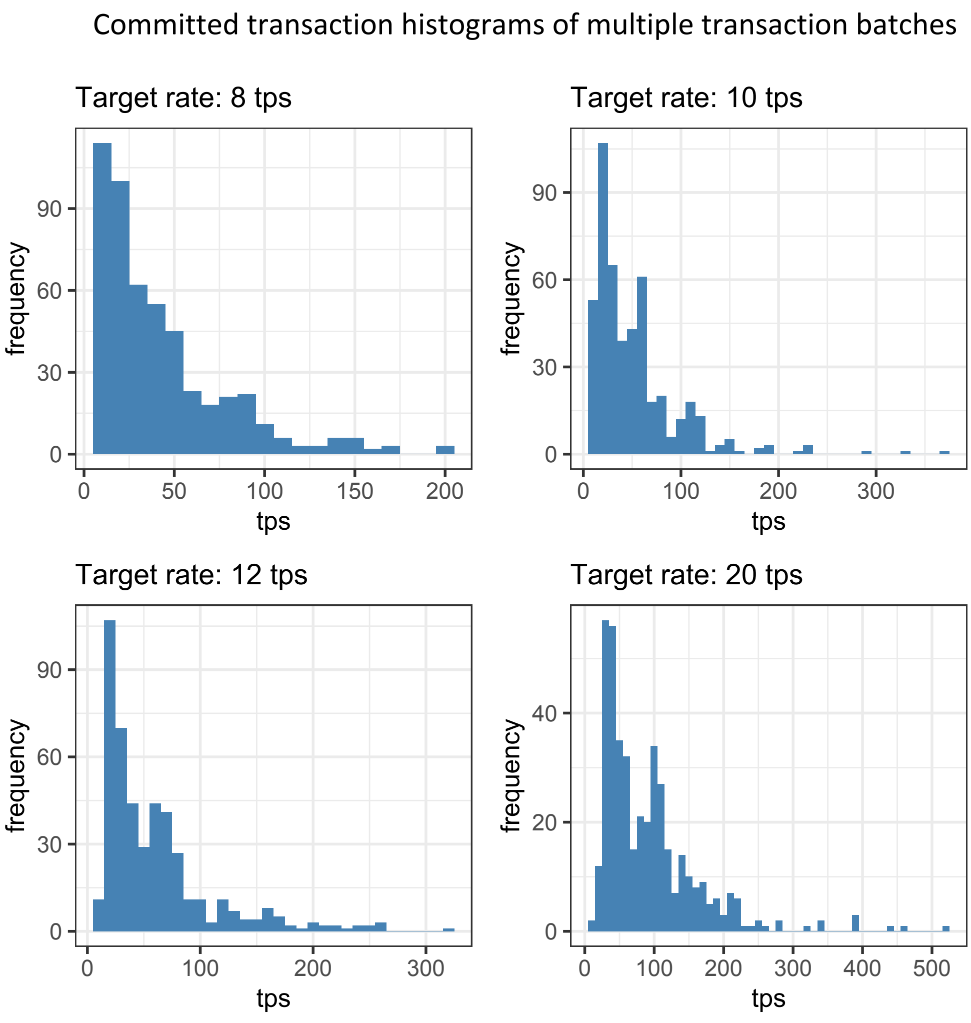


Figure 28: Committed transaction rates for multiple transaction batches

4.2.2 Transactions per block

It was observed in the results presented in Table 7 that the variance for the number of committed transactions per second increased as the rate of target tps increased. It was also observed from Figure 27 that transactions were committed at a rate up to the maximum permitted for the default setting for maximum number of batches per block.

The next step is to explore further the relationship between blocks and transactions. There is no data available that specifically reports on the number of transactions per block, but it can be inferred by comparing the number of blocks and the number of transactions, both of which are available separately.

The results for the total number of blocks and mean number of transactions per block are presented in Table 8. They show that the mean number of transactions per block increased as the target tps increased. The increase in mean number of transactions per block coincides with increases in the standard deviation seen in Table 7.

The results also show that the total number of blocks decreased in absolute terms with increases in the target tps, though the number was relatively stable throughout. Interestingly, the number of blocks was consistent between the single and multiple transaction batches, e.g., at a target tps of 10, there were 475 blocks for the single transaction batches versus 473 for the multiple transaction batches (see Figure 29).

The mean number of transactions per block was higher for multiple transaction batches than it was for single transaction batches. Given that the number of blocks was relatively constant for single and multiple batch transactions at a given target tps, the higher mean transactions per block for multiple transaction batches reflects the higher throughput achieved.

Target tps	Transactions per batch	Total blocks	Mean transactions per block
2	1	550	8.945
4	1	538	12.827
6	1	518	17.264
8	1	483	28.424
10	1	475	34.238
12	1	464	38.841
8	10	503	42.604
10	10	473	51.628
12	10	455	60.400
20	10	415	92.682

Table 8: Total blocks and mean transactions per block

Results presented in Table 7, Figure 27 and Figure 28 indicate that the high variance for committed transactions was driven in some part by the variance in the number of transactions per block. This raised the question if reducing the maximum number of transactions per block may help to reduce the variance, by forcing the network to commit transactions more frequently in smaller blocks.

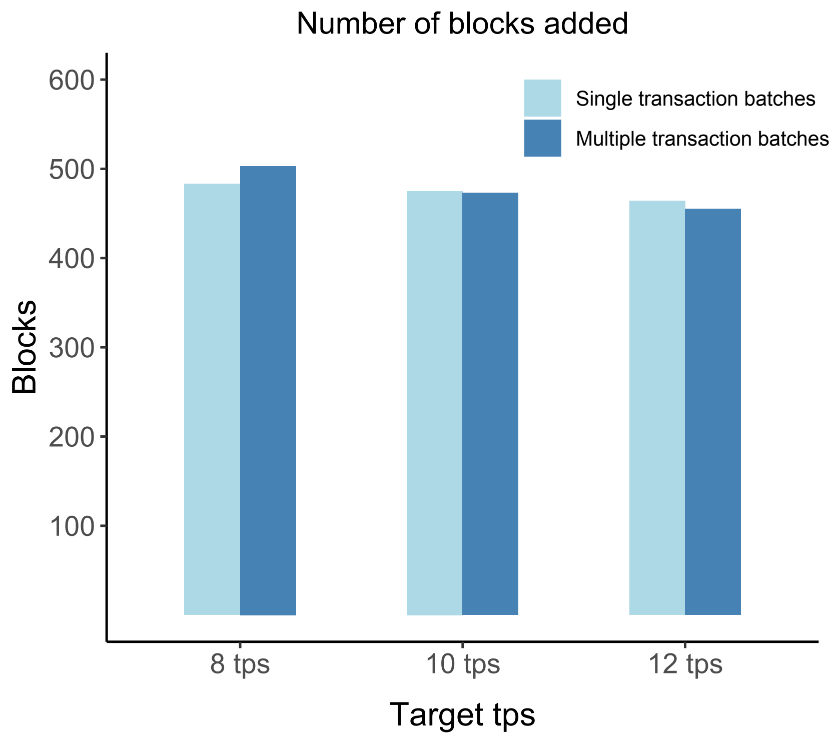


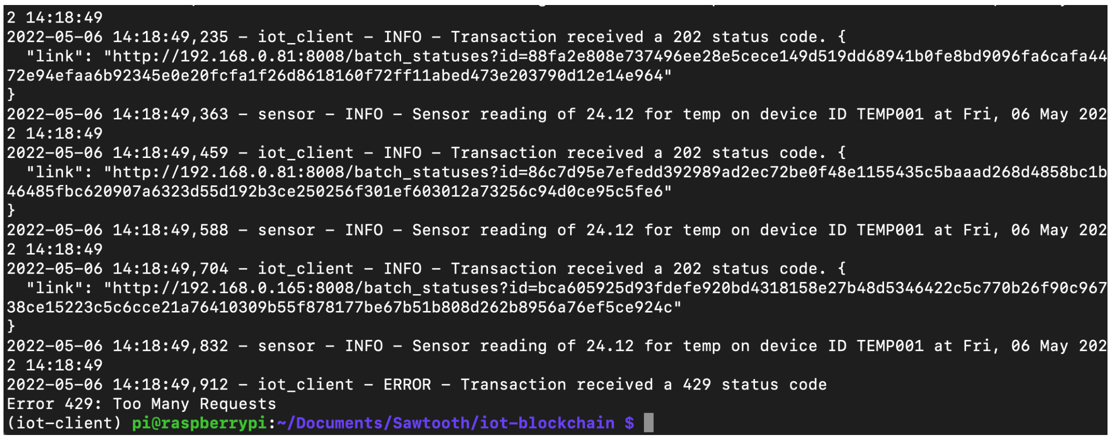
Figure 29: Blocks committed for single and multiple transaction batches

To evaluate if this approach could offer an effective solution for reducing variance, a series of follow up experiments were performed with the maximum number of batches per block set at 10, and the results are presented as Table 9. For all but three of the experiments, failure occurred after the network became overwhelmed when the same number of input transactions were submitted at the lower maximum batches per block. In many cases, failure occurred within a few minutes of starting the experiment.

Target tps	Transactions per batch	Mean tps	Standard deviation
2	1	Error	-
4	1	Error	-
6	1	Error	-
8	1	Error	-
10	1	Error	-
12	1	Error	-
8	10	5.894	18.411
10	10	6.717	20.918
12	10	7.736	23.483
20	10	Error	-

Table 9: Experiment results with reduced maximum batches per block

This type of failure can occur when batches are submitted faster than they can be processed, and they accumulate as pending batches. If the number of pending batches reaches a certain threshold, then batches will be rejected to reduce pressure on the network, in a feature known as back-pressure[16] in Sawtooth. This is a security measure that helps to prevent Distributed-Denial-of-Service (DDoS) attacks. Figure 30 shows a HTTP 429 status code indicating the error at the IoT client. This error halted execution of the IoT client in the current network design.



```
2 14:18:49
2022-05-06 14:18:49,235 - iot_client - INFO - Transaction received a 202 status code. {
  "link": "http://192.168.0.81:8008/batch_statuses?id=88fa2e808e737496ee28e5cece149d519dd68941b0fe8bd9096fa6cafa44
72e94efaa6b92345e0e20fcfa1f26d8618160f72ff11abed473e203790d12e14e964"
}
2022-05-06 14:18:49,363 - sensor - INFO - Sensor reading of 24.12 for temp on device ID TEMP001 at Fri, 06 May 202
2 14:18:49
2022-05-06 14:18:49,459 - iot_client - INFO - Transaction received a 202 status code. {
  "link": "http://192.168.0.81:8008/batch_statuses?id=86c7d95e7efedd392989ad2ec72be0f48e1155435c5baaad268d4858bc1b
46485fbc620907a6323d55d192b3ce250256f301ef603012a73256c94d0ce95c5fe6"
}
2022-05-06 14:18:49,588 - sensor - INFO - Sensor reading of 24.12 for temp on device ID TEMP001 at Fri, 06 May 202
2 14:18:49
2022-05-06 14:18:49,704 - iot_client - INFO - Transaction received a 202 status code. {
  "link": "http://192.168.0.165:8008/batch_statuses?id=bca605925d93fdefe920bd4318158e27b48d5346422c5c770b26f90c967
38ce15223c5c6cce21a76410309b55f878177be67b51b808d262b8956a76ef5ce924c"
}
2022-05-06 14:18:49,832 - sensor - INFO - Sensor reading of 24.12 for temp on device ID TEMP001 at Fri, 06 May 202
2 14:18:49
2022-05-06 14:18:49,912 - iot_client - ERROR - Transaction received a 429 status code
Error 429: Too Many Requests
(iot-client) pi@raspberrypi:~/Documents/Sawtooth/iot-blockchain $
```

Figure 30: HTTP status code 429 received at IoT client

The results suggest that when the maximum batches per block is set to 10, the network can only reliably process around one batch per second, as this is the rate of batch submission when the target tps is between 8 and 12 and the batch size is 10 transactions per batch. Even at 2 batches per second the experiment failed relatively quickly, as is the case for a target tps of 2 or greater in a single transaction batch and for a target tps of 20 in the 10 transactions batch.

Another interesting result is that for the experiments that were successful, there is no real observable improvement in the mean tps compared to the equivalent target tps with the default setting for maximum batches per block, as can be seen in Figure 31. However, there is some reduction in variance as measured by the standard deviation. Future studies could evaluate performance with different maximum batches per block to find the optimal configuration for this setting.

[16] https://sawtooth.hyperledger.org/faq/rest.html#what-is-the-back-pressure-test

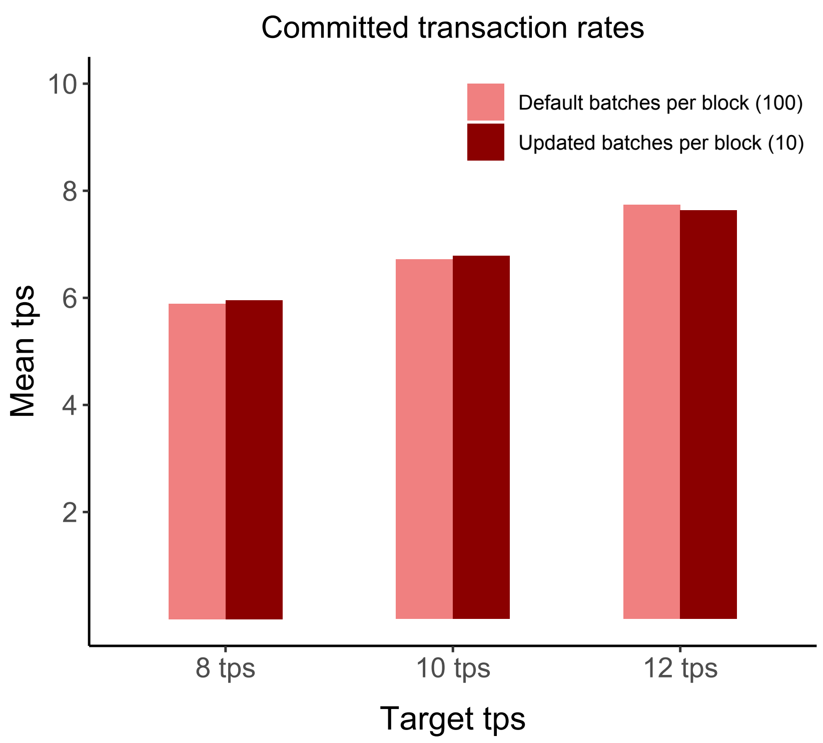


Figure 31: Committed transaction rates comparison of default and updated minimum batches per block

4.3 Hypothesis test

The highest mean tps of any experiment was achieved when the target tps was 20 with a batch size of 10 and using the default setting for maximum batches per block (see Table 7). The data obtained during this experiment will be used to test the hypothesis that Sawtooth can process IoT data at a rate of at least 10 transactions per second.

A one-sample *t*-test was selected to test if the mean of the committed transactions was statistically significantly greater than a hypothesised mean. The mean of the hypothesised distribution in this scenario is 10 transactions per second (tps). Testing the hypothesis with a one-sample *t*-test requires several assumptions to be fulfilled. Normally distributed data is one of the assumptions.

However, it is clear from the density plot shown in Figure 32 (a) that the assumption of data being normally distributed is not valid for the committed transactions data. Normality of the data was also shown to be unlikely by performing a Shapiro-Wilk test in R.

The high number of one-second intervals where zero transactions are being committed is a significant contributor to the non-normality of the data. This can be seen on the density plot as the very prominent peak that is centred around zero committed transactions. Attempts to transform the data, using a range of methods, including square-root and log transforms, did not produce a distribution that was normal or even approximately normal. An example of the log transformed committed transactions data is shown as Figure 32 (b).

Without the assumption of normality, an alternate approach to test the hypothesis was required. The approach selected was the central limit theorem. The central limit theorem states that for sufficiently large random samples, taken (with replacement) from a population with mean μ and variance σ^2, the distribution of the sample means will approach normality (Kwak & Kim, 2017). This occurs regardless of the distribution of the parent population.

The central limit theorem was applied to the committed transactions data by taking 5,000 samples of size 300, resulting in a distribution of sample means that is shown in Figure 32 (c). The distribution of sample means appears approximately normal, and with a mean of 10.69 tps it is very close to the true population mean of 10.68 tps.

To improve the normality of the data even further, a square root transformation was applied to the central limit theorem data before completing the hypothesis test (see Figure 32 (d)). The transformed data was checked using a Shapiro-Wilk test which provided strong evidence that the data was normally distributed.

Applying the one-sample *t*-test to the transformed data obtained a *p* value less than the significance level of $\alpha = 0.05$, meaning we reject the null hypothesis and conclude that it is plausible that if Hyperledger Sawtooth blockchain distributed ledger is implemented on an edge computing server, then it will process and securely store internet-of-things (IoT) data transactions, in the form of temperature readings from a sensor connected to a Raspberry Pi computer and transmitted via REST API, at a rate of at least 10 transactions per second.

4.4 Discussion of results

The results support the alternate hypothesis and answer the research question posed at the outset. A possible limitation was having to use a target rate of 20 tps to achieve a rate of throughput at the level required to reject the null hypothesis. The decision to apply this methodology was made on the basis that 20 was the *target* and not the actual rate of submission. It follows that the actual rate is equivalent to the mean tps, which has achieved the level required to reject the null hypothesis.

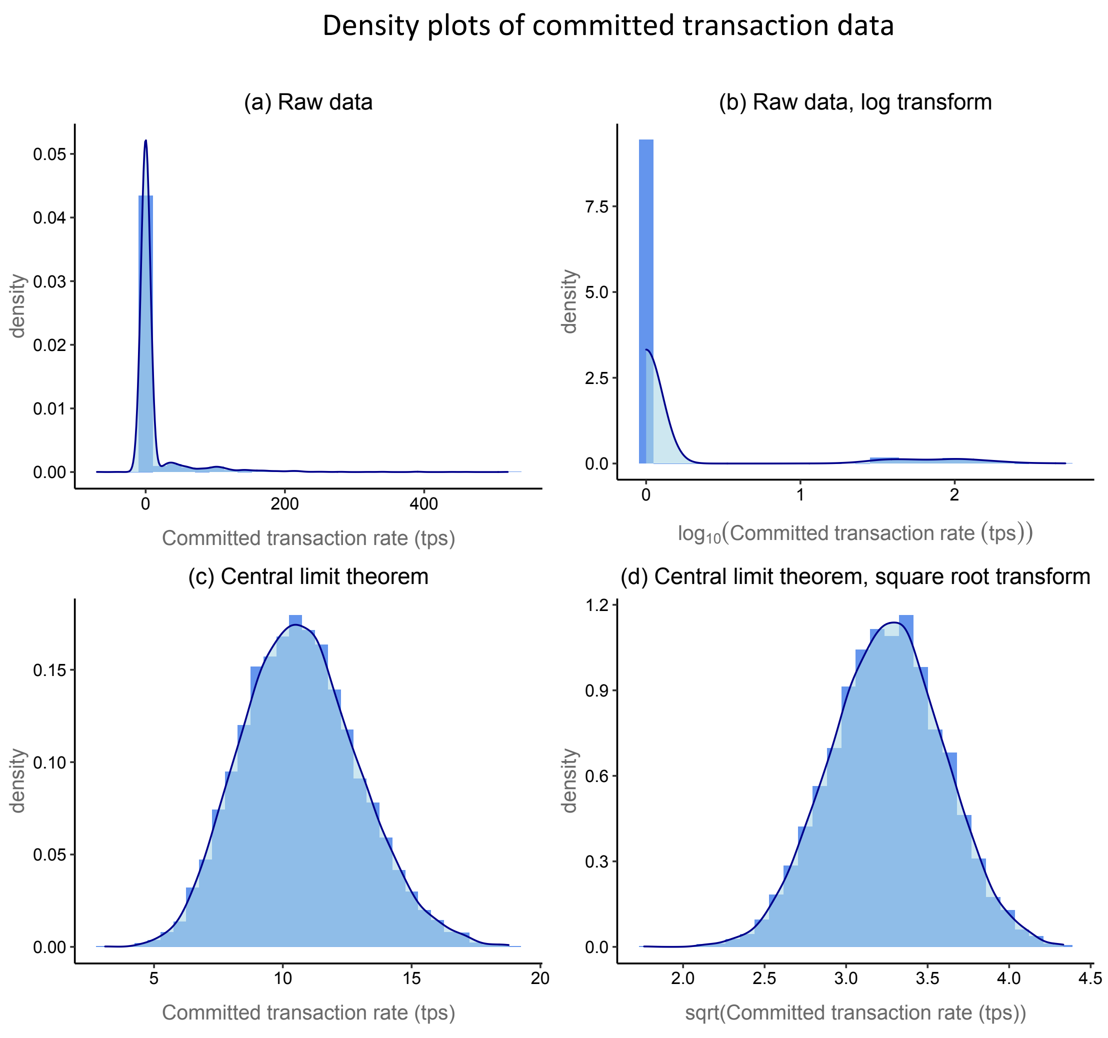


Figure 32: Density plots of committed transaction data

This gap between the actual rate of transaction throughput and the target level of throughput was one of the main findings in the results. Another finding was the very high level of variance in the rate at which transactions were committed to the blockchain. Each of these findings will now be discussed further.

4.4.1 Transaction throughput

The first of the additional findings to be discussed in detail, was the failure of the mean tps to achieve the level specified in the target tps. This is a potential limitation of the current network design, and possibly of Hyperledger Sawtooth itself, that requires a better understanding of why it occurred to propose possible solutions and directions for future research.

Some of the difference between target tps and mean tps was due to the design of the network and the method for setting the rate of input transaction submission at the IoT client. A time delay was specified at run time by providing an interval at the IoT client through a command line argument, where the interval was a programmed delay between submitting successive transactions, e.g., for 10 transactions per second the interval was 100 milliseconds.

The input transaction submission interval did not leave room for additional processing times within any of the software components or latencies in the REST API response. This will have contributed to a lower mean tps relative to the target tps rate due to the total round-trip time of sending a transaction being longer than the specified interval between transactions.

Differences between target tps and mean tps could also be due to internal delays within the Sawtooth components, including the consensus mechanism. Delays in the Sawtooth components or consensus mechanism would represent a more binding constraint for achieving a target level of performance.

Changing the design of the IoT client to submit transactions asynchronously may remove some of the delays due to the REST API response latencies. Sawtooth SDKs are available for JavaScript that may offer a better solution for asynchronously posting the HTTP requests. This approach could be applied in future studies aimed at improving and optimising performance. Such a redesign would help highlight the extent to which delays were occurring internally in the Sawtooth components by removing any possible latencies arising from the REST API.

Another redesign that could be explored in future work would be to use a lower-level messaging protocol to submit transactions. Sawtooth natively supports the ZMQ messaging protocol for this task. This solution is more complex than using the REST API but offers a lighter-weight and more efficient and robust solution[17].

One approach to reduce the delta between target tps and actual tps that was evaluated during the experiments was to batch together multiple transactions before submitting them to the REST API. This approach results in fewer round trips per transaction. The effect of increasing the number of transactions per batch is visible in

[17] https://sawtooth.hyperledger.org/docs/1.2/architecture/rest_api.html

Figure 25, where the results indicate that multiple transaction batches do achieve a higher mean tps compared to single transaction batches, for a given target tps.

However, increasing the number of transactions per batch does also present some potential risks to overall performance. Multiple transaction batches incur a higher cost if a batch is rejected by a timeout resulting from an excess of pending batches, as all transactions in the batch will be rejected together.

More transactions per batch also increases latency in the network by increasing the time for some transactions to be committed, as the first transaction added to the batch is not submitted to the REST API until the last transaction has been added to the batch. For example, if transactions are added to a 10 transactions batch, at the rate of one per second, then at least 10 seconds will pass from the time of adding the first transaction until the last transaction is added to complete the batch and send to the REST API.

4.4.2 Performance variability

Variability in performance was another finding of the experiment results. This can be seen in Table 7 where it presents as the large standard deviations relative to the corresponding means, and throughout Figure 27 and Figure 28, where it presents as a long tail in the histogram distributions. Variance was also observed during the experiments, where it presented as unevenness in the performance of the network. This unevenness, which has been discussed already, presents as pauses and bursts of activity in the rate of committed transactions, visible through the console logging messages and on the Grafana dashboard.

Under ideal conditions, a linear rate of input transactions would have an approximately corresponding linear rate of transactions being committed to the blockchain. Such linearity may be unrealistic in real-world conditions, as latencies and bottlenecks occur at various stages of processing. However, even allowing for latency, it is reasonable to expect that the blockchain should achieve a steady state of transaction throughput that has a linear relationship with the rate of input transactions.

Attempts to reduce the amount of variance by forcing the transactions into smaller blocks resulted in failure for most rates of input transaction, though the experiments that completed successfully did have some reduction in their standard deviations compared to the initial results. Repeating the experiment in a higher resource environment may improve performance for smaller blocks but that goes against the goal of using low-powered edge computing resources. Ultimately, it seems that variance at higher rates of input transaction is a characteristic of Hyperledger Sawtooth as other studies have shown the same finding (Shi et al., 2019).

High variance creates a limitation on the types of applications and use cases for Sawtooth at the business solution layer. For non-time critical applications, with lower throughputs, Sawtooth may be an appropriate blockchain platform to provide a secure log of IoT data transactions. On the other hand, for time sensitive applications and use cases with higher throughputs, then Sawtooth may not be an appropriate blockchain platform, at least not with the current network design.

4.5 Summary

Results were presented for multiple experiments across a range of input transaction rates and for different configurations. These results provided some interesting findings and insights on how transactions are committed to the blockchain.

One finding was the failure of the network to support the specified rate of input transactions, as measured by the target tps. This finding relates more to the design of the network rather than the performance of Sawtooth.

Another finding was the variability in the rate at which transactions were committed to the blockchain. Such variance has been observed in other studies (Shi et al., 2019), indicating that this finding relates more to an inherent performance characteristic of Sawtooth. Attempts to reduce the variance by reducing the number of transactions per block were largely unsuccessful.

A one-sample *t*-test was applied to the committed transaction data to test the hypothesis. Application of the central limit theorem and a square root transform was required to fulfil the assumption of normally distributed data that is required for a one-sample *t*-test.

The result of the test supports the hypothesis that a Hyperledger Sawtooth blockchain distributed ledger implemented on an edge computing server will process and securely store internet-of-things (IoT) data transactions, in the form of temperature readings from a sensor connected to a Raspberry Pi computer and transmitted via REST API, at a rate of at least 10 transactions per second.

5. CONCLUSION

This final chapter will begin with a summary of the research from the literature review, before restating the problem definition that framed the research question at the outset of the work. This will be followed with an overview of the results and findings from an experiment designed to address the research problem, including the implications and limitations. Finally, the chapter will offer recommendations for future work.

5.1 Research overview

The research conducted in the literature review focused on three main elements that were integrated into the research question and hypothesis test. These three elements were IoT security, blockchain applications for IoT security, and edge computing in the context of IoT security and blockchain.

To begin with, the research focused on gaining an understanding of the security challenges facing IoT and the existing proposed solutions, which includes blockchain. Blockchain has been shown to offer a feasible solution for IoT security (Dorri et al., 2017), but limitations have been encountered in the rate of transaction throughput due to the time taken for the consensus mechanism to approve and add transactions to a new block (Huh et al., 2017).

The research then explored how the transaction throughput limitations can potentially be overcome using private blockchains with more efficient consensus mechanisms. One such private blockchain is Hyperledger Sawtooth, which has shown promising results in performance benchmarks compared to other blockchain platforms (Rasolroveicy & Fokaefs, 2020).

Finally, proposals for integrating IoT and blockchain with edge computing were explored to establish the benefits, and to understand the patterns and reference architectures that have previously been applied. A proposed architecture consisting of an IoT layer, an edge layer, and a business solution layer (Sittón-Candanedo et al., 2019), was selected as a blueprint for the experiment design.

5.2 Problem definition

A gap existed in the literature for a performance evaluation of Hyperledger Sawtooth in a practical application of blockchain for IoT data security that included a detailed design and a solution architecture. A practical application using real data collected by real sensors and submitted over a physical network provides better generalisation of results compared to experiments that use test transactions.

Results obtained in an evaluation of a practical application can be taken as more indicative of real-world performance than results obtained from simulated networks and test transactions. As such, a practical application serves as a better indicator of performance for development of use cases, especially when combined with a detailed design and solution architecture that potentially can be scaled.

5.3 Experimentation, evaluation & results

To address the research problem, an experiment was designed for an edge computing implementation of Hyperledger Sawtooth blockchain to store data from an IoT device. The edge computing network consisted of a MacBook Air laptop representing the edge server and a Raspberry Pi computer connected to a temperature sensor representing the IoT device. The edge server hosted three Sawtooth blockchain nodes on three virtual machines. Communication between the IoT device and the blockchain nodes on the edge server was over a home Wi-Fi network using Sawtooth's REST API.

Experiments were run over a range of input transaction rates and with different configurations to evaluate performance under different workloads. Data for each of the experiments was captured in an InfluxDB instance on one of the virtual machines. After the experiments were complete, data was exported to R for analysis, visualisation, and hypothesis testing.

The minimum rate of throughput required to reject the null hypothesis was at least 10 transactions per second. The hypothesis test was performed using a one-sample *t*-test at an $\alpha = 0.05$ level of significance. Prior to testing the hypothesis, the central limit theorem and a square root transformation was applied to the results data. This was required to achieve the assumption of normality required for the one-sample *t*-test.

The outcome of the hypothesis test indicates that if Hyperledger Sawtooth blockchain distributed ledger is implemented on an edge computing server, then it will process and securely store IoT data transactions, in the form of temperature readings from a sensor connected to a Raspberry Pi computer and transmitted via REST API, at a rate of at least 10 transactions per second.

The level of throughput observed in the results would support many use cases, such as a smart home environment, where IoT devices could communicate with the centralised edge server to store and exchange data securely, e.g., the temperature data could be accessed by the heating system as a thermostatic control.

However, results also showed a high level of variance in the rate of transaction throughput that appears to be characteristic of Sawtooth as similar studies have also reported this finding (Shi et al., 2019). Variance in throughput was the result of intermittency in the rate at which transactions were committed to the blockchain. This variance could be an issue for some use cases and applications at the business solution layer that depend on a steady rate of data availability.

Unsuccessful attempts were made to reduce the amount of variance by altering the configuration to add fewer batches and hence fewer transactions per block. Using this approach, the network quickly became overwhelmed and experienced a queue timeout leading to the rejection of transactions. The few experiments that did complete successfully did show a slight reduction in variance but no overall improvement in throughput.

There were also some limitations in achieving the desired target rate of transaction throughput from the IoT client. This was due to the round-trip time of submitting a transaction via the Sawtooth REST API being longer than anticipated. Typically, the actual rate achieved for throughput was about half the target rate. Some or most of this delay is related to the design of the experiment and cannot be attributed to the performance of Sawtooth itself.

Crucially, every transaction that was submitted to the REST API was either committed or was reported as an error in the cases where the minimum transaction per batch was lowered leading to failure of the experiment. The importance of this cannot be overstated in a series of experiments where 100s of thousands of transactions were submitted by the IoT client. In this regard, it can be concluded that the reliability of Sawtooth and the network design was very high.

Results and findings must be viewed within the context of the edge computing implementation that was designed for the experiments. The host machine representing the centralised edge server for the Sawtooth network was a basic specification MacBook Air laptop. This edge server was running three blockchain nodes as that is the minimum number required by the PoET consensus mechanism.

This is a heavy workload for the host machine as each blockchain node was running within its own virtual machine (VM). If the network was extended to three or more edge servers in a larger area blockchain network, then each edge server would need to run only a single blockchain node, thereby reducing the workload on each edge server.

A final point relates to the setup and configuration of both the software and hardware components, which was a significant effort. The Sawtooth documentation available online was to a high standard but there were some gaps. As Sawtooth is somewhat of a niche software application, the user community is small and there are relatively few resources available for support and discussion. Troubleshooting during installation and operation was a challenge requiring long hours of testing, investigation, and debugging.

5.4 Contribution and Impact

The experiments conducted have addressed a gap in the existing literature to provide a performance evaluation of a practical application of Hyperledger Sawtooth for IoT data security. Notwithstanding the limitations that have been discussed, the outcome of the hypothesis test indicates that an edge computing implementation of Sawtooth can achieve a throughput rate of at least 10 transactions per second in the experiment that was designed and evaluated.

Performance was evaluated using a physical IoT device, consisting of a Raspberry Pi connected to a temperature sensor, sending real data over a REST API to the Sawtooth nodes on a basic specification MacBook Air that acted as a centralised edge server. This improves the generalisability of the results compared to other performance assessments of Sawtooth that applied test cases only. It also extends the scope of Sawtooth applications for IoT to just about any environment that has an internet connection and some basic hardware.

A detailed analysis on the sources of variance in transaction throughput was presented alongside the experiment results, something which wasn't provided in other studies of Sawtooth. Understanding the sources of variance paves the way for future efforts to reduce it and achieve a more stable performance.

The detailed network and experiment design provides a scalable reference architecture and blueprint for future development of use cases and applications that can utilise the data stored on the blockchain. Design improvements have also been suggested, notably in relation to reducing the round-trip time of submitting transactions.

5.5 Future Work and Recommendations

Further work will be required to understand the limitations and other observations that were not fully explored in the results as they were not within scope. One observation that was noted but not explored was the forking of the blockchain that occurred in all experiments. Forking could have implications on performance at the business solution layer and would benefit from further investigation and understanding before use cases and applications are developed.

Another observation that was noted but not fully explored was the occasional reduction in the number of committed transactions on individual blockchain nodes. This may well have been related to forking of the blockchain and self-correcting mechanisms to ensure no duplicate transactions were added in blocks. Comfort is taken from the fact that the experiments concluded with the expected number of transactions committed on each blockchain node. However, it is worth understanding the reasons why this occurred before proceeding to development of use cases and applications.

Limitations relating to achieving the target level of throughput could potentially be overcome by redesigning some of the features, such as by sending transactions to Sawtooth's REST API asynchronously to reduce waiting time on HTTP responses. This redesign should be evaluated and benchmarked against the existing design to see if it offers any improvement. Sawtooth transactions can also be submitted using the ZMQ protocol, and this could also be included in future benchmarking performance evaluation of transaction submission.

Results showed that reducing the batches per block does reduce performance variance slightly, though the setting that was evaluated in the experiments was perhaps too ambitious and caused the network to quickly get overwhelmed. Further analysis should be performed to find the optimal configuration in terms of batch size and batches per block to find a balance between minimising variance while avoiding overwhelming the network with too many batches.

Given the initial success of the experiments conducted in this study, use cases and applications at the business solution layer can be prototyped, tested, and evaluated, in parallel to work addressing the limitations and observations discussed. The network for these prototypes should be scaled to encompass a wider area of multiple sites, with each site a separate blockchain node. Each site or blockchain node can in turn have multiple IoT devices. Performance of the network with multiple nodes supporting multiple IoT devices can then be re-evaluated to measure throughput for scaled applications.

A final recommendation to the Hyperledger developers and community would be to maintain support through the available channels for troubleshooting and debugging. Community support is crucial for a niche application like Sawtooth as other learning resources are very limited. While resources were often hard to find, the quality of documentation and resources that were available from official channels was of an excellent standard. Some gaps exist so regular updates will be required to ensure these are addressed and closed off.

BIBLIOGRAPHY

Alfandi, O., Khanji, S., Ahmad, L., & Khattak, A. (2021). A survey on boosting IoT security and privacy through blockchain. *Cluster Computing 24(1)*, 37-55.

Alladi, T., Chamola, V., Parizi, R. M., & Choo, K.-K. R. (2020). Blockchain applications for industry 4.0 and industrial IoT: A review. *IEEE Access 7*, 176935-176951.

Alladi, T., Chamola, V., Sikdar, B., & Choo, K.-K. (2020). Consumer IoT: Security vulnerability case studies and solutions. *IEEE Consumer Electronics Magazine, 9(2)*, 17-25.

Amanullah, M. A., Habeeb, R. A., Nasaruddin, F. H., Gani, A., Ahmed, E., Nainar, A. S., . . . Imran, M. (2020). Deep learning and big data technologies for IoT security. *Computer Communications 151* , 495-517.

Ampel, B., Patton, M., & Chen, H. (2019, July). Performance modeling of hyperledger sawtooth blockchain. *2019 IEEE International Conference on Intelligence and Security Informatics (ISI)* (pp. 59-61). IEEE.

Baralla, G., Pinna, A., & Corrias, G. (2019, May). Ensure traceability in European food supply chain by using a blockchain system. *2019 IEEE/ACM 2nd International Workshop on Emerging Trends in Software Engineering for Blockchain (WETSEB)* (pp. 40-47). IEEE.

Dorri, A., Kanhere, S. S., Jurdak, R., & Gauravaram, P. (2017). Blockchain for IoT Security and Privacy: The Case Study of a Smart Home. *2017 IEEE international conference on pervasive computing and communications workshops (PerCom workshops)* (pp. 618-623). IEEE.

Elrawy, M. F., Awad, A. I., & Hamed, H. F. (2018). Intrusion detection systems for IoT-based smart environments: a survey. *Journal of Cloud Computing 7, (1)*, 1-20.

Gadekallu, T. R., Pham, Q. V., Nguyen, D. C., Maddikunta, P. K., Deepa, N., Prabadevi, B., . . . Hwang, W. J. (2021). Blockchain for edge of things: applications, opportunities, and challenges. *IEEE Internet of Things Journal 9, no. 2*, 964-988.

Hassija, V., Chamola, V., Saxena, V., Jain, D., Goyal, P., & Sikdar, B. (2019). A survey on IoT security: application areas, security threats, and solution architectures. *IEEE Access 7*, 82721-82743.

Huh, S., Cho, S., & Kim, S. (2017). Managing IoT devices using blockchain platform. *2017 19th international conference on advanced communication technology (ICACT)* (pp. 464-467). IEEE.

Iftekhar, A., Cui, X., Tao, Q., & Zheng, C. (2021). Hyperledger fabric access control system for internet of things layer in blockchain-based applications. *Entropy, 23(8)*, 1054.

Kromes, R., Gerrits, L., & Verdier, F. (2019, October). Adaptation of an embedded architecture to run Hyperledger Sawtooth Application. *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)* (pp. 0409-0415). IEEE.

Kullig, N., Lämmel, P., & Tcholtchev, N. (2020). Prototype Implementation and Evaluation of a Blockchain Component on IoT Devices. *Procedia Computer Science 175* , 379-386.

Kwak, S., & Kim, J. (2017). Central limit theorem: the cornerstone of modern statistics. *Korean journal of anesthesiology, 70(2)*, 144-156.

Malik, A. A., Tosh, D. K., & Ghosh, U. (2019, June). Non-intrusive deployment of blockchain in establishing cyber-infrastructure for smart city. *2019 16th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)* (pp. 1-6). IEEE.

Misra, S., Mukherjee, A., Roy, A., Saurabh, N., Rahulamathavan, Y., & Rajarajan, M. (2020). Blockchain at the edge: Performance of resource-constrained IoT networks. *IEEE Transactions on Parallel and Distributed Systems, 32(1)*, 174-183.

Monrat, A. A., Schelén, O., & Andersson, K. (2020). Performance Evaluation of Permissioned Blockchain Platforms. *2020 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)* (pp. 1-8). IEEE.

Moschou, K., Theodouli, A., Terzi, S., Votis, K., Tzovaras, D., Karamitros, D., & Diamantopoulos, S. (2020, November). Performance Evaluation of different Hyperledger Sawtooth transaction processors for Blockchain log storage with varying workloads. *2020 IEEE International Conference on Blockchain (Blockchain)* (pp. 476-481). IEEE.

Nakamoto, S. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System.* Retrieved from bitcoin.org: https://bitcoin.org/bitcoin.pdf

Pahl, C., Helmer, S., Miori, L., Sanin, J., & Lee, B. (2016). A container-based edge cloud paas architecture based on raspberry pi clusters. *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)* (pp. 117-124). IEEE.

Pavithran, D., Shaalan, K., Al-Karaki, J. N., & Gawanmeh, A. (2020). Towards building a blockchain framework for IoT. *Cluster Computing, 23(3)*, 2089-2103.

Rasolroveicy, M., & Fokaefs, M. (2020, July). Performance evaluation of distributed ledger technologies for iot data registry: A comparative study. *2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4)* (pp. 137-144). IEEE.

Sedlmeir, J., Buhl, H. U., Fridgen, G., & Keller, R. (2020). The Energy Consumption of Blockchain Technology: Beyond Myth. *Business & Information Systems Engineering, 62(6)*, 599-608.

Shi, Z., Zhou, H., Hu, Y., Jayachander, S., de Laat, C., & Zhao, Z. (2019, June). Operating permissioned blockchain in clouds: A performance study of hyperledger sawtooth. *019 18th International Symposium on Parallel and Distributed Computing (ISPDC)* (pp. 50-57). IEEE.

Sittón-Candanedo, I., Alonso, R. S., Corchado, J. M., Rodríguez-González, S., & Casado-Vara, R. (2019). A review of edge computing reference architectures and a new global edge proposal. *Future Generation Computer Systems, 99*, 278-294.

Wang, S., Yuan, Y., Wang, X., Li, J., Qin, R., & Wang, F.-Y. (2018). An Overview of Smart Contract: Architecture, Applications, and Future Trends,. *2018 IEEE Intelligent Vehicles Symposium (IV)* (pp. 108-113). IEEE.

Yang, R., Yu, F. R., Si, P., Yang, Z., & Zhang, Y. (2019). Integrated blockchain and edge computing systems: A survey, some research issues and challenges. *IEEE Communications Surveys & Tutorials, 21(2)*, 1508-1532.