



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

A Thesis

For the Degree of Doctor of Philosophy

Transaction Traffic Control Based on Fuzzy Logic for
Improving Performance in Blockchain Network

Lei Hang

Department of Computer Engineering

Graduate School

Jeju National University

August 2020

Transaction Traffic Control Based on Fuzzy Logic for Improving
Performance in Blockchain Network

Lei Hang

(Supervised by professor Do-Hyeun Kim)

A thesis submitted in partial fulfillment of the requirement for the degree of Doctor of Philosophy in Computer Science

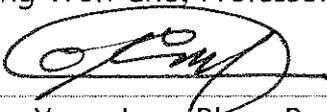
2020. 08.

This thesis has been examined and approved.


Thesis Committee Chair, See-Kyun Kim, Professor, Jeju National University


Thesis Committee Member, Nam Je Park, Professor, Jeju National University


Thesis Committee Member, Jung Won Cho, Professor, Jeju National University


Thesis Committee Member, Yoon Jung Rhee, Professor, Jeju National
University


Thesis Supervisor, Do-Hyeun Kim, Professor, Jeju National University

August 2020

Department of Computer Engineering
GRADUATE SCHOOL

*Dedicated to my families for being a constant source
of support and encouragement!*

Acknowledgment

First of all, I would like to express my gratitude to all those who helped me during the writing of this thesis. I gratefully acknowledge the help of my supervisor, Prof. Do-Hyeun Kim, who has offered me valuable suggestions in the academic studies. In the preparation of this thesis, he has spent much time reading through each draft and provided me with inspiring advice. Without his patient instruction, insightful criticism, and expert guidance, the completion of this thesis would not have been possible.

Second, I would like to thank my thesis evaluation committee for their insightful comments and valuable suggestions during the thesis defense. Their input helped me in elevating the quality of this thesis.

Last, I appreciate the support and encouragement given to my current lab mates Wenquan Jin, Rongxu Xu, Azimbek Khudoyberdiev, Faisal Jamil, Shabir Ahmad, Imran Jamal, Naeem Iqbal, DongHyun Oh and to former lab mates Dr. Israr Ullah, Dr. Muhammad Fayaz, Dr. Sehrish Malik, Faisal Mehmood for their kind support and help during this endeavor.

To all of you, a heartfelt thanks for everything you did!

August 2020

Lei Hang

Table of Contents

Abstract	1
1. Introduction	5
2. Related Work.....	10
2.1 Blockchain Technology and Platform	10
2.2 Existing Studies of Performance Improvement in Blockchain Network	16
2.3 Performance Evaluation Architecture in Blockchain Network	25
3. Performance Analysis of Configurable Parameters in Blockchain Network	27
3.1 Blockchain Network Performance Metrics.....	27
3.2 Configurable Parameters in Blockchain Network	28
3.3 Experiment Environment in Blockchain Network.....	30
3.4 Performance Evaluation of Configurable Parameters in Blockchain Network	31
4. Transaction Traffic Control Mechanism based on Fuzzy Logic in Blockchain Network	48
4.1 Proposed Transaction Traffic Control Mechanism based on Fuzzy Logic.....	48
4.2 Development of the Transaction Traffic Control Mechanism based on Fuzzy Logic	59
4.3 Performance Analysis of the Transaction Traffic Control Mechanism based on Fuzzy Logic	63
5. Transaction Traffic Control Mechanism based on Learning to Prediction in Blockchain Network	77

5.1	Proposed Transaction Traffic Control Mechanism based on Learning to Prediction	77
5.2	Development of the Transaction Traffic Control Mechanism based on Learning to Prediction	88
5.3	Performance Analysis of the Transaction Traffic Control Mechanism based on Learning to Prediction	96
6.	Performance Evaluation of the Proposed Approach in Clinical Trial Testbed	110
6.1	Clinical Trial Testbed Environment for Blockchain Performance Evaluation	110
6.2	Experiment Environment of Clinical Trial Testbed	126
6.3	Evaluation Results of the Proposed Approach in Clinical Trial Testbed	132
6.3.1	Evaluation Results of the Optimized Network in Clinical Trial Testbed	132
6.3.2	Evaluation Results of the Fuzzy Logic in Clinical Trial Testbed	135
6.3.3	Evaluation Results of the Learning to Prediction in Clinical Trial Testbed	141
7.	Conclusion	152
	References	155

List of Figures

Figure 1: The basic structure of the blockchain.....	11
Figure 2: Blockchain platform transaction workflow (e.g., Hyperledger Fabric)	16
Figure 3: Endorsement channel function and generation process in endorser peer.....	18
Figure 4: Optimized ordering service process in blockchain network.....	19
Figure 5: Conceptual architecture of Nexledger Accelerator	20
Figure 6: An example of transactions processing with cache.....	21
Figure 7: BFT-smart ordering service architecture.....	22
Figure 8: Optimized validation phase for Hyperledger Fabric	23
Figure 9: The behavior of two new efficient API functions.....	24
Figure 10: Performance evaluation framework architecture of blockchain using Hyperledger Caliper	26
Figure 11: Evaluation of the impact of block size on transaction throughput	32
Figure 12: Evaluation of the impact of block size on transaction latency.....	33
Figure 13: Evaluation of the impact of block frequency on transaction throughput	34
Figure 14: Evaluation of the impact of block frequency on transaction latency	34
Figure 15: Evaluation of the impact of the use of TLS on transaction throughput	35
Figure 16: Evaluation of the impact of the use of TLS on transaction latency	36
Figure 17: Evaluation of the impact of the ledger database on transaction throughput.....	37
Figure 18: Evaluation of the impact of the ledger database on transaction latency	37
Figure 19: Evaluation of the impact of the ordering service on transaction throughput....	38
Figure 20: Evaluation of the impact of the ordering service on transaction latency	39
Figure 21: Evaluation of the impact of the programming language of smart contract on transaction throughput.....	40

Figure 22: Evaluation of the impact of the programming language of smart contract on transaction latency	40
Figure 23: Evaluation of the impact of the number of clients on transaction throughput..	41
Figure 24: Evaluation of the impact of the number of clients on transaction latency	42
Figure 25: Evaluation of the impact of the number of endorser peers on transaction throughput	43
Figure 26: Evaluation of the impact of the number of endorser peers on transaction latency	43
Figure 27: Evaluation of the impact of the number of organizations on transaction throughput	44
Figure 28: Evaluation of the impact of the number of organizations on transaction latency	45
Figure 29: Evaluation of the impact of the endorsement policy on transaction throughput	46
Figure 30: Evaluation of the impact of the endorsement policy on transaction latency.....	47
Figure 31: Conceptual architecture of the transaction traffic control mechanism based on fuzzy logic	49
Figure 32: Development configuration of the transaction traffic control mechanism based on fuzzy logic	50
Figure 33: Block diagram of the network configuration	51
Figure 34: Flow chart of the transaction traffic control mechanism based on fuzzy logic..	56
Figure 35: Sequence diagram of the blockchain network configuration	58
Figure 36: Sequence diagram of the transaction traffic control mechanism based on fuzzy logic.....	59
Figure 37: Fuzzy set definition in the smart contract	61
Figure 38: Fuzzy rule definition in the smart contract.....	62

Figure 39: Function to invoke fuzzy inference system in the smart contract	63
Figure 40: Sample of transaction control policy in terms of the acceptance rate	63
Figure 41: Evaluation of transaction throughput in 1org2peer network with 1 client (baseline and fuzzy logic).....	64
Figure 42: Evaluation of transaction latency in 1org2peer network with 1 client (baseline and fuzzy logic).....	65
Figure 43: Evaluation of transaction throughput in 2org1peer network with 1 client (baseline and fuzzy logic)	66
Figure 44: Evaluation of transaction latency in 2org1peer network with 1 client (baseline and fuzzy logic).....	66
Figure 45: Evaluation of transaction throughput in 2org2peer network with 1 client (baseline and fuzzy logic).....	67
Figure 46: Evaluation of transaction latency in 2org2peer network with 1 client (baseline and fuzzy logic).....	68
Figure 47: Evaluation of transaction throughput in 3org2peer network with 1 client (baseline and fuzzy logic).....	69
Figure 48: Evaluation of transaction latency in 3org2peer network with 1 client (baseline and fuzzy logic).....	69
Figure 49: Evaluation of transaction throughput in 1org2peer network with 5 clients (baseline and fuzzy logic).....	70
Figure 50: Evaluation of transaction latency in 1org2peer network with 5 clients (baseline and fuzzy logic).....	71
Figure 51: Evaluation of transaction throughput in 2org1peer network with 5 clients (baseline and fuzzy logic).....	72
Figure 52: Evaluation of transaction latency in 2org1peer network with 5 clients (baseline and fuzzy logic).....	72

Figure 53: Evaluation of transaction throughput in 2org2peer network with 5 clients (baseline and fuzzy logic).....	73
Figure 54: Evaluation of transaction latency in 2org2peer network with 5 clients (baseline and fuzzy logic).....	74
Figure 55: Evaluation of transaction throughput in 3org2peer network with 5 clients (baseline and fuzzy logic).....	75
Figure 56: Evaluation of transaction latency in 3org2peer network with 5 clients (baseline and fuzzy logic).....	76
Figure 57: Conceptual architecture of the transaction traffic control mechanism based on learning to prediction.....	78
Figure 58: Development configuration of the transaction traffic control mechanism based on learning to prediction	79
Figure 59: Overview architecture of the learning to prediction module	80
Figure 60: Flow chart of the transaction throughput prediction using Kalman Filter	81
Figure 61: Detailed diagram of the learning to prediction module	83
Figure 62: Flow chart of the transaction traffic control based on learning to prediction... 	85
Figure 63: Sequence diagram of the transaction traffic control mechanism based on learning to prediction.....	86
Figure 64: 4-fold cross-validation model for training and testing data	89
Figure 65: Function of ANN in the learning to prediction module	92
Figure 66: Function of Kalman Filter in the learning to prediction module.....	93
Figure 67: Function of transaction traffic control in the learning to prediction module	94
Figure 68: Learning to prediction module main interface	95
Figure 69: Execution results of the learning to prediction module	96
Figure 70: Evaluation of transaction throughput in 1org2peer network with 1 client (baseline and learning to prediction).....	97

Figure 71: Evaluation of transaction latency in 1org2peer network with 1 client (baseline and learning to prediction).....	98
Figure 72: Evaluation of transaction throughput in 2org1peer network with 1 client (baseline and learning to prediction).....	99
Figure 73: Evaluation of transaction latency in 2org1peer network with 1 client (baseline and learning to prediction).....	99
Figure 74: Evaluation of transaction throughput in 2org2peer network with 1 client (baseline and learning to prediction).....	100
Figure 75: Evaluation of transaction latency in 2org2peer network with 1 client (baseline and learning to prediction).....	101
Figure 76: Evaluation of transaction throughput in 3org2peer network with 1 client (baseline and learning to prediction).....	102
Figure 77: Evaluation of transaction latency in 3org2peer network with 1 client (baseline and learning to prediction).....	102
Figure 78: Evaluation of transaction throughput in 1org2peer network with 5 clients (baseline and learning to prediction).....	103
Figure 79: Evaluation of transaction latency in 1org2peer network with 5 clients (baseline and learning to prediction).....	104
Figure 80: Evaluation of transaction throughput in 2org1peer network with 5 clients (baseline and learning to prediction).....	105
Figure 81: Evaluation of transaction latency in 2org1peer network with 5 clients (baseline and learning to prediction).....	105
Figure 82: Evaluation of transaction throughput in 2org2peer network with 5 clients (baseline and learning to prediction).....	106
Figure 83: Evaluation of transaction latency in 2org2peer network with 5 clients (baseline and learning to prediction).....	107

Figure 84: Evaluation of transaction throughput in 3org2peer network with 5 clients (baseline and learning to prediction).....	108
Figure 85: Evaluation of transaction latency in 3org2peer network with 5 clients (baseline and learning to prediction).....	109
Figure 86: Layer-based system architecture of the clinical trial service platform.....	111
Figure 87: Service scenario of clinical trial service based on blockchain	112
Figure 88: System configuration of the blockchain-based clinical trial service	114
Figure 89: The workflow of the blockchain-based clinical trial service.....	115
Figure 90: Execution process of the proposed blockchain-based clinical trial service	116
Figure 91: Blockchain network topology of the clinical trial service	118
Figure 92: Sample code of the smart contract in clinical trial service	120
Figure 93: Sample code of the clinical trial service server	121
Figure 94: Sample code of the REST API in clinical trial service	122
Figure 95: Implementation results of clinical trial service	125
Figure 96: Blockchain network configuration in clinical trial service (configtx.yaml)	127
Figure 97: Blockchain network credentials configuration in clinical trial service (crypto-config yaml)	128
Figure 98: Blockchain network container configurations in clinical trial service (docker-compose-cli yaml).....	129
Figure 99: Blockchain network benchmark configuration in clinical trial service.....	130
Figure 100: Blockchain network initialization in clinical trial service.....	131
Figure 101: Performance testing results in clinical trial service.....	132
Figure 102: Evaluation of transaction throughput with one client (baseline and optimized network).....	133
Figure 103: Evaluation of transaction latency with one client (baseline and optimized network).....	134

Figure 104: Evaluation of transaction throughput with five clients (baseline and optimized network).....	134
Figure 105: Evaluation of transaction latency with five clients (baseline and optimized network).....	135
Figure 106: Evaluation of transaction throughput with 1 client (baseline and fuzzy logic).....	136
Figure 107: Evaluation of transaction latency with 1 client (baseline and fuzzy logic).....	136
Figure 108: Evaluation of transaction throughput with 5 clients (baseline and fuzzy logic).....	137
Figure 109: Evaluation of transaction latency with 5 clients (baseline and fuzzy logic).....	138
Figure 110: Evaluation of transaction throughput with one client (optimized network and fuzzy logic).....	139
Figure 111: Evaluation of transaction latency with one client (optimized network and fuzzy logic).....	139
Figure 112: Evaluation of transaction throughput with five clients (optimized network and fuzzy logic).....	140
Figure 113: Evaluation of transaction latency with five clients (optimized network and fuzzy logic).....	141
Figure 114: Evaluation of transaction throughput with one client (baseline and learning to prediction).....	142
Figure 115: Evaluation of transaction latency with one client (baseline and learning to prediction).....	143
Figure 116: Evaluation of transaction throughput with five clients (baseline and learning to prediction).....	144
Figure 117: Evaluation of transaction latency with five clients (baseline and learning to prediction).....	144

Figure 118: Evaluation of transaction throughput with 1 client (optimized network and learning to prediction) 145

Figure 119: Evaluation of transaction latency with 1 client (optimized network and learning to prediction) 146

Figure 120: Evaluation of transaction throughput with 5 clients (optimized network and learning to prediction) 147

Figure 121: Evaluation of transaction latency with 5 clients (optimized network and learning to prediction) 147

List of Tables

Table 1: Comparison of some well-known blockchain platforms.....	12
Table 2: Comparison of different types of blockchain platforms	13
Table 3: Configurable parameters that affect the blockchain performance	29
Table 4: Default configuration for the experiment unless otherwise stated	31
Table 5: Fuzzy set definition in smart contract.....	53
Table 6: Fuzzy rules definition in smart contract	54
Table 7: Development environment of transaction traffic control based on fuzzy logic	60
Table 8: Fuzzy rules definition for learning to prediction	84
Table 9: Development environment of transaction traffic control based on learning to prediction	87
Table 10: Dataset for the learning to prediction module.....	88
Table 11: RMSE for different configurations using the 4-fold cross-validation model.....	89
Table 12: Defined transactions in the smart contract.....	119
Table 13: Defined assets in the smart contract.....	119
Table 14: Sample REST API endpoints in clinical trial service	124
Table 15: Experiment parameter configurations.....	126
Table 16: Comparison analysis of the performance evaluation.....	149
Table 17: Comparison analysis of the performance evaluation with Accelerator	151

Abbreviations

TPS	Transaction Per Second
RPS	Read Per Second
IoT	Internet of Things
AI	Artificial Intelligence
MVCC	Multi-version Concurrency Control
VSCC	Verification System Chaincode
MSP	Membership Service Providers
CA	Certificate Authority
SDK	Software Development Kit
BFT	Byzantine Fault Tolerant
PBFT	Practical Byzantine Fault Tolerance
API	Application Programmers Interface
P2P	Peer to Peer
REST	Representational State Transfer
CRA	Clinical Research Associate
PI	Principal Investigator
CRC	Clinical Research Coordinator
CRO	Contract Research Organization

BGM	Blood Glucose Meter
FLC	Fuzzy Logic Control
ANN	Artificial Neural Networks
RMSE	Root Mean Square Error
LAN	Local Area Network
WAN	Wide-Area-Network
IT	Information Technology
AWS	Amazon Web Services
JSON	JavaScript Object Notation
PoW	Proof of Work
LAN	Local Area Network
SUT	System Under Test

Abstract

The blockchain technology is known as an innovation that powers the cryptocurrency Bitcoin. Blockchain is a potential technology for migrating the processing burden from the central server into a decentralized, secure, and transparent manner. This technology is expected to make a significant impact and lead to a revolution in various types of industries. However, one issue holding them back is their limited transaction throughput, especially compared to established solutions such as distributed database systems. Transaction per second (tps) is one of the performance indexes widely used to evaluate the transaction processing capability of applications in enterprise use cases. Many well-known blockchain platforms such as Bitcoin, Ethereum have been widely adopted into different application domains such as the Internet of Things (IoT), supply chain, healthcare, etc. So far, there has been much confusion about whether the blockchain performs with scale, and admittedly, a lack of information about best practices that can improve the performance and scale. Besides, more analysis and evaluation of the performance of these platforms are urgent.

In this paper, we propose two transaction traffic control approaches using fuzzy logic to improve the blockchain performance, such as increasing the transaction throughput while reducing transaction latency. Besides, this paper conducts a comprehensive evaluation of various configurable parameters that can affect network performance. For the first approach, we propose the transaction traffic control based on a fuzzy controller in the smart contract. The proposed fuzzy controller is implemented in the smart contract that makes to adjust the transaction traffic flow according to the network conditions collected in real-time. For the second approach, we propose additional learning to prediction module to enhance the performance of the fuzzy controller in the smart contract. The learning to prediction is implemented in an external module, which is

extensible enough to adopt different algorithms. To demonstrate the significance of the proposed transaction traffic control approaches, we apply these two approaches into a blockchain-based clinical trial testbed from our previous work. Lastly, we deploy the proposed approaches with existing performance-enhancing tools, and the results indicate that our approaches are flexible and scalable to be used with any other projects.

To be more precise, configurable parameters can be mainly summarized into two categories: software and hardware. For the case of software, it contains various parameters, including block size, block frequency, ledger database, ordering service, the programming language of smart contract, use of Transport Layer Security (TLS), number of clients, number endorser peers, number of organizations, and the endorsement policy. For the case of hardware, it contains the number of vCPUS, memory allocation, disk type and speed, network speed, and CPU speed. We evaluated each configurable parameter to analyze the impact on network performance. For example, the blockchain performance can be significantly affected by increasing the number of peers and the use of TLS since more messages are generated within the entire network.

For the first transaction traffic control approach, we deploy the fuzzy controller into the smart contract to regulate the transaction traffic across the network automatically. The fuzzy controller is comprised of the fuzzy inference system and the transaction control module. We observe the blockchain network benchmark results in real-time, and these values are passed to the smart contract. These benchmark results are used as the input variables of the fuzzy controller, and the control command is computed to perform different transaction control operations on the received transactions accordingly.

For the second transaction traffic control approach, we extend the architecture of the fuzzy controller by introducing additional learning to prediction module. We use the Kalman Filter as the prediction module to estimate the actual transaction throughput. In contrast, the Artificial Neural Network (ANN) is used as the learning module to tune the parameter of the Kalman Filter.

The dataset for the learning to prediction is the performance benchmark results collected in one week. Besides, we perform a 4-fold cross-validation test on the training and testing data to select the best model for the ANN algorithm.

We set up an optimized network in terms of the evaluation results of each configurable parameter to achieve better performance. For example, we set the block size to 30 transactions per block, block frequency to 250ms, the number of clients to 5, and use OR endorsement policy to achieve higher transaction throughput and lower transaction latency. We evaluated the performance of the proposed approaches by applying them into a clinical trial testbed from the previous work using different performance metrics. The results of the proposed approaches are compared with two other schemes: the baseline scheme and an optimized scheme. The evaluation results show that the proposed approaches can enhance network performance compared to the baseline and optimized schemes.

For the case of baseline scheme using one client, the network with optimized configurable parameters increases the transaction throughput by 19.6% and decrease the transaction latency by 20% compared to the baseline. The transaction throughput is increased by 27.6% and 29.7%; the transaction latency is decreased by 41.5% and 47.7% with the fuzzy logic and learning to prediction approaches, respectively.

For the case of the network with optimized configurable parameters using one client, the transaction throughput is increased by 10.9% and 13.7%; the transaction latency is decreased by 40.4% and 46.2% with the fuzzy logic and learning to prediction approaches, respectively. For the case of baseline scheme using five clients, the network with optimized configurable parameters increases the transaction throughput by 9.3% and decrease the transaction latency by 17.4% compared to the baseline network. The transaction throughput is increased by 21.8% and 26.5%; the transaction latency is decreased by 26.8% and 40% with the fuzzy logic and learning to prediction approaches, respectively. For the case of the network with optimized configurable

parameters using five clients, the transaction throughput is increased by 13.7% and 18.4%; the transaction latency is decreased 17% and 36.6% with the fuzzy logic and learning to prediction approach.

Furthermore, the proposed approaches are deployed with one of the existing performance-enhancing tools called Accelerator. For the case of using one client, the network with Accelerator increases the transaction throughput by 67.2% and decrease the transaction latency by 35.4% compared to the baseline. The transaction throughput is increased by 77.7% and 80.7%; the transaction latency is decreased by 52.3% and 56.9% with the fuzzy logic and learning to prediction approaches, respectively. For the case of using five clients, the network with Accelerator increases the transaction throughput by 82.1%and decreases the transaction latency by 20% compared to the baseline. The transaction throughput is increased by 96.9% and 99.9%; the transaction latency is decreased by 37.9% and 43.6% with the fuzzy logic and learning to prediction approaches, respectively. The results indicate that the proposed approaches are flexible enough to integrate with other approaches.

1. Introduction

Distributed ledger technologies such as blockchain offer a way to conduct transactions in a secure and verifiable manner without the need for a trusted third party [1]. Blockchain technology is progressively turning into in style, with applications in varied domains like finance, supply chains, real estate, etc. A blockchain is a distributed ledger of transactions, which is maintained by all the taking part nodes of the blockchain network. The transactions represent the business logic and are formed into a chain of blocks that are attached to the ledger. Every node within the network updates its copy of the ledger with the new block when the agreement is reached amongst the nodes.

As such, it is widely believed that blockchain will significantly impact industries ranging from finance and real estate to public administration, energy, and transportation [2]. However, to be viable in practice, blockchain must support transaction rates comparable to those supported by existing database management systems, which can provide some of the same transactional guarantees. Performance is considered as one of the main challenges in adopting blockchain implementations as an alternative to current centralized servers [3]. The inefficient transaction processing capability and lack of standardization can put an obstacle in the development of the blockchain, such as limited scalability, throughput bottleneck, transaction latency, and storage constraints [4]. For example, the block size of Bitcoin is limited to 1 MB, and a new block is created every 10 minutes. Subsequently, the Bitcoin network is restricted to a rate of 7 transactions per second, which is incapable of dealing with high-frequency trading. Besides, a transaction of bitcoin needs six affirmations before it is confirmed that it can take around an hour on average. The transaction confirmation time of Ethereum takes around 15 seconds; however, the average time would increment exponentially as indicated by differed network situations. In a permissionless blockchain like Bitcoin and Ethereum, anybody is permitted to take part, and each

member is mysterious. This implies neither can there be a privacy of the agreements themselves, nor of the exchange information that they procedure.

All in all, these platforms issue their tokens to incent exorbitant mining or to fuel transaction execution to relieve the absence of privacy. The transaction cost and speed can be altogether influenced by a negative relationship with the use of native digital currencies. Furthermore, it hinders the cooperation with other decentralized platforms, as the token utilized in both platforms must be consistent.

Due to its relatively poor performance, many observers do not believe that blockchain technology is suitable for large-scale applications. In contrast to permissionless blockchains, which do not restrict network membership, we focus on permissioned blockchains, in which the identities of all participating nodes are known. A permissioned blockchain provides a way to secure the interactions among a group of entities that have a common goal but which may not fully trust each other. By relying on the identities of the participants, a permissioned blockchain can use more traditional crash fault tolerant (CFT) or byzantine fault tolerant (BFT) consensus protocols that do not require costly mining.

Recent developments in blockchain technology are creating new opportunities for artificial intelligence (AI) applications [5]. AI technologies could help solve many blockchain challenges. For instance, there is always a supervisor who is responsible for determining whether the contract condition is satisfied. In the case of self-driving, AI technologies can be applied in the smart contract to make autonomous decisions in terms of the rules of the road and traffic laws to limit misbehaviors made by driverless cars. The authors in [6] discussed how the integration of AI and blockchain could help in developing a new decentralized ecosystem. Besides, multiple use cases of AI applications and implementations utilizing blockchain are discussed, covering intelligent transportation [7], intelligent precision farming [8], supply chain industry [9], swarm robotic system [10], etc. However, most of the existing blockchain-enabled AI applications only utilize

the blockchain infrastructure to provide users with qualitatively new data models, shared control of AI training data and models, and leads to improved trustworthiness on data. None of these studies explore the use of AI to enhance the blockchain performance.

To the best knowledge of the author, this paper is the first attempt to discuss how the use of AI techniques can help in improving the network performance of blockchain. This paper introduces two transaction traffic control approaches based on fuzzy logic. For the first approach, we implement the fuzzy controller in the smart contract. For the second approach, we implement additional learning to prediction module to enhance the performance of the fuzzy controller in the smart contract. The former mechanism implements a fuzzy controller specified to control the transaction traffic flow according to network feedback collected in real-time. The fuzzy controller is implemented in the smart contract to automate the process of transaction traffic control without third-party intervention. The latter mechanism extends the structure of the first mechanism by introducing additional learning to prediction module. We use the Kalman Filter as the prediction model to estimate the transaction throughput and use the artificial neural network (ANN) as the learning model to tune the parameter of the Kalman Filter. The learning to prediction module estimates the transaction throughput according to the real-time network conditions. The predicted transaction throughput is used as the fuzzy input for the fuzzy controller in the smart contract to automate the process of transaction traffic control. In this paper, we implement the learning to prediction module as an external module since it requires time-consuming and challenging training that can tax the smart contract.

Additionally, we perform a comprehensive experiment for various configurable network parameters, which have an impact on blockchain performance. These configurable parameters can be mainly summarized into two categories: software and hardware. For the case of software, it contains various parameters, including block size, block frequency, ledger database, ordering service, the programming language of smart contract, use of TLS, number of clients, number

endorser peers, number of organizations, and the endorsement policy. For the case of hardware, it contains the number of vCPUS, memory allocation, disk type and speed, network speed, and CPU speed. According to the evaluation results, optimized values of each parameter are selected to form an optimized network to achieve optimum performance.

We deploy the proposed transaction traffic control mechanisms in a clinical trial testbed from our previous work. The clinical trial testbed is built on a permissioned network, where the identities of all the network participants are known to each other. The results of the proposed approaches are compared with two other schemes: the baseline scheme and an optimized scheme. The evaluation results indicate that the proposed transaction traffic control mechanisms can significantly improve transaction throughput while decreasing network latency compared to the baseline and optimized schemes. We also apply the design approaches with one of the existing performance-enhancing tools, and the results show that our approaches are flexible enough to be with existing studies. The contributions of this paper are discussed as follows:

- **Novelty:** This paper introduces the use of machine learning techniques to enhance the blockchain performance. The fuzzy controller is defined in the smart contract to automate the transaction traffic control. The proposed system also supports the interaction with the external machine learning module. Learning to prediction module is proposed in this paper to improve the performance of the fuzzy controller further.
- **Universality:** This paper considers various configurable network parameters that can affect the blockchain performance. A comprehensive experiment is performed to analyze the impact of each configurable parameter on the blockchain performance. According to the evaluation results, an optimized network is set up to achieve better network performance. Although some of the parameters may be specified to particular blockchain platforms, this experiment points out a clear indicator to evaluate the performance of the blockchain network.

- Scalability: The proposed system is implemented in a modular and extensible architecture, which supports an interface to interact with different external machine learning modules. It can also interact with the existing performance-enhancing tool without much modification of the original system.
- Usability: The proposed mechanisms have been tested in a case study of the clinical trial using a permissioned network. Many other smart contracts enabled blockchain platforms can also potentially profit by the significance of this work.

The remainder of this paper is organized as follows, Chapter 2 gives a brief of the blockchain technology, and blockchain platforms analyze some of the related studies that focus on improving the blockchain performance and introduces blockchain performance benchmark tool. Chapter 3 introduces the performance metrics, overviews configurable parameters that affect blockchain performance, and illustrates the evaluation results of each configurable parameter. Chapter 4 presents the proposed transaction traffic control mechanism based on fuzzy logic, provides the details of the implementation of the proposed mechanism, and evaluates the performance. Chapter 5 presents the proposed transaction traffic control mechanism based on learning to prediction, provides the details of the implementation of the proposed mechanism, and evaluates the performance. Chapter 6 describes the clinical trial testbed, which is used as the simulation environment to evaluate the proposed approaches and illustrates the evaluation results. Finally, Chapter 7 concludes the paper and discusses future research directions.

2. Related Work

2.1 Blockchain Technology and Platform

Figure 1 shows the structure of blockchain at an abstract level. Blockchain is an entirely secure, decentralized info comprised of various peers, which provides storage to record data from an outsized form of entities [11]. It is a series of linked blocks of transferred data between different connected nodes that form the network of the blockchain. Transparency is recognized as one of the most important characteristics of blockchain since it enables instant access to data since it is replicated on all nodes without interventions of third parties. The blockchain technology has been widely applied to hatch practical use cases in varied application fields [12], like intelligent transport system [13], health sector [14, 15, 16], distributed applications [17, 18], and prediction platforms [19, 20].

In the case of permissionless blockchain, there is no centralized authority, and no party has more power than the remainder. Here every member seems to be hospitable, be a part of or leave as they want. The blockchain is publicly open, and everybody has the proper to validate a group action. In the case of Bitcoin [22], one of the best examples of a permissionless network, it is the miners who can validate the transaction. They get bitcoins within the sort of transaction validation fees, and therefore the new bitcoins are generated for the effort they place into solving the puzzle problem. Ethereum [23] is incredibly almost like Bitcoin; however, it differs in many aspects. As an example, it is a versatile programming language for smart contracts [24].

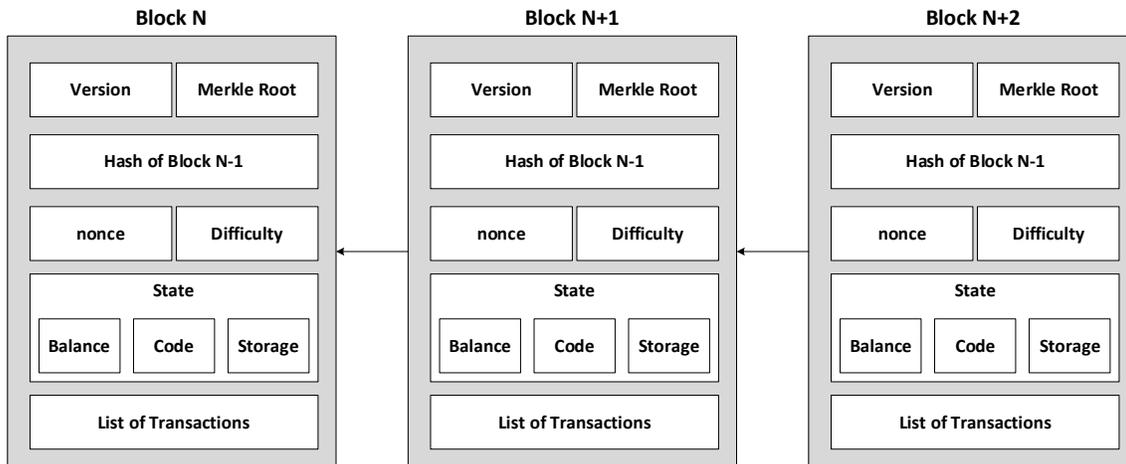


Figure 1. The basic structure of the blockchain

The consortium blockchain [25] belongs to a permissioned network; not everybody has equal rights to the validation of transactions. Only several selected individuals are permitted to perform validation on transactions. A slightly different version between the consortium blockchain and the permissioned blockchain is that the permissioned blockchain is based on a centralized structure with one entity, and this entity conjointly controls the validation processing. The centralized head can check that the consensus that's followed is that the one it projected. This is often additional, like having a centralized body just like the government in several nations. Permissioned blockchains are quicker, additional energy-efficient, and only implementable compared to permissionless blockchains. Relying upon the requirement and implementation atmosphere, one ought to prefer that algorithmic rule to deploy. Algorithms supported permissioned blockchain are the foremost used, and additional applications supported it are being researched on.

Ripple [26] connects banks and payment suppliers via RippleNet to produce one resistance expertise for causation and receiving cash globally. It is ascendible, secure, and interoperates with totally different networks. Banks, payment suppliers, and digital asset exchanges method and supply liquidity for payments on RippleNet, making new, competitive cross-border payments services for customers. Stellar [27] is another platform that connects banks, payments systems,

and people. Integrate to maneuver cash quickly, reliably, and at virtually no cost. Corda [28] could be a blockchain implementation that focuses on automating money documents besides as maintains records of them. They embody legal documents sculpturesque as planned by law, which is predicated on laptop codes and is mechanically generated. These legal documents contain info on the rights of the individual mentioned. This technology may revolutionize the money and legal sectors in several countries. Table 1 compares the various options of some well-known blockchain platforms.

In contrast to the Bitcoin and Ethereum, which are permissionless-based networks, Hyperledger Fabric [29] is a permissioned-based, where all of the participants are trusted to each other. One of the most critical differences in support of multiple consensus algorithms that make this platform additional effectively to be adaptable for specific use cases. Fabric leverages consensus algorithms that do not need a native cryptocurrency to incent expensive mining or to fuel transaction execution. The rejection of a cryptocurrency reduces some vital risk/attack vectors. Therefore, the absence of cryptologic mining operations implies that the platform may be deployed with roughly a similar operational cost as the other distributed system. It also supports the employment of general programming languages like Java and Go instead of domain-specific language to smart contracts.

Table 1. Comparison of some well-known blockchain platforms

Name	Support of Smart Contract	Consensus Algorithm	Native Cryptocurrency	Blockchain Type
Bitcoin	No	Proof of Work	BTC	Permissionless
Ethereum	Yes (Solidity)	Proof of Work	ETH	Permissionless
Hyperledger Fabric	Yes (Java, Go, etc.)	Pluggable Consensus	No	Permissioned

Table 2 presents a comparative analysis by comparing different types of blockchains. The results indicate that the permissioned network has high efficiency compared to the permissionless network since the consensus is determined by one or a selected set of nodes. However, a permissioned network can result in a fully centralized architecture against the idea of decentralization. This paper focuses on the consortium blockchain, Hyperledger Fabric, as it is a permissioned network with a decentralized characteristic. So far, Hyperledger Fabric is considered as an ideal blockchain platform to implement distributed applications for enterprise use cases.

Table 2. Comparison of different types of blockchain platforms

Property	Public Blockchain (Permissionless)	Private Blockchain (Permissioned)	Consortium Blockchain (Permissioned)
Consensus Participant	All nodes	Single node	A selected number of nodes
Permission	Public	Either public or private	Either public or private
Efficiency	Low	High	High
Centralized	No	Yes	Partial
Consensus	Permissionless based	Permissioned based	Permissioned based
Use Case	Bitcoin, Ethereum	MultiChain [30], OpenChain [31]	Hyperledger Fabric

This paper focuses on the permissioned network, and this section provides a concise description of one of the most well-known consortium blockchain platforms, Hyperledger Fabric. The Fabric is comprised of miscellaneous components such as peers, membership service providers (MSPs), clients, and ordering services. The basic transaction workflow goes through the following stages: the endorsement stage, the ordering stage, and the validation stage. Each stage runs independently and does not affect the other stages. Performance bottlenecks are slightly different at each stage, depending on the network configuration environment. Many existing

studies on blockchain performance have emphasized the transaction validation as one of the main bottlenecks except as the bottleneck in the ordering service. For the case of transaction validation, current optimization approaches include parallel transaction validation, bulk reading of slow CouchDB, and identity certificate caching. However, these studies modify the original architecture of the Fabric network either by updating some stages of the transaction process or adding external user-specific modules. As mentioned earlier, the development of the Hyperledger Fabric is still in progress, and there are times when new versions outside of the user-specified release cycles that may lead to instability and compatibility issues.

The business logic of Fabric is provided by the smart contracts that serve as a trustworthy decentralized program, gaining its trust and security from the blockchain and conjointly the fundamental agreement across the entire network. Fabric introduces a brand-new approach known as an execute-order-validate to perform transactions in three different stages. In simple terms, a submitted transaction will be executed, thus being endorsed, ordered in a block, and before appending to the ledger, these endorsed transactions must be validated against the predefined endorsement policy.

Figure 2 illustrates the workflow of the transaction execution taken place across the whole network. The client application should have credentials issued by the Certificate Authority (CA) to induce the approved permission for submitting dealings proposals. The CA issues credentials to clients who want to submit transaction proposals and authenticates the identities of these clients before they are allowed to participate in the network. Transactions are generated from client applications, and the application software development kit (SDK) is used to form connections between the client application and the peers within the network. These peers are the basic units to form the network, and they can be separated into endorser peers or committer peers depending on the type of task. Endorser peers simulate and sign proposals with their signatures, reply to grant, or deny approvals. Committer peers validate each endorsed transaction against the endorsement

policy and append the block of transactions to the ledger. The orderer is an individual node that is responsible for sorting the transaction that happened on a first-come-first-serve basis over the whole network. Furthermore, it supports various ordering service implementations like Kafka and Raft and contains the cryptographic credentials linked to each network entity.

Endorser peers execute the received transaction proposal by invoking the associated function defined by the smart contract in a simulated environment. It is worth noting that these transaction execution results will not be mirrored within the ledger in the current stage. Every endorser peer simulates the received transaction, signs the Read and Write (RW) set using their endorsing signatures, and returns proposal responses to the client application for inspection. The client verifies the endorsing signatures to check if the required endorsement policy (e.g., the required number of endorser peers that have to endorse the transaction execution results) has been consummated. Besides, it extracts the results from the RW sets to compare whether the transaction execution result simulated by each endorser peer is consistent or not. Afterward, these signed transactions are packaged and submitted along with RW sets to the orderer by the client application. The batched data is ordered into a block by the orderer and delivered to any or all committer peers. Every committer peer validates the transaction by checking whether the RW sets match the current state or not. Once the committer peer validates the transaction, it writes the transaction to the ledger, and also update the state by using the Write data from the RW set. Finally, the committer peers send events to the client application to inform whether the submitted transaction succeeded or not. The client application can subscribe to events to be notified by every committer peer once an event happens.

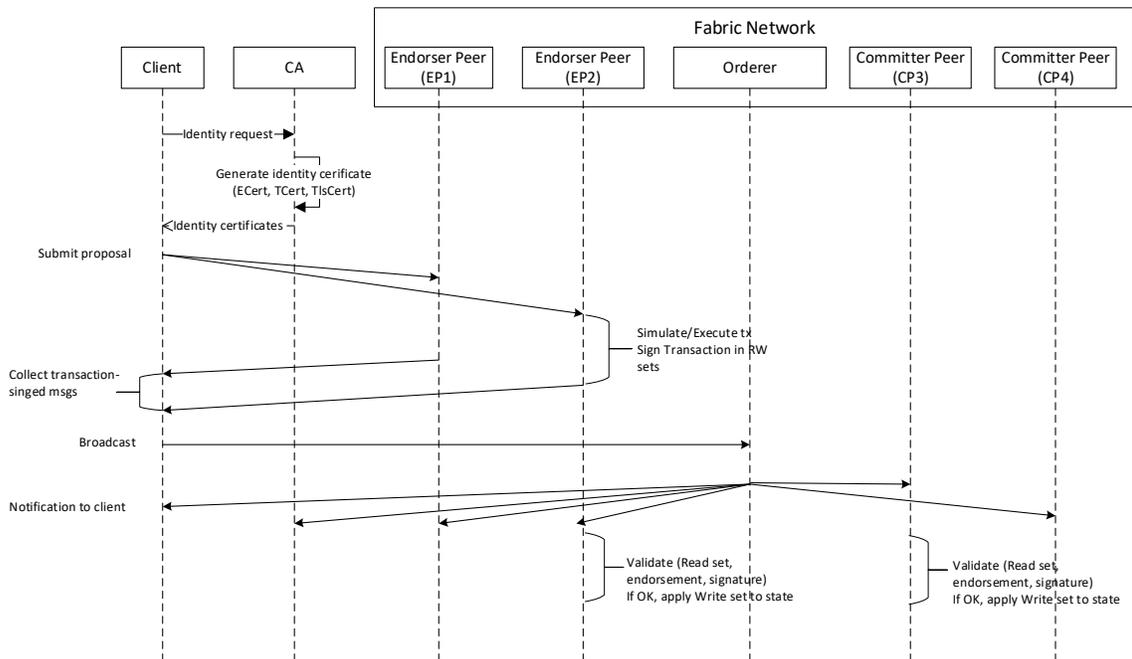


Figure 2. Blockchain platform transaction workflow (e.g., Hyperledger Fabric)

2.2 Existing Studies of Performance Improvement in Blockchain Network

In recent years, blockchain technology has emerged as a powerful technology to provide robust data integrity guarantees, especially in trust-less networks. The data integrity of blockchain can be guaranteed by the use of consensus algorithms such as Proof of Work (PoW), a consensus schema based on solving a CPU cost function that can deter denial-of-service attacks and any other service abuses such as network spam. Despite the characteristic of data integrity, utilizing blockchain to enhance distributed systems implies coping with mandatory performance penalties such as high latency and low throughput [32].

This paper concentrate on the performance improvement of one of the permissioned based blockchain networks called Hyperledger Fabric. The architecture of Hyperledger Fabric is still on the way, undergoing multiple changes in development and bug fixing. Therefore, there are

relatively few studies on fabric performance analysis or architecture optimization. The authors in [33] analyzed the performance of different Hyperledger Fabric versions (v0.6 and v1.0). The results indicate that Fabric v1.0 performs better than Fabric v0.6 in terms of throughput and latency. The authors in [34] investigated the performance of the Fabric network by varying different block sizes, resource allocation, state DB, and endorsement policies. In [35], the impact of peer CPU, disk type is evaluated on blockchain latency and throughput. These studies go partway towards explaining some parameters that can affect the blockchain performance and scale; however, not comprehensive. This paper first conducts a full survey on configurable parameters that can affect the Hyperledger Fabric network performance. Multiple rounds of experiments for each configurable parameter are performed to obtain the optimized configurable parameter value, which is used to set up the network for the experiment.

TPS is an essential factor for industrial application, and stable operation is possible. However, due to the nature of the blockchain, it is difficult to improve performance due to the high complexity of the consensus process, and the operation and maintenance costs are high because the block data is redundantly stored in the nodes constituting the blockchain. Currently, most of the researchers have tried to build the blockchain network based on high-performance hardware to improve throughput. However, due to the high cost, it is challenging to use blockchain in small and medium-sized enterprises with insufficient funds. To solve these difficulties, the authors propose a configurable blockchain system with a new consensus algorithm that can adjust the verification process [36]. They improve the tps performance according to the network scale-up/out, space efficiency, and security level. As a constraint, the basic authentication procedure is performed with a private blockchain, and general network security is in place. Besides, all P2P data transmission utilizes the PKI encryption module.

In the existing blockchain, the nodes participating in the consensus process are fixed in both ways, in which all nodes participate in consensus or select a specific leader. On the other side, in

the proposed blockchain, the consensus is achieved by selecting a random node based on a preset value defined according to the CPU utilization rate and an arbitrary validate node to reduce the computational overload of a specific node due to a fixed consensus node. Hyperledger Fabric is built on a modular architecture composed of multiple components, such as peers, ordering node, consensus, and secure storage. These components play an independent role in committing the block to the hash chain through the steps of assurance, ordering, verification, and modification. Each step is performed as a container and does not affect other steps. The bottleneck is slightly different in terms of the network configuration, but the main latency stage can be considered as the endorsement stage and the ordering stage.

As shown in Figure 3, when a client requests a transaction from a peer in the endorsement stage, it always checks whether the requested client is a node in the blockchain network. When the same client repeatedly makes a request, the node verification process may cause a decrease in performance. To reduce this process, a client that has been authenticated once performs processing for a transaction without verifying the node at the peer for a certain period [37].

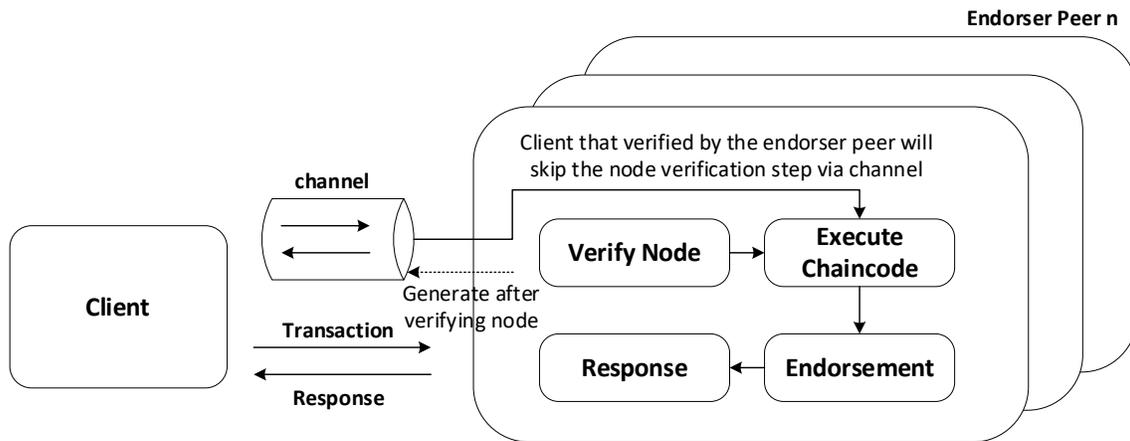


Figure 3. Endorsement channel function and generation process in endorser peer [37]

Hyperledger Fabric uses Kafka as an external project plug-in for Crash Fault Tolerance (CFT) during the sequencing phase to create blocks. Because it uses an external project that is heavier

than the local environment, a bottleneck may occur while performing the ordering step in the ordering node. Therefore, the proposed optimization works directly in the ordering node and proposes a protocol that can satisfy the CFT. As shown in Figure 4, when a transaction arrives from the ordering node, it separates the channel ID from the transaction contents, executes the thread corresponding to the channel, and manages and collects the transaction in a different file for each channel. Transactions are stored in the ordering node's file system. Even if a fault occurs in the ordering node, the fault can be repaired using another ordering node that has stored the transaction as a file using the proposed protocol. Depending on the ordering node setting, this thread also clears transactions when the maximum number of transactions is accumulated or when the waiting time is reached. The ordered node creates a block by including its signature in the organized transaction. The order service node then delivers the block to the leader peer, and the leader peer delivers the block to the commit peer on the same channel. This optimization method

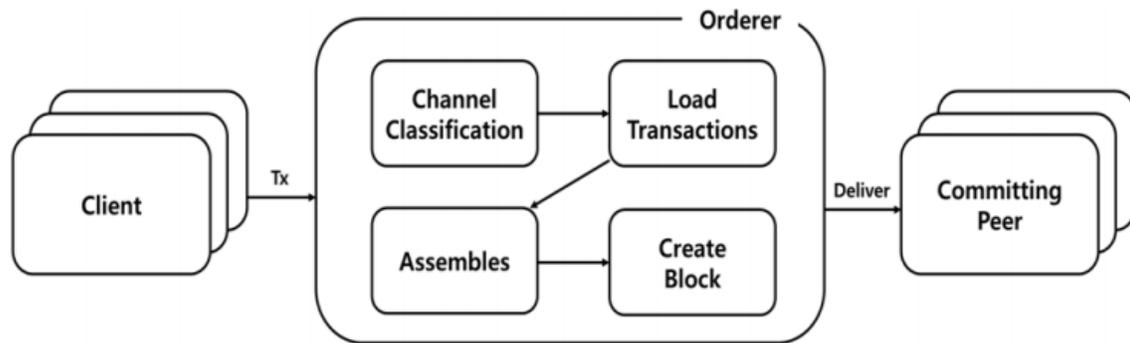


Figure 4. Optimized ordering service process in blockchain network [37]

Nexledger Accelerator [38] is a novel transaction processing engine, that has an independent and modular structure, acts as an intermediate between application clients and the blockchain network, as shown in Figure 5. Such a feature is appropriate for adopting IoT applications since IoT devices might not have enough computing power to run a novel approach for improving blockchain performance. The accelerator provides a straightforward, however robust transaction processing algorithm by using the batch scheduling. Accelerator classifies the incoming

transactions into a batched transaction. To this end, Accelerator is fastidiously designed to settle on the self-adaptive batch size, counting on the characteristic of requested transactions and also the remaining computing resource of the blockchain network.

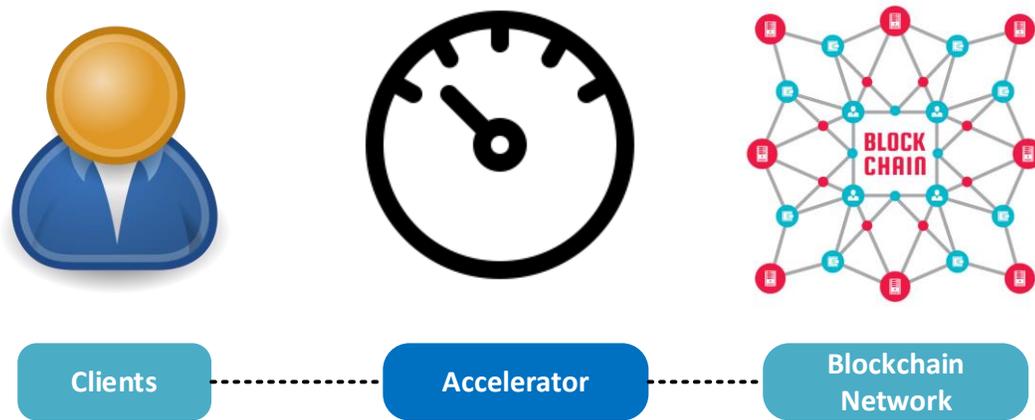


Figure 5. Conceptual architecture of Nexledger Accelerator [38]

As an essential technology that realizes decentralized and extremely trust database management, Blockchain has recently attracted in-depth attention. However, the high latency and low throughput in significantly concurrent environments is considered as the main performance bottleneck of blockchain technology, and therefore delays its deployment. Though multiple studies are dedicated to addressing the problem of low throughput, these studies only pay attention to the use cases with giant transaction volume, whereas neglecting the problem of conflicting transactions. Conflicting transactions are those transactions initiated by the client with the constant primary key. The system performance of Hyperledger Fabric may be severely degraded if various conflicting transactions existed. The authors [39] propose a cache-enabled endorser to discover the existence of conflict transactions before executing the smart contract. The system integrates with cache takes fewer steps to discard the conflict transactions compared to the system without the use of the cache. The cache is enforced on the local endorser, as shown in Figure 6. For every arrived transaction, the account code is extracted from the received data as the key information and stored within the cache before it being recorded. If the account code exists within the cache,

the endorser can discard the processing of such transactions and at once send a notification to clients to shorten the turnaround.

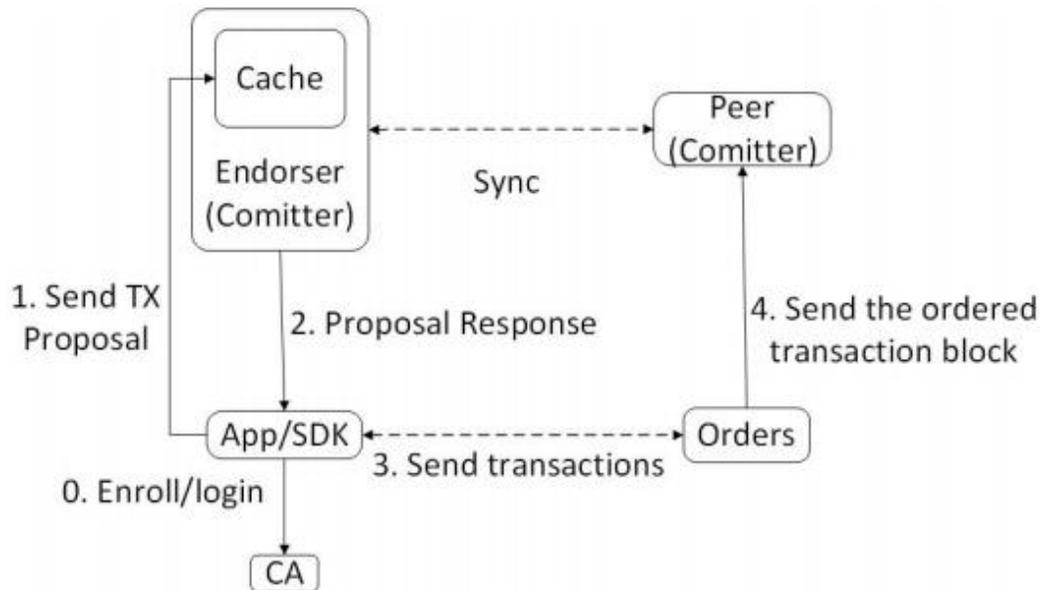


Figure 6. An example of transactions processing with cache [39]

Hyperledger Fabric is built with flexibility and generality as key style issues, supporting a significant form of non-deterministic smart contracts and pluggable services. However, version 1.0 comes without any BFT ordering service implementation, providing only a crash fault-tolerant ordering service. The authors [40] present the design, implementation, and evaluation of a new BFT ordering service, which is similar to the Practical Byzantine Fault Tolerance (PBFT) protocol. The evaluation, conducted both on a local cluster and in a geo-distributed setting, shows that BFT-S MART ordering service is able to achieve up to 10k representative transactions per second and write a transaction irrevocably within the blockchain in half a second, even with ordering nodes unfold through totally different continents. As shown in Figure 7, the frontend consists of the Fabric codebase and a BFT shim. The Fabric codebase (implemented in Go) provides an associate interface for clients to submit envelopes. These envelopes are relayed to the BFT shim using UNIX sockets. This shim is enforced in Java and maintains (1) a client thread pool that receives envelopes

and relays them to the ordering cluster, and (2) a receiver thread that collects blocks from the cluster. Envelopes (resp.blocks) are sent to (resp. received from) the cluster through the BFT-SMaRt proxy.

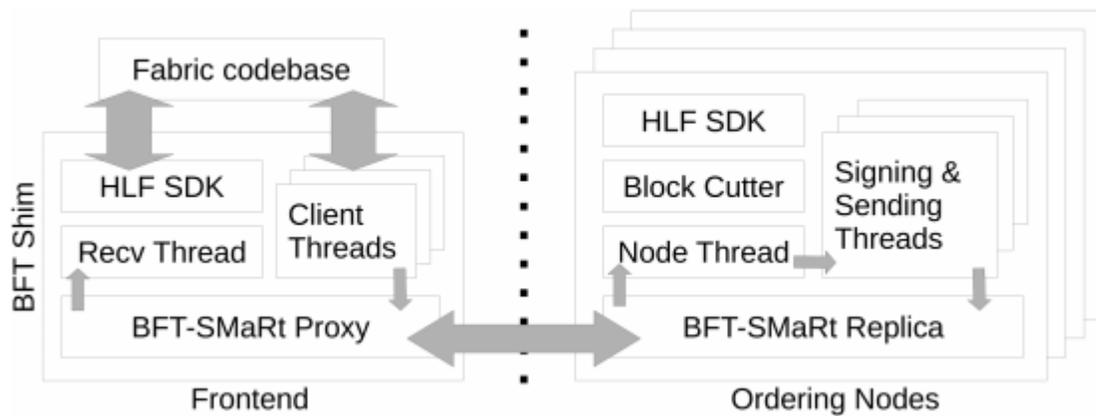


Figure 7. BFT-smart ordering service architecture [40]

Every peer node is linked to a distributed repository, so-called state database, to store the latest values of ledger data. Currently, Hyperledger Fabric supports two state databases: LevelDB and CouchDB. LevelDB is a database that resides in the peer that permits comparatively quick accesses. CouchDB is a client-server model, which is accessed by REST API over HTTP. The transaction validation is redesigned by authors in [41] to offer parallel operations on transaction validation, and a chaincode cache is proposed as well. Figure 8 describes each process in four steps. In the first step, the transactions are validated by committer peers, which set the transaction flag to invalid if the transaction is ungrammatical. Afterward, in the second step, the verification system chaincode (VSCC) and the operation to read the state database state are called together. In the third step, the multi-version concurrency control (MVCC) is further applied to check the transactions validated by the VSCC to avoid double-spend issues. In the last step, the ledger write and history database write operations are performed in parallel. In the case of LevelDB, the history database write operation is executed after completing the state database write operation. In contrast

to LevelDB, the history database write operation and the ledger write operations are performed at the same time.

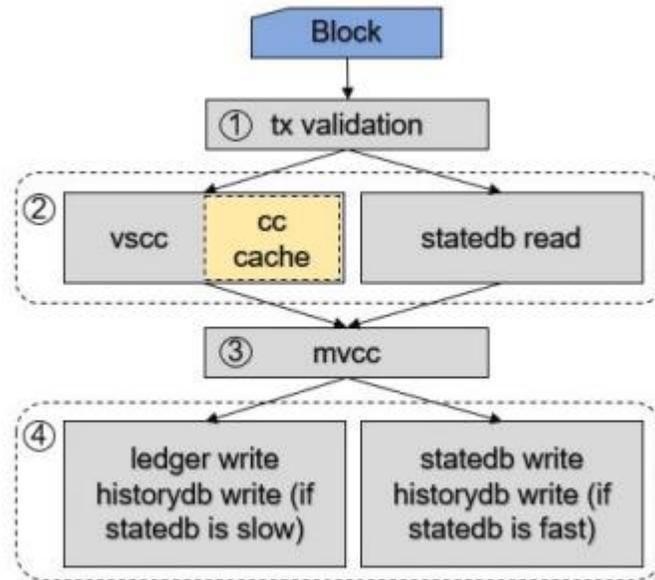


Figure 8. Optimized validation phase for Hyperledger Fabric [41]

As shown in Figure 9, two additional competent APIs between the chaincode and the peer are proposed [42]. The first API is called the Differential Update State (DUS), which can reduce the process of reading the state of the key before writing the updated value. As the name implies, the DUS API provides a specified set of operations to compute the updated values via different operations and writes the commutated value to the ledger. The second API is called the Compound Request (CR), which supports read, write, and their combined function. It executes all the requests in a specified order and further removes the number of requests compared to the DUS API. This feature makes it suitable in use cases that require frequent parameter read and initialization operations.

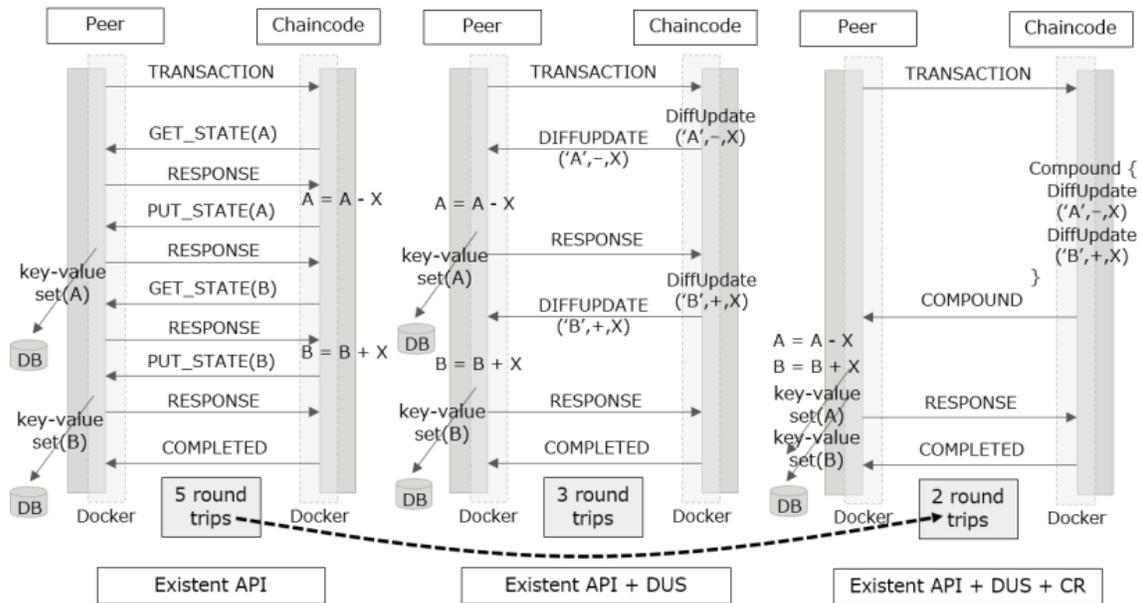


Figure 9. The behavior of two new efficient API functions [42]

As mentioned above, most of the existing systems change the original architecture of Hyperledger Fabric, including endorser peer, validating peer, and ordering service. This may result in incompatibility issues, especially when a new version released since the development of Fabric is ongoing. The proposed approach in this paper does not make any changes to the original system as the fuzzy logic can be directly deployed into the smart contract or implemented as an external module that is flexible enough to be extended. The Accelerator is built on an independent and modular architecture; however, it is only specified to Fabric network. The configuration of Accelerator is also involved, which makes it difficult to be used, especially for people who know little about blockchain. The proposed approach can be integrated with any other blockchain platform, which supports the use of smart contracts. Developers can choose the proper language to depend on themselves to implement the smart contract without concern about the infrastructure of the blockchain.

2.3 Performance Evaluation Architecture in Blockchain Network

Performance evaluation is the process of measuring the performance of a system under test. The goal of any performance evaluation is to understand and document the performance of the system or subsystem being tested. This often involves measuring what happens when dependent variables are altered; for example, measuring the throughput of the system as the number of concurrent requests is varied. Measure the performance of the blockchain network has been considered as one of the most concerning points from blockchain developers and researchers. A typical configuration of the performance evaluation in the blockchain network consists of test harness and system under test (SUT). The test harness is the system and program used to run the performance evaluation and to generate workload against the SUT. In general, the test harness can be multiple types of clients that can generate workload and observe the statistics of each peer node. The blockchain network is comprised of various peer nodes, which communicate with other nodes to work together corporately to execute transactions.

In this study, Hyperledger Caliper [43] is used to evaluate the performance of the designed solutions. Caliper is a unified blockchain benchmark framework that provides a standard layer to integrate with existing blockchain platforms (Hyperledger Fabric, Ethereum, etc.). Caliper provides a performance test report containing several performance indicators, such as transaction throughput, transaction latency, resource utilization, etc. It offers a variety of blockchain configurations, network setup as well as specific use-cases to test user-customized purposes.

Figure 10 presents the Caliper framework architecture that consists of four layers: benchmark layer, interface and core layer, adaptation layer, and the network layer. The benchmark layer provides various use cases to test the backend functions of the Fabric network. The interface and core layer offers a CLI-based interface for interacting with Fabric network, such as starting the network. It also provides additional functions, including resource monitoring, performance analysis, and report generation. The adaptation layer is used to integrate Fabric network with

external client applications by using corresponding blockchain SDKs to map operations such as invoking and querying states from the ledger. It also supports the interaction with other blockchain solutions by using platform specified adaptors to translate blockchain backend operations into associated blockchain protocol. The network layer provides the blockchain infrastructure of the Fabric network. The blockchain infrastructure contains various peers that hold distributed ledgers and smart contracts.

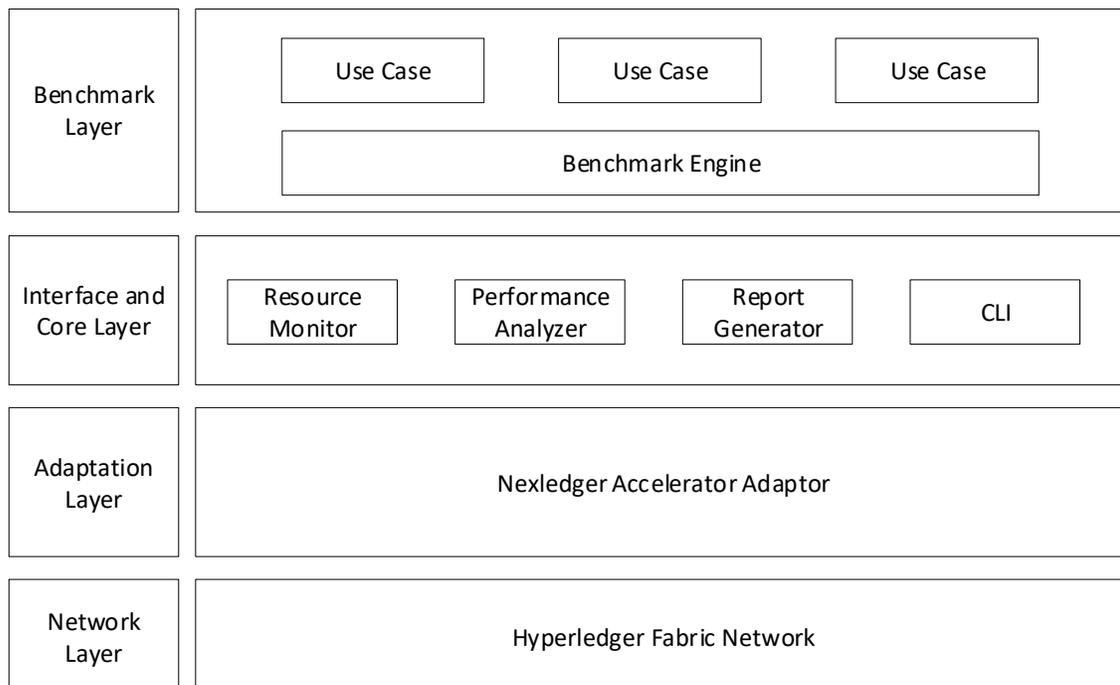


Figure 10. Performance evaluation framework architecture of blockchain using Hyperledger Caliper [43]

3. Performance Analysis of Configurable Parameters in Blockchain Network

3.1 Blockchain Network Performance Metrics

The throughput and latency are two standard performance metrics to evaluate the performance of the blockchain network. The throughput can be further divided into two subcategories concerning the operations to deal with. Read throughput is a specific measure to count the number of read operations completed in a defined time period, expressed as read per second (rps). Read throughput is not used as a central performance to measure the blockchain since most of the systems are typically deployed adjacent to the blockchain to achieve significant efficiency in reading and queries. Transaction throughput is the rate at which valid transactions are committed by the blockchain in a defined time period, expressed as tps. Transaction throughput is not the measure at a single node but across all nodes of the whole network.

- $Read\ Throughput = \frac{Total\ read\ operations}{total\ time\ in\ seconds}$ (1)

- $Transaction\ Throughput = \frac{Total\ valid\ transactions}{total\ time\ in\ seconds}$ (2)

Latency can also be separated into two subcategories in terms of the type of operations. Read latency measures the total time to submit a read request and receive the reply. Transaction latency measures the time that the entire network takes to validate a transaction, covering the broadcasting time as well as the allocation time spent by the consensus algorithm. The definition of the network threshold is specified in [44], which represents the quantity of time spent for a proportion of the network to commit a transaction. In this paper, we set the network threshold to one hundred as the utilization of the non-probabilistic protocol like PBFT.

- $Read\ Latency = Response\ received\ time - submission\ time$ (3)

- $Transaction\ Latency = Confirmation\ time * network\ threshold - submission\ time$ (4)

3.2 Configurable Parameters in Blockchain Network

This section discusses various configurable parameters, which can affect the blockchain performance. As shown in Table 3, the configurable parameters can be mainly summarized into two categories: software and hardware. For the case of software, it contains various parameters, including block size, block frequency, ledger database, ordering service, the programming language of smart contract, use of TLS, number of clients, number endorser peers, number of organizations, and the endorsement policy. Block size consists of two parts: message count and byte size. Message count determines the maximum number of transactions to form a new block. Byte size specifies the maximum number of bytes allowed for serialized transactions in a block. Block frequency determines the amount of time to wait after the first transaction arrives for additional transactions before cutting a block. In general, decreasing this value will improve latency, but decreasing it too much may decrease throughput by not allowing the block to fill to its maximum capacity. Ledger database stores the latest values of all keys. It includes several options, such as LevelDB and CouchDB. LevelDB is an embedded database that resides on each peer and stores data in the form of key-value pairs.

Ordering service is executed by a particular node called orderer that sorts the order of transactions. There exist server ordering service implementations: Solo, Raft, and Kafka. The Solo ordering service only features a single orderer node. It does not support fault-tolerant but is the right choice for testing blockchain applications and functionalities of smart contracts before putting into the production environment. Raft ordering service is implemented on the Raft protocol, which features the CFT. Kafka ordering service is a similar CFT-based implementation; however, it utilizes a ZooKeeper ensemble for managing a cluster of ordering nodes.

Currently, Fabric offers several SDKs to develop a smart contract in various programming languages such as Go, Java, and Node.js. Fabric supports the use of Transport Layer Security to secure data communication between nodes. TLS communication can be used in both one-way (server only) and two-way (server and client) authentication. A peer node can act as a TLS server and a TLS client at the same time. For example, it acts as a server receives messages from other nodes while acts as the client when initializing connections to other nodes. The client can connect to a specified peer according to the user-specific definition and thereby makes a connection to a peer to invoke transactions. A peer node is responsible for receiving the ledger state updates from the orderer and holds a copy of the ledger to maintain the consistency of the whole network. Endorser peer features a unique role concerning a specific smart consist and consists of endorsing the seal of approval to a transaction before committing the transaction. The organization is also known as a member that is invited to join the blockchain network by a blockchain network provider. The organization is known as a member of the network. An organization contains one or more peers, which is the transaction endpoint. The user can specify a set of organizations to form a consortium. Endorsement policy specifies the peers that must execute transactions and the required combination of endorsement responses.

For the case of hardware, it contains the number of vCPUS, memory allocation, disk type and speed, network speed, and CPU speed. The descriptions for each hardware parameter are depicted, as shown in Table 3.

Table 3: Configurable parameters that affect the blockchain performance

Category	Component	Description
	Block size	The maximum limit of a blockchain to be filled up with transactions.
	Block frequency	The amount of time to generate a new block.
	Ledger database	A decentralized database that records the newest values for all keys appeared in the transaction history.

Software	Ordering service	A specific kind of node called orderer node that performs transaction ordering, which, along with other nodes, forms an ordering service.
	Programming language of smart contract	A set of universal programming languages to code the smart contract, including Go, Node.js, and Java.
	Use of TLS	Represent whether to use Transport Layer Security
	Number of clients	The client represents the entity that acts on behalf of an end-user.
	Number of endorser peers	Endorser peers can endorse the seal of approval to a transaction when it is proposed.
	Number of organizations	A member that is invited to join the blockchain network.
	Endorsement policy	Define peers that ought to agree on the results of a transaction before appending it to the ledger.
Hardware	Number of vCPUs	A vCPU stands for a virtual central processing unit. One or more vCPUs are assigned to every virtual machine.
	Memory allocation	Memory allocation is a process by which computer programs and services are assigned with physical or virtual memory space.
	Disk type and speed	There are two types of disk drives: HDD (hard disk drive) and SSD (solid-state drive). Disk speed refers to the speed of reading and writing data, expressed in megabit per second (Mbps).
	Network speed	Network speed can be defined as the total no of packets being exchanged by the client and the server per second, which is usually calculated in megabit per second (Mbps).
	CPU speed	A CPU's clock speed rate is a measure of how many clock cycles a CPU can perform per second, is generally measured in Hertz, or GHz.

3.3 Experiment Environment in Blockchain Network

This section discusses the experiment setup and workload for evaluating the impact of various configurable parameters mentioned above. The default network for this experiment contains one channel, which consists of two organizations, each with one endorser peer for a total of two peers. The default block size is set to 10 transactions per block, and a new block is formed every 250 ms. The default ordering service is in solo mode, which consists only of a single ordering node. We utilized the LevelDB as the default state database in this experiment. The rest experiment parameters are specified, as shown in Table 4. The evaluation tests presented in this section were

averaged over multiple rounds to reduce errors resulting from the network congestion. We utilized the Hyperledger Caliper to evaluate the blockchain network performance. In this experiment, the sample network and benchmark configurations provided by Hyperledger Caliper were utilized to evaluate all configurable parameters. The smart contract used for this experiment is called simple, which is specified for a banking scenario. We tested the open function that creates a new account with the given amount of money.

Table 4: Default configuration for the experiment unless otherwise stated

Parameters	Values
Number of Orgs	2
Number of Endorser Peers	2
Endorsement Policy	AND (a, b, c)
Ordering Service	Solo
Block Size	10 transactions per block
Block Frequency	250 ms
Ledger database	LevelDB
Programming language of smart contract	GO
Use of TLS	No
Number of clients	1

3.4 Performance Evaluation of Configurable Parameters in Blockchain Network

The section describes the evaluation results of each configurable parameter, as discussed in section 3.2. We evaluated the impact of block size on the performance by varying the block size (30, 50, and 100) over different transaction send rate (range from 25 – 200 tps). Figure 11 plots the experimental results in terms of average transaction throughput. The transaction throughput increased linearly with the increase in send rate until it reached around 175 tps. However, the growth of transaction throughput decreased significantly and approached a flat when the send rate

was above this point. When the send rate was 200 tps, the throughput of each block size set was 180.4 tps, 172.3 tps, and 167.5 tps, respectively. Figure 12 plots the experimental results in terms of average transaction latency. The transaction latency decreases with the increase in the send rate. For example, when the block size was 30, and the send rate increased from 25 to 100 tps, the latency decreased from 170 to 90 ms.

This experiment results indicate that the settings of block size have a small effect on performance. Smaller block size shows better application performance in throughput and latency, but the difference is modest.

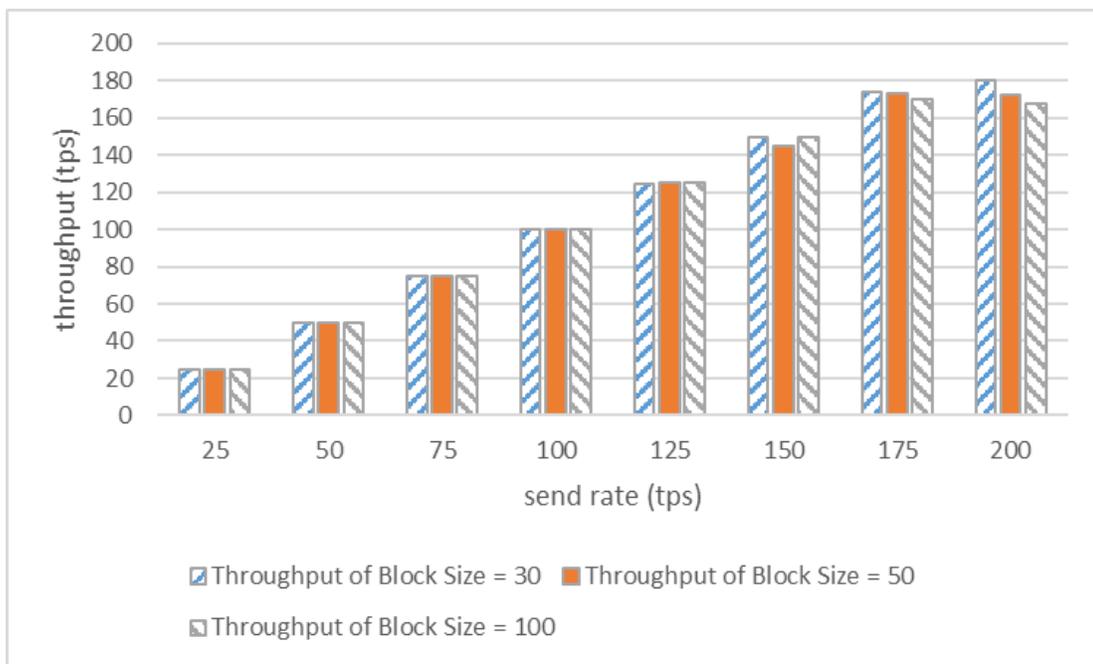


Figure 11: Evaluation of the impact of block size on transaction throughput

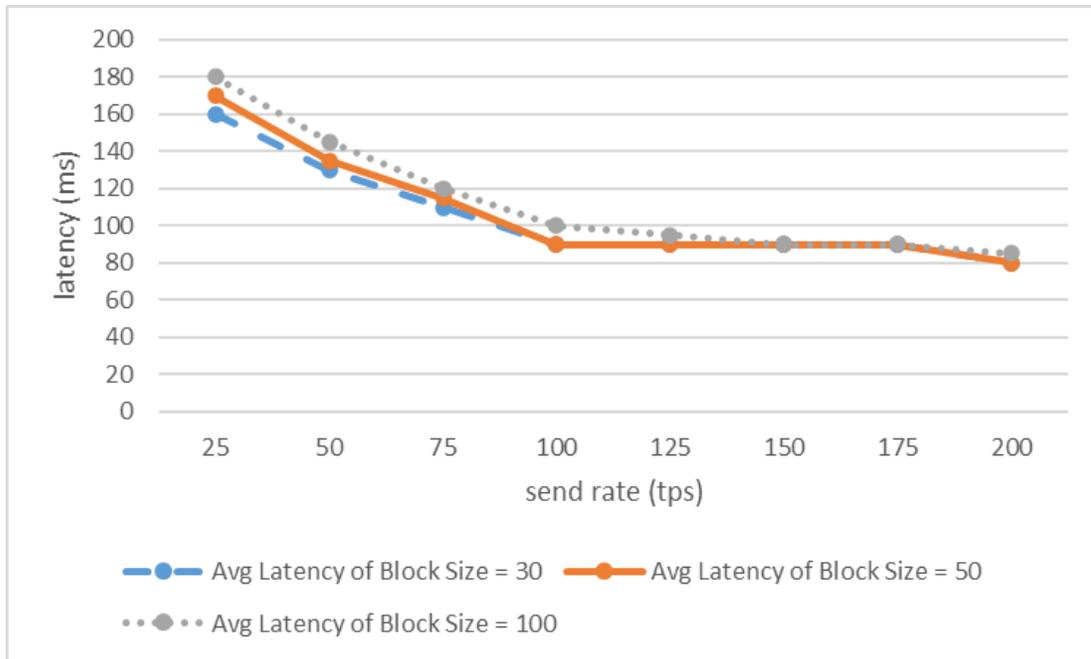


Figure 12: Evaluation of the impact of block size on transaction latency

We evaluated the impact of block frequency on performance by varying the block frequency (30, 50, 100) over different transaction send rate (range from 25 – 200 tps). Figure 13 plots the experimental results in terms of average transaction throughput. The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. However, the growth of transaction throughput decreased significantly and approached a flat when the send rate was above this point.

When the send rate was 175 tps, the throughput of each block frequency set was 165.8 tps, 170.6 tps, and 157.4 tps, respectively. Figure 14 plots the experimental results in terms of average transaction latency. The transaction latency decreases as the send rate increases. For example, when the block frequency was 250 ms, and the send rate increased from 25 to 100 tps, the latency decreased from 160 to 100 ms. This experiment results indicate that the settings of block frequency have a small effect on performance. Lower block frequency shows better application performance in throughput and latency, but the difference is modest.

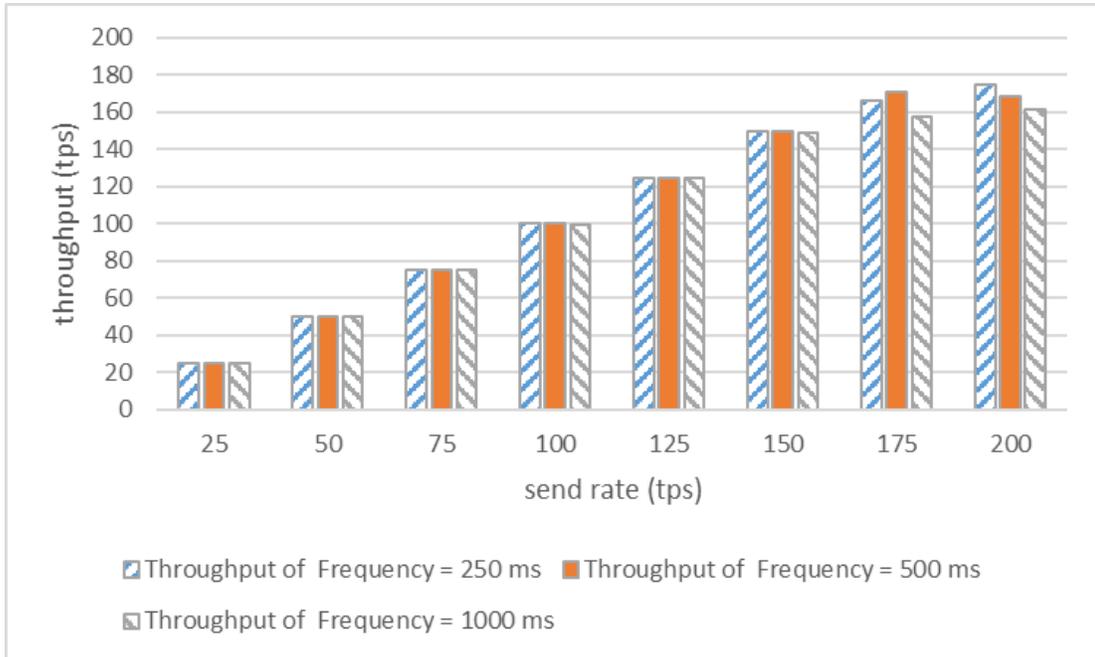


Figure 13: Evaluation of the impact of block frequency on transaction throughput

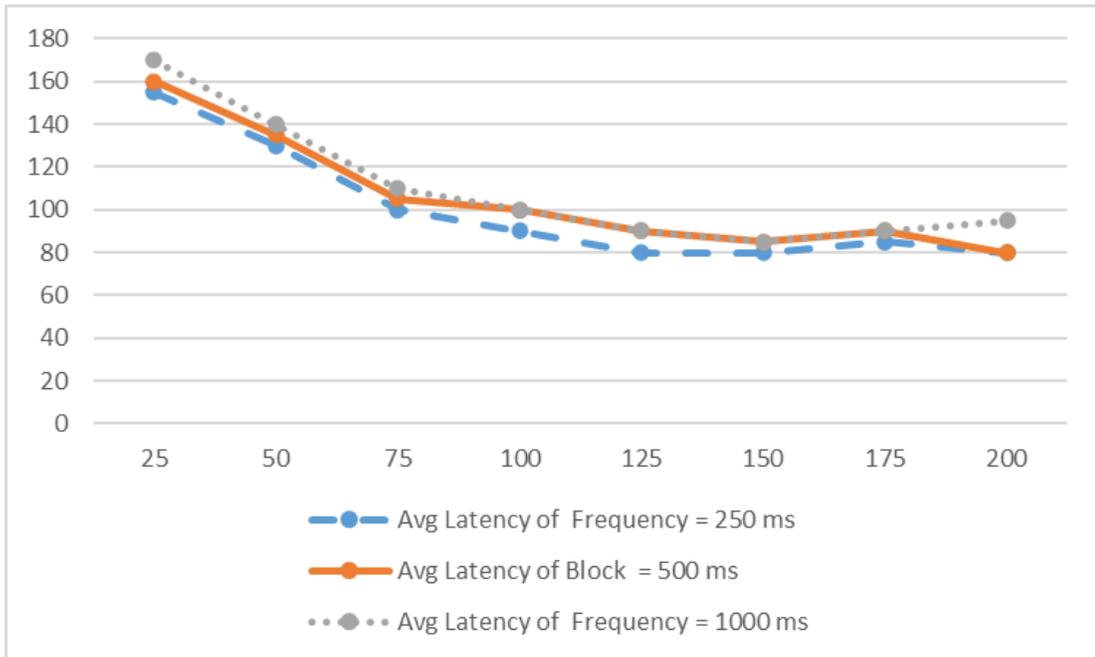


Figure 14: Evaluation of the impact of block frequency on transaction latency

In this experiment, we evaluated the impact of TLS on performance over different transaction send rate (range from 25 – 200 tps). Figure 15 plots the experimental results in terms of average

transaction throughput. The transaction throughput increased linearly with the increase in send rate until it reached around 75 tps. However, the growth of transaction throughput with TLS decreased significantly and approached to a flat when the send rate was above this point. For the transaction throughput, network without TLS has a higher throughput than the network with TLS when the send rate greater than 75 tps; for example, when the send rate was 100 tps, the throughput without TLS was 99.7 tps while the throughput with TLS was 81.1 tps. Figure 16 plots the experimental results in terms of average transaction latency. The network with TLS generates more latency than the network without TLS, but the variation of latency is tiny. This experiment results indicate that the use of TLS has a significant effect on performance. The network without TLS shows better application performance than the network with TLS since the network traffic is low.

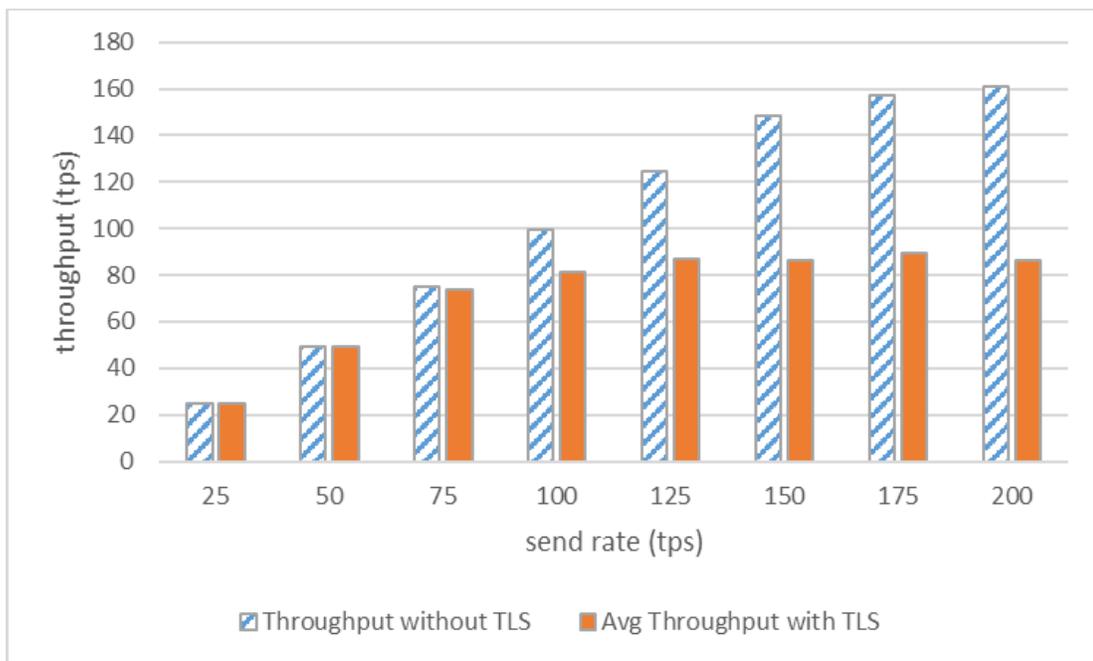


Figure 15: Evaluation of the impact of the use of TLS on transaction throughput

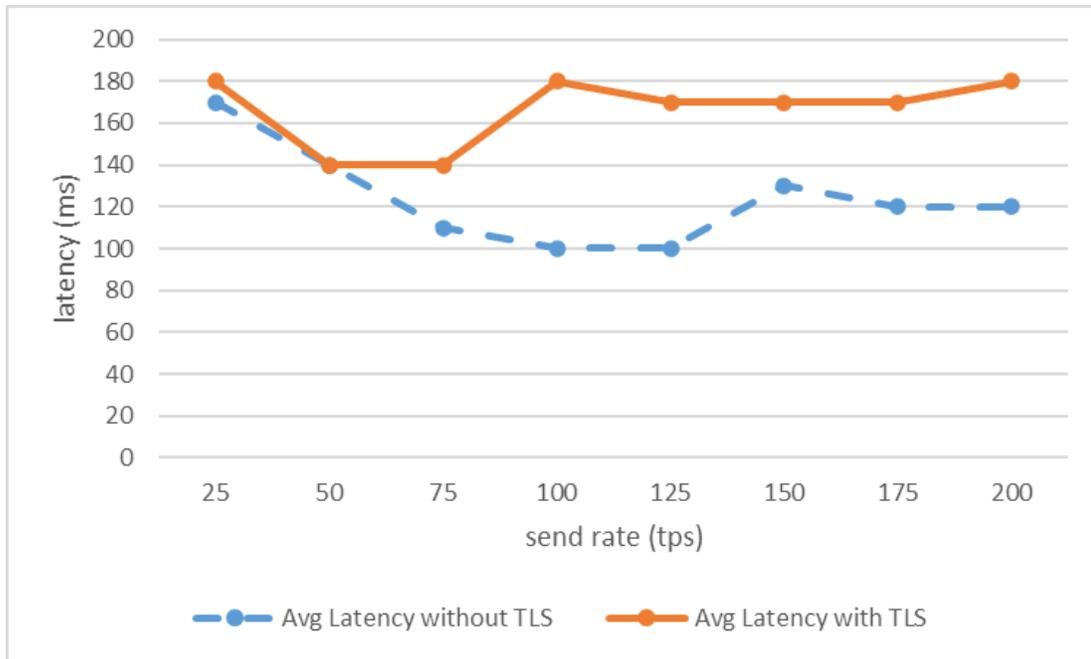


Figure 16: Evaluation of the impact of the use of TLS on transaction latency

In this experiment, we evaluated the LevelDB and CouchDB to analyze the impact of the ledger database over different transaction send rate (range from 25 – 200 tps). Figure 17 plots the experimental results in terms of average transaction throughput. The transaction throughput increased linearly with the increase in send rate until it reached around 75 tps. However, the growth of transaction throughput with CouchDB decreased significantly and approached to a flat when the send rate was above this point. For the transaction throughput, LevelDB has a higher throughput than the CouchDB when the send rate was greater than 75 tps, for example, when the send rate was 100 tps, the throughput using the LevelDB was 99.6 tps while the throughput using CouchDB was 65.8 tps. Figure 18 plots the experimental results in terms of average transaction latency. The CouchDB results in more significant latency than the LevelDB when the send rate was above 50 tps. The LevelDB database performs better than CouchDB because the peer can directly manipulate it while using CouchDB requires HTTP communication that generates additional network latency. The LevelDB performs better than CouchDB, generally, but is not as

effective at supporting a rich schema for the world state. It is appropriate to choose the LevelDB if the ledger data is in simple key-pair and does not require rich queries.

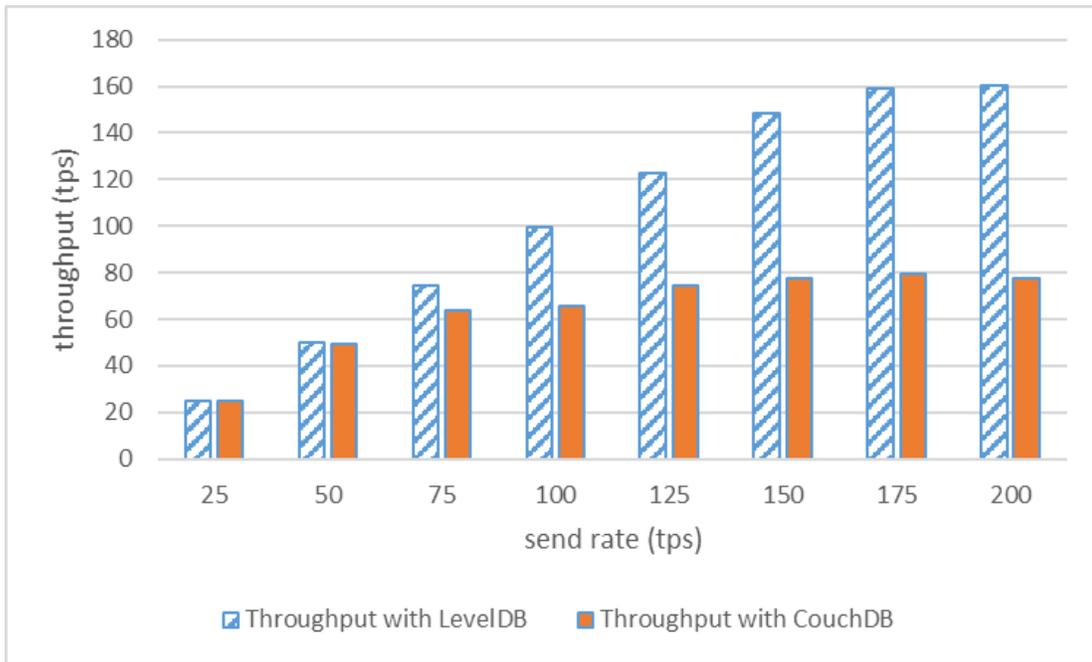


Figure 17: Evaluation of the impact of the ledger database on transaction throughput

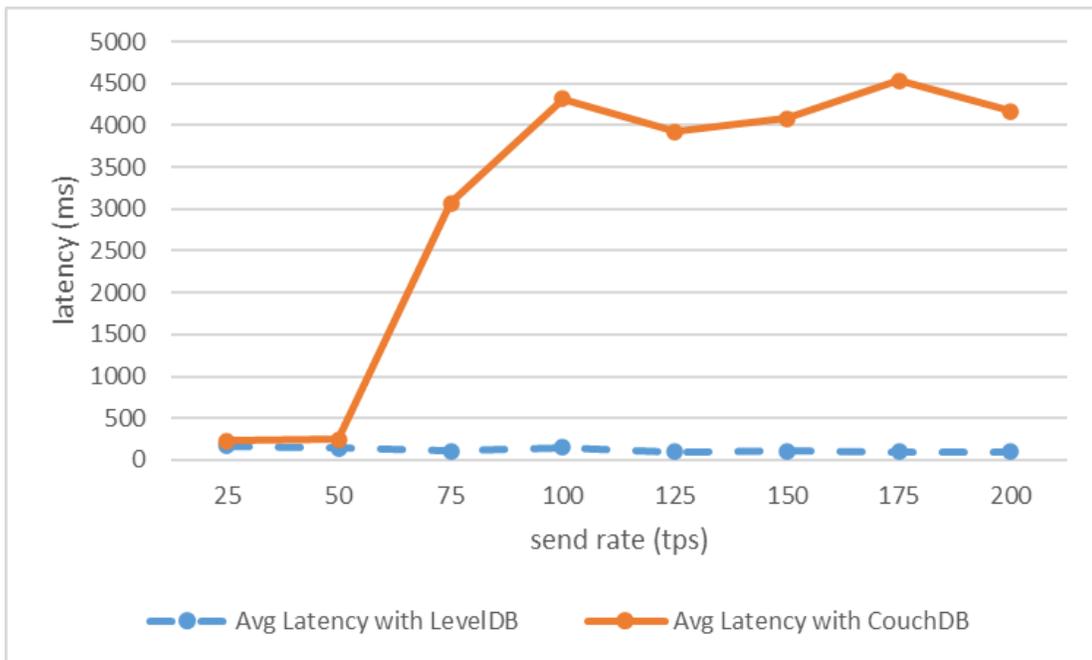


Figure 18: Evaluation of the impact of the ledger database on transaction latency

In this experiment, we evaluated three kinds of ordering services: Solo, Solo Raft (a single node Raft network), Raft to analyze the impact of ordering services over different transaction send rate (range from 25 – 200 tps). Figure 19 plots the experimental results in terms of average transaction throughput. The transaction throughput increased linearly with the increase in send rate until it reached around 75 tps, the throughput growth of Solo Raft and Raft decreased significantly and approached to a flat. For the throughput, Solo ordering service has a higher throughput than Solo Raft and Raft when the send rate greater than 75 tps; for example, when the send rate was 100 tps, the throughput of Solo was 99.8 tps. In comparison, the throughput of Solo Raft and Raft were 90.1 tps and 87.8 tps, respectively. Figure 20 plots the experimental results in terms of average transaction latency. Solo ordering service generates much less latency than Solo Raft and Raft ordering services. This experiment results indicate that the choice of ordering service does have a significant impact on performance. The Solo ordering shows better application performance than Solo Raft and Raft ordering services since it is a single node and does not require the process of TLS.

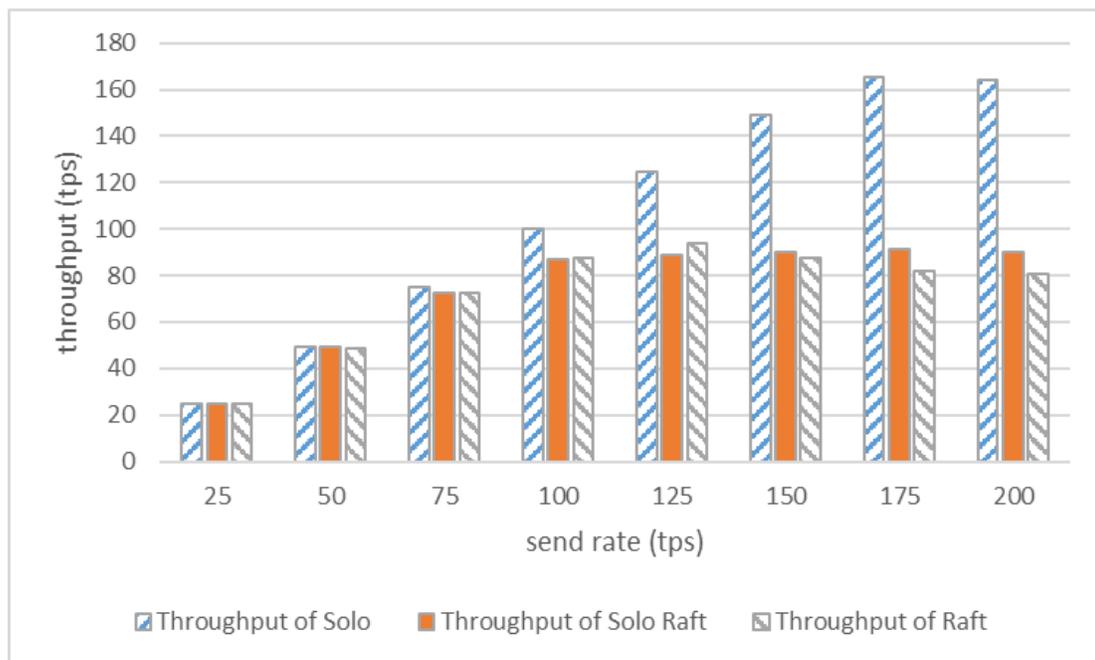


Figure 19: Evaluation of the impact of the ordering service on transaction throughput

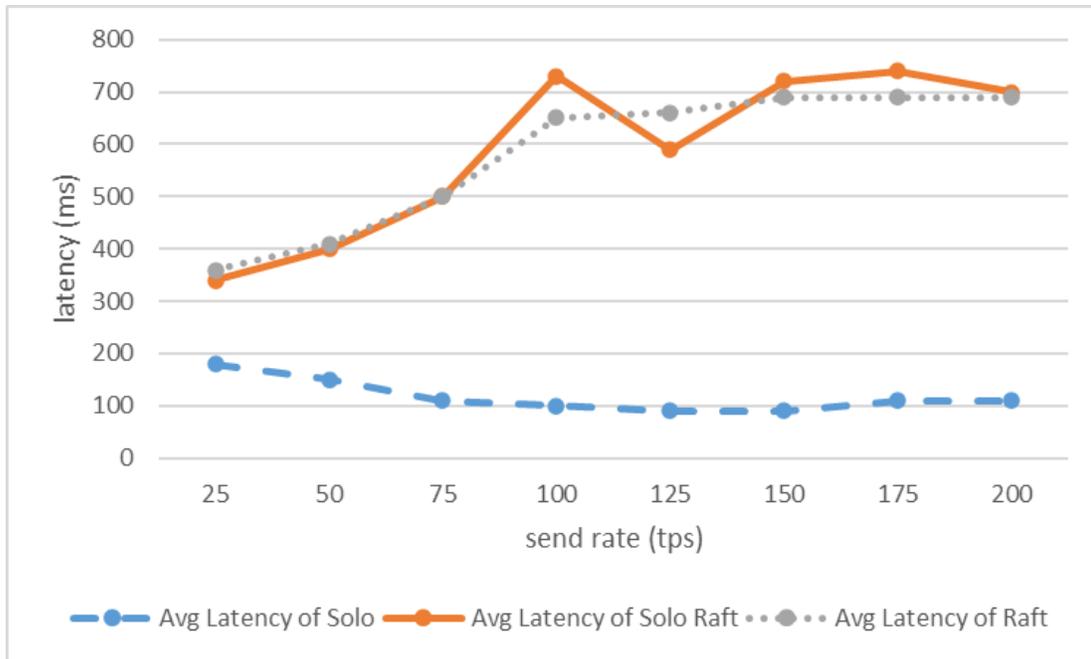


Figure 20: Evaluation of the impact of the ordering service on transaction latency

In this experiment, Go and Node.js languages are used to analyze the impact of the programming language of smart language over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly as expected with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. For the throughput, Go based smart contract has a higher throughput than Node.js based smart contract when the send rate was greater than the saturation point, for example, when the send rate was 175 tps, the throughput using Go based smart contract was 158 tps. In contrast, the throughput using Node.js based smart contract was 146 tps. Figure 22 plots the experimental results in terms of average transaction latency. Node.js based smart contract generates more latency than Go based smart contract, but the variation of latency is tiny. This experiment results indicate that the choice of programming language does affect performance. The Go language shows better application performance than the Node.js as it is compiled and supports multiple threads of execution.

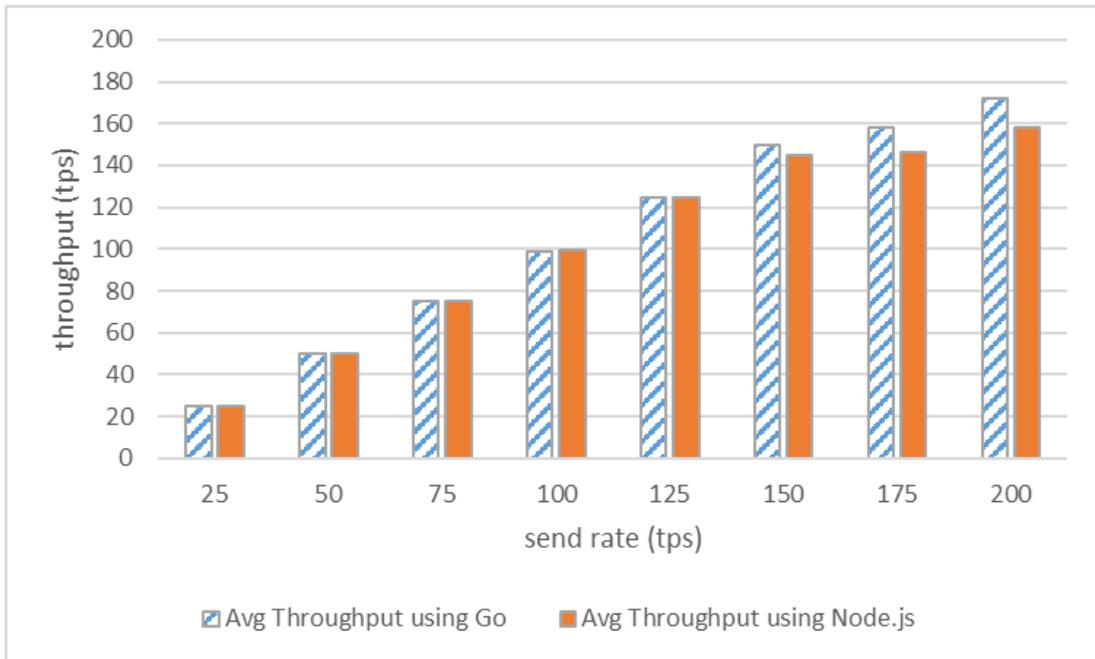


Figure 21: Evaluation of the impact of the programming language of smart contract on transaction throughput

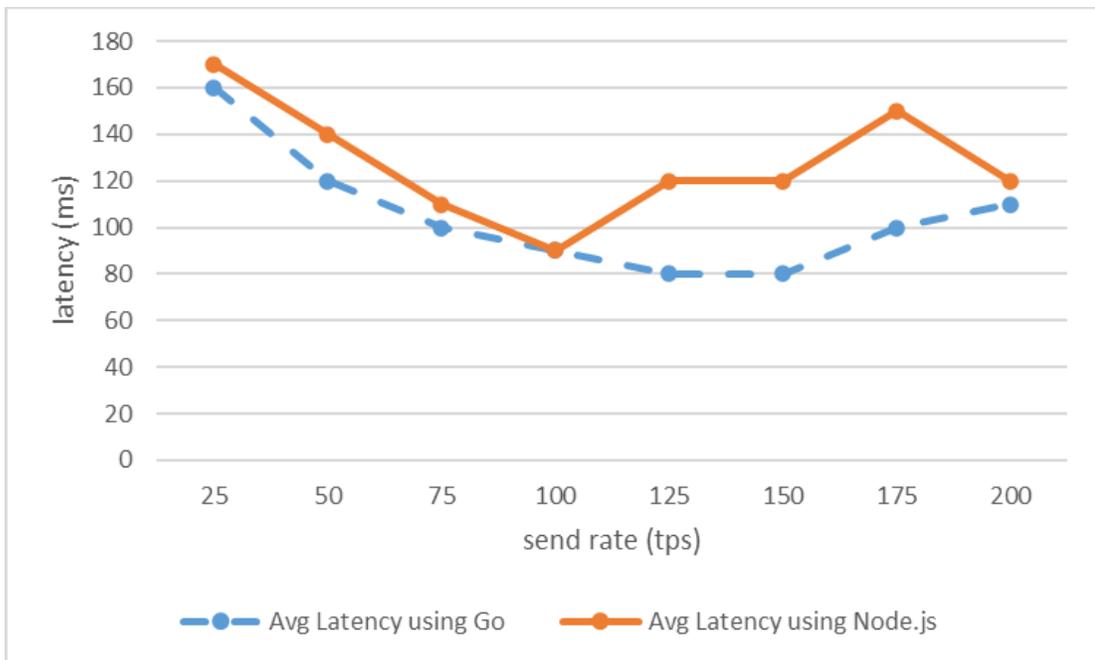


Figure 22: Evaluation of the impact of the programming language of smart contract on transaction latency

This experiment evaluated the impact of the number of clients on performance. Figure 23 plots the average transaction throughput for a various number of clients (1, 5, 10) over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the transaction throughput of each client set was 157.4 tps, 173.4 tps, and 162 tps, respectively. Figure 24 plots the average transaction latency for a various number of clients (1, 5, 10) over different transaction send rate (range from 25 – 200 tps). The transaction latency increases as the number of clients increases. For example, when the number of clients was 10, and the send rate increased from 125 to 200 tps, the latency increased from 130 to 550 ms. This experiment results indicate that the number of clients does have a significant effect on performance. Increasing the number of clients can improve the throughput, but increasing it too much can significantly increase the latency due to the increase in network traffic volume.

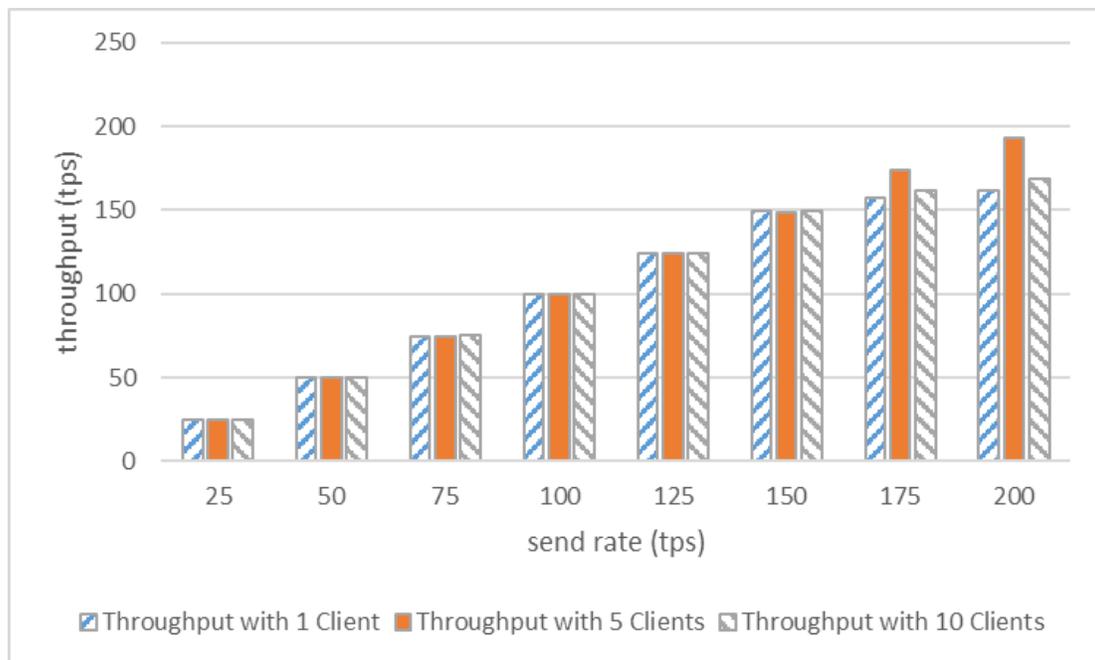


Figure 23: Evaluation of the impact of the number of clients on transaction throughput

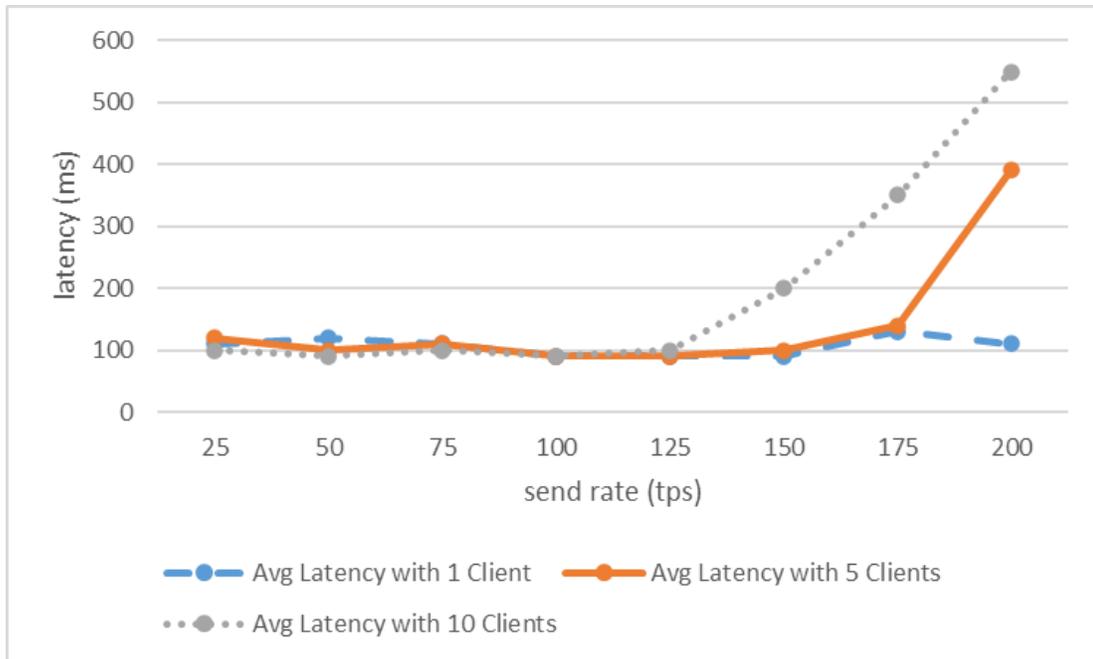


Figure 24: Evaluation of the impact of the number of clients on transaction latency

This experiment evaluated the impact of the number of endorser peers on performance. Figure 25 plots the average transaction throughput for a various number of endorser peers (2, 4, 6) over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each endorser peer set was 149.1 tps, 142.4 tps, and 123.7 tps, respectively. Figure 26 plots the average transaction latency for a various number of endorser peers (2, 4, 6) over different transaction send rate (range from 25 – 200 tps). The network with more endorser peers generates more latency, but the variation of latency is not apparent. This experiment results indicate that the number of endorser peers does have a significant effect on performance. Increasing the number of endorser peers can decrease the throughput and increase the latency due to the increase in network traffic volume.

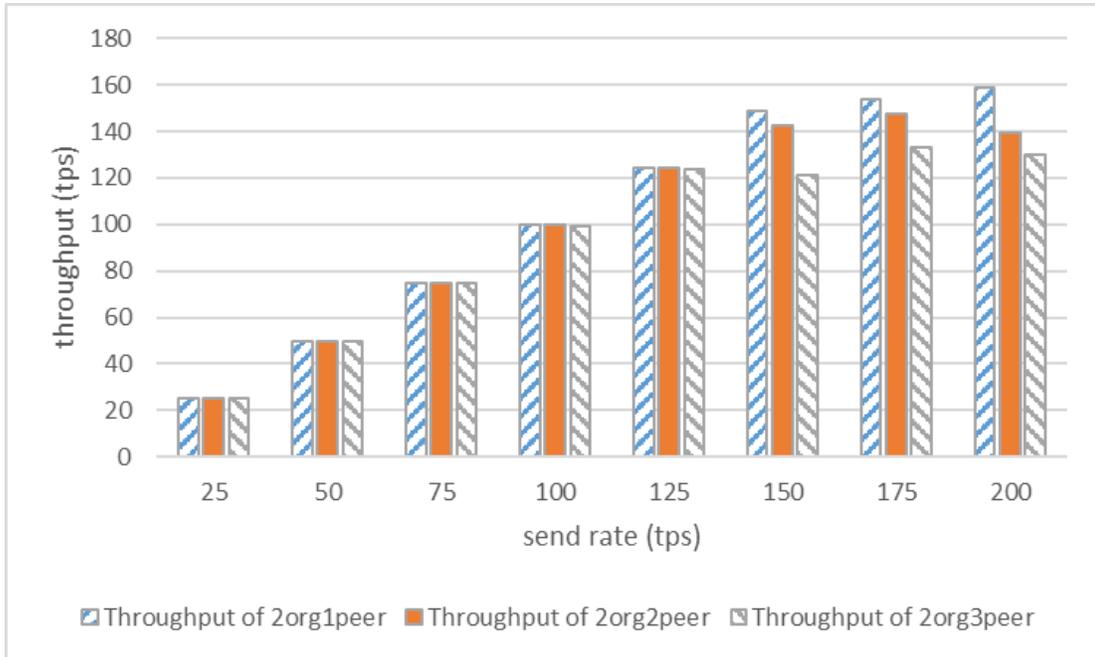


Figure 25: Evaluation of the impact of the number of endorser peers on transaction throughput

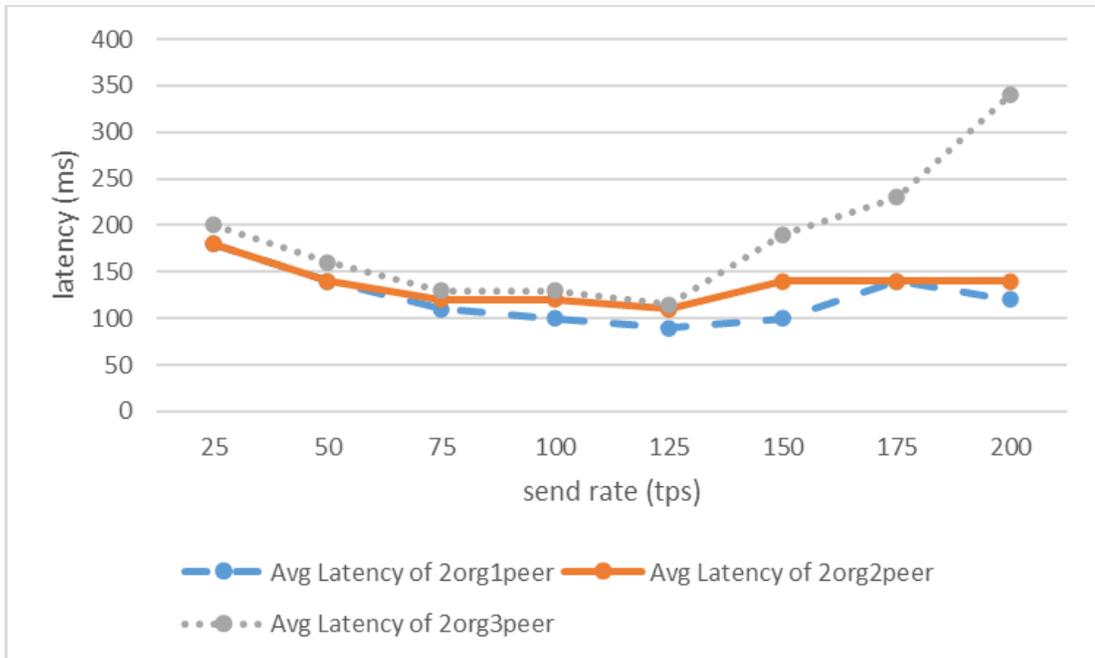


Figure 26: Evaluation of the impact of the number of endorser peers on transaction latency

This experiment evaluated the impact of the number of organizations on performance. Figure 27 plots the average transaction throughput for a various number of organizations peers (1, 2, 3) over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each endorser peer set was 149.4 tps, 138.8 tps, and 119.5 tps, respectively. Figure 28 plots the average transaction latency for a various number of organizations peers (1, 2, 3) over different transaction send rate (range from 25 – 200 tps). The network with more organizations generates more latency, and the variation of latency is enormous. For example, when the number of organizations was 3, and the send rate increased from 125 to 200 tps, the latency increased from 200 to 430 ms. This experiment indicates that increasing the number of organizations can decrease the throughput and increase the latency because the network becomes complicated.

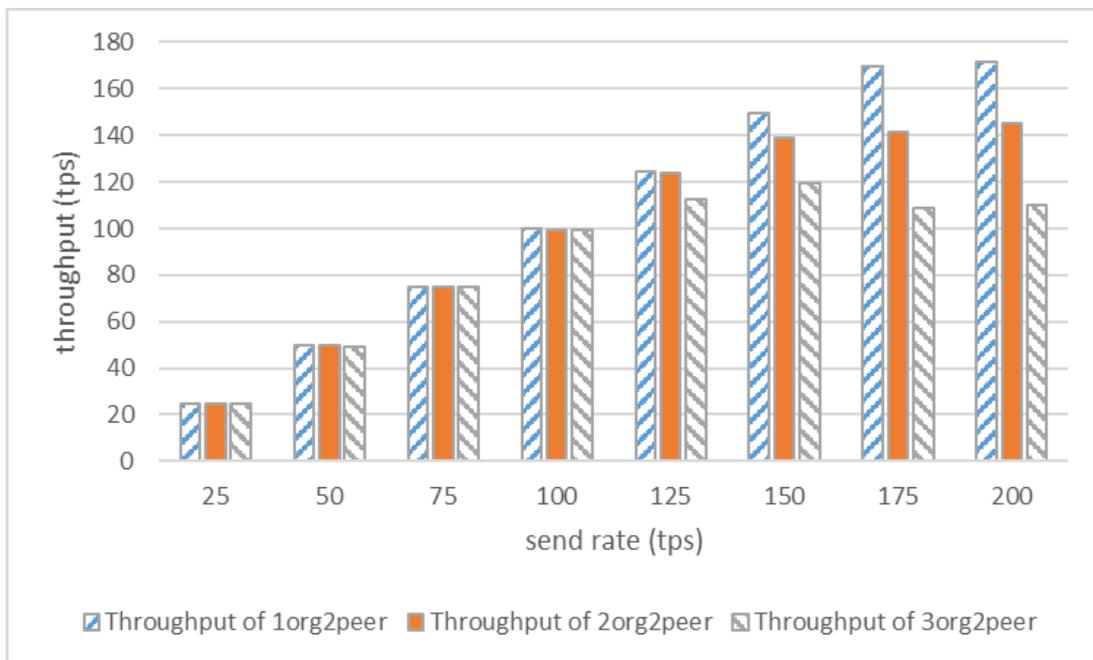


Figure 27: Evaluation of the impact of the number of organizations on transaction throughput

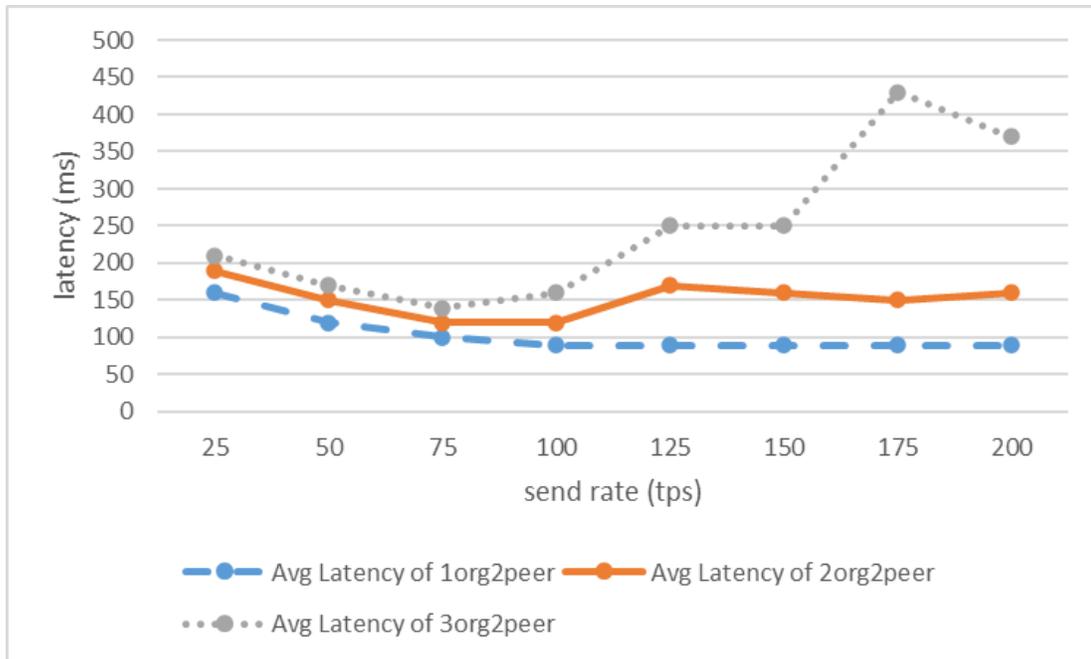


Figure 28: Evaluation of the impact of the number of organizations on transaction latency

For the experiment of endorsement policy, we configured a sample network with 3 organizations, and each organization has 2 endorser peers. Besides, three endorsement policies were specified: (1) at least one of the organizations much endorse transactions (policy 1), (2) at least two of the organizations much endorse transactions (policy 2), (3) all of the organizations must endorse transactions (policy 3). Figure 29 plots the average transaction throughput for various endorsement policies over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each endorser peer set was 149.4 tps, 138.8 tps, and 119.5 tps, respectively. Figure 30 plots the average transaction latency for various endorsement policies over different transaction send rate (range from 25 – 200 tps). The network with more organizations generates more latency, and the variation of latency is enormous. For example, when the number of organizations was 3, and the send rate increased from 125 to 200 tps, the latency increased from 200 to 430 ms.

This experiment results indicate that the endorsement policy does have a significant effect on performance. Increasing the number of organizations that must endorse transactions in the endorsement policy can decrease the throughput and increase the latency since each endorser peer has to exchange a message for every endorser peer, and this could significantly add process efforts for the network communication.

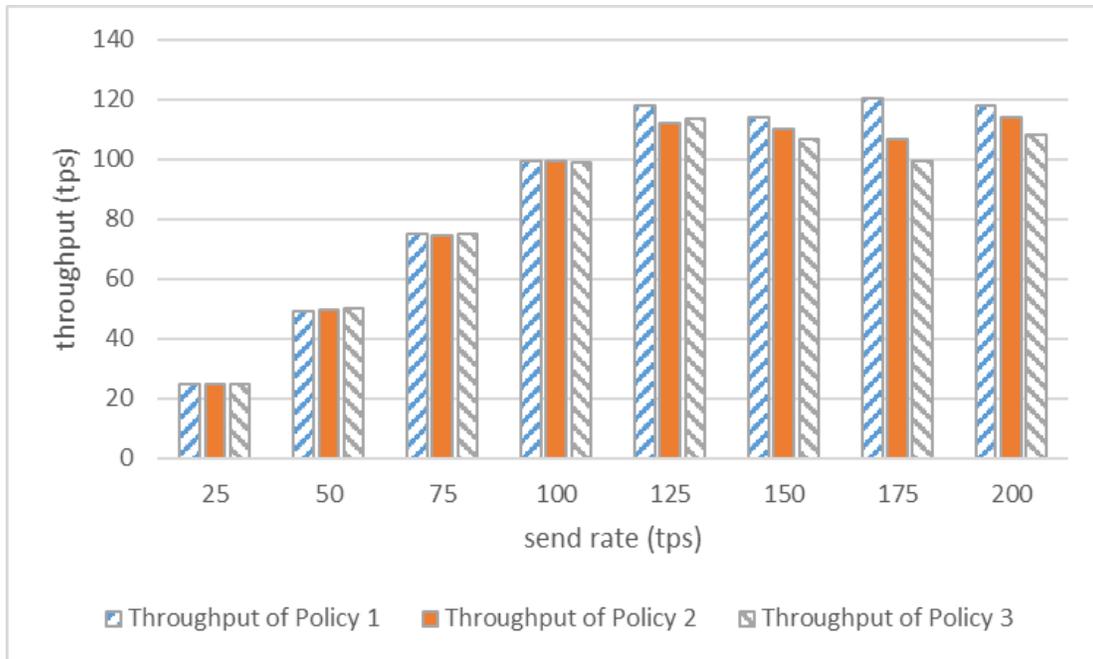


Figure 29: Evaluation of the impact of the endorsement policy on transaction throughput

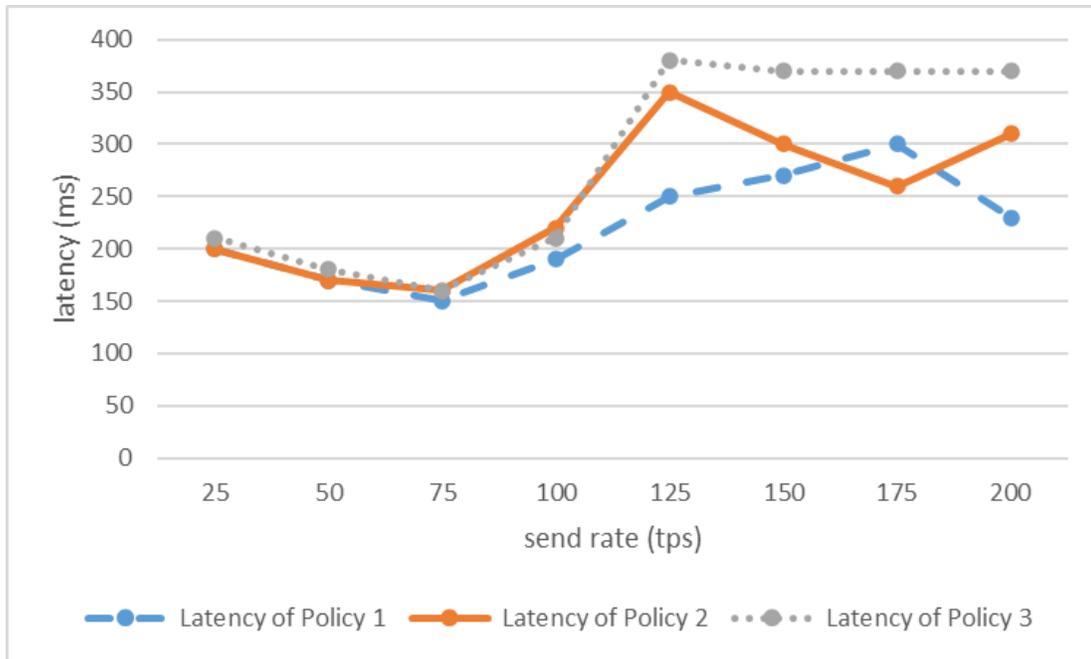


Figure 30: Evaluation of the impact of the endorsement policy on transaction latency

It is not the purpose of this paper to address all the impact of hardware on performance as these experiments were performed in a single-host virtual machine. Due to our hardware limitations, we could not validate the impact of hardware by varying the number of CPUs for validating peers or expanding the bandwidth of the network card. However, the spec of hardware does indeed scale performance according to some existing studies. The authors in [45] stated that critical dimensions for a distributed ledger system include peer hardware and software capabilities. The authors in [46] evaluated the impact of boosting the number of CPUs to the peers to improve the performance. The authors in [47] also indicated that if the hardware configuration has a higher spec, a higher number of transactions can be supported. Hence, there is, in fact, no limit on the number of transactions that can take part in the Hyperledger blockchain network. It depends on the selected hardware and the blockchain network configurations. Further studies on the impact of hardware should be conducted on different hardware components such as memory allocation, disk type and speed, network speed, and CPU speed. This will be pursued in future work by deploying the Fabric network on a cloud service like Amazon Web Services (AWS).

4. Transaction Traffic Control Mechanism based on Fuzzy Logic in Blockchain Network

4.1 Proposed Transaction Traffic Control Mechanism based on Fuzzy Logic

The conceptual architecture of the transaction traffic control mechanism based on fuzzy logic is described in Figure 31. The blockchain network is comprised of various nodes, which provide the host environment of smart contracts and hold a copy of the distributed ledger to maintain the consistency of the whole network. There exist multiple clients that can submit transactions by invoking the functions specified in the smart contract. The fuzzy controller is embedded in the smart contract to regulate the transaction traffic across the network automatically. The fuzzy controller consists of the fuzzy inference system and the transaction control modules. Benchmark results of the network are observed in real-time, and these values are transmitted to the smart contract. The fuzzy controller computes the control commands to make decisions on the received transactions. The consensus is achieved within the whole network, and the execution results are returned to the clients.

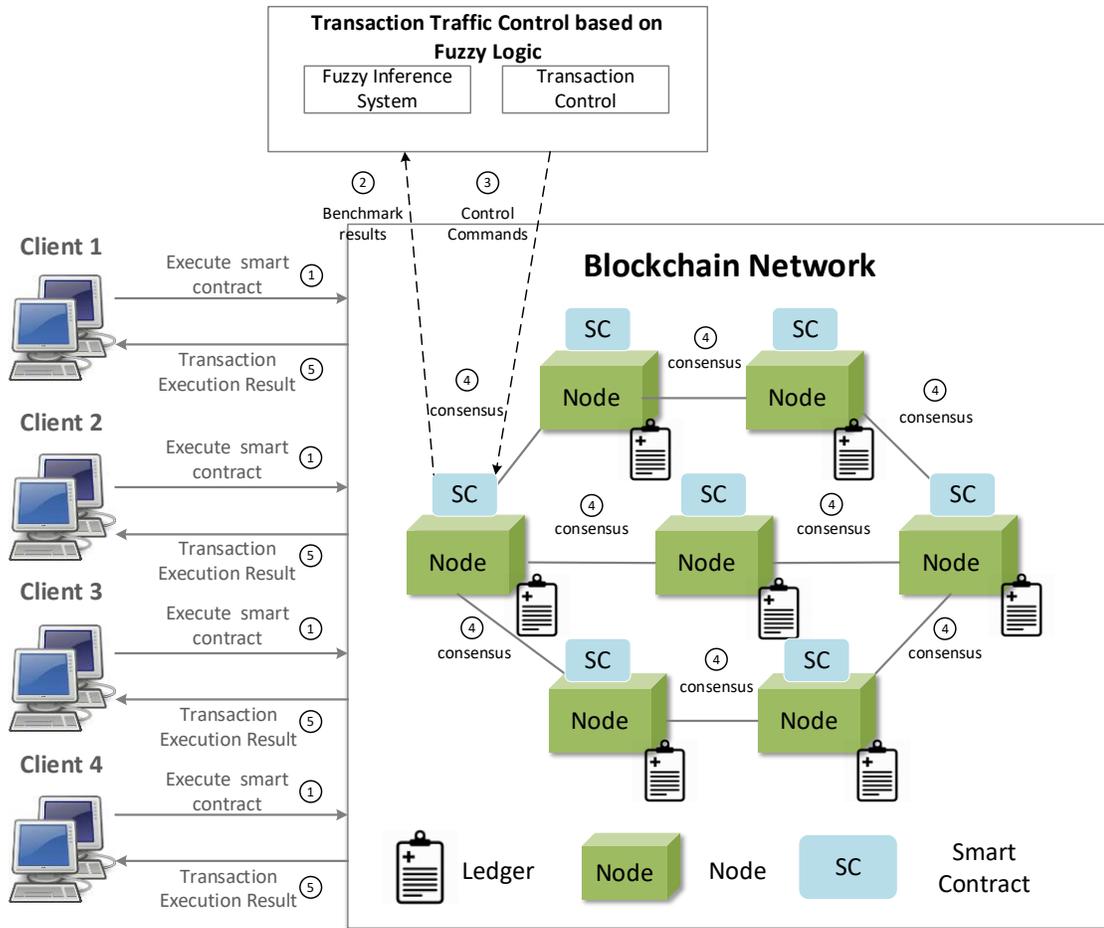


Figure 31: Conceptual architecture of the transaction traffic control mechanism based on fuzzy logic

Figure 32 illustrates the implementation diagram of the proposed system, which is comprised of the admin, transaction traffic measurement analyzer, client, blockchain adaptor, benchmark DB, and the blockchain network. The blockchain network consists of various nodes that hold a copy of the distributed ledger and a smart contract. The admin can configure the benchmark and network files for the transaction performance evaluation. A network configuration file describes the system under test and provides connection requirements for the network. A benchmark configuration file describes the performance benchmark workload and user-specified test files. The blockchain adaptor not only generates the transactions and passes to the client where the workload happens but also sends commands to initialize the blockchain network. The client submits transactions to

the blockchain network and returns the transaction responses. The transaction traffic measurement analyzer reads predefined performance statistics (TPS, latency, number of successful transactions, etc.) and stores benchmark results into the benchmark DB. The fuzzy controller adjusts the transaction acceptance rate by comparing transaction throughput, transaction latency with the acceptance rate. Transaction throughput and transaction latency are input parameters of the fuzzifier. Rules are evaluated in the inference engine. The defuzzifier converts output data (acceptance rate) into non-fuzzy values. The output value is obtained by the transaction control module to adjust the transaction acceptance rate. The whole process is repeated, and the throughput of the blockchain network can be dynamically maintained at a suitable level.

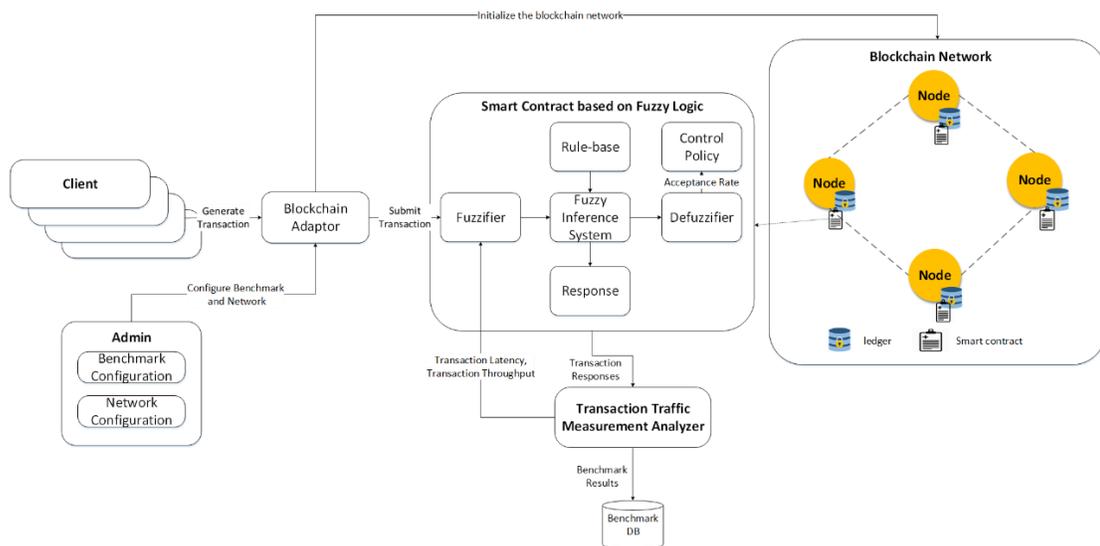


Figure 32: Development configuration of the transaction traffic control mechanism based on fuzzy logic

Figure 33 details the block diagram of the network configuration. The admin creates crypto certificates for each network entity and updates the network configuration, which specifies the topology of the network. The blockchain adaptor consists of config validator, client factory, client worker, blockchain SDK, and network configuration modules. The config validator validates each network configuration object. The client factory spawns the client worker to generate workloads.

The client worker is the client instance generated by the client factory. The blockchain SDK provides the interface to connect with the network. The network configuration is used to access information in the connection profile configuration. The blockchain adaptor can initialize the network (channel, peer) and install the smart contract to the network.

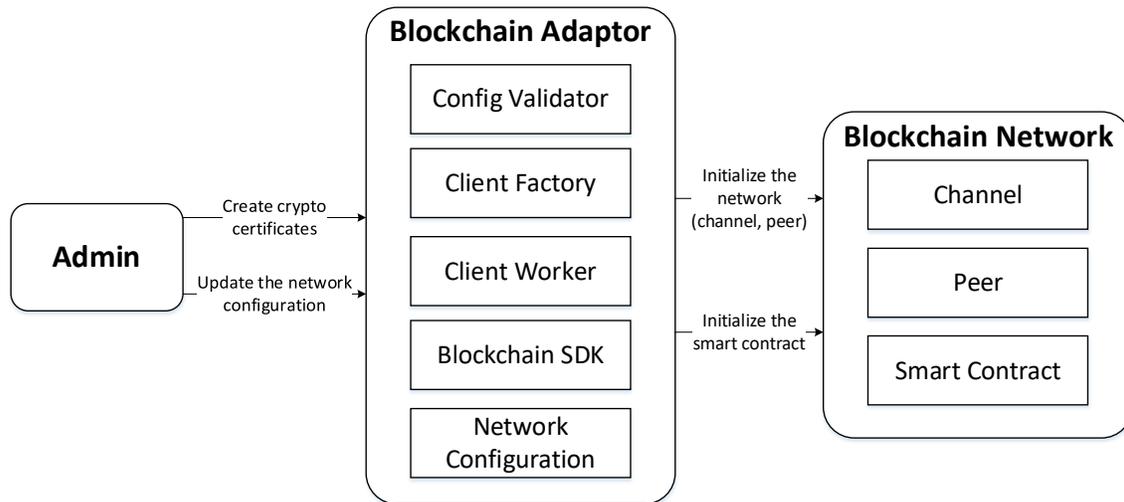


Figure 33: Block diagram of the network configuration

A fuzzy controller is a fuzzy logic-based control system that has been broadly utilized in numerous fields, for example, cooling, refrigeration, and automated control frameworks. Some different methodologies, like neural networks and genetic algorithms, can achieve just as fuzzy logic. Fuzzy logic keeps a favorable position by utilizing the general knowledge or experience that humans can easily understand.

The Mamdani fuzzy system is one of the most well-known theories in the field of fuzzy logic control (FLC) [48]. The linguistic control strategy is born-again into an automatic control strategy supported up-to-date data by FLC. Linguistic expression labeling information granular, like temperature for the weather or age for persons, is expressed as a linguistic variable. It is familiar and comfortable to convert linguistic values by using adverbs or adjectives since natural languages do not continuously contain enough worth terms to define a fuzzy variable scale. In this paper, we

utilize the Mamdani rule structure to set up linguistic modeling for regulating the transaction traffic control.

We use both triangular or trapezoidal membership functions to outline the fuzzy variables within the fuzzy system. The trapezoidal fuzzy set A performs $\mu_A(x)$, which is assigned by four quantified variables (a, b, c, d). The mathematical illustration of the fuzzy membership function is interpreted, as shown in Equation (1):

$$\mu_A(x) \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a < x < b \\ 1, & b < x < c \\ \frac{d-x}{d-c}, & c < x < d \\ 0, & x > d \end{cases} \quad (a < b \leq c \leq d) \quad (1)$$

It is worth noting that the trapezoidal function is thought to be a triangular one when b equals c. Equation (2) describes the fuzzy intersection operation between two fuzzy sets A and B, where $A, B \in U$ and x is any component within the U universe:

$$\mu_{(A \cap B)}(x) = \min\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in U \quad (2)$$

Besides, Equation (3) defines the fuzzy union operation:

$$\mu_{(A \cup B)}(x) = \max\{\mu_A(x), \mu_B(x)\}, \quad \forall x \in U \quad (3)$$

Two input variables, transaction throughput, and transaction latency, are considered in the proposed fuzzy controller, and the output variable generated by the fuzzy controller is the acceptance rate of the transaction. The quantitative results of the given fuzzy sets and corresponding membership degrees are calculated by Equation (4):

$$mCoA = \frac{\int f(x) \cdot x dx}{\int f(x) dx} \quad (4)$$

The linguistic terms of the input and output variables and their corresponding fuzzy sets are defined in Table 5.

Table 5: Fuzzy set definition in smart contract

Fuzzy variables	Linguistic terms	Fuzzy sets (a, b, c, d)
Transaction Throughput	Very Low	0, 0, 20, 60
	Low	20, 60, 100
	Acceptable	60, 100, 140
	High	100, 140, 180
	Very High	140, 180, 200, 200
Network Latency	Very Low	0, 0, 0.15, 0.45
	Low	0.15, 0.45, 0.75
	Acceptable	0.45, 0.75, 1.05
	High	0.75, 1.05, 1.3
	Very High	1.05, 1.3, 1.5, 1.5
Acceptance Rate	Very Low	0, 0, 10, 30
	Low	10, 30, 50
	Medium	30, 50, 70
	High	50, 70, 90
	Very High	70, 90, 100, 100

The proposed fuzzy controller aims to hold the acceptance rate of the transaction at an optimum level. For example, the acceptance rate is medium if the transaction throughput is in an exceedingly high condition, and the transaction latency is incredibly low. In a word, the fuzzy controller serves as a regulator of transaction traffic in line with transaction throughput and latency. Table 6 gives a list of specified fuzzy rules, and in total, twenty-five rules are defined.

Table 6: Fuzzy rules definition in smart contract

Latency Throughput	Very Low	Low	Acceptable	High	Very High
Very Low	Very High	Very High	High	High	Medium
Low	Very High	Very High	High	Medium	Medium
Acceptable	High	High	Medium	Medium	Medium
High	High	High	Medium	Medium	Low
Very High	Medium	Medium	Low	Very Low	Very Low

Figure 34 describes the workflow of the proposed transaction traffic control mechanism based on fuzzy logic. At the beginning of each test, the admin should configure the network and benchmark profiles to fulfill the requirements of the test scenario. The benchmark file describes how the evaluation test should be executed, including the number of rounds, send rate of the transaction, and settings about monitoring the test network. The network configuration file

describes the topology of the test network, such as the configuration of nodes, number of clients, and smart contracts deployed to the test network.

After configuring the network and benchmark profiles, the user can start the test. The system extracts the configuration details from the network configuration profile to set up the blockchain network. Meanwhile, it creates a required number of client workers to generate the workload to the network. Afterward, the clients start to submit transactions to the network, and the benchmark results are observed. These results are further analyzed to compute the transaction throughput and transaction latency, which are stored in an external data storage. These two parameters are used as the input parameters of the fuzzy controller. The fuzzy inference engine evaluates the input parameters according to defined fuzzy rules. The fuzzy controller produces the acceptance rate as the output value, which is used to control the transaction traffic flow. The transaction execution response is generated and returned to the client. This process is repeated across the entire benchmark experiment until the admin stops the test. Finally, all of the network entities and the smart contract will be removed.

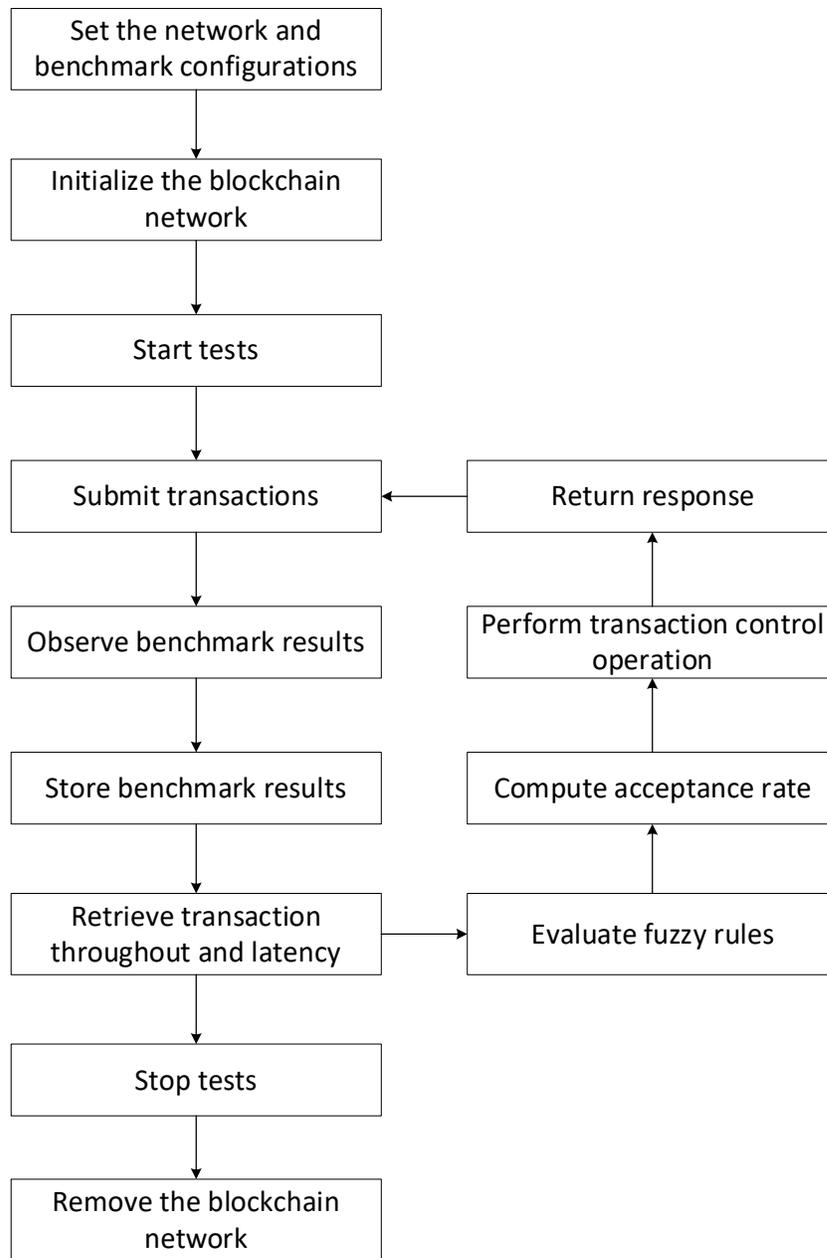


Figure 34: Flow chart of the transaction traffic control mechanism based on fuzzy logic

Figure 35 illustrates the execution process of the network configuration. The network admin uses the cryptogen tool to generate the required certificate for each network entity. These generated network certificates are stored in the local file system. Then the configtxgen tool is used to create configuration artifacts, including orderer genesis block, channel configuration transaction, and anchor peer transactions. After that, the admin modifies the docker compose files according

to his requirement and execute these files to start the blockchain network in docker containers. The channel configuration transaction is submitted to the blockchain network via the adaptor to create the channel in the blockchain network. As a response, a genesis block is returned, which will be used to join the channel. After the channel is created, each peer is joined the channel, and anchor peer transactions are executed by the orderer to specify anchor peer for each organization. The adaptor enrolls clients and obtains key pairs for each client. It also submits the smart contract install transaction to the network, and the network will initialize the smart contract accordingly.

Figure 36 describes the execution process of the transaction control based on fuzzy logic. The admin can start the script to start the benchmark test. One or more clients generate transactions to the adaptor, in turn, the adaptor submits transactions to the Fabric network. Meanwhile, the benchmark results are observed and collected by the transaction traffic measurement analyzer. The analyzer calculates the benchmark statistics and stores the results in the benchmark DB. The fuzzifier retrieves the transaction throughput and network latency as the input parameters of the fuzzy inference system. The inference engine evaluates the input parameters according to the fuzzy rules. The defuzzifier produces the acceptance rate as the output value and sends this value to the transaction control module. The transaction module performs transaction traffic control operations with respect to the acceptance rate. The transaction execution response is generated and returned to the client. This process is repeated across the entire benchmark experiment until the user stops the test. Finally, all of the network entities and the smart contract will be removed.

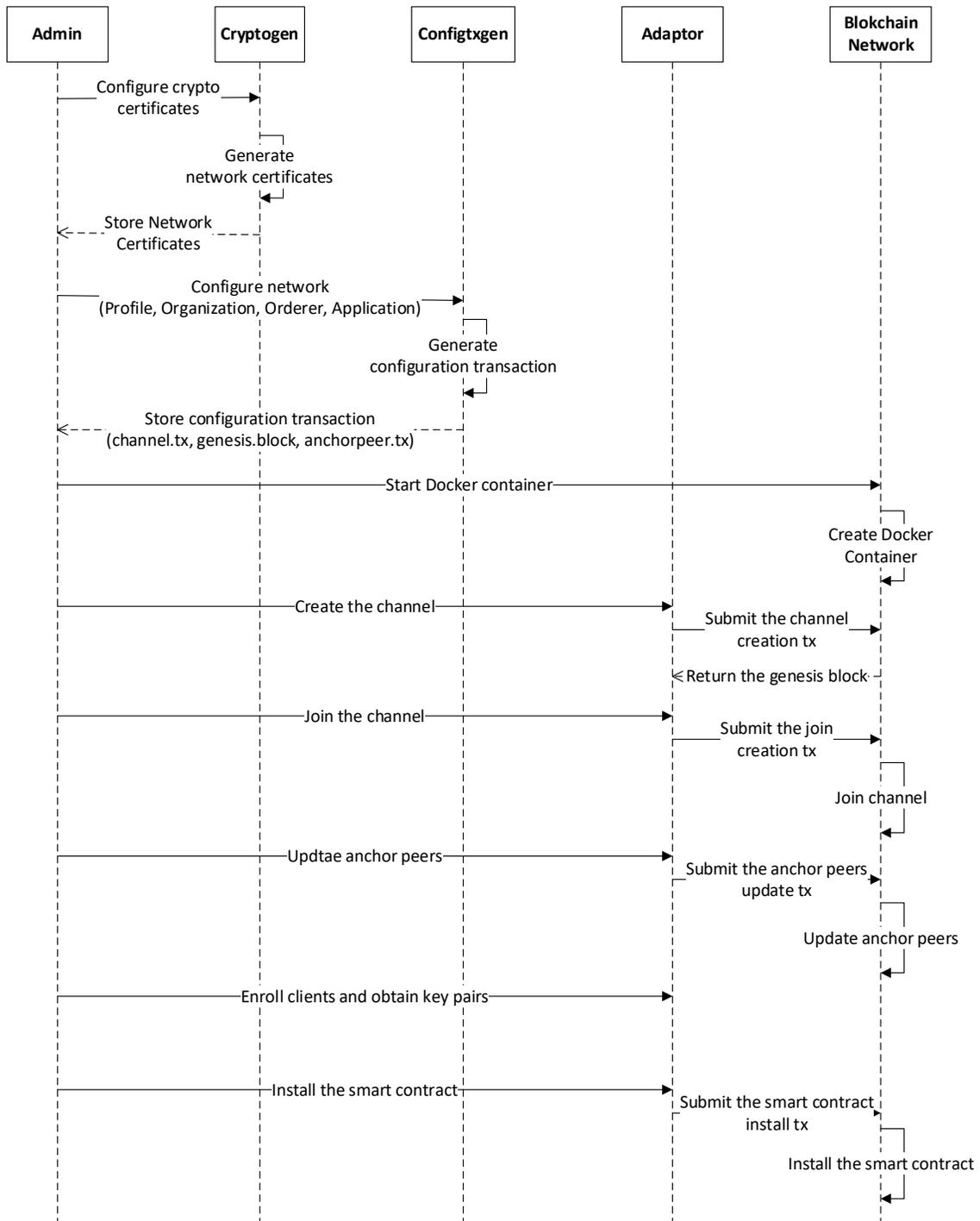


Figure 35: Sequence diagram of the blockchain network configuration

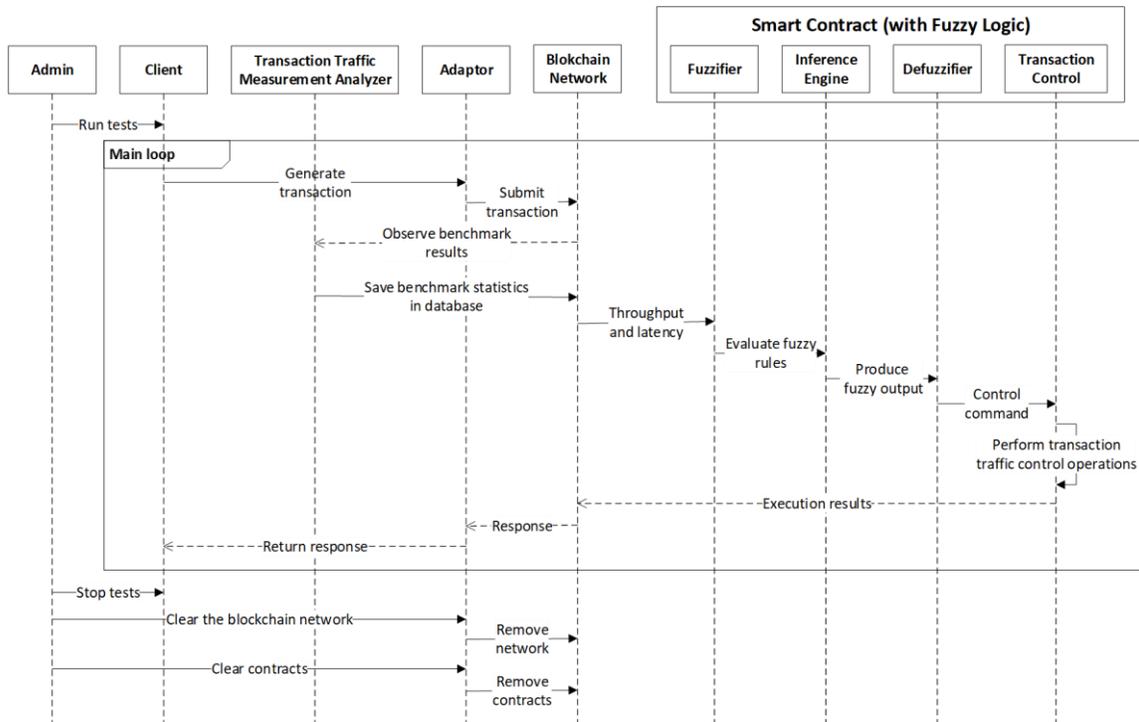


Figure 36: Sequence diagram of the transaction traffic control mechanism based on fuzzy logic

4.2 Development of the Transaction Traffic Control Mechanism based on Fuzzy Logic

Table 7 presents the technology stack used to implement transaction traffic control based on fuzzy logic. The Hyperledger Fabric (v1.4.1) is used as the blockchain infrastructure, which is deployed in the Ubuntu Linux (18.04 LTS) operating system. All the network elements of Hyperledger Fabric are encapsulated as Docker images in Docker containers, which are running in the virtual machine. The Node SDK enables interactions between external applications and the Fabric blockchain network via a group of APIs to submit transactions to the ledger or query content data from the ledger. Hyperledger Caliper (v2.0.0) is an open-source blockchain benchmark tool that allows users or developers to measure different performance indexes of blockchain

implementation. FuzzyIS is a JavaScript library for building a fuzzy inference system in smart contracts that utilize Node.js. MongoDB is a NoSQL database used to store the benchmark results in the JSON-like document with a schema. Express.js is a Node.js based web server framework to build web applications, which provides various REST APIs to manipulate the MongoDB.

Table 7: Development environment of transaction traffic control based on fuzzy logic

Component	Description
CPU	Intel Core i5-8500 @ 3.00 GHz
Memory	12 GB
OS	Ubuntu Linux 18.04 LTS
Docker Engine	v19.03.8
Docker-Composer	v1.24.0
SDK	Node.js v8.17.0
Blockchain Infrastructure	Hyperledger Fabric v1.4.1
TPS traffic measurement tool	Hyperledger Caliper V2.0.0
FIS Library	fuzzyIS
DBMS	MongoDB
Web Server	Express.js
Programming Language	JavaScript
IDE	VSCode

As shown in Figure 37, the fuzzy controller in the smart contract contains 4 core objects. Linguistic Variable initializes and adds input and output linguistic variables into the system, such as acceptance rate, transaction throughput, and network latency. The term describes fuzzy terms for each variable like high/low, very high/very low, etc. The rule describes the connection between input and output linguistic variables. These are conditions like: "if transaction throughput is very low AND network latency is very low, then accept rate should be very high," which describes how the system works. FIS – fuzzy inference system is created with input and output linguistic variables along with described rules. It calculates precise values for output variables referring to the rules given.

```

// fuzzy output
var acceptRate;

// describe new system, input and output variables
const system = new FIS('TPS control system');

// init and add variables into system

const ACCEPT_RATE = new LinguisticVariable('accept rate', [0, 100]);
system.addOutput(ACCEPT_RATE);

const TRANSACTION_THROUGHPUT = new LinguisticVariable('transaction throughput', [0, 200]);
const NETWORK_LATENCY = new LinguisticVariable('network latency', [0, 2]);

system.addInput(TRANSACTION_THROUGHPUT);
system.addInput(NETWORK_LATENCY);

// describe terms for each variable
TRANSACTION_THROUGHPUT.addTerm(new Term('very low', 'trapeze', [0, 0, 20, 60]));
TRANSACTION_THROUGHPUT.addTerm(new Term('low', 'triangle', [20, 60, 100]));
TRANSACTION_THROUGHPUT.addTerm(new Term('acceptable', 'triangle', [60, 100, 140]));
TRANSACTION_THROUGHPUT.addTerm(new Term('high', 'triangle', [100, 140, 180]));
TRANSACTION_THROUGHPUT.addTerm(new Term('very high', 'trapeze', [140, 180, 200, 200]));

NETWORK_LATENCY.addTerm(new Term('very low', 'trapeze', [0, 0, 0.15, 0.45]));
NETWORK_LATENCY.addTerm(new Term('low', 'triangle', [0.15, 0.45, 0.75]));
NETWORK_LATENCY.addTerm(new Term('acceptable', 'triangle', [0.45, 0.75, 1.05]));
NETWORK_LATENCY.addTerm(new Term('high', 'triangle', [0.75, 1.05, 1.3]));
NETWORK_LATENCY.addTerm(new Term('very high', 'trapeze', [1.05, 1.3, 1.5, 1.5]));

ACCEPT_RATE.addTerm(new Term('very low', 'trapeze', [0, 0, 10, 30]));
ACCEPT_RATE.addTerm(new Term('low', 'triangle', [10, 30, 50]));
ACCEPT_RATE.addTerm(new Term('medium', 'triangle', [30, 50, 70]));
ACCEPT_RATE.addTerm(new Term('high', 'triangle', [50, 70, 90]));
ACCEPT_RATE.addTerm(new Term('very high', 'trapeze', [70, 90, 100, 100]));

```

Figure 37: Fuzzy set definition in the smart contract

Figure 38 describes the structure of fuzzy rules in the smart contract. The rule preserves the same order in the term description. The first two values present the input variables, transaction throughput, and network latency, respectively. The third value presents the output variable accept rate. The fourth value represents the connection variable that can be and/or.


```

async Invoke(stub) {
  let url = "mongodb://192.168.0.24:27017/";

  MongoClient.connect(url, function (err, db) {
    if (err) throw err;
    var dbo = db.db("benchmarkdb");
    dbo.collection("benchmarks").findOne({}, { sort: { _id: -1 } }, function (err, result) {
      if (err) throw err;
      //console.log(result);
      acceptRate = system.getPreciseOutput([result.throughput, result.latency]);
      console.log("acceptRate: " + acceptRate);
      db.close();
    });
  });

  var funcAndParams = stub.getFunctionAndParameters();

  var method = this[funcAndParams.fcn];
  if (!method) {
    return getErrorResponse('Invoke', ERROR_WRONG_FORMAT);
  }

  try {
    return await method(stub, funcAndParams.params, acceptRate);
  } catch (err) {
    return getErrorResponse('Invoke', ERROR_SYSTEM, err);
  }
}

```

Figure 39: Function to invoke fuzzy inference system in the smart contract

```

// Drop the tx
if (acceptRate >= 0 && acceptRate < 30) {
  try {
    console.log("Drop the transaction!");
    // expand enumerable Buffer to byte array with the ... operator
    //await stub.putState(account, Buffer.from(params[1]));
  } catch (err) {
    return getErrorResponse('open', ERROR_SYSTEM, err);
  }
}

```

Figure 40: Sample of transaction control policy in terms of the acceptance rate

4.3 Performance Analysis of the Transaction Traffic Control Mechanism based on Fuzzy Logic

This section illustrates the evaluation results of the transaction traffic control mechanism based on fuzzy logic compared to the baseline network. The default block size is set to 10 transactions per block, and a new block is formed every 250 ms. The default ordering service is in solo mode, which consists only of a single ordering node. The LevelDB is used as the default state

database in this experiment. The evaluation tests presented in this section were averaged over multiple rounds to reduce errors resulting from the network congestion. Figure 41 plots the evaluation results of transaction throughput in a 1org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the throughput of each block size set was 152.9 tps and 157.7 tps, with a 3.1% increase of transaction throughput. Figure 42 plots the evaluation results of transaction latency in a 1org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 80 ms and 100ms, with a 20% reduction of transaction latency.

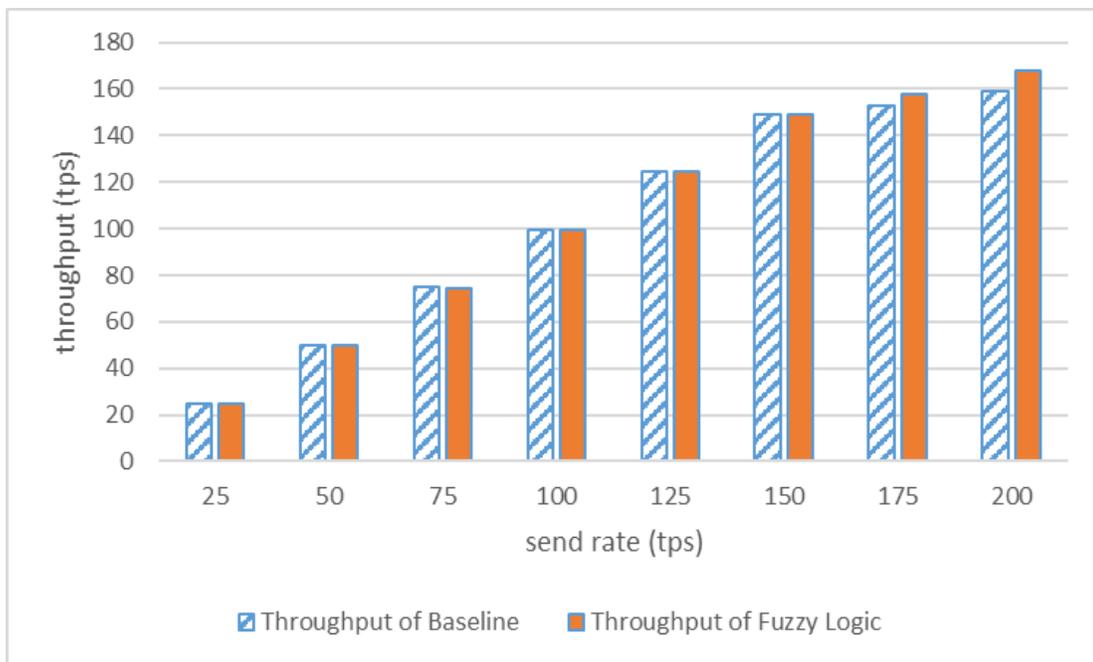


Figure 41: Evaluation of transaction throughput in 1org2peer network with 1 client (baseline and fuzzy logic)

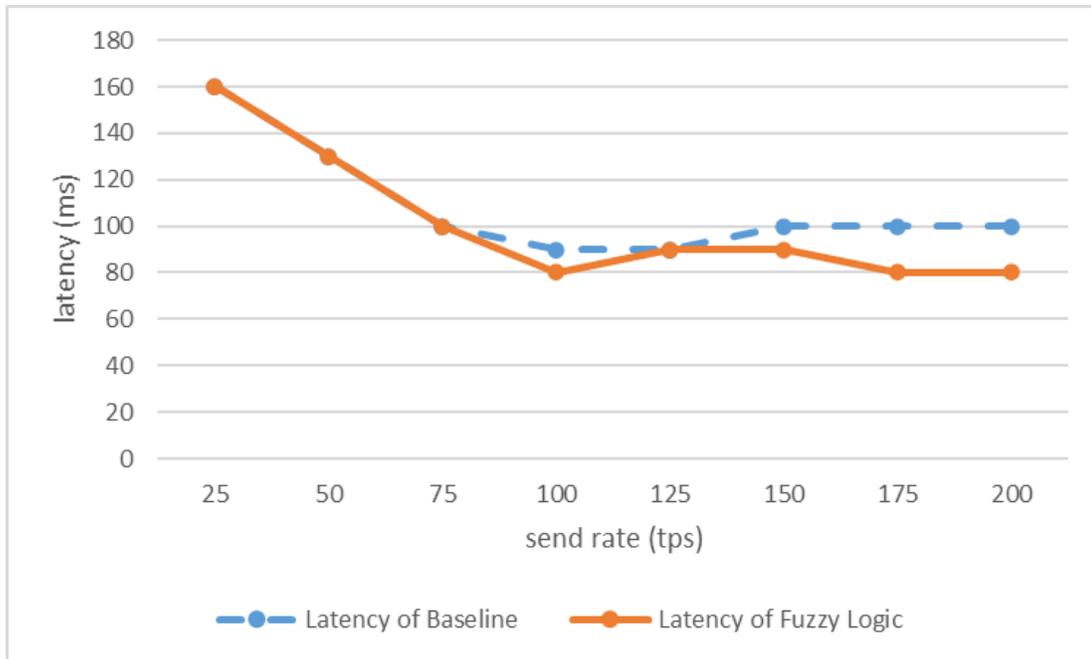


Figure 42: Evaluation of transaction latency in 1org2peer network with 1 client (baseline and fuzzy logic)

Figure 43 plots the evaluation results of transaction throughput in a 2org1peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it flattened out at around 150 tps, as shown in Figure 43. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the transaction throughput of each block size set was 132.6 tps and 141.1 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 6.4% compared to the baseline. Figure 44 plots the evaluation results of transaction latency in a 2org1peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 100 ms and 150 ms, with a 33.3% reduction of transaction latency.

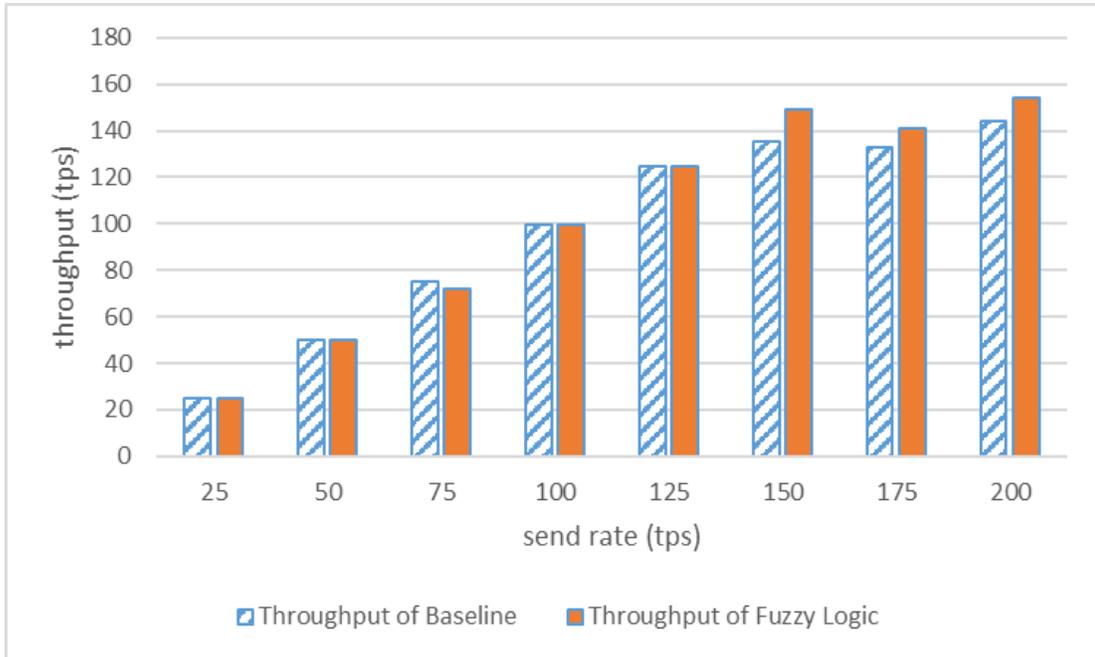


Figure 43: Evaluation of transaction throughput in 2org1peer network with 1 client (baseline and fuzzy logic)

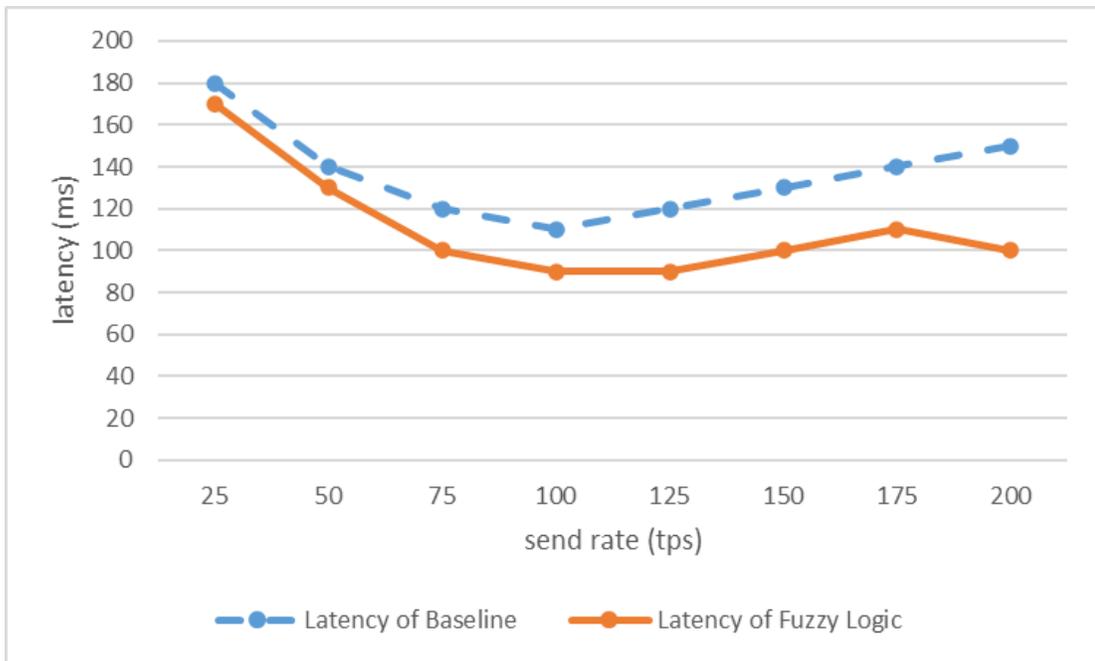


Figure 44: Evaluation of transaction latency in 2org1peer network with 1 client (baseline and fuzzy logic)

Figure 45 plots the evaluation results of transaction throughput in a 2org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 132.7 tps and 135.5 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 2.1% compared to the baseline. Figure 46 plots the evaluation results of transaction latency in a 2org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 130 ms and 170 ms, with a 23.5% reduction of transaction latency.

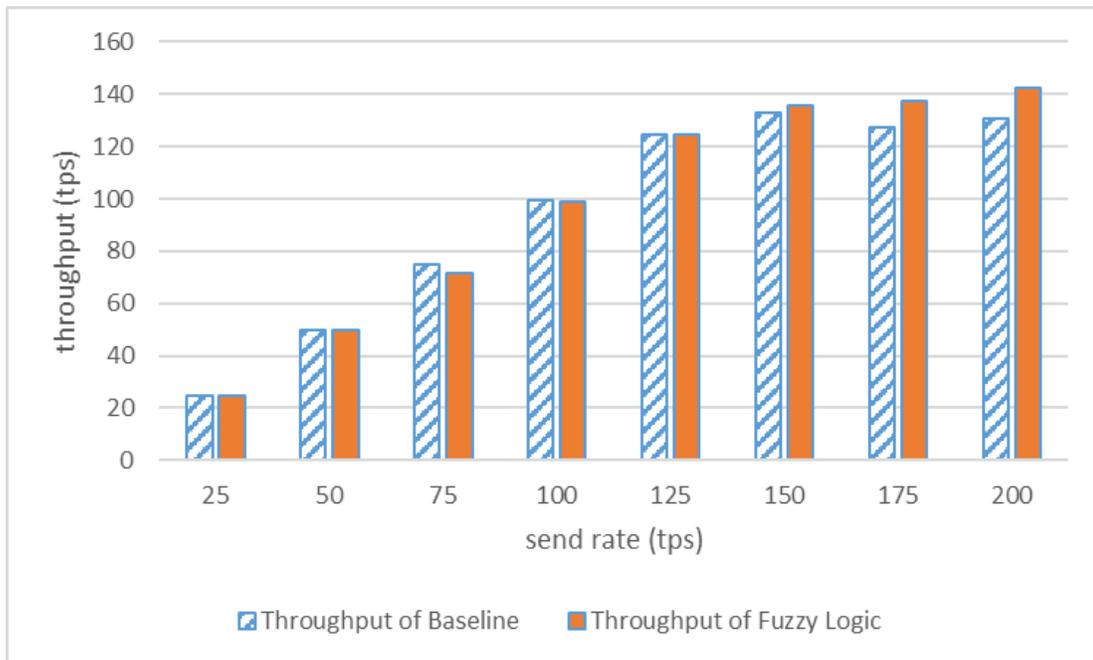


Figure 45: Evaluation of transaction throughput in 2org2peer network with 1 client (baseline and fuzzy logic)

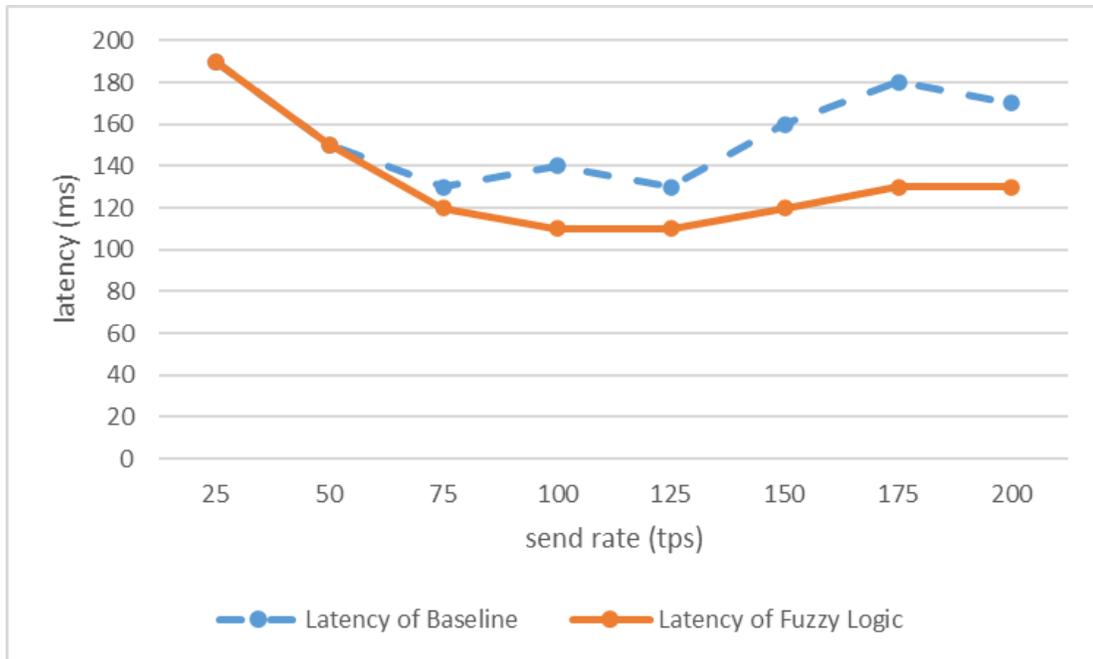


Figure 46: Evaluation of transaction latency in 2org2peer network with 1 client (baseline and fuzzy logic)

Figure 47 plots the evaluation results of transaction throughput of a 3org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 112.2 tps and 122.3 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 9% compared to the baseline. Figure 48 plots the evaluation results of transaction latency of a 3org2peer network with 1 client by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 230 ms and 320 ms, with a 34.4% reduction of transaction latency.

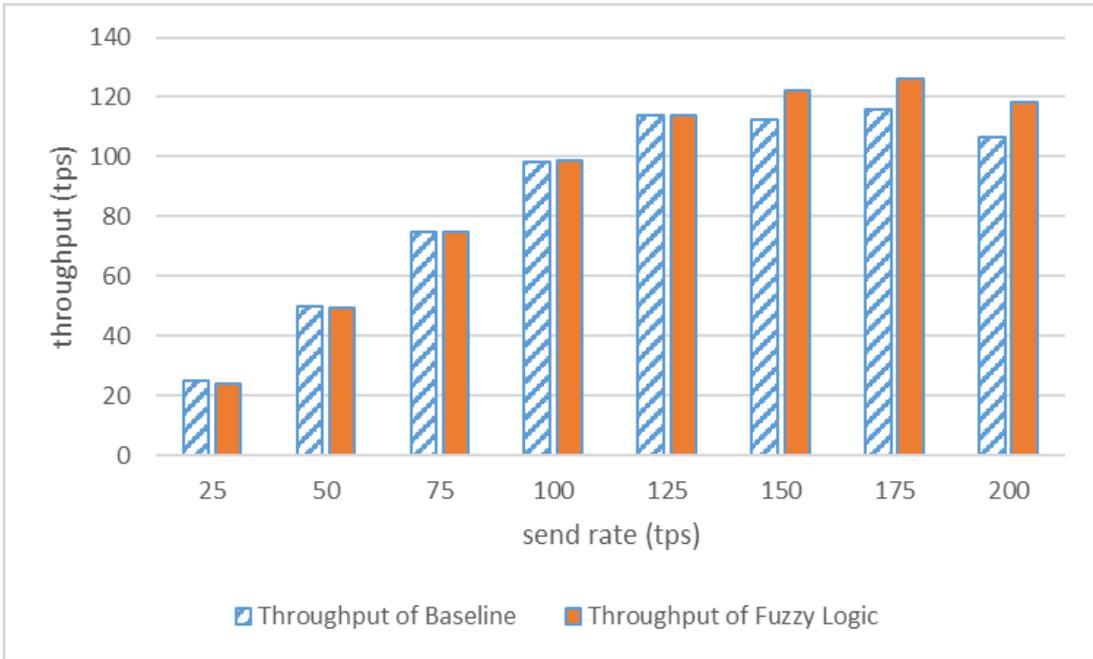


Figure 47: Evaluation of transaction throughput in 3org2peer network with 1 client (baseline and fuzzy logic)

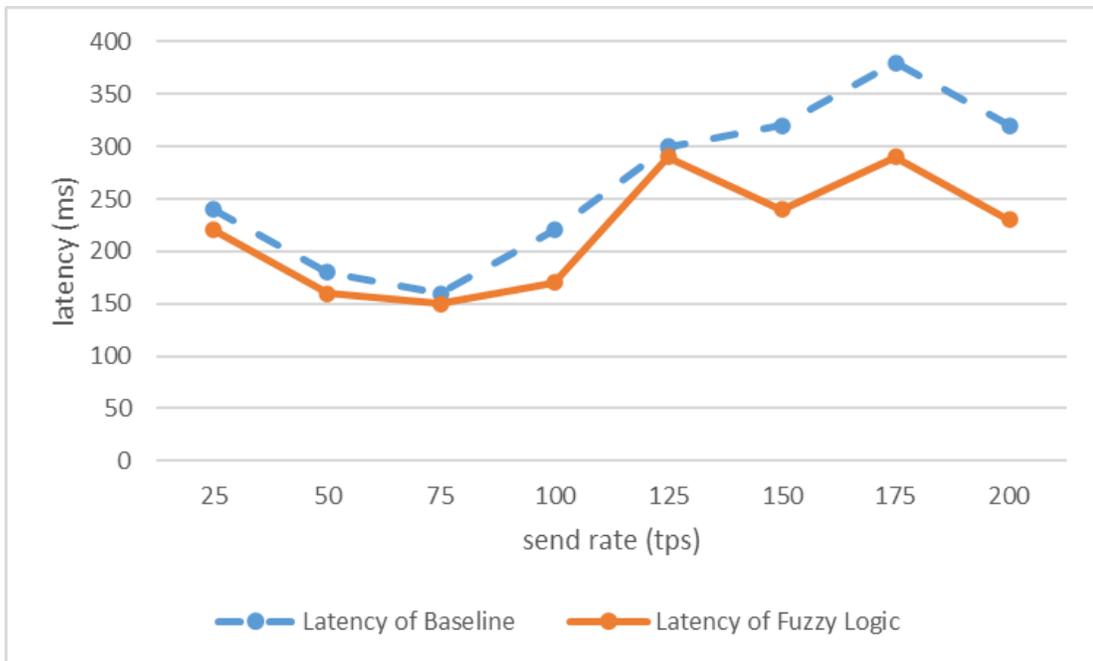


Figure 48: Evaluation of transaction latency in 3org2peer network with 1 client (baseline and fuzzy logic)

Figure 49 plots the evaluation results of transaction throughput in a 1org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). For the case of the baseline network, the transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the transaction throughput of the baseline network and the fuzzy-based mechanism was 163.6 tps and 174.6 tps. Respectively. The fuzzy-based mechanism can increase transaction throughput by 6.7% compared to the baseline. Figure 50 plots the evaluation results of transaction latency in a 1org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 480 ms and 680 ms, with a 29.4% reduction of transaction latency.

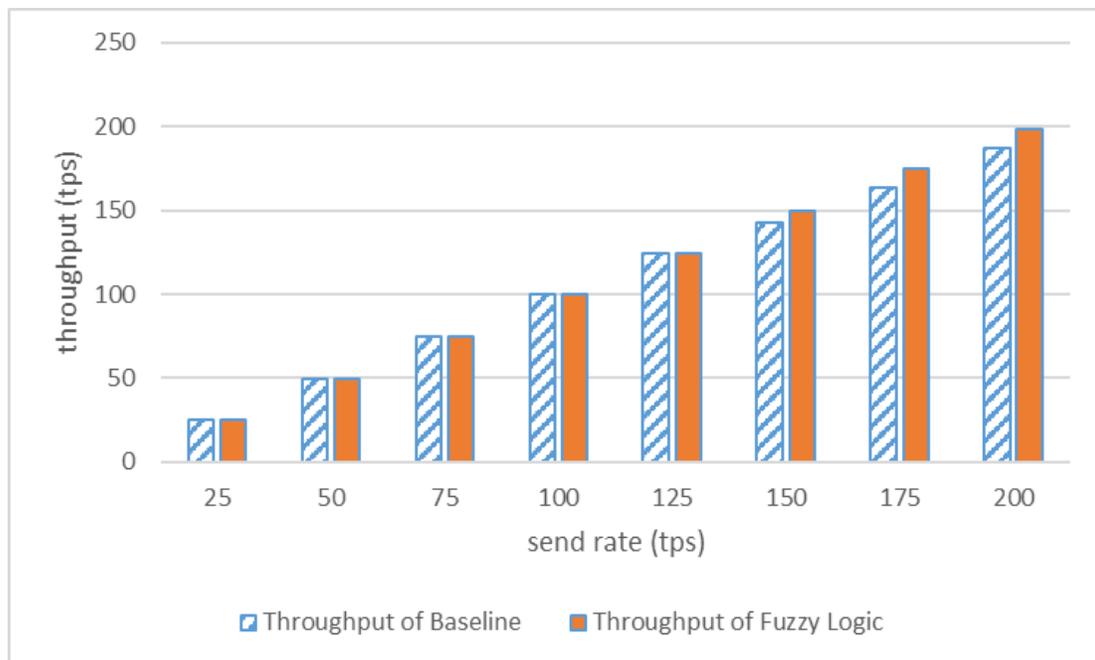


Figure 49: Evaluation of transaction throughput in 1org2peer network with 5 clients (baseline and fuzzy logic)

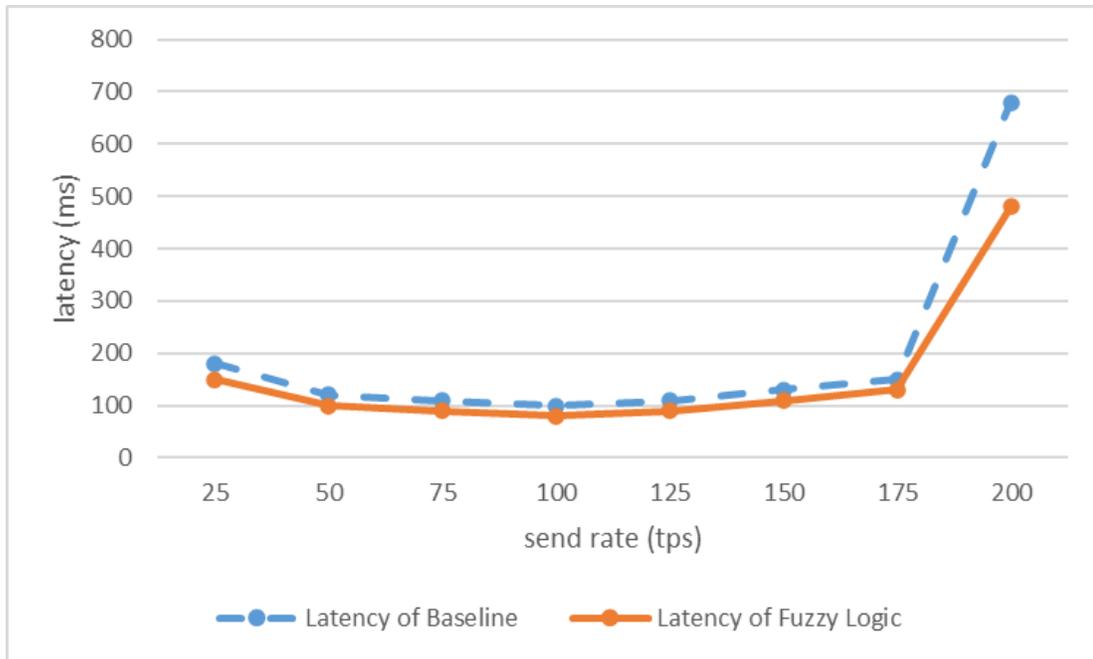


Figure 50: Evaluation of transaction latency in 1org2peer network with 5 clients (baseline and fuzzy logic)

Figure 51 plots the evaluation results of transaction throughput in a 2org1peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 135.1 tps and 149.1 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 9.4% compared to the baseline. Figure 52 plots the evaluation results of transaction latency in a 2org1peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 820 ms and 950 ms, with a 13.7% reduction of transaction latency.

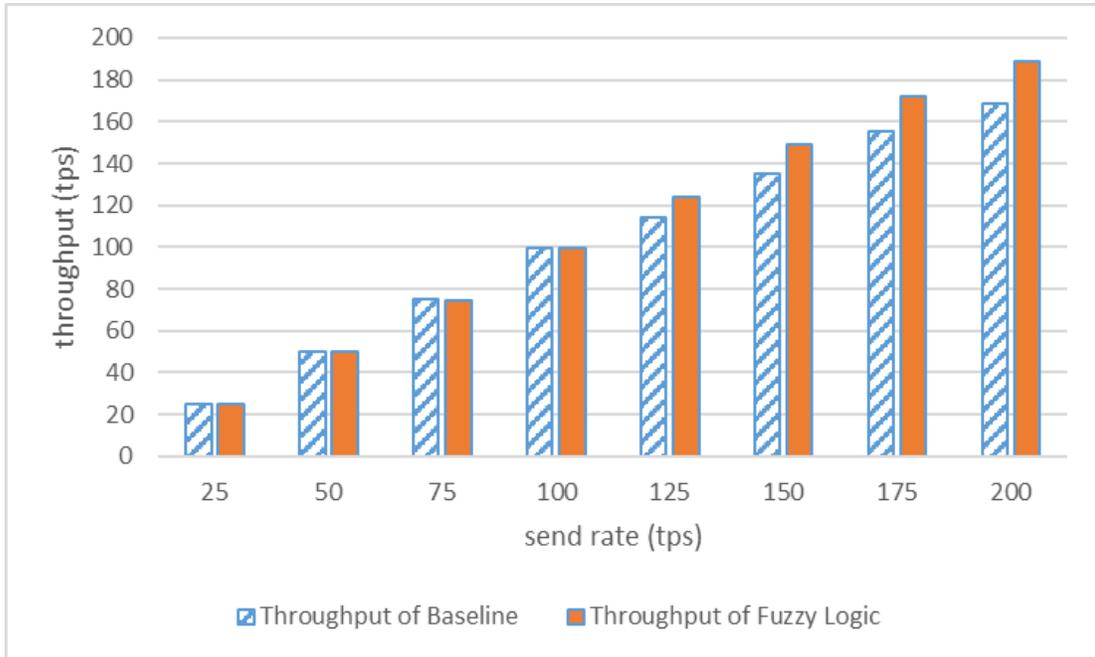


Figure 51: Evaluation of transaction throughput in 2org1peer network with 5 clients (baseline and fuzzy logic)

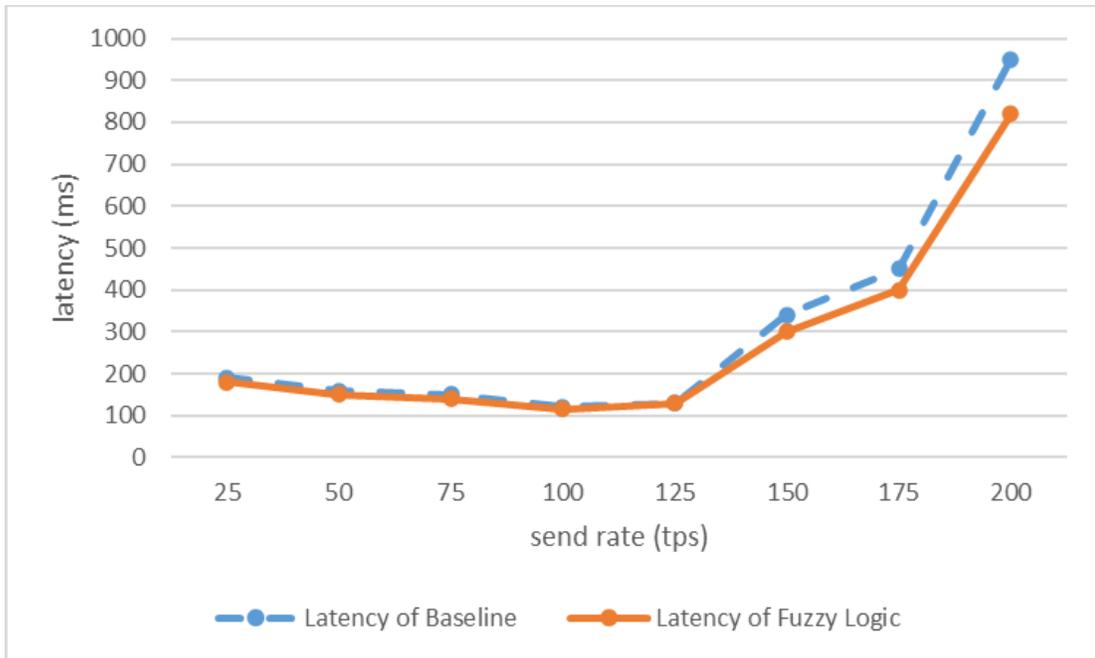


Figure 52: Evaluation of transaction latency in 2org1peer network with 5 clients (baseline and fuzzy logic)

Figure 53 plots the evaluation results of transaction throughput in a 2org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 128.6 tps and 138.4 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 7.6% compared to the baseline. Figure 54 plots the evaluation results of transaction latency in a 2org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 1760 ms and 2260 ms, with a 22.1% reduction of transaction latency.

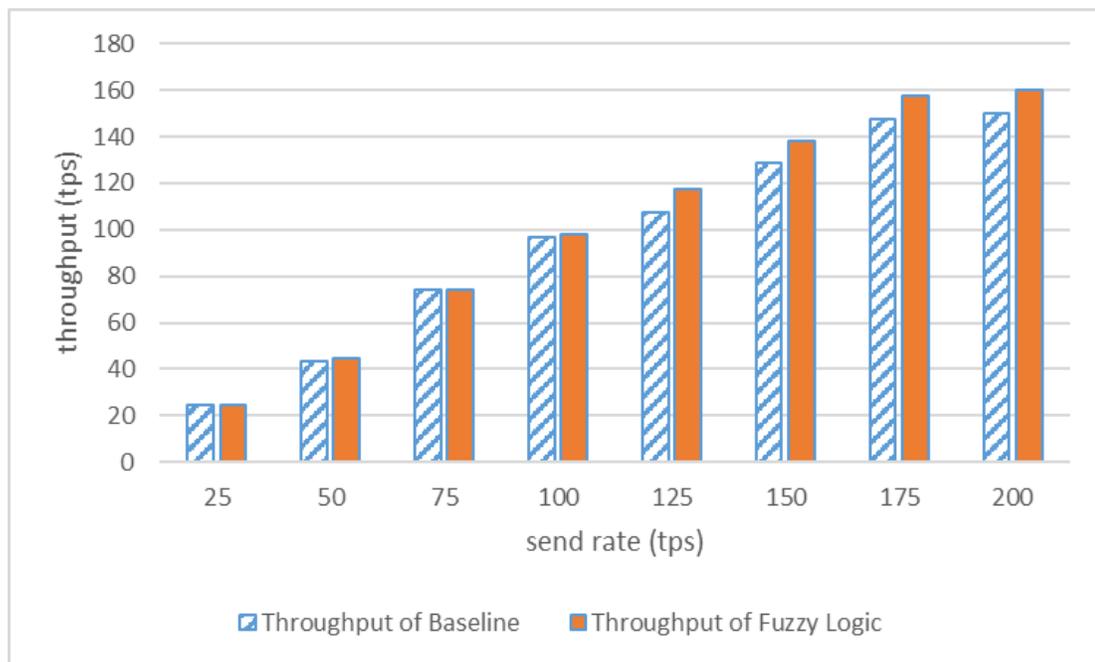


Figure 53: Evaluation of transaction throughput in 2org2peer network with 5 clients (baseline and fuzzy logic)

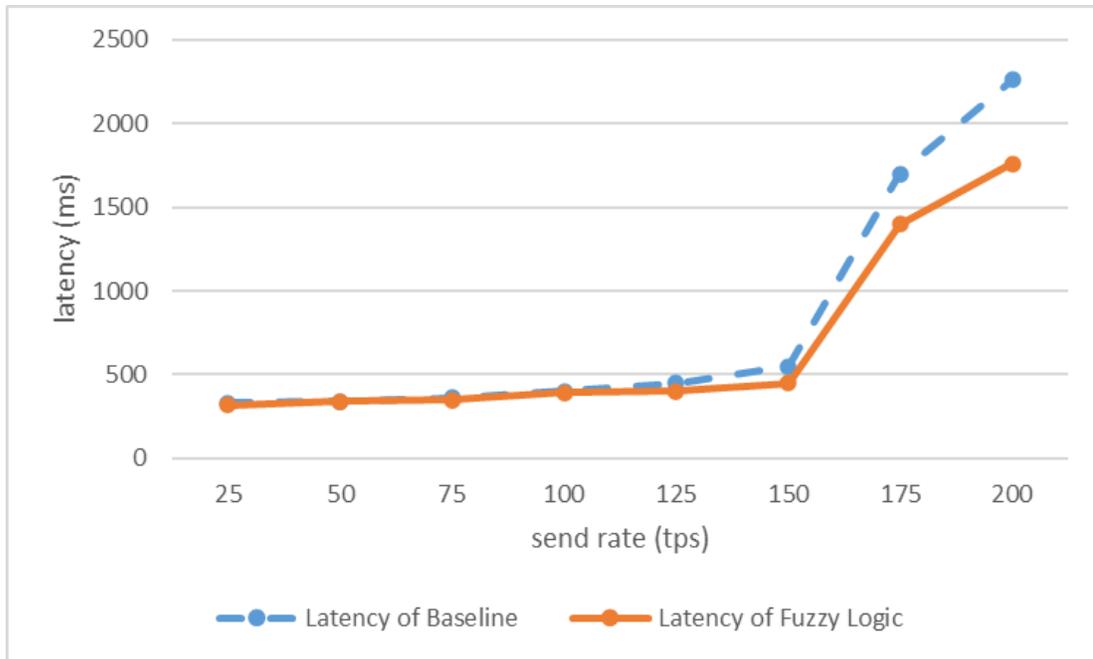


Figure 54: Evaluation of transaction latency in 2org2peer network with 5 clients (baseline and fuzzy logic)

Figure 55 plots the evaluation results of transaction throughput in a 3org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 100 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 125 tps, the transaction throughput of each block size set was 98.6 tps and 108.4 tps, respectively. The fuzzy-based mechanism can increase transaction throughput by 9.9% compared to the baseline. Figure 56 plots the evaluation results of transaction latency in a 3org2peer network with 5 clients by comparing the network using proposed fuzzy logic-based mechanism with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of fuzzy-based mechanism and the baseline was 2320 ms and 3020 ms, with a 23.2% reduction of transaction latency.

The proposed transaction traffic control approach based on fuzzy logic was tested in different network configurations by varying the network scale and number of clients. The experiment results indicate that the proposed transaction traffic control mechanism based on fuzzy logic can improve the blockchain performance concerning transaction throughput and transaction latency. In all cases, the evaluation using the fuzzy logic-based transaction traffic control mechanism outperforms the baseline by increasing the transaction throughput while decreasing the transaction latency.

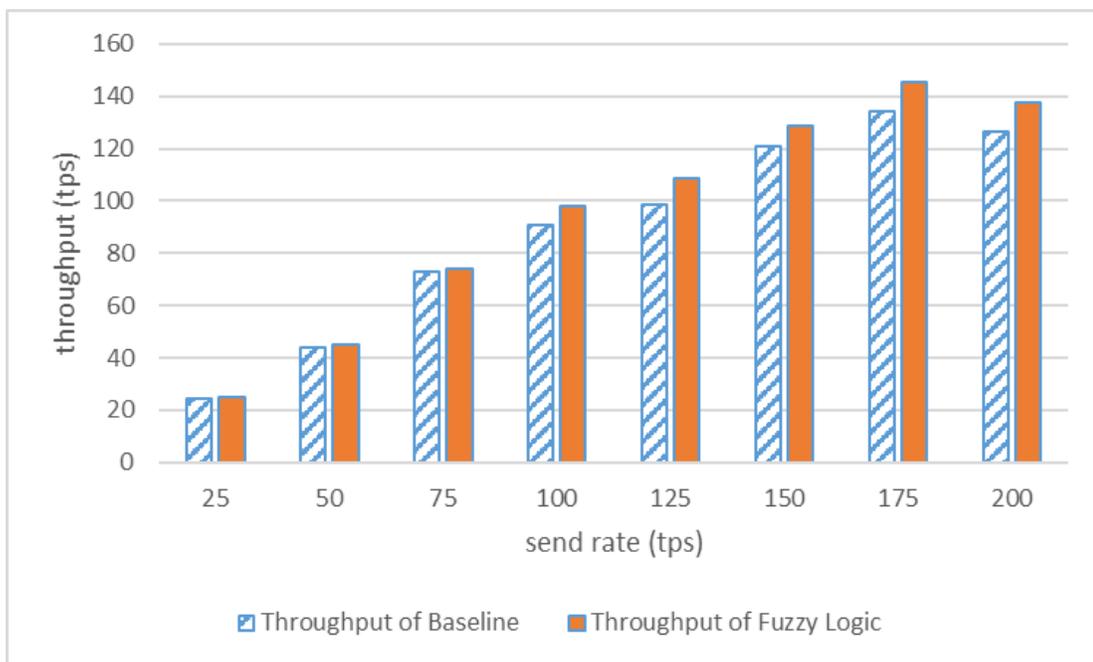


Figure 55: Evaluation of transaction throughput in 3org2peer network with 5 clients (baseline and fuzzy logic)

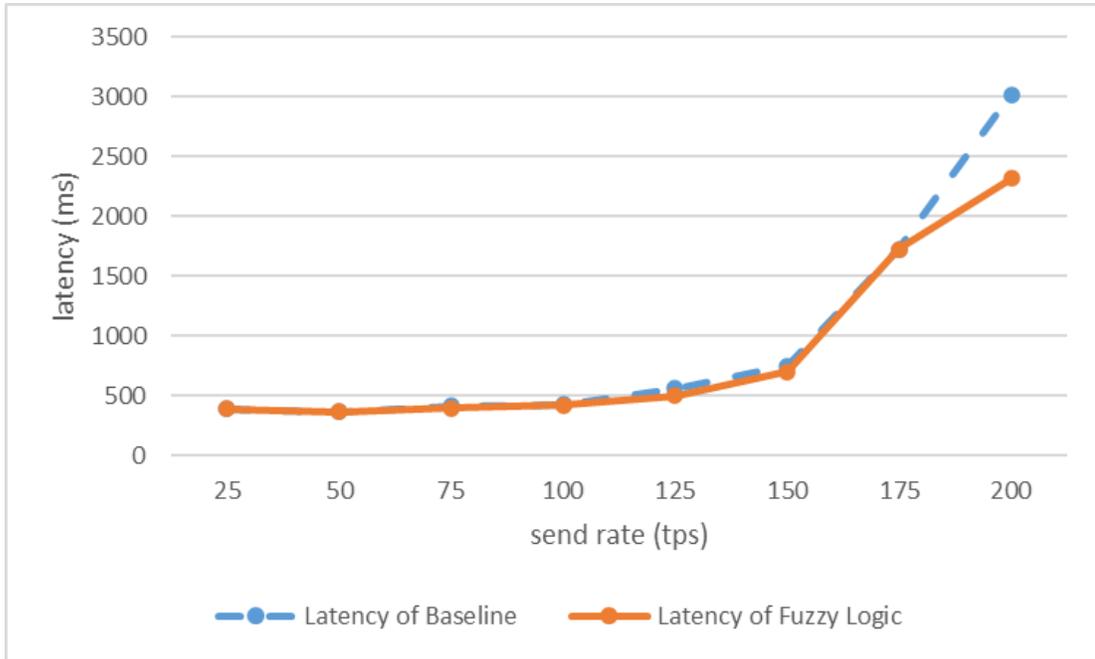


Figure 56: Evaluation of transaction latency in 3org2peer network with 5 clients (baseline and fuzzy logic)

5. Transaction Traffic Control Mechanism based on Learning to Prediction in Blockchain Network

5.1 Proposed Transaction Traffic Control Mechanism based on Learning to Prediction

The conceptual architecture of the transaction traffic control mechanism based on learning to prediction is described in Figure 57. The concept of the learning to prediction is first proposed in [49] to improve the prediction accuracy of temperature readings due to the changeable humidity level in the case study of a greenhouse. This paper utilizes this idea to improve the prediction accuracy of network conditions from the noisy network measurement environment. As the learning to prediction module performs data training that is performance sensitive and time-consuming, it is not a proper way to directly deploy this module into the smart contract. The blockchain network is comprised of various nodes, which provide the host environment of smart contracts and hold a copy of the distributed ledger to maintain the consistency of the whole network. The learning to prediction module is an external module that can make connections with the tps traffic measurement module and the blockchain network. The transaction traffic measurement module consists of the adaptor and the performance analyzer. The clients can submit transactions by invoking the functions specified in the smart contract. The learning module is used to tune the parameter of the prediction module to improve the prediction policy. The learning module takes the historical data such as network conditions observed in previous. Based on the history data, the learning module establishes the forecasting model to predict the tuning parameter dynamically. The predicted value of the prediction module is used as the input parameter of the fuzzy controller. Similar to the first mechanism, the fuzzy controller is also implemented in the smart contract to automate the process of transaction traffic flow across the network. The fuzzy

controller consists of the fuzzy inference system and the transaction control modules. Benchmark results of the network are observed in real-time, and these values are transmitted to the smart contract. The fuzzy controller computes the control commands to make decisions on the received transactions. The consensus is achieved within the whole network, and the execution results are returned to the clients.

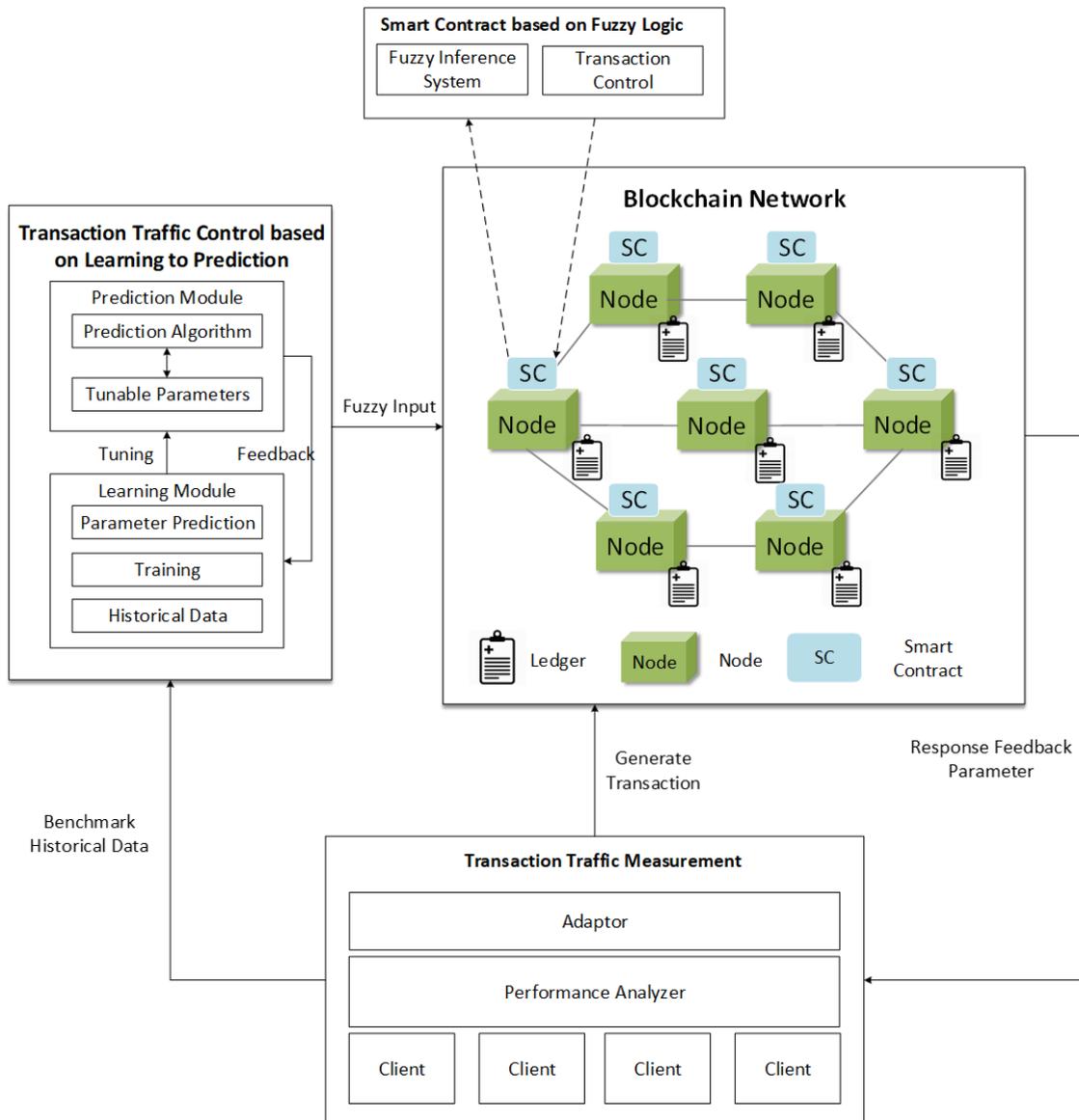


Figure 57: Conceptual architecture of the transaction traffic control mechanism based on learning to prediction

Figure 58 illustrates the detailed architecture of the proposed system, which is comprised of the admin, transaction traffic measurement analyzer, benchmark DB, predicted result DB, learning to prediction module, and the blockchain network. Based on the fuzzy-based approach, learning to prediction module is additionally implemented to improve the performance of the fuzzy controller. Kalman Filter-based prediction model is used to estimate the transaction throughput in the next stage. The ANN-based learning model is used to optimize the prediction algorithm by tuning the parameter of the Kalman Filter. The predicted throughput values are persisted in the predicted result DB. The proposed fuzzy controller adjusts the transaction acceptance rate of the smart contract by comparing transaction throughput, with the acceptance rate. The fuzzy controller only has one single input parameter, the predicted throughput. The inference engine evaluates the input against predefined fuzzy rules. The defuzzifier converts output data (acceptance rate) into non-fuzzy values. The output value is obtained by the transaction control module to adjust the transaction acceptance rate. The whole process is repeated, and the throughput of the Fabric network can be dynamically maintained at a suitable level.

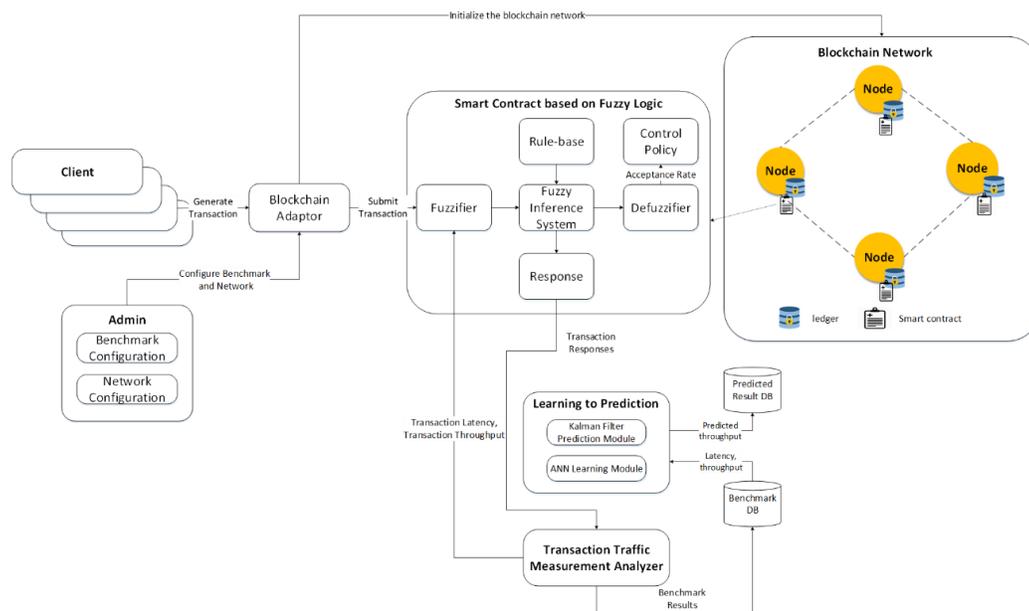


Figure 58: Development configuration of the transaction traffic control mechanism based on learning to prediction

The training module is used to adjust the prediction rule to enhance its performance concerning the prediction accuracy, as shown in Figure 59. We tend to use the Kalman Filter as the prediction module, and the artificial neural network (ANN) as the learning module. The Kalman Filter is employed to predict the transaction throughput of the network from a noisy environment. Noise in the environment is introduced in a situation wherever the transaction latency profoundly impacts transaction throughput. Transaction throughput, transaction latency, and the actual transaction throughput are used as input parameters for the ANN-based learning module. The Kalman Filter obtains the transaction throughput at time t , i.e., z_t , and can predict the transaction throughput T_t by eliminating noise. The performance of the Kalman Filter algorithm is mainly affected by a configurable parameter called Kalman gain (K), which is renewed on each new iteration in terms of the variance matrix (P) and the calculable error (R). The ANN-based learning module aims to estimate the calculable error (R) to update the Kalman gain (K) dynamically.

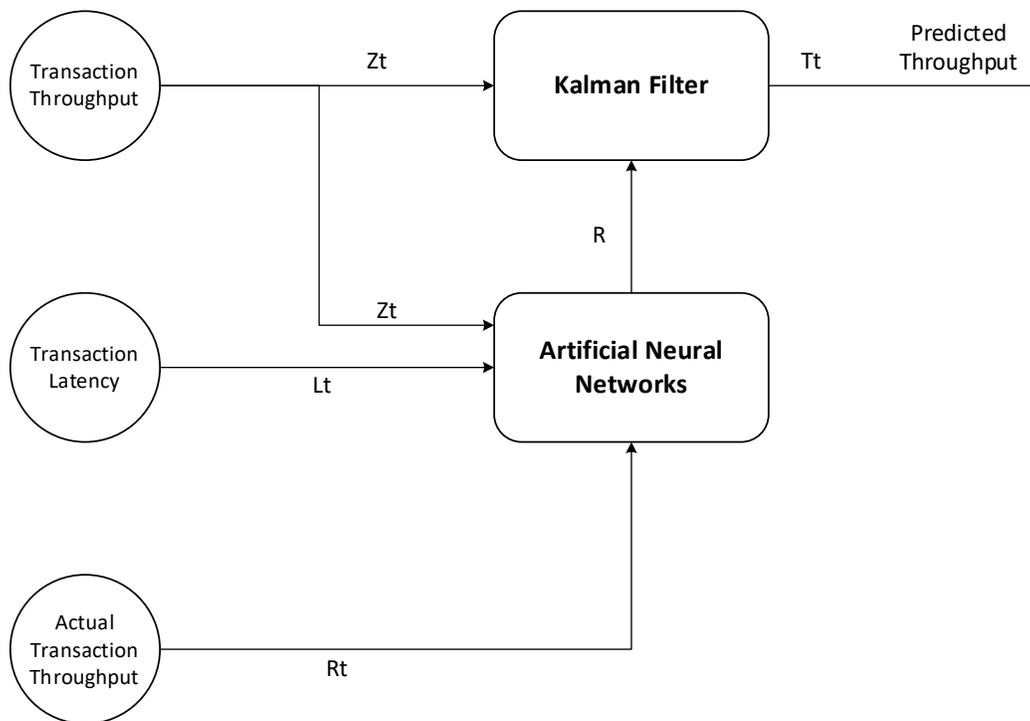


Figure 59: Overview architecture of the learning to prediction module

The Kalman Filter can predict the actual state of the system concerning only the previous state information. It updates the value of Kalman gain (K) in terms of the condition to regulate weights given to the system's own estimated state or sensing values. The essential parts and workflow of the Kalman Filter are described in Figure 60.

Every blockchain network has its noise factors, which may seriously affect throughput measurement. In this paper, we tend to contemplate a throughput measurement having noise and allow us to assume T_t is that the transaction throughput at time t. The Kalman Filter contains the model that can make a prediction of the system state, i.e., estimated transaction throughput, and then, this value is compared to the current measured transaction throughput value to predict the transaction throughput T_{t+1} at time t+1.

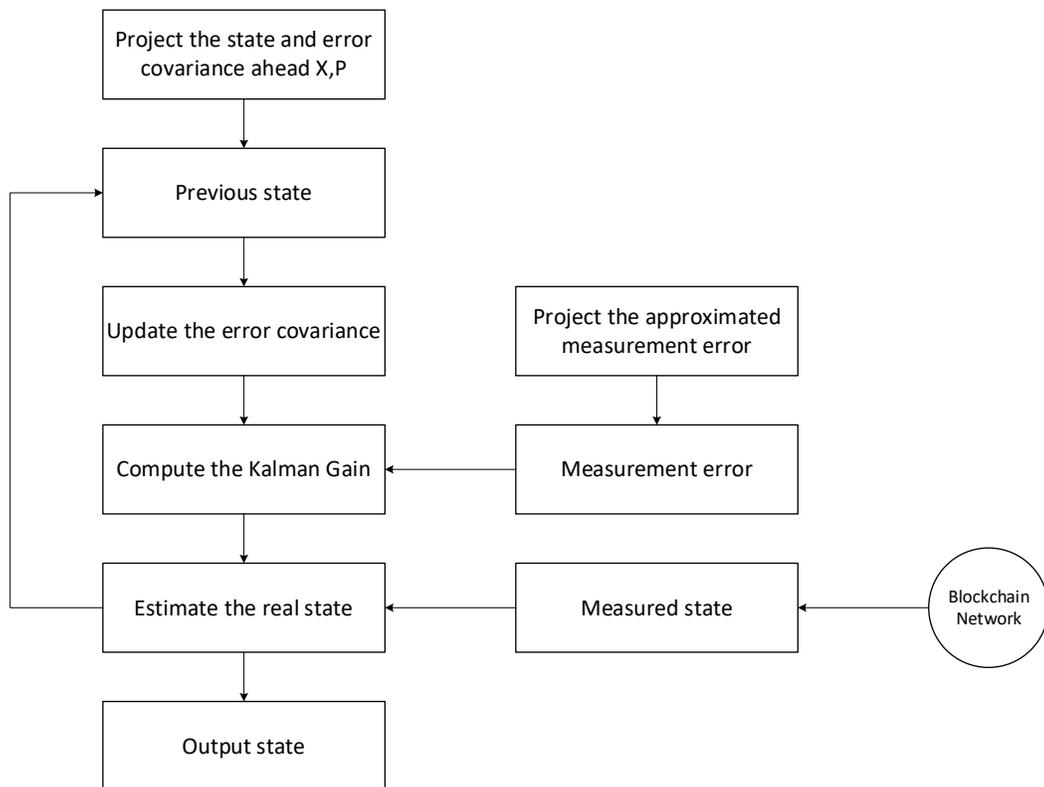


Figure 60: Flow chart of the transaction throughput prediction using Kalman Filter

In the commencement, the estimated transaction throughput is computed from the antecedently estimated value by equation (5):

$$T_k = A \cdot T_{k-1} + B \cdot u_k \quad (5)$$

T_k represents the estimated transaction throughput, A , and B present the state transition and control matrixes, respectively. T_{k-1} represents the transaction throughput at time $k-1$, and u_k is the control vector. The estimated transaction throughput T_k is determined by P_k , which represents the covariance factor.

$$P_k = A \cdot P_{k-1} \cdot A^T + Q \quad (6)$$

A and A^T represent the state transition matrix as well as its transpose, and P_{k-1} is the previous covariance value with a process error Q . The estimated transaction throughput and the updated covariance value are used to compute the Kalman gain (K), expressed in equation (7):

$$K_k = \frac{P_k \cdot H^T}{H \cdot P_k \cdot H^T + R} \quad (7)$$

H and H^T are the observation matrix as well as its transpose, and the measurement error is represented by R . The current measured transaction throughput at time k is presented as z_k . The updated transaction throughput for the next stage is calculated, as expressed in equation (8):

$$T_e = T_k + K_k (z_k - H \cdot T_k) \quad (8)$$

The covariance value is updated for the next iteration by equation (9):

$$P_e = (I - K_k \cdot H) P_k \quad (9)$$

Figure 61 details the architecture of the learning to prediction module to control the transaction traffic flow. Three parameters, transaction latency, transaction throughput, and actual transaction throughput, are used as inputs for the ANN-based learning module. The ANN algorithm predicts

the transaction throughput measurement error; in turn, this value is divided by a constant factor (F) to calculate the estimated error R, which is passed to the Kalman Filter. The Kalman gain (K) is updated accordingly to adjust the accuracy of the prediction module.

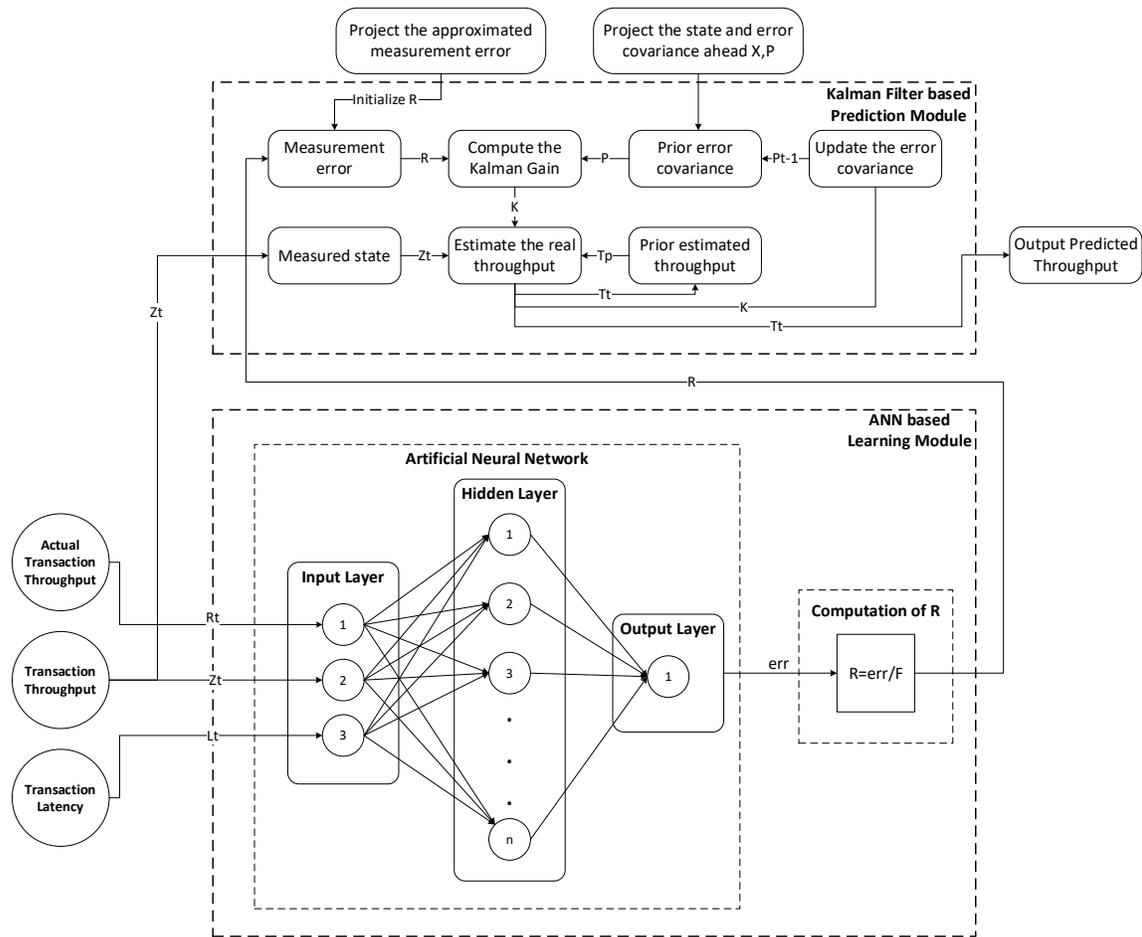


Figure 61: Detailed diagram of the learning to prediction module [49]

Table 8 describes the rule definitions for the learning to prediction module. The fuzzy controller for the learning to prediction just takes one input parameter, predicted transaction throughput. As a consequence, 5 rules are built for the fuzzy controller in total.

Table 8: Fuzzy rules definition for learning to prediction

Predicted Transaction Throughput	Acceptance Rate
Very Low	Very High
Low	High
Acceptable	Medium
High	Low
Very High	Very Low

Figure 62 describes the workflow of the proposed transaction traffic control mechanism based on learning to prediction. At the beginning of each test, the user should configure the network and benchmark profiles to fulfill the requirements of the test scenario. The benchmark file describes how the evaluation test should be executed, including the number of rounds, send rate of the transaction, and settings about monitoring the test network. The network configuration file describes the topology of the test network, such as the configuration of nodes, number of clients, and smart contracts deployed to the test network.

After configuring the network and benchmark profiles, the user can start the test. Afterward, the clients submit transactions to the network, and the benchmark results are observed. These results are further analyzed to compute the transaction throughput and latency, which are stored in an external data storage. These two parameters are used as the input parameters ANN-based learning module to predict the measurement error. The estimated error is used to tune the measurement error factor of the Kalman Filter algorithm. Actual transaction throughput is predicted and stored in the database. The smart contract retrieves the actual transaction throughput as the input parameter. The fuzzy inference engine evaluates the input parameters according to defined fuzzy rules. The fuzzy controller produces the acceptance rate as the output value, which is used to control the transaction traffic flow. The transaction execution response is generated and returned to the client. This process is repeated across the entire benchmark experiment until the user stops the test. Finally, all of the network entities and the smart contract will be removed.

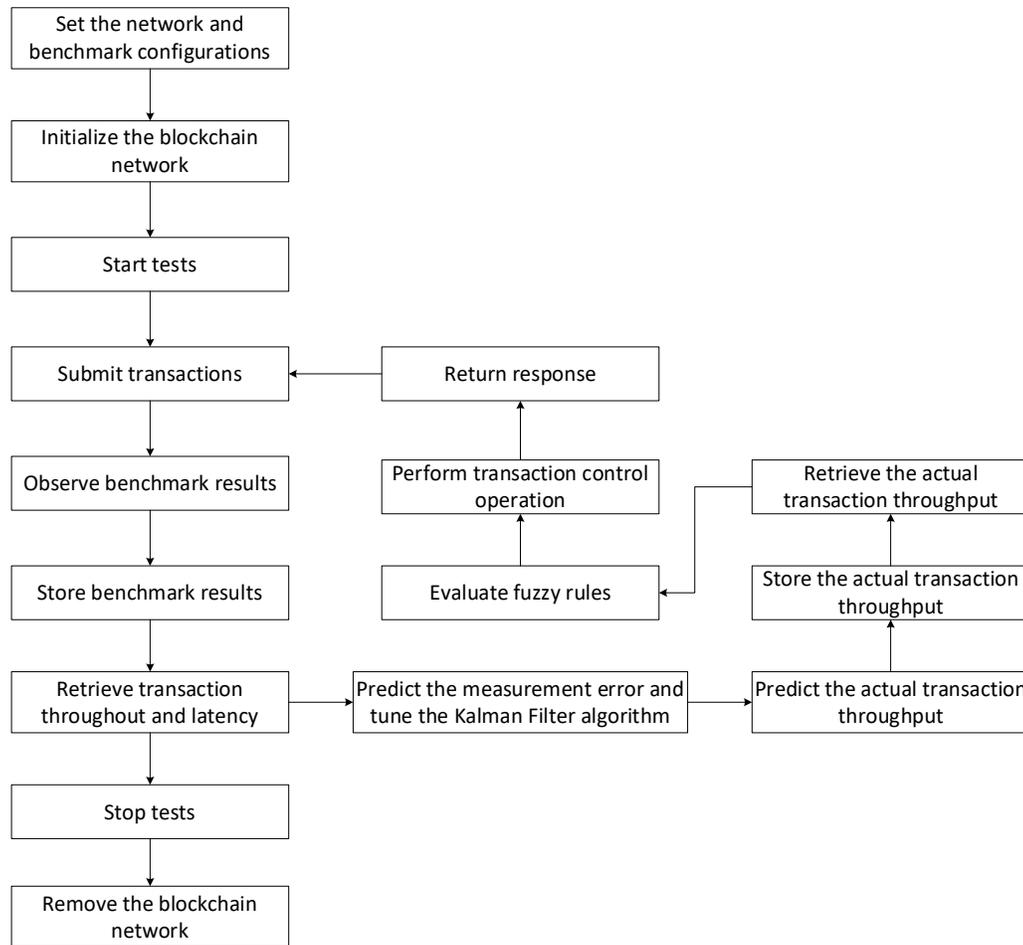


Figure 62: Flow chart of the transaction traffic control based on learning to prediction

Figure 63 illustrates the execution process of the transaction traffic control mechanism based on learning to prediction. First of all, the admin needs to configure the network and benchmark files to set up the blockchain network. The blockchain network will be initialized, and the smart contract will be installed according to the network configuration. Afterward, the admin can start the script to start the benchmark test. One or more clients generate transactions to the adaptor, in turn, the adaptor submits transactions to the Fabric network. Meanwhile, the benchmark results are observed and collected by the performance analyzer. The analyzer calculates the benchmark statistics and stores the results in the benchmark DB. The learning module retrieves the transaction latency and transaction throughput values to predict the measurement error used for tuning the Kalman Filter algorithm. The actual transaction throughput is estimated and stored in the predicted

result database. The fuzzifier obtains the predicted transaction throughput as the input parameter of the fuzzy inference system. The inference engine evaluates the input parameter according to the fuzzy rules. The defuzzifier produces the acceptance rate as the output value and sends this value to the transaction control module. The transaction module performs transaction traffic control operations with respect to the acceptance rate. The transaction execution response is generated and returned to the client. This process is repeated across the entire benchmark experiment until the user stops the test. Finally, all of the network entities and the smart contract will be removed.

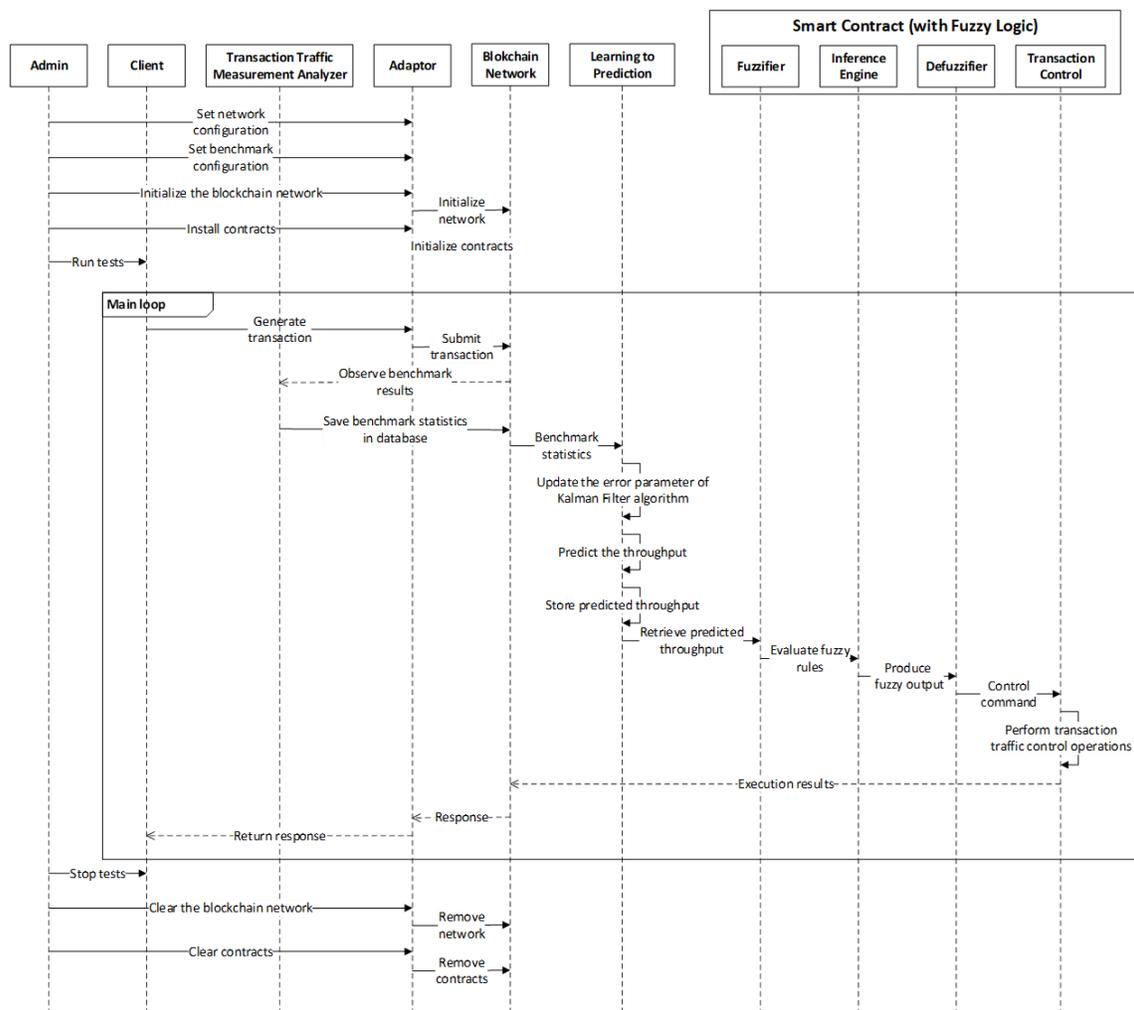


Figure 63: Sequence diagram of the transaction traffic control mechanism based on learning to prediction

5.2 Development of the Transaction Traffic Control Mechanism based on Learning to Prediction

Table 9 presents the technology stack used to implement the transaction traffic control based on learning to prediction. Similar to the fuzzy-based approach, the Hyperledger Fabric is used as the blockchain infrastructure. Hyperledger Caliper (v2.0.0) is used to measure different performance indexes of the blockchain implementation. The learning to prediction module is developed in Visual Studio Community with C#. For implementing the ANN-based learning module, Accord. Neuro is used. The Newtonsoft.Json is a JSON framework for .NET.

Table 9: Development environment of transaction traffic control based on learning to prediction

Component	Description
CPU	Intel Core i5-8500 @ 3.00 GHz
Memory	12 GB
OS	Ubuntu Linux 18.04 LTS
Docker Engine	v19.03.8
Docker-Composer	v1.24.0
SDK	Node.js v8.17.0
Blockchain Infrastructure	Hyperledger Fabric v1.4.1
TPS traffic measurement tool	Hyperledger Caliper V2.0.0
Library	fuzzyIS, Accord. Neuro, Newtonsoft.Json
DBMS	MongoDB
Programming Language	JavaScript, C#
IDE	VSCode, Visual Studio Community

As shown in Table 10, the learning to prediction dataset consists of four features: network latency, send rate, transaction throughput, and the error. To make this dataset, we utilize the Hyperledger Caliper and modify the sample benchmark configuration file to meet our scenario. Each row describes the benchmark statistics in one minute, and in total, this dataset contains 10080 rows, which represent the performance profile in a week. The transaction latency measures the time for the entire network to validate a transaction, including the propagation time and any

settling time due to the consensus in place. The send rate is the rate at which clients submit transactions. The transaction throughput is the rate at which the blockchain commits valid transactions in a defined time period. The error presents the difference between the send rate and transaction throughput.

Table 10: Dataset for the learning to prediction module

No	Network Latency (s)	Send Rate (tps)	Transaction Throughput (tps)	Error (tps)
1	0.22	146.5	145.8	0.7
2	0.22	146.9	146.3	0.6
3	0.19	151.3	150.7	0.6
4	0.18	152.1	151.4	0.7
5	0.19	151.1	150.3	0.8
6	0.17	161.9	161.1	0.8
7	0.15	158	157.3	0.7
8	0.16	157.6	156.9	0.7
9	0.19	161.4	160.5	0.9
10	0.13	160.6	159.8	0.8
...
...
10080	0.88	161.1	159.7	1.4

To set up the best training module for the ANN, different configurations are tested by varying the quantity of neurons within the hidden layer, learning rates, and activation functions. For each network configuration, experiments were conducted in multiple rounds for training, and average results are recorded to analyze the random factor for initializing weights of the ANN network. Besides, to avoid bias within the training, a 4-fold cross-validation technique is applied for each configuration across all experiments. In this experiment, we divided the original dataset into four

equal-sized subsets (2520 instances in every subset). 75% of the data were utilized for training, and the remaining 25% was used for testing with the defined configuration in every experiment, as shown in Figure 64.

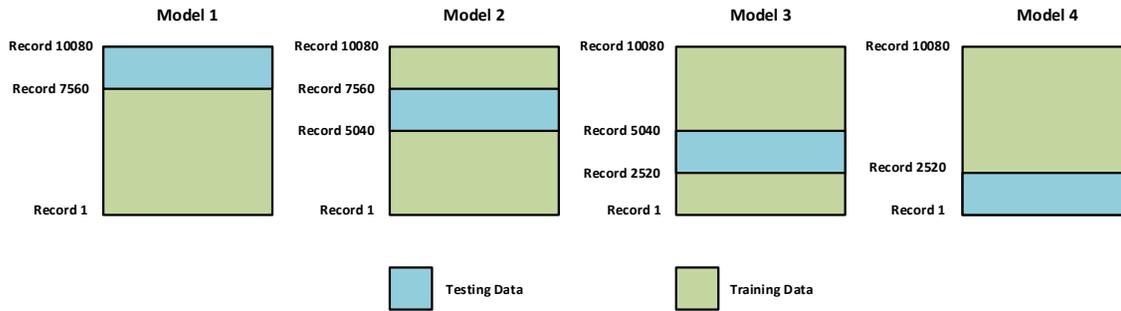


Figure 64: 4-fold cross-validation model for training and testing data

Table 11 provides elaborated info concerning the chosen configuration for ANN; therefore, the associated prediction accuracy concerning Root Mean Square Error (RMSE) evaluating the network configuration in every model. The training process was supported by the Levenberg–Marquardt algorithm, considered as one of the most effective and quickest methodologies for moderately-sized neural networks [50]. We set the maximum number of epochs to 50 for training the ANN network.

Table 11: RMSE for different configurations using the 4-fold cross-validation model

Hidden Layer Size	Activation Function	Learning Rate	Experiment ID	Average (Test Cases)	Experiment Average (Test Cases)
5	Linear	0.1	1	4.69	4.50
5	Linear	0.1	2	4.48	
5	Linear	0.1	3	4.31	
5	Linear	0.1	4	4.52	
5	Linear	0.2	1	4.69	4.51
5	Linear	0.2	2	4.48	
5	Linear	0.2	3	4.31	
5	Linear	0.2	4	4.56	
5	Sigmoid	0.1	1	0.69	0.50
5	Sigmoid	0.1	2	0.48	
5	Sigmoid	0.1	3	0.31	

5	Sigmoid	0.1	4	0.53	
5	Sigmoid	0.2	1	0.59	0.64
5	Sigmoid	0.2	2	0.57	
5	Sigmoid	0.2	3	0.66	
5	Sigmoid	0.2	4	0.75	
5	Linear	0.1	1	4.41	4.61
5	Linear	0.1	2	4.53	
5	Linear	0.1	3	4.74	
5	Linear	0.1	4	4.76	
5	Linear	0.2	1	4.57	4.55
5	Linear	0.2	2	4.60	
5	Linear	0.2	3	4.50	
5	Linear	0.2	4	4.53	
5	Sigmoid	0.1	1	0.51	0.61
5	Sigmoid	0.1	2	0.72	
5	Sigmoid	0.1	3	0.53	
5	Sigmoid	0.1	4	0.68	
5	Sigmoid	0.2	1	0.58	0.64
5	Sigmoid	0.2	2	0.47	
5	Sigmoid	0.2	3	0.77	
5	Sigmoid	0.2	4	0.75	
10	Linear	0.1	1	4.47	4.47
10	Linear	0.1	2	4.47	
10	Linear	0.1	3	4.47	
10	Linear	0.1	4	4.76	
10	Linear	0.2	1	4.47	4.47
10	Linear	0.2	2	4.47	
10	Linear	0.2	3	4.47	
10	Linear	0.2	3	4.47	
10	Sigmoid	0.1	1	0.38	0.48
10	Sigmoid	0.1	2	0.58	
10	Sigmoid	0.1	3	0.50	
10	Sigmoid	0.1	4	0.45	
10	Sigmoid	0.2	1	0.52	0.64
10	Sigmoid	0.2	2	0.73	
10	Sigmoid	0.2	3	0.46	
10	Sigmoid	0.2	4	0.86	
15	Linear	0.1	1	4.47	4.47
15	Linear	0.1	2	4.47	
15	Linear	0.1	3	4.47	
15	Linear	0.1	4	4.47	

15	Linear	0.2	1	4.47	4.47
15	Linear	0.2	2	4.47	
15	Linear	0.2	3	4.47	
15	Linear	0.2	4	4.47	
15	Sigmoid	0.1	1	0.64	0.67
15	Sigmoid	0.1	2	0.64	
15	Sigmoid	0.1	3	0.74	
15	Sigmoid	0.1	4	0.64	
15	Sigmoid	0.2	1	0.44	0.58
15	Sigmoid	0.2	2	0.67	
15	Sigmoid	0.2	3	0.52	
15	Sigmoid	0.2	4	0.67	

Figure 65 presents the code of training function in the learning to prediction module. The network configuration for training is set up with 2 inputs, 10 neurons in the hidden layer, and one output. We use the Sigmoid activation function and the Levenberg–Marquardt algorithm for learning. The process of training is a loop in which we set the epochs at 20. Figure 66 presents the code of the Kalman Filter algorithm in the learning to prediction module. We initialize the value for each parameter in the first round. Afterward, the result in the first round is used as the input for the next iteration. In this way, the Kalman Filter estimates the actual transaction throughput based on transaction throughput measurements observed over time.

```

private void StartTraining()
{
    //prepare Data
    latencyDataNormalized = Normalize(latencyData);
    throughputDataNormalized = Normalize(throughputData);
    errDataNormalized = Normalize(errData);

    network = new ActivationNetwork(new SigmoidFunction(2),
                                    2, // two inputs in the network
                                    10, // two neurons in the first layer
                                    1);

    // create teacher
    LevenbergMarquardtLearning teacher = new LevenbergMarquardtLearning(network);
    teacher.LearningRate=0.1;

    input=new double[SIZE][];
    for(int i = 0; i < SIZE; i++)
    {
        input[i] = new double[2];
        input[i][0] = latencyDataNormalized[i];
        input[i][1] = throughputDataNormalized [i];
    }

    output = new double[SIZE][];
    for (int i = 0; i < SIZE; i++)
    {
        output[i] = new double[1];
        output[i][0] = errDataNormalized[i];
    }

    _pauseEvent.Set();

    // Training
    // loop
    int n = 0;
    while (n<20)
    //while (true)
    {
        // run epoch of learning procedure
        double error = teacher.RunEpoch(input, output);
        //Thread.Sleep(100);
        updateLearningLoss(error);
        _pauseEvent.WaitOne(Timeout.Infinite);
        n++;
    }
}

```

Figure 65: Function of ANN in the learning to prediction module

```

public KalmansFilterModule()
{
    Q= 1;
    R= 10;
    lastPredictedWL = 0;
    firstCall = true;
}

public double prediction(double curData)
{
    double curDataKF;
    if (firstCall) {
        //Console.WriteLine("is first call?:" + firstCall);
        K = 0;
        P = 0;
        Pmin = P + Q;
        lastPredictedWL = curData;
        //curDataKF = Convert.ToInt32(lastPredictedWL);
        curDataKF = lastPredictedWL;
        firstCall = false;
        return curDataKF;
    }
    K = Pmin / (Pmin + R);

    lastPredictedWL = lastPredictedWL+K*(curData -lastPredictedWL );

    P = (1 - K) * Pmin;
    Pmin = P + Q;
    //curDataKF = Convert.ToInt32 (lastPredictedWL );
    curDataKF = lastPredictedWL;

    return curDataKF;
}

public void updateLastPredictedValue(double val)
{
    lastPredictedWL = val;
}

```

Figure 66: Function of Kalman Filter in the learning to prediction module

Figure 67 shows the code of transaction traffic control based on learning to prediction. The whole process of transaction traffic control includes the following steps: obtain the input parameters (network latency, transaction throughput), predict the measurement error using the ANN learning model, and predict the actual transaction throughput using Kalman Filter. The application retrieves the benchmark results from the mongo db by calling the URL of the mongo db. These values are normalized and then passed to the ANN learning module to predict the error

factor. Afterward, the predicted error factor is divided by a constant value to compute the normalized error. After denormalization, the error is divided by a constant factor and passed to the Kalman Filter. Lastly, the Kalman Filter predicts the actual transaction throughput using the updated error factor and stores the predicted result in the mongo db.

```
private void StartTpsControl()
{
    //int duration = Convert.ToInt16(txtTpsDuration.Text);
    appKalmanFilterModule4Tps = new KalmansFilterModule2();
    double max, min;
    max = Max(errData);
    min = Min(errData);
    _isContinue = true;

    while (_isContinue)
    {
        // Retrieve benchmark results from MongoDB directly
        var client = new MongoClient("mongodb://192.168.0.24:27017");
        var database = client.GetDatabase("benchmarkdb");
        var collection = database.GetCollection<BsonDocument>("benchmarks");

        var latestRecord = collection.Find(new BsonDocument()).Sort("{natural: -1}").FirstOrDefault();
        latencyData[0] = Convert.ToDouble(latestRecord["latency"]);
        throughputData[0] = Convert.ToDouble(latestRecord["throughput"]);
        Console.WriteLine("measured throughput:" + throughputData[0]);
        Console.WriteLine("measured latency:" + latencyData[0]);

        double maxLatency, minLatency, maxThroughput, minThroughput;
        maxLatency = Max(latencyData);
        minLatency = Min(latencyData);
        maxThroughput = Max(throughputData);
        minThroughput = Min(throughputData);
        double normalizedLatency = NormalizeSingleValue(latencyData[0], maxLatency, minLatency);
        double normalizedThroughput = NormalizeSingleValue(throughputData[0], maxThroughput, minThroughput);
        Console.WriteLine("normalized latency:" + normalizedLatency);
        Console.WriteLine("normalized throughput:" + normalizedThroughput);

        double[] input = new double[2] { normalizedLatency, normalizedThroughput };

        double[] normalizedErr = network.Compute(input);
        Console.WriteLine("normalized err:" + normalizedErr[0]);

        double preErr = normalizedErr[0] * (max - min) + min;
        Console.WriteLine("pre err:" + preErr);
        appKalmanFilterModule4Tps.RR = Math.Sqrt(preErr / Convert.ToDouble(txtErrFactor.Text));
        Console.WriteLine("kalman gain:" + appKalmanFilterModule4Tps.RR);
        double preidictedThroughput = appKalmanFilterModule4Tps.prediction(throughputData[0]);
        Console.WriteLine("predicted throughput:" + preidictedThroughput);

        // Create the predicted throughput record in MongoDB
        var predictedThroughputs = database.GetCollection<BsonDocument>("predictedThroughputs");
        predictedThroughputs.InsertOne(new BsonDocument("predicted_throughput", preidictedThroughput));

        Thread.Sleep(65000);
    }
}
```

Figure 67: Function of transaction traffic control in the learning to prediction module

The learning to prediction module is implemented using window forms in C#, as shown in Figure 68. We implement various buttons to provide entries for performing operations provided by the learning to prediction module. The data load obtains the dataset in text format and stores it in the application. The Kalman Filter removes the measurement noise and predicts the transaction throughput. ANN learning module tunes the Kalman Filter algorithm to improve its prediction accuracy. Prediction with learning to Kalman Filter utilizes the optimized prediction parameter to predict the transaction throughput. TPS control utilizes the optimized Kalman Filter algorithm to predict the transaction throughput and stores the predicted value into the database. The results panel visualizes the prediction results and accuracy of the learning to prediction module.

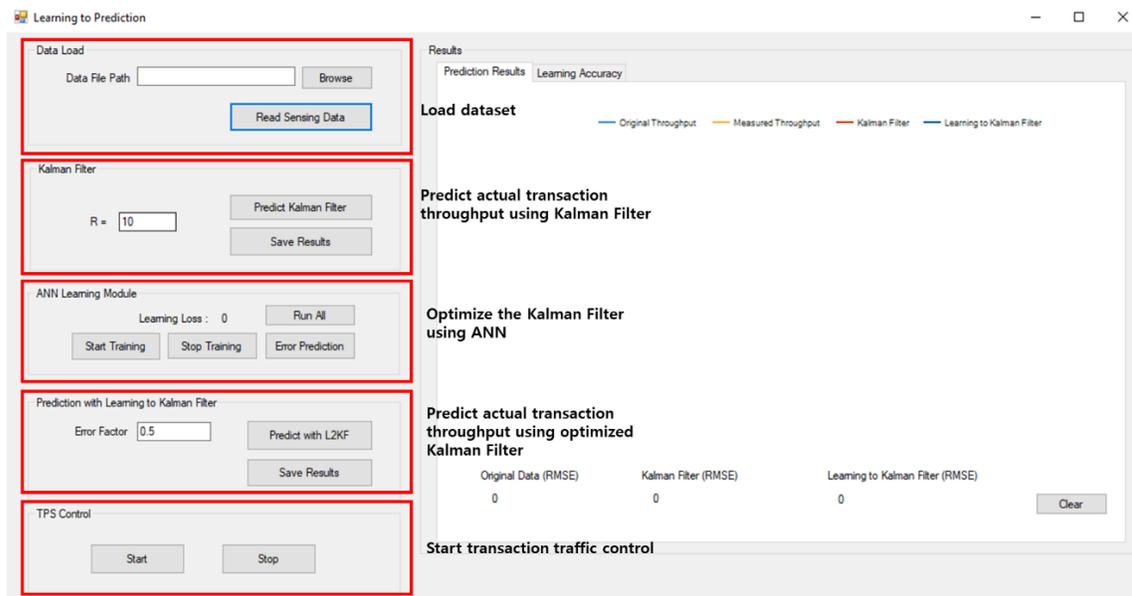


Figure 68: Learning to prediction module main interface

Figure 69 illustrates the execution results of the learning to prediction module. The dataset is loaded from an external text file and stored inside the application for further processing. The RMSE is computed by comparing the measured transaction throughput with the actual send rate. The RMSE for the measurement error is very high, with a value of 20.25. The RMSE error is at

17.41, with a reduction of 14% by using the Kalman Filter with $R=20$. This error is further reduced by 20.3% at 13.88 by using the optimized Kalman Filter.

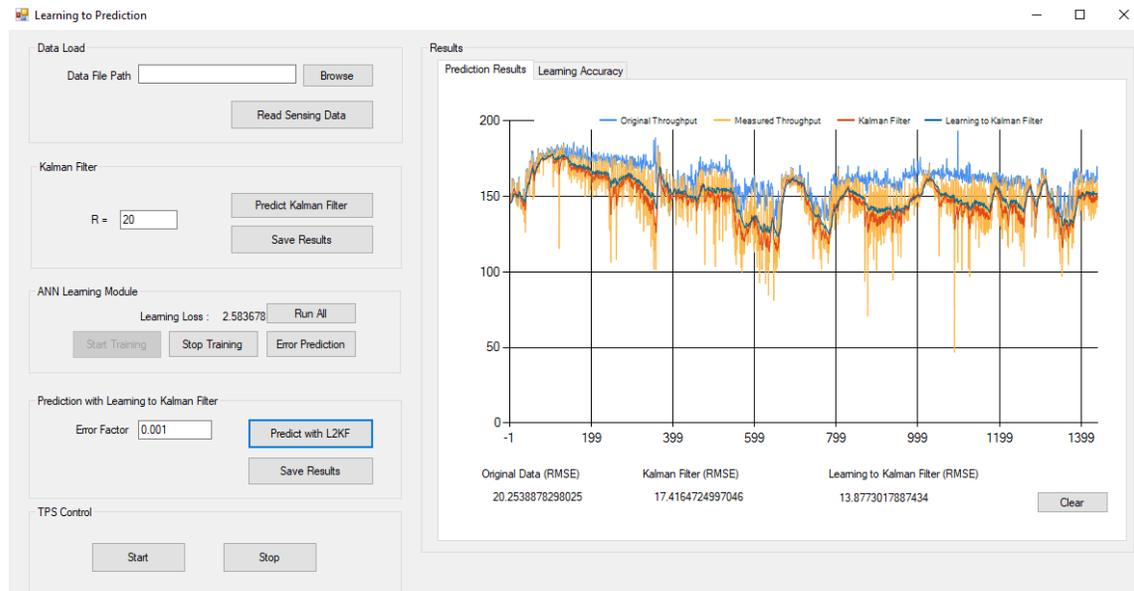


Figure 69: Execution results of the learning to prediction module

5.3 Performance Analysis of the Transaction Traffic Control Mechanism based on Learning to Prediction

This section illustrates the evaluation results of the transaction traffic control mechanism based on learning to prediction compared to the baseline network. We used the sample smart contract called simple to test the backend functionalities of the blockchain network. The default block size is set to 10 transactions per block, and a new block is formed every 250 ms. The default ordering service is in solo mode, which consists only of a single ordering node. The LevelDB is used as the default state database in this experiment. The evaluation tests presented in this section were averaged over multiple rounds to reduce errors resulting from the network congestion.

Figure 70 plots the evaluation results of transaction throughput in a 1org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the transaction throughput of each block size set was 152.9 tps and 159.7 tps, with a 4.4% increase of the transaction throughput. Figure 71 plots the evaluation results of transaction latency in a 1org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 80 ms and 100 ms, with a 20% reduction of transaction latency.

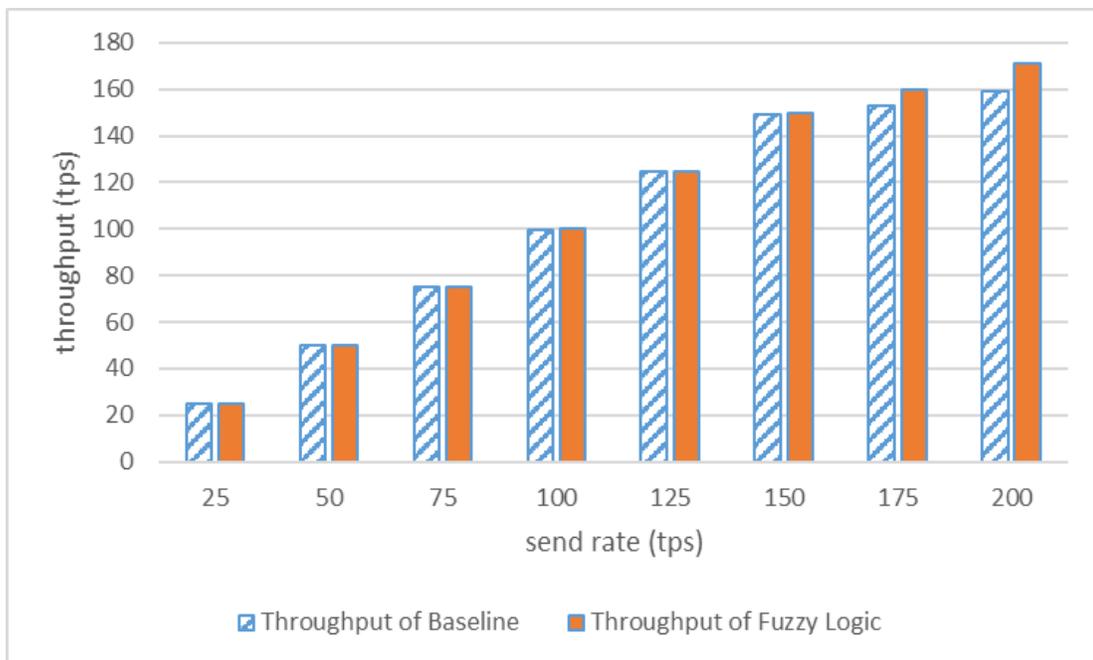


Figure 70: Evaluation of transaction throughput in 1org2peer network with 1 client (baseline and learning to prediction)

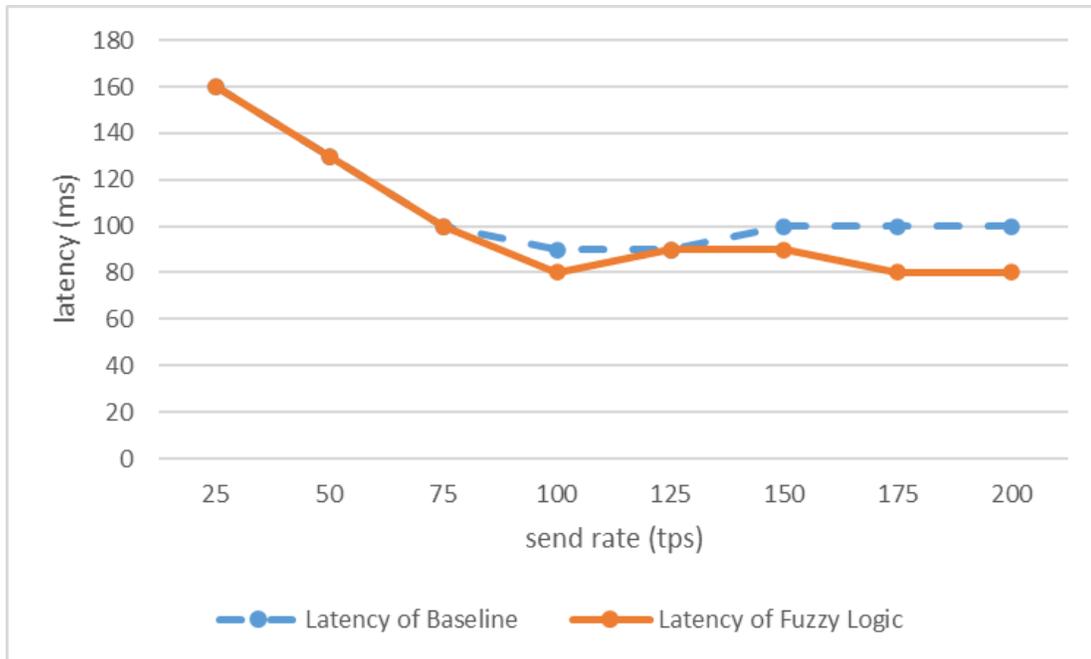


Figure 71: Evaluation of transaction latency in 1org2peer network with 1 client (baseline and learning to prediction)

Figure 72 plots the evaluation results of transaction throughput in a 2org1peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 175 tps, the transaction throughput of each block size set was 132.6 tps and 144.1 tps, with an 8.7% increase of the transaction throughput. Figure 73 plots the evaluation results of transaction latency in a 2org1peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 95 ms and 150 ms, with a 36.7% reduction of transaction latency.

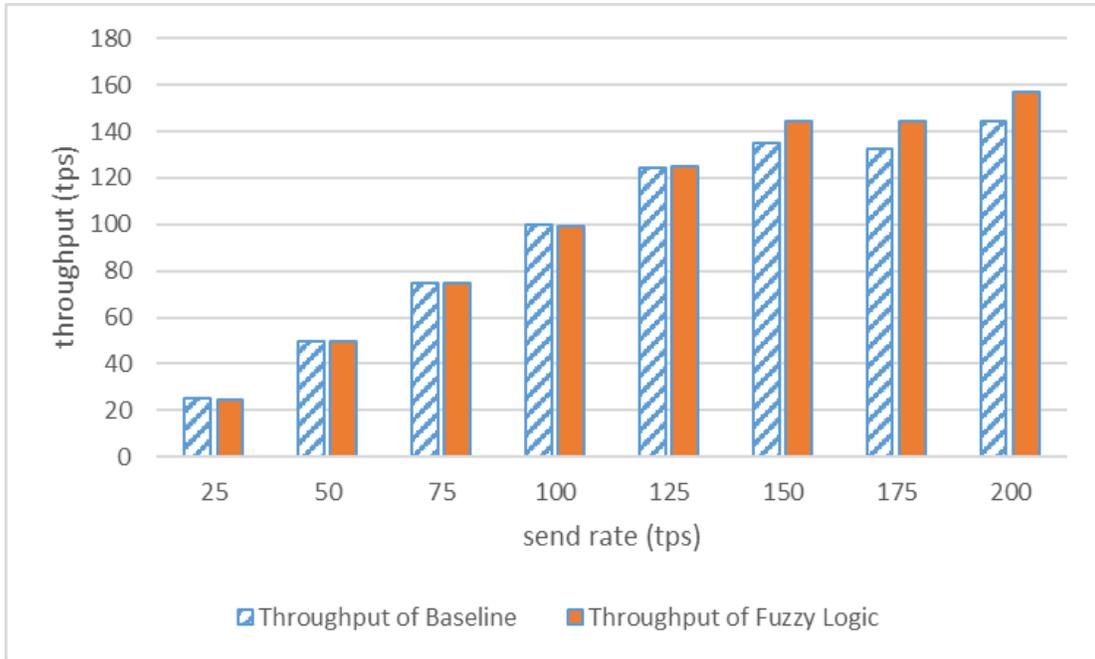


Figure 72: Evaluation of transaction throughput in 2org1peer network with 1 client (baseline and learning to prediction)

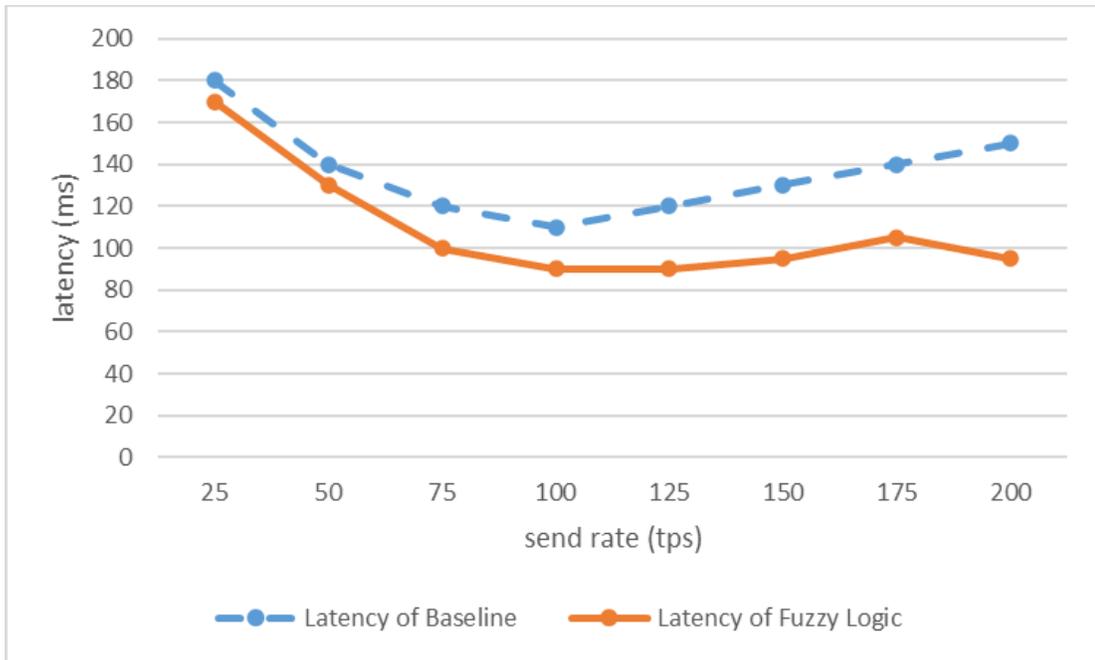


Figure 73: Evaluation of transaction latency in 2org1peer network with 1 client (baseline and learning to prediction)

Figure 74 plots the evaluation results of transaction throughput in a 2org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 132.7 tps and 139.5 tps, with a 5.1% increase of the transaction throughput. Figure 75 plots the evaluation results of transaction latency in a 2org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 125 ms and 170 ms, with a 26.5% reduction of transaction latency.

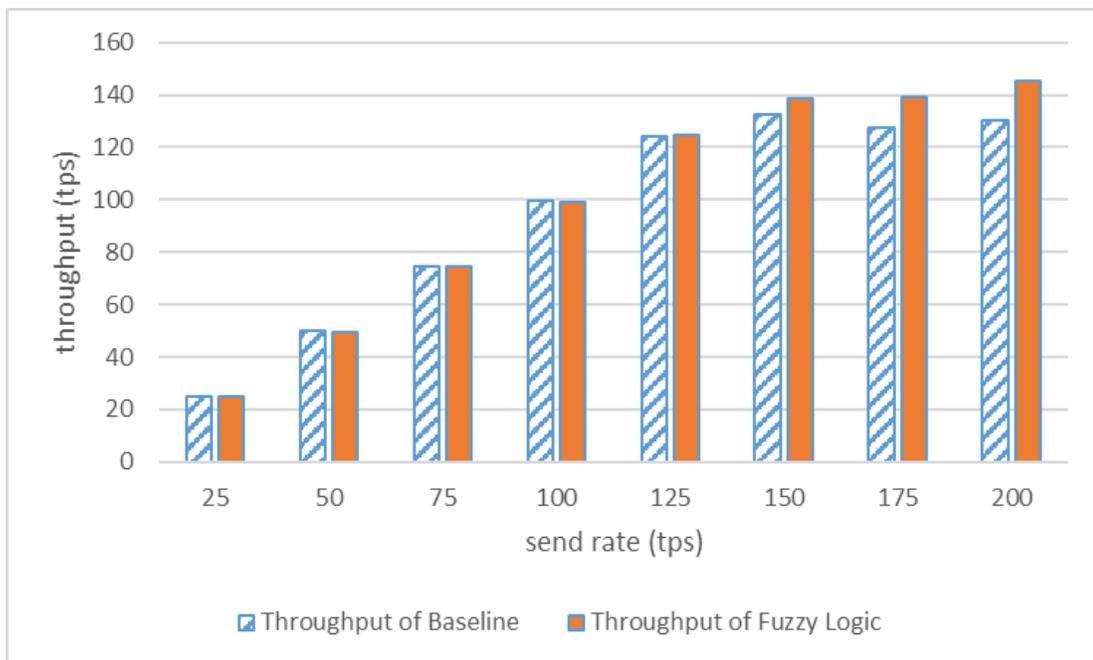


Figure 74: Evaluation of transaction throughput in 2org2peer network with 1 client (baseline and learning to prediction)

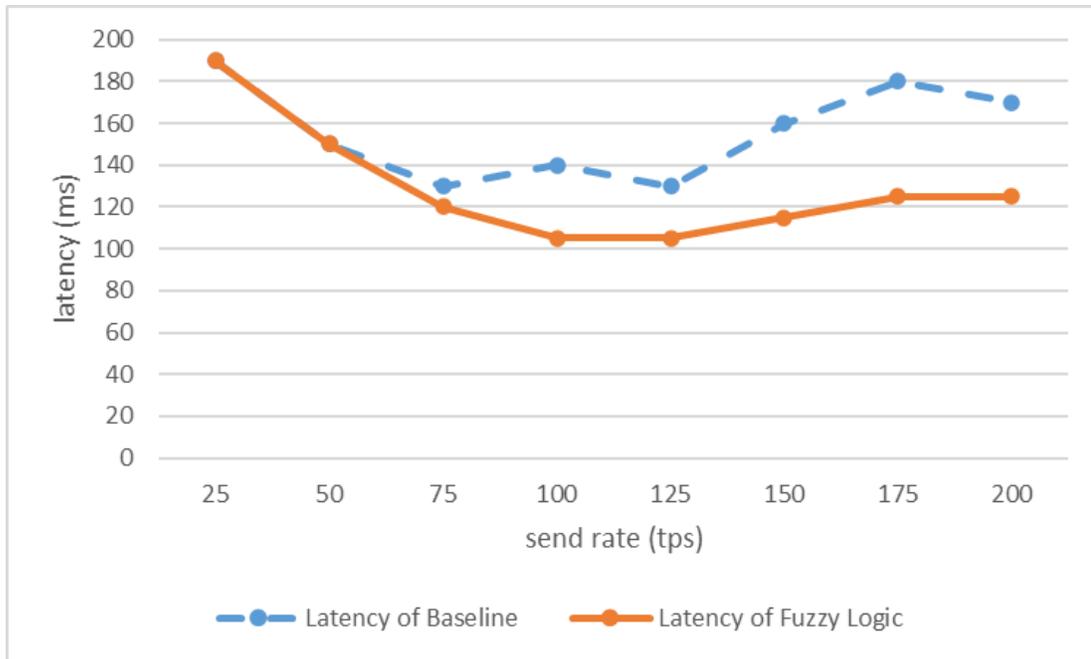


Figure 75: Evaluation of transaction latency in 2org2peer network with 1 client (baseline and learning to prediction)

Figure 76 plots the evaluation results of transaction throughput in a 3org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of each block size set was 112.2 tps and 125.3 tps, with an 11.7% increase of the transaction throughput. Figure 77 plots the evaluation results of transaction latency in a 3org2peer network with one client by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 225 ms and 320 ms, with a 29.7% reduction of transaction latency.

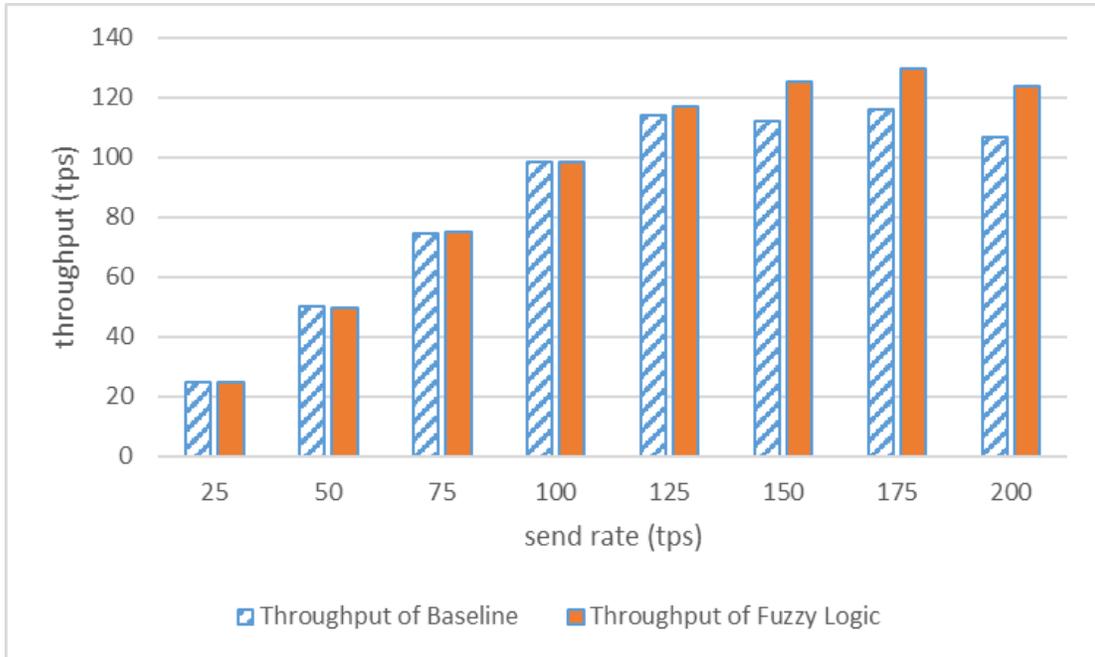


Figure 76: Evaluation of transaction throughput in 3org2peer network with 1 client (baseline and learning to prediction)

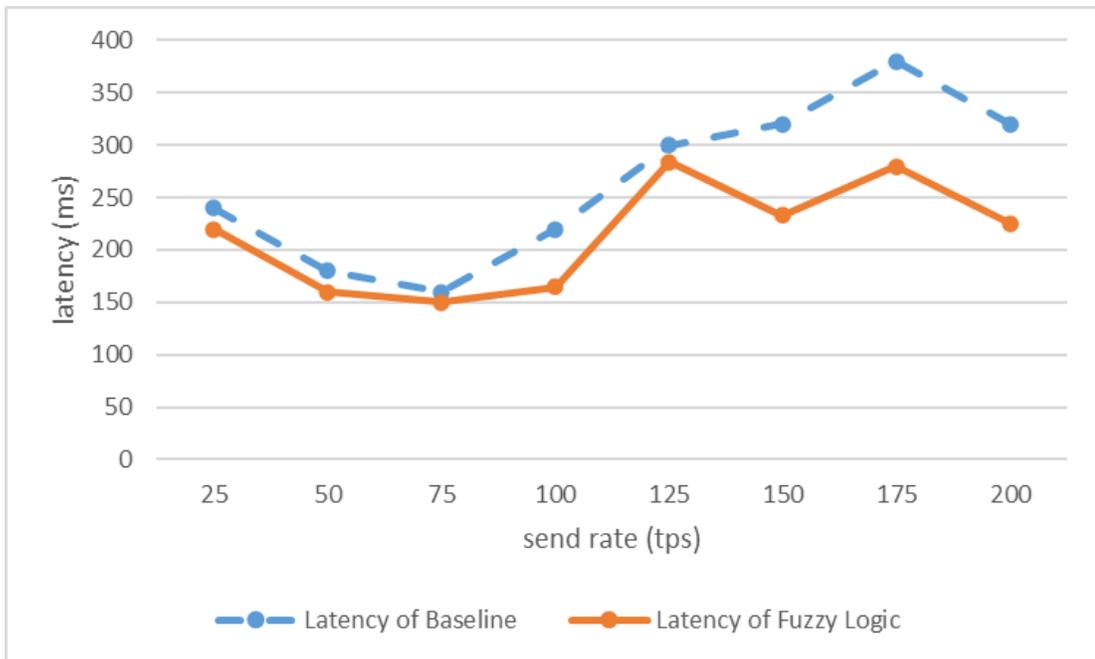


Figure 77: Evaluation of transaction latency in 3org2peer network with 1 client (baseline and learning to prediction)

Figure 78 plots the evaluation results of transaction throughput in a 1org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 175 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the learning to prediction mechanism and the baseline was 183.1 tps and 173.1 tps with a 5.8% increase of the transaction throughput. Figure 79 plots the evaluation results of transaction latency in a 1org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 440 ms and 680 ms, with a 35.2% reduction of transaction latency.

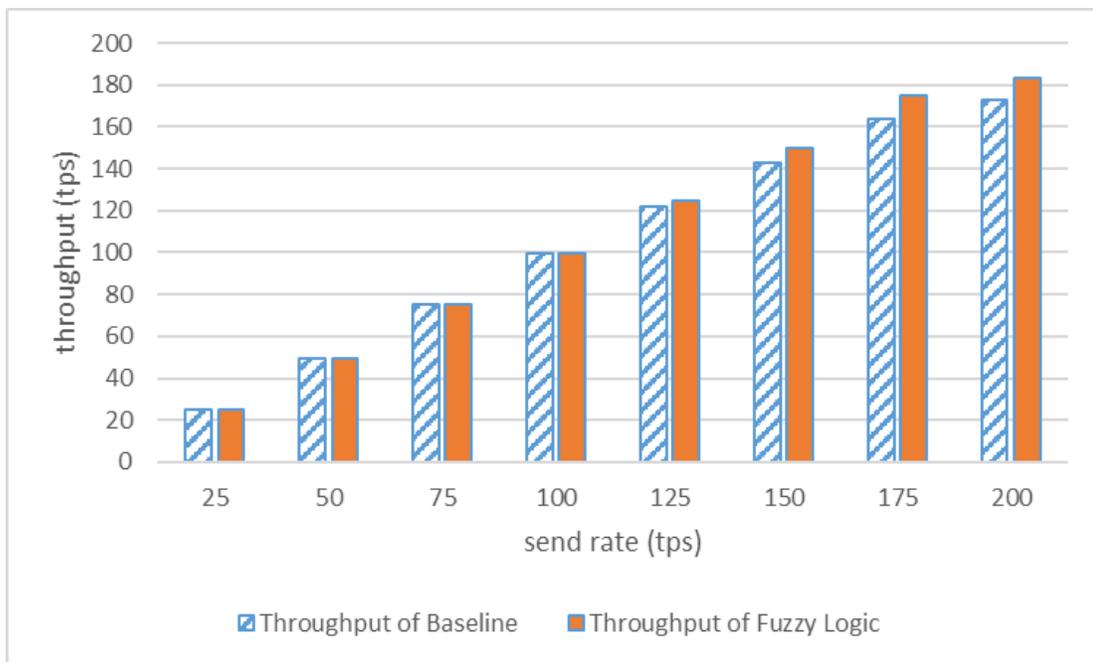


Figure 78: Evaluation of transaction throughput in 1org2peer network with 5 clients (baseline and learning to prediction)

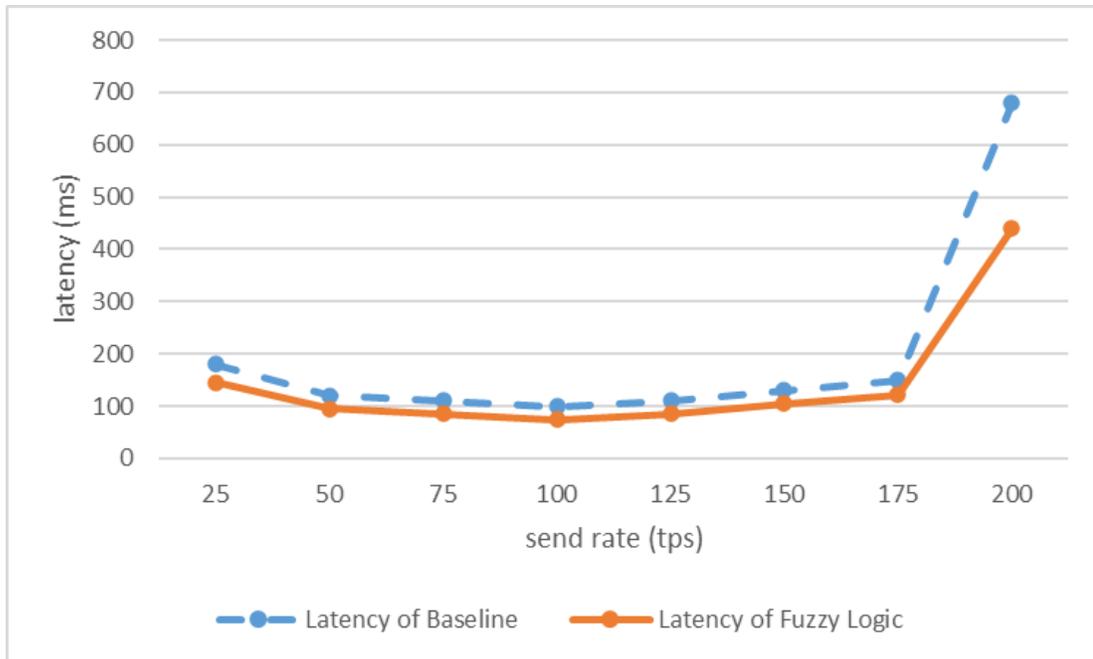


Figure 79: Evaluation of transaction latency in 1org2peer network with 5 clients (baseline and learning to prediction)

Figure 80 plots the evaluation results of transaction throughput in a 2org1peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of the learning to prediction mechanism and the baseline was 135.1 tps and 149.6 tps with a 10.7% increase of the transaction throughput. Figure 81 plots the evaluation results of transaction latency in a 2org1peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 800 ms and 950 ms, with a 15.8% reduction of transaction latency.

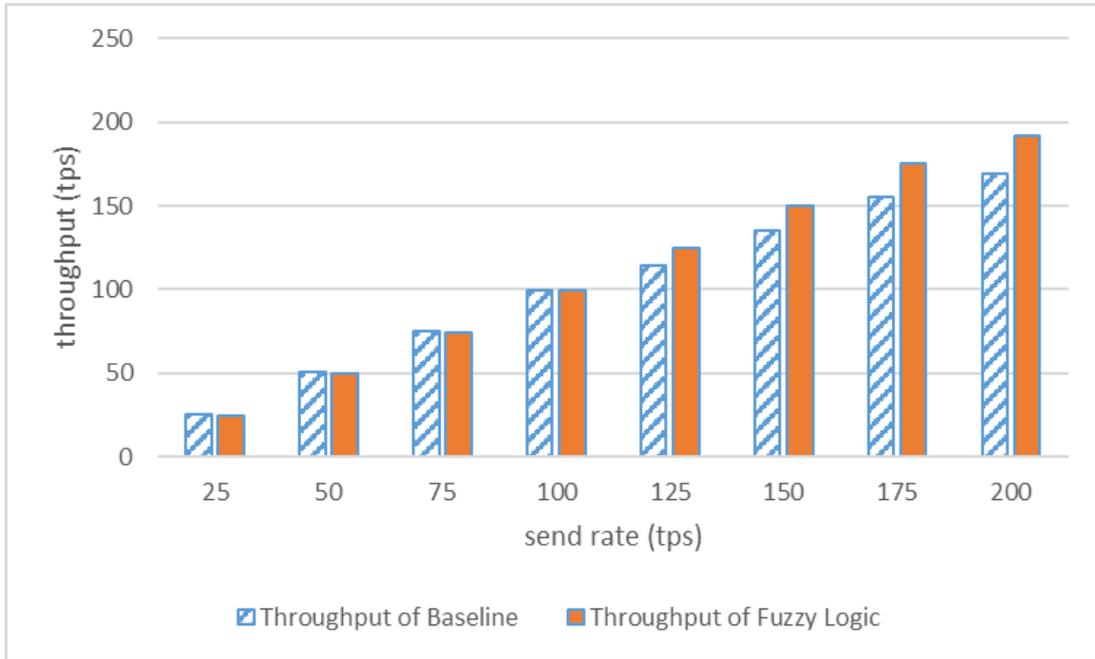


Figure 80: Evaluation of transaction throughput in 2org1peer network with 5 clients (baseline and learning to prediction)

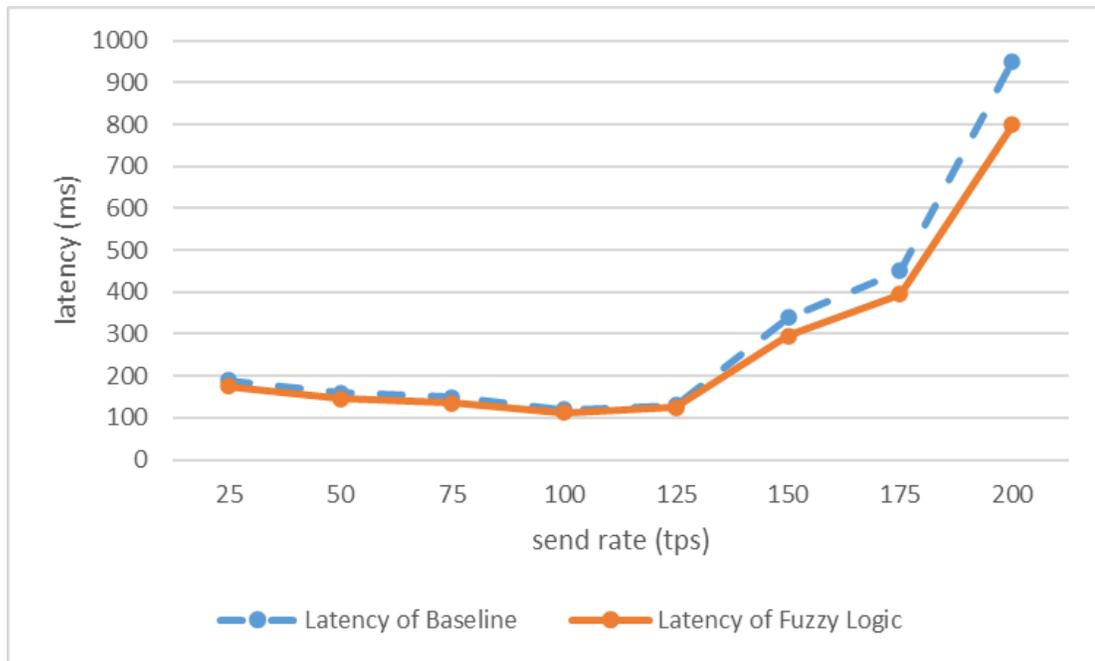


Figure 81: Evaluation of transaction latency in 2org1peer network with 5 clients (baseline and learning to prediction)

Figure 82 plots the evaluation results of transaction throughput in a 2org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the throughput of the learning to prediction mechanism and the baseline was 128.6 tps and 142.4 tps with a 10.7% increase of the transaction throughput. Figure 83 plots the evaluation results of transaction latency in a 2org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 1620 ms and 2260 ms, with a 28.3% reduction of transaction latency.

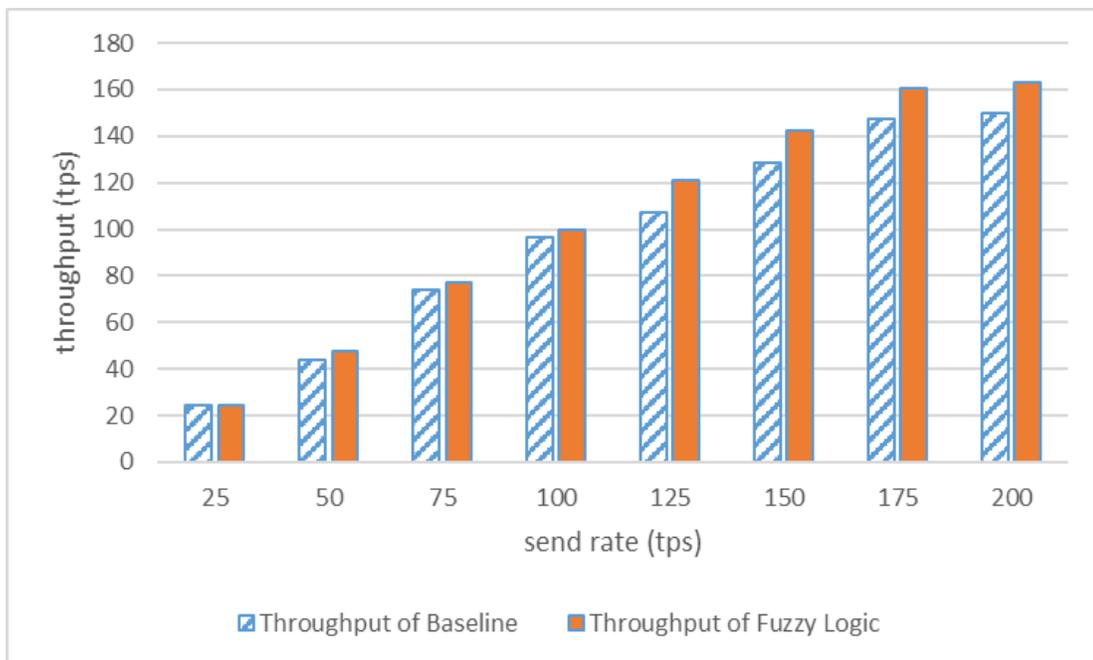


Figure 82: Evaluation of transaction throughput in 2org2peer network with 5 clients (baseline and learning to prediction)

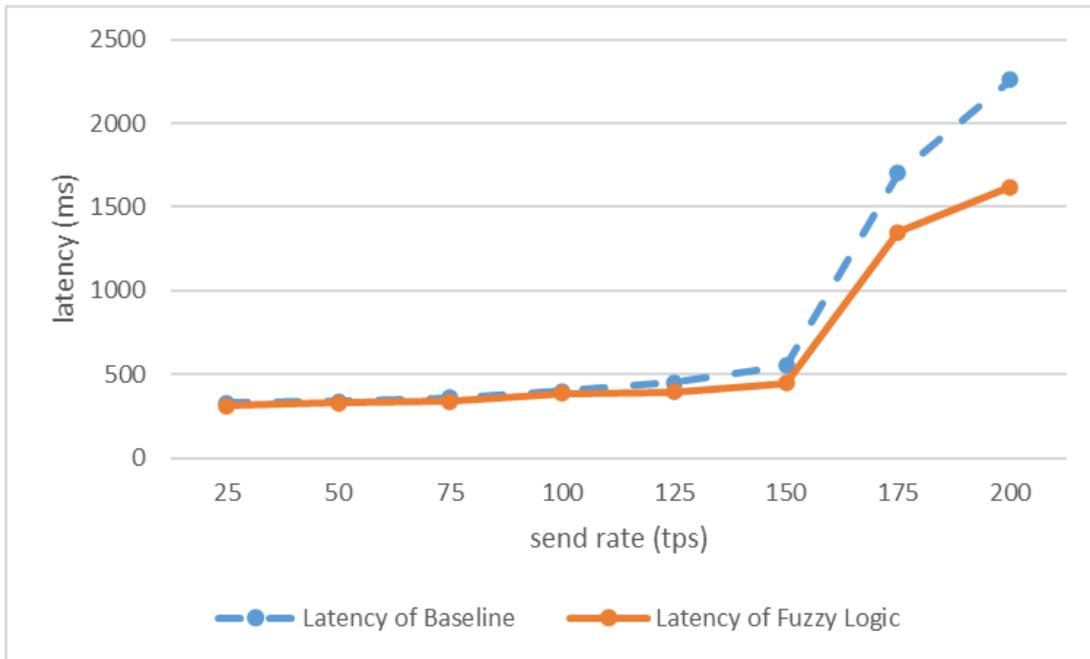


Figure 83: Evaluation of transaction latency in 2org2peer network with 5 clients (baseline and learning to prediction)

Figure 84 plots the evaluation results of transaction throughput in a 3org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). The transaction throughput increased linearly with the increase in send rate until it reached around 100 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 125 tps, the transaction throughput of the learning to prediction mechanism and the baseline was 98.6 tps and 111.4 tps with a 13% increase of the transaction throughput. Figure 85 plots the evaluation results of transaction latency in a 3org2peer network with five clients by comparing the network using the proposed transaction traffic control mechanism based on learning to prediction with the baseline network over different transaction send rate (range from 25 – 200 tps). When the send rate was 200 tps, the transaction latency of the learning to prediction mechanism and the baseline was 2210 ms and 3020 ms, with a 26.8% reduction of transaction latency.

The proposed transaction traffic control mechanism based on learning to prediction was tested in different network configurations by varying the network scale and number of clients. The experiment results indicate that the learning to prediction approach can improve the blockchain performance concerning transaction throughput and transaction latency. In all cases, the evaluation using the transaction traffic control mechanism based on learning to prediction outperforms the baseline by increasing the transaction throughput while decreasing the transaction latency.

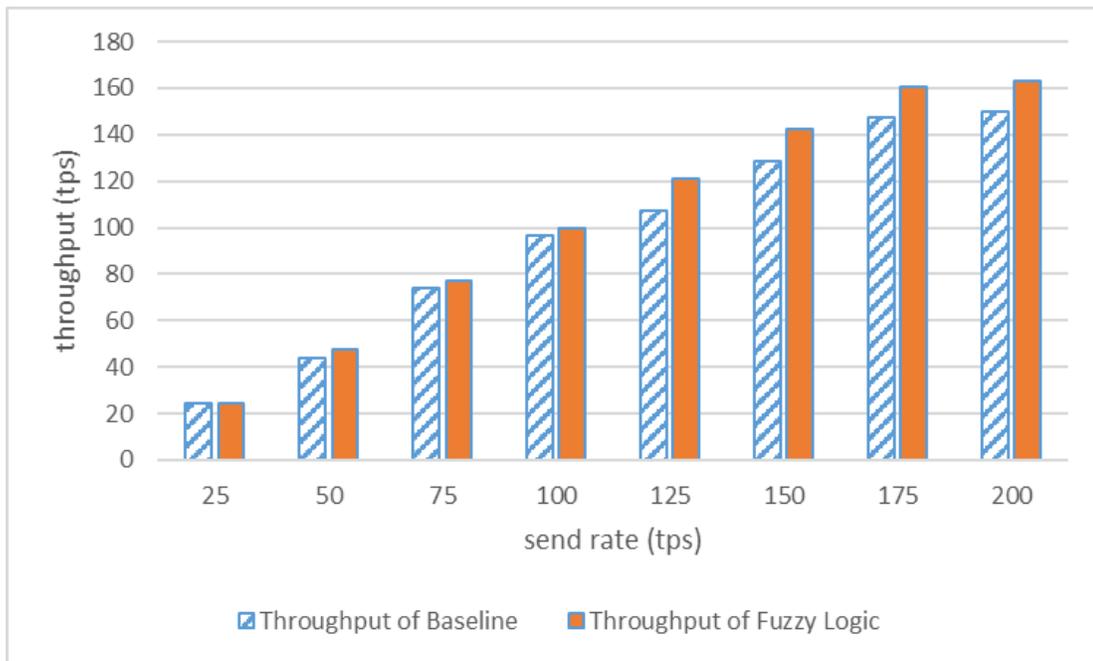


Figure 84: Evaluation of transaction throughput in 3org2peer network with 5 clients (baseline and learning to prediction)

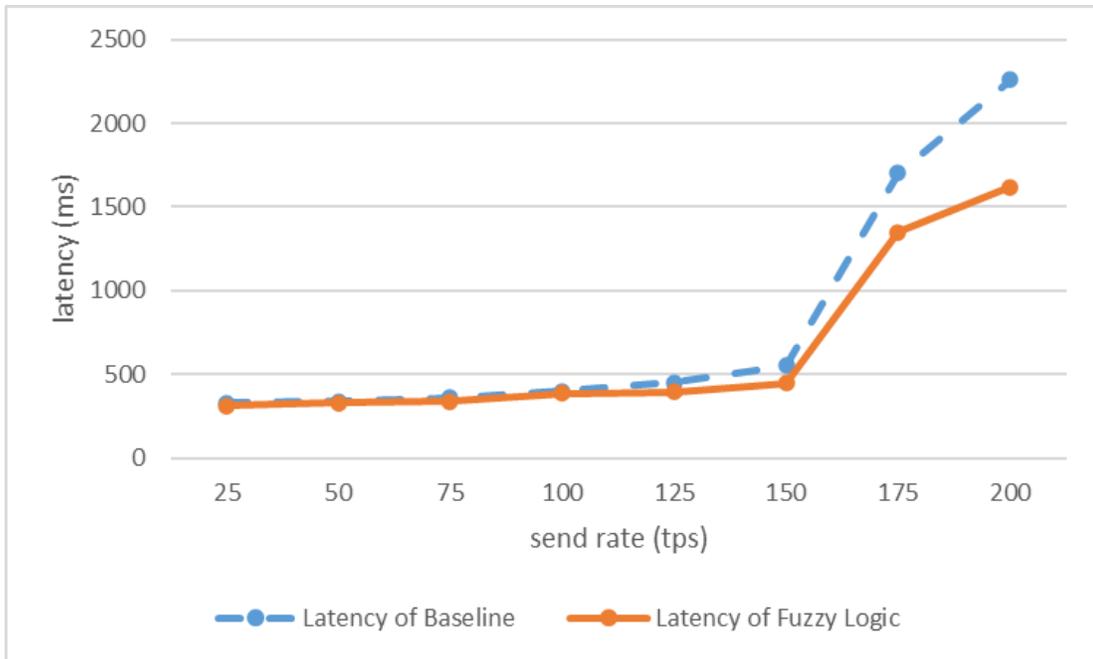


Figure 85: Evaluation of transaction latency in 3org2peer network with 5 clients (baseline and learning to prediction)

6. Performance Evaluation of the Proposed Approach in Clinical Trial Testbed

6.1 Clinical Trial Testbed Environment for Blockchain Performance Evaluation

The system architecture of the clinical trial testbed consists of three layers: the physical layer, the service layer, and the application layer. As shown in Figure 86, the bottom layer is the physical layer, which is comprised of various smart devices for collecting the vital signals from subjects. These devices enable the collection of objective measures of intervention effects both in-clinical and in remote settings. The service layer adopts the modular design that makes the blockchain network more natural to maintain and extend. This layer encapsulates various characteristics of blockchain technologies into individual modules, including peer-to-peer (P2P) protocol, certificate authority, and consensus. The blockchain network consists of various entities, including distributed ledger, certificate authority, P2P protocol, consensus, smart contract, etc. The ledger is decentralized storage to maintain the replicated and shared data distributed across the entire network. The smart contract defines the business logic concerning all clinical trial-related operations, such as creating a patient record. The orderer is a particular node that is performing a consensus algorithm to guarantee the stable operation of the blockchain network and ensure that all peers maintain the data consistency. The event hub is responsible for emitting events whenever a new block is generated, or the condition defined in the smart contract is triggered. The functions specified in the smart contract are encapsulated into REST APIs. The external smart devices and applications can integrate with the network by calling these APIs. The application layer describes the way that services provided by the blockchain are visualized to the end-user. The blockchain

network can be accessed either using responsive web-based applications or native applications on smartphones.

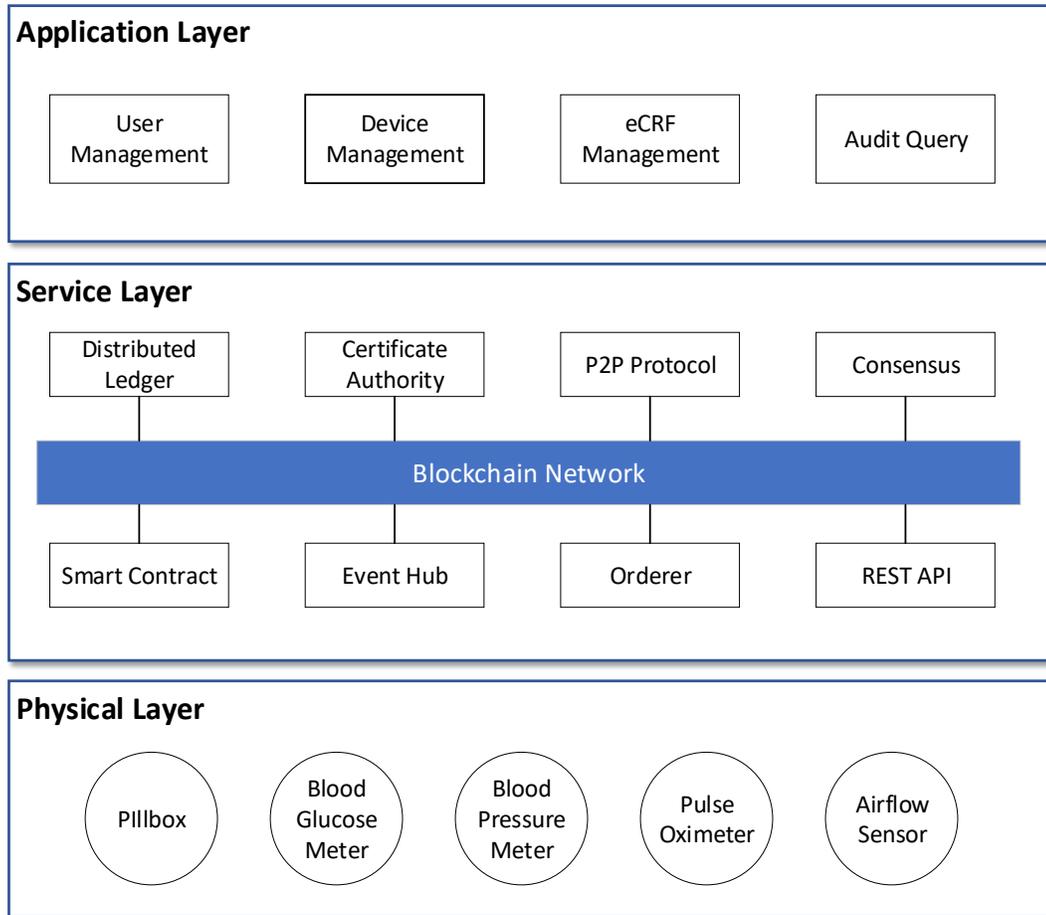


Figure 86: Layer-based system architecture of the clinical trial service platform

There are admins, principal investigators (PIs), clinical research coordinators (CRCs), clinical research associates (CRAs), subjects, and smart devices, all of which form the stakeholders in the clinical trial, as shown in Figure 87. The pharmaceutical company is the sponsor that takes responsibility for creating experiment plans, developing clinical protocols, preparing instruments and medicines of a clinical trial. The pharmaceutical company is not considered as the stakeholder since the contract research organization (CRO) provides clinical trial services for the pharmaceutical company on an outsource basis. In this paper, the admin of the blockchain

management company can set up the network to initiate a clinical trial but cannot perform transactions on the blockchain. Besides, admin is the network manager of blockchain, who is responsible for user registration and enrollment. CRC and PI are investigators who are responsible for the conduct of the clinical trial at a trial site. Generally, a clinical trial is conducted by a team of individuals at a clinical site. CRC interacts heavily with subjects, doing things like collecting and entering data. PI is the lead individual of the team that is ultimately responsible for all trial-related activities at the site. Their job is to ensure the protocol is executed precisely as written and may delegate trial-related activities to members of the clinical team. CRA is the regulator who works in CRO, with authority to review submitted clinical data and those that conduct inspections. The subject is a direct participant of the clinical trial, either as a recipient of the investigational product or as a control. The process of subject enrollment is performed by CRC, who has to screen the recruited subjects to see whether they meet the inclusion and exclusion criteria. As the most fundamental part of the clinical trial system, subjects need to transmit the biomedical data collected from smart devices throughout clinical trials. The distributed data lake serves as an isolated storage that resides on the blockchain, also known as off-chain data storage. It is used to preserve all clinical data, covering user, device, eCRF, and audit data.

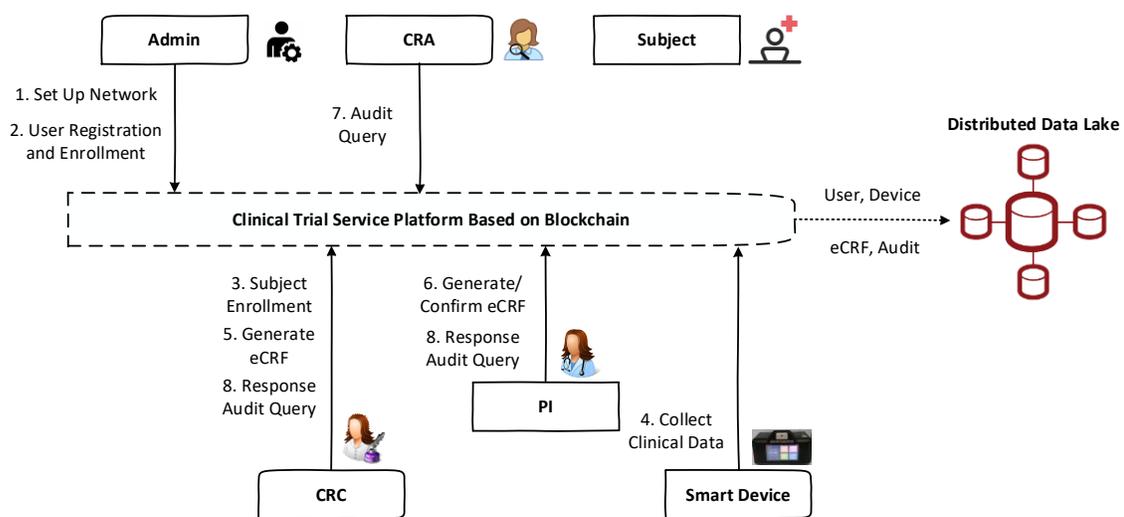


Figure 87: Service scenario of clinical trial service based on blockchain

As shown in Figure 88, these participants can access the blockchain network through the client applications that can communicate with REST APIs. REST APIs serve as an intermediate between external applications and the blockchain network. The smart contract (SC) is a decentralized application that defines the business logic of the blockchain network according to the clinical protocol and automates the whole clinical trial process. The blockchain network appends an immutable record in the ledger to reflect changes resulting from transactions proposed by external applications, and a transaction response is returned as the response. The key-value database (K-V DB) holds the current states of the ledger. Each time a new transaction is agreed upon and added, the K-V DB will update to reflect the latest transaction. The blockchain network is comprised of multiple channels, which can have various organizations, different identities, and data visibility rules. This system is appropriate for multiple clinical trials as the channel is a private network that data is shared only between the participants within the same channel. Organization refers to a business entity (blockchain management company, hospital, contract research organization) that participates in the network. In this paper, each channel consists of four related organizations, and each of them holds a copy of the ledger to maintain consistency.

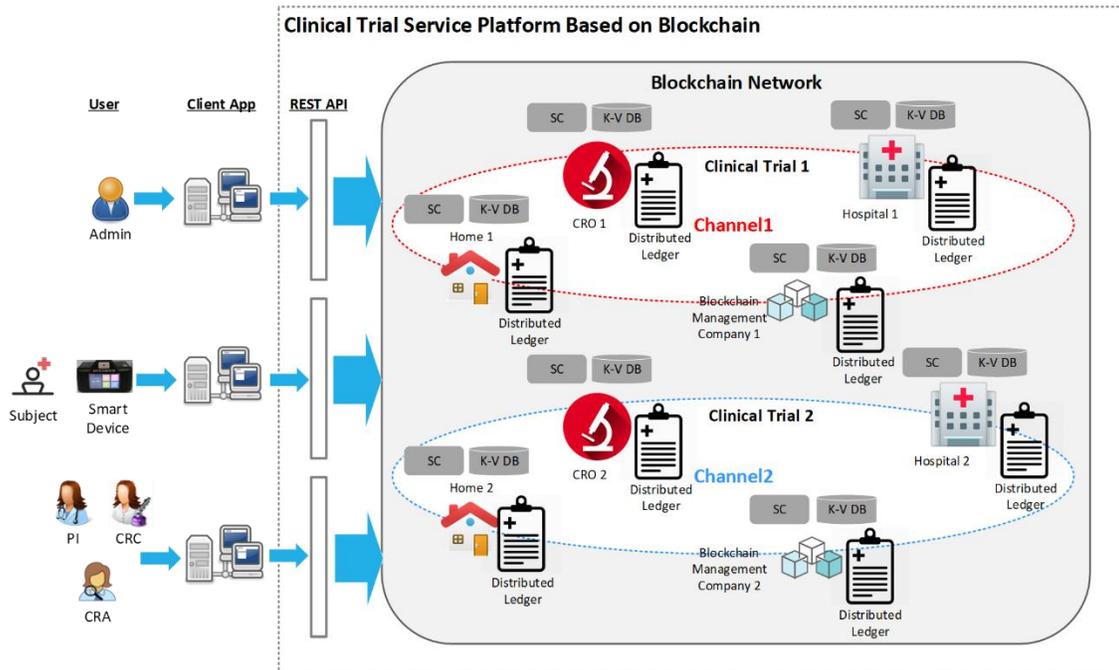


Figure 88: System configuration of the blockchain-based clinical trial service

The detailed workflow of the proposed system is illustrated, as shown in Figure 89. Each participant must have credentials to get the authorized permission for submitting a transaction to the blockchain network. The PI, CRC, and CRA can only read and update their profiles. The PI and CRC can create profiles for new subjects who participate in the clinical trial. They can also set profiles for devices (pillbox, bgm), and these devices will update the settings accordingly. The devices collect biomedical data from subjects and generate eCRF pillbox/bgm data in the blockchain. The eCRF PI consult data and lab test data are created by the CRC when the subject visits the clinical site. After confirmation by the PI, these data cannot be modified. The CRA can review the data and generate audit queries if there exist errors in data. Afterward, the PI and CRC can access the audit and correct the data accordingly.

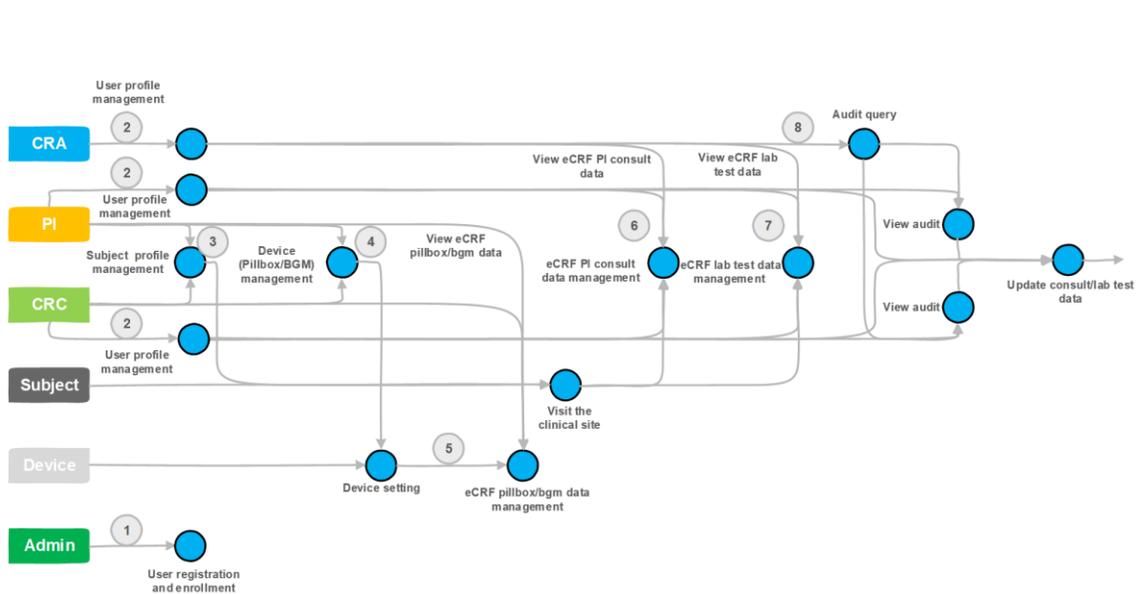


Figure 89: The workflow of the blockchain-based clinical trial service

Figure 90 describes the execution processes of various service scenarios in the proposed system. Users such as PI, CRC, and CRA can read and update their profiles, but they cannot delete these profiles as this operation is allowed by the admin. Subject profiles created by either PI or CRC are accessible to the CRA. The PI and CRC can create the profile of a device that is visible to the CRA. To get medical data from a specified subject, the PI and CRC can bind the device to the subject by updating the device profile. The device periodically read the profile and configure the settings (time to alarm, report interval) accordingly. The medical data collected by the device is generated in the blockchain, and these data are visible to PI and CRC. In clinical trials, the subject needs to visit the clinical site regularly. Whenever the subject visits the clinical site, either PI or CRC can create the eCRF PI consult data and lab test data. The CRA reviews these data and informs a specified PI or CRC of the change by creating the audit query. The PI and CRC receive the message and update the eCRF data accordingly.

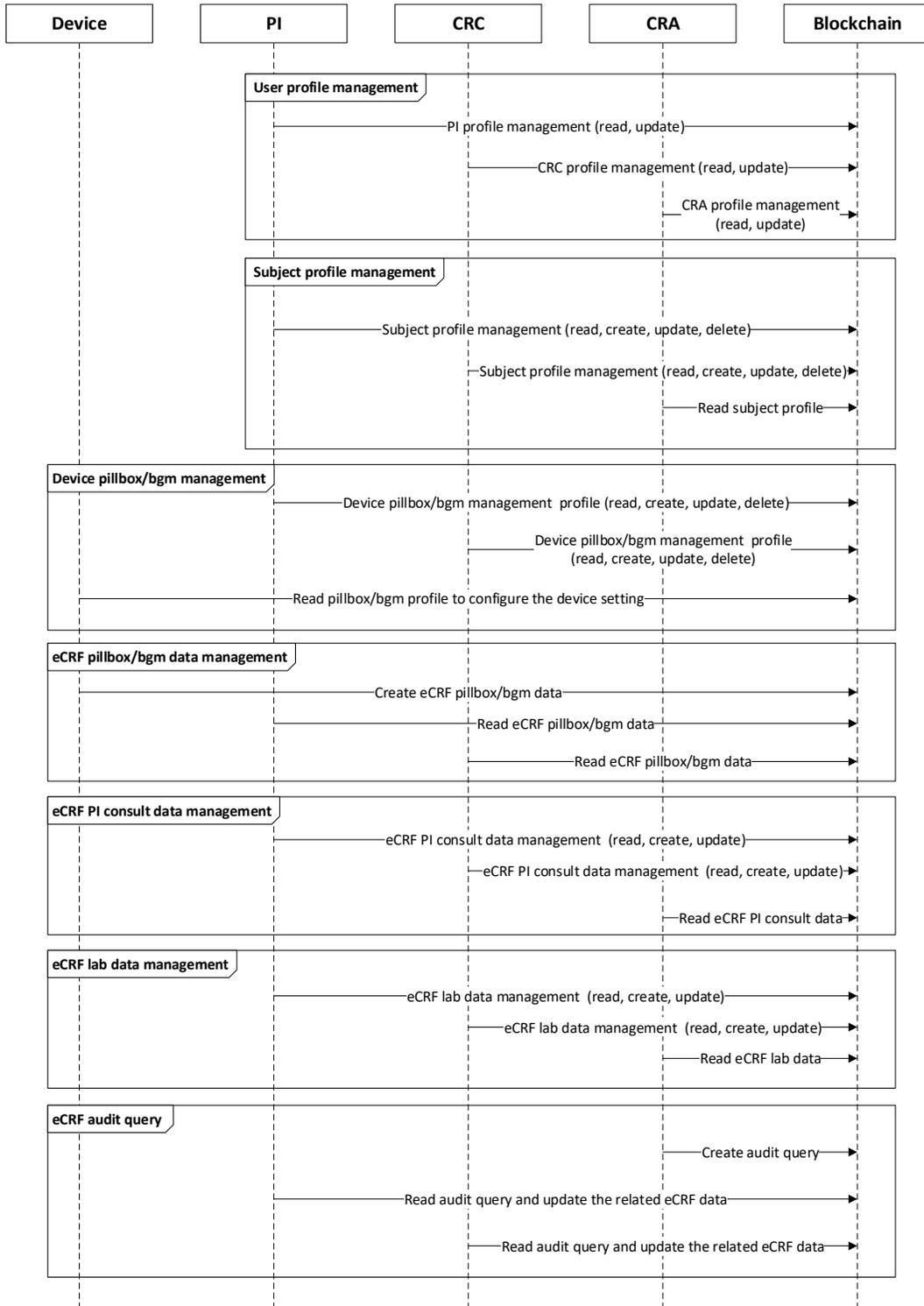


Figure 90: Execution process of the proposed blockchain-based clinical trial service

As shown in Figure 91, the Fabric network is set up and exploited by eight organizations that have corporately decided and signed agreements. An organization refers to a managed group of members, such as the blockchain management company, home, hospital, and CRO. In this experiment, the organizations manage their members under multiple MSPs, which represent different organizational groups in independent clinical trials. It is worth noting that the different MSPs can be used to present the same organization group. For example, the company organization consists of two MSPs, ORG1.MSP and ORG5.MSP, which represents the same blockchain management company that performs different clinical trials. Organizations R1 and R5, refer to the blockchain management companies, have been empowered to initialize the network.

Each organization (e.g., Organization R1) in channel C1 is connected with a client application that can submit transactions, as do other organizations within the same channel. Similarly, client applications connected with organizations in channel C2 can perform transactions within channel C2. It is worth mentioning that one organization can also have multiple client applications such as organizations R2 and R6. Each peer in channel C1 keeps the same copy of the ledger L1 while peer nodes in channel C2 keep the same copy of the ledger L2. The network is under the control of policy rules specified in network configuration NC, which governed by organizations R1 and R5. Channel C1 is governed in terms of the rules defined by channel policy CP1. Similarly, channel C2 is governed in terms of the rules defined by channel policy CP2. The ordering service supports application in both channels, and orders transactions into blocks. Besides, each of the eight organizations has a preferred CA.

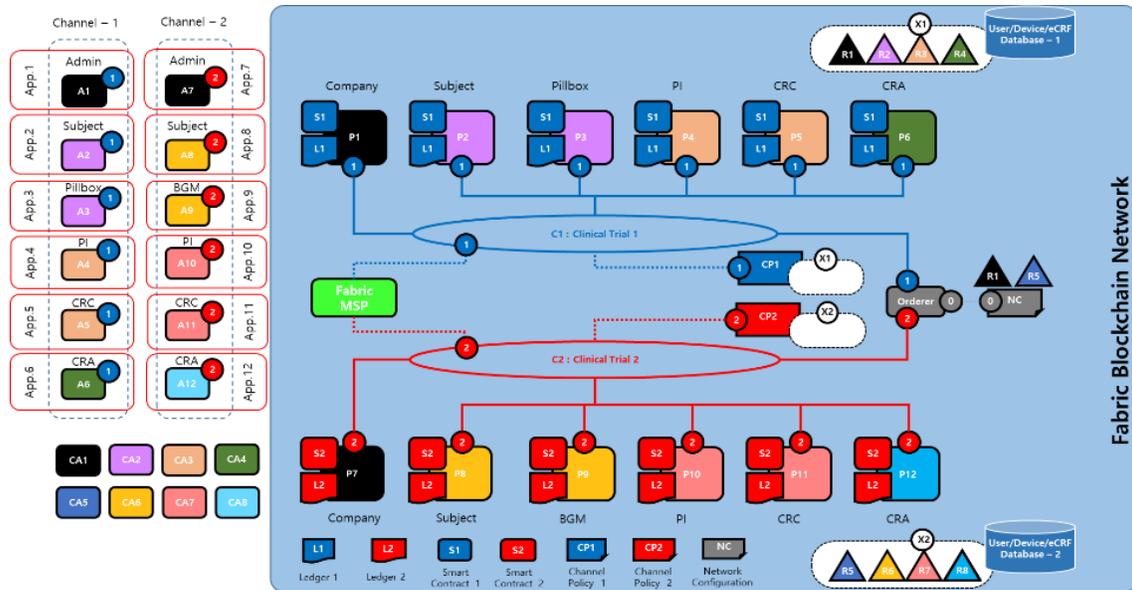


Figure 91: Blockchain network topology of the clinical trial service

The smart contract for the clinical trial testbed contains seven participants, five assets, and nine transactions, as shown in Table 12. The participants are CRC, PI, CRA, subject, pillbox, bgm (Blood Glucose Meter), and last but not least, the admin of the network. Table 12 gives a list of transactions and describes the transaction structure, which is comprised of the participant, operation, and the resource. Participants are users who can submit the transaction to the business network. The operation specifies the action (e.g., Create, Read) that the transaction can perform on the resource. ALL represents that the transaction can support all kinds of actions. Resources represent either participant (e.g., CRC, CRA) or assets such as eCRF pillbox data and eCRF bgm data. Transactions submitted by a participant is to perform the specified operation against the resource.

Table 12: Defined transactions in the smart contract

Transaction	Participant	Operation	Resource (Participant, Asset)
User Profile Management	Admin	ALL	CRC, CRA, PI
Subject Management	CRC, PI	ALL	Subject
Device Pillbox Profile Management	CRC, PI	ALL	Pillbox
Device BGM Profile Management	CRC, PI	ALL	BGM
eCRF Pillbox Data Management	CRC, PI, Pillbox	READ, CREATE	eCRF pillbox data
eCRF BGM Data Management	CRC, PI, BGM	READ, CREATE	eCRF BGM data
eCRF PI Consult Data Management	CRC, PI, CRA	ALL	eCRF PI consult data
eCRF LAB Data Management	CRC, PI, CRA	ALL	eCRF lab data
eCRF CRA Audit	CRC, PI, CRA	ALL	eCRF audit

Table 13 describes the defined assets in the smart contract. Assets are represented as key-value pairs to record state changes of the ledger. The assets are eCRF pillbox data, eCRF bgm data, eCRF PI consult data, eCRF lab data, and eCRF audit. The user of the business network can perform operations on these assets by submitting transactions.

Table 13: Defined assets in the smart contract

Component	Type	Role
eCRF Pillbox	Asset	Record bio/non-bio data of subjects collected by pillbox
eCRF BGM	Asset	Record bio/non-bio data of subjects collected by BGM
eCRF PI Consult	Asset	Record consult data of the subject
eCRF LAB	Asset	Record lab test data of the subject
eCRF CRA Audit	Asset	Record audit data generated by CRA

We use the fabric-contract-api to implement the smart contract for the clinical trial testbed. It provides a JavaScript high-level API to write the business logic and supports communication with peers in the Fabric network. As shown in Figure 92, two transactions, createSubject and querySubject will be invoked by the smart contract whenever receive the request from clients. We use the stub interface to make the connection between the smart contract and the network peers. For instance, the putState function of the stub interface is used to write the state variable to the state database. In contrast, the getState function of the stub interface is used to retrieve the current value of the state variable.

```
'use strict';

const { Contract } = require('fabric-contract-api');

class ClinicalTrial extends Contract {

  // Create a subject
  async createSubject(ctx, sid, subjectName, subjectAge, subjectGender,
    console.info('===== START : Create Subject ====='));

    const subject = {
      name: subjectName,
      age: subjectAge,
      gender: subjectGender,
      ct_cd: ct_cd,
      ct_name: ct_name,
      n_device: n_device,
      activePillboxId: activePillboxId,
      activeBgmId: activeBgmId
    };

    await ctx.stub.putState(sid, Buffer.from(JSON.stringify(subject)))
    console.info('===== END : Create Subject =====');
  }

  // Query a specific subject
  async querySubject(ctx, sid) {
    const subjectAsBytes = await ctx.stub.getState(sid); // get the s
    if (!subjectAsBytes || subjectAsBytes.length === 0) {
      throw new Error(`${sid} does not exist`);
    }
    console.log(subjectAsBytes.toString());
    return subjectAsBytes.toString();
  }
}
```

Figure 92. Sample code of the smart contract in clinical trial service

The client-side of the clinical trial testbed is implemented using Node.js SDK. We use the express framework to implement the web server, which makes it possible to use REST APIs to interact with the Fabric network. As shown in Figure 93, various modules such as fabric-ca-client, fabric-network should be imported to enable the connection with the Fabric network. The fabric-ca-client module allows the client application to enroll and register users to establish trusted identities on the network. The fabric-network module is responsible for submitting transactions and performing queries against the ledger. FileSystemWallet is used to manage all the user identities stored in the wallet. X509WalletMixin creates the identity that has metadata comprised of a certificate and a private key. Gateway is responsible for connecting to the smart contract that resides on the peer node.

```
// Import modules
const express = require('express');
const app = express();
var bodyParser = require('body-parser');

// Import fabric modules
const FabricCAServices = require('fabric-ca-client');
const { FileSystemWallet, X509WalletMixin, Gateway } = require('fabric-network');
const fs = require('fs');
const path = require('path');

const ccpPath = path.resolve(__dirname, '..', '..', 'first-network', 'connection-org1.json');
const ccpJSON = fs.readFileSync(ccpPath, 'utf8');
const ccp = JSON.parse(ccpJSON);

// Configure server
const port = 3000;

// app.use
app.use(express.static(path.join(__dirname + '/vendor')));
app.use(express.static(path.join(__dirname + '/css')));
app.use(express.static(path.join(__dirname + '/views')));
app.use(express.static(path.join(__dirname + '/js')));
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: false }));

// Routing
// 1. Page routing
app.get('/', (req, res) => {
  res.sendFile(__dirname + '/views/login.html');
})

app.get('/subject', (req, res) => {
  res.sendFile(__dirname + '/views/subject.html');
})

app.get('/pillbox', (req, res) => {
  res.sendFile(__dirname + '/views/pillbox-data.html');
})
```

Figure 93. Sample code of the clinical trial service server

```

// 2. REST routing
app.post('/api/login', async (req, res) => {
  const userID = req.body.userID;

  // Create a new file system based wallet for managing identities.
  const walletPath = path.join(process.cwd(), '/javascript/wallet');
  const wallet = new FileSystemWallet(walletPath);
  console.log(`Wallet path: ${walletPath}`);

  // Check to see if we've already enrolled the user.
  const userExists = await wallet.exists(userID);
  if (!userExists) {
    console.log('An identity for the user ' + userID + ' does not exist in the wallet');
    console.log('Run the registerUser.js application before retrying');
    res.sendFile(__dirname + '/views/login.html');
    return;
  } else {
    res.sendFile(__dirname + '/views/subject.html');
  }
});

app.get('/api/Subject', async (req, res) => {
  // Create a new file system based wallet for managing identities.
  const walletPath = path.join(process.cwd(), '/javascript/wallet');
  const wallet = new FileSystemWallet(walletPath);
  console.log(`Wallet path: ${walletPath}`);

  // Check to see if we've already enrolled the user.
  const userExists = await wallet.exists('user1');
  if (!userExists) {
    console.log('An identity for the user "user1" does not exist in the wallet');
    console.log('Run the registerUser.js application before retrying');
    return;
  }

  // Create a new gateway for connecting to our peer node.
  const gateway = new Gateway();
  await gateway.connect(ccpPath, { wallet, identity: 'user1', discovery: { enabled: true, },

  // Get the network (channel) our contract is deployed to.
  const network = await gateway.getNetwork('mychannel');

  // Get the contract from the network.
  const contract = network.getContract('clinical_trial');

  // Evaluate the specified transaction.
  const result = await contract.evaluateTransaction('queryAllSubjects');
  console.log(`Transaction has been evaluated, result is: ${result.toString()}`);
}

```

Figure 94. Sample code of the REST API in clinical trial service

The detailed implementation of the REST API is shown in Figure 94. The workflow of the API can be summarized into three stages: user authentication, connection, and transaction execution. First, the API will check whether the user who requests the API exists in the wallet. Afterward, it sets the connection with the peer node and gets the contract from the network channel. Lastly, it executes the transaction and returns the transaction execution results.

Table 14 presents the sample list of some REST API endpoints that are used to call transactions provided by the smart contract. Each API contains a URI and verbs such as GET, POST, PUT, and DELETE. The URI specifies the path of the endpoint, and the verb presents the specific operation to be performed on the resource. These APIs are provided in the clinical trial web server. CreateBgm API generates the BGM profile that contains BGM metadata, subject, and device configuration such as the alarm time and report interval. UpdateBgm API is responsible for updating the specified property of the BGM profile. DeleteBgm API deletes the BGM profile stored in the blockchain. CreatePillbox API generates the pillbox profile that contains pillbox metadata, subject, and device configuration like the alarm time and report interval. UpdatePillbox API is responsible for updating the specified property of the pillbox profile. DeletePillbox API deletes the pillbox profile stored in the blockchain.

CreateSubject API generates the subject profile that contains subject metadata, device info, and clinical trial info such as clinical code and clinical name. UpdateSubject API is responsible for updating the specified property of the subject profile. DeleteSubject API deletes the subject profile stored in the blockchain. CreateECRFpillbox API collects various medical info like the dosage, taking time, and subject as well as store the record into the blockchain. CreateECRFbgm API collects various medical info like the blood glucose value, testing time, and subject as well as generate the record into the blockchain. CreateECRFpiConsult API generates various info like the consult metadata, clinical trial info such as clinical code and clinical name, acting note, and memo note as well as store the record into the blockchain. CreateECRFlab API generates various info like clinical trial info such as clinical code and clinical name, subject metadata, device, and biological signals such as blood pressure, heartbeat, and electrolytes. CreateCRAaudit generates various info like clinical trial info such as clinical code and clinical name, subject, test, and audit metadata like acting note.

Table 14. Sample REST API endpoints in clinical trial service

URI	Verb	Description
/api/CreateBgm	POST	Create device BGM profile
/api/UpdateBgm	POST	Update device BGM profile
/api/DeleteBgm	POST	Delete device BGM profile
/api/CreatePillbox	POST	Create device pillbox profile
/api/UpdatePillbox	POST	Update device pillbox profile
/api/DeletePillbox	POST	Delete device pillbox profile
/api/CreateSubject	POST	Create a subject profile
/api/UpdateSubject	POST	Update subject profile
/api/DeleteSubject	POST	Delete subject profile
/api/CreateECRFpillbox	POST	Create eCRF pillbox data
/api/CreateECRFbgm	POST	Create eCRF bgm data
/api/CreateECRFpiConsult	POST	Create eCRF PI consult data
/api/CreateECRFlab	POST	Create eCRF lab data
/api/CreateCRAaudit	POST	Create eCRF audit data

Figure 95 represents the implementation results of the clinical trial service, presenting the following features (a) eCRF pillbox data, (b) pillbox, (c) eCRF lab data, and (d) eCRF PI consult data. The eCRF pillbox data dashboard is used to manage the eCRF pillbox data, which includes various medical info. The pillbox dashboard is responsible for managing the pillbox metadata, subject, and device configuration, like the alarm time and report interval. The eCRF lab data dashboard is used to manage various info related to lab test data. The eCRF PI consult data dashboard is used to manage various info like the consult metadata, clinical trial info, acting note, and memo note. The web client provides an entry that not only allows the user of the business network to perform different operations on the blockchain but also visualizes different data on the blockchain. For example, the CRC and PI can create, read, update, and delete the profile of the pillbox. The web client handles the user requests by invoking the REST APIs provided by the web server. Afterward, the server calls the Fabric SDK to submit the specified transaction or query the ledger and returns the transaction execution results.

Dashboard / eCRF Pillbox

eCRF Pillbox Table Add eCRF Pillbox

Show 10 entries Search:

ID	Subject ID	Clinical Code	Pillbox ID	Taken Date	Taken Status
gubwl	resource.org.clinical.trial.Subject#subject002	100	resource.org.clinical.trial.Pillbox#pillbox002	2019-08-29T02:32:59.349Z	Y
REHyg	resource.org.clinical.trial.Subject#subject002	100	resource.org.clinical.trial.Pillbox#pillbox002	2019-08-29T02:22:59.349Z	Y

Showing 1 to 2 of 2 entries Previous 1 Next

(a)

Dashboard / Pillbox

Pillbox Table Add Pillbox

Show 10 entries Search:

ID	Type	PI ID	CRC ID	Subject ID	Dose Start Date	Dose End Date	Alarm Start Time	Alarm End Time
pillbox001	100	resource.org.clinical.trial.PI#pi001	undefined	resource.org.clinical.trial.Subject#subject001	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z
pillbox002	100	resource.org.clinical.trial.PI#pi001	undefined	resource.org.clinical.trial.Subject#subject002	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z
pillbox003	100	undefined	resource.org.clinical.trial.CRC#crc001	resource.org.clinical.trial.Subject#subject003	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z
pillbox004	100	undefined	resource.org.clinical.trial.CRC#crc001	resource.org.clinical.trial.Subject#subject004	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z	2019-08-28T13:00:10.000Z	2019-08-28T14:00:10.000Z

Showing 1 to 4 of 4 entries Previous 1 Next

(b)

Dashboard / eCRF LAB

eCRF LAB Table Add eCRF LAB Confirm eCRF LAB

Show 10 entries Search:

ID	CRC ID	PI ID	Clinical Code	Clinical Name	Subject ID	Subject Name	Subject Age	Subject Gender	Active Device IDs	LAB Test Date
0AMu0	undefined	resource.org.clinical.trial.PI#pi001	100	xxx test	resource.org.clinical.trial.Subject#subject001	bbb	20	M	pillbox001	2019-08-30T07:55:08.1
7W127	resource.org.clinical.trial.CRC#crc001	undefined	100	xxx test	resource.org.clinical.trial.Subject#subject002	bbb	20	M	pillbox002	2019-08-30T11:55:08.1
MOmbD	resource.org.clinical.trial.CRC#crc001	undefined	100	xxx test	resource.org.clinical.trial.Subject#subject002	bbb	20	M	pillbox002	2019-08-30T10:55:08.1
piTK	undefined	resource.org.clinical.trial.PI#pi001	100	xxx test	resource.org.clinical.trial.Subject#subject001	bbb	20	M	pillbox001	2019-08-30T13:55:08.1

Showing 1 to 4 of 4 entries Previous 1 Next

(c)

Dashboard / eCRF PI Consult

eCRF PI Consult Table Add eCRF PI Consult Confirm eCRF PI Consult

Show 10 entries Search:

ID	CRC ID	PI ID	Clinical Code	Clinical Name	Subject ID	Active Device IDs	Consult Date	Consult Note	Act Note	Memo Note
bmGAp	undefined	resource.org.clinical.trial.PI#pi001	100	xxx test	resource.org.clinical.trial.Subject#subject002	pillbox002	2019-08-29T08:32:25.388Z	xxx	xxx	xxx
GtXO	undefined	resource.org.clinical.trial.PI#pi001	100	xxx test	resource.org.clinical.trial.Subject#subject002	pillbox002	2019-08-29T06:32:25.388Z	xxx	xxx	xxx
SeZyl	resource.org.clinical.trial.CRC#crc001	undefined	100	xxx test	resource.org.clinical.trial.Subject#subject004	pillbox004	2019-08-29T11:32:25.388Z	xxx	xxx	xxx
XSF9U	resource.org.clinical.trial.CRC#crc001	undefined	100	xxx test	resource.org.clinical.trial.Subject#subject003	pillbox003	2019-08-29T10:32:25.388Z	xxx	xxx	xxx

Showing 1 to 4 of 4 entries Previous 1 Next

(d)

Figure 95. Implementation results of clinical trial service

6.2 Experiment Environment of Clinical Trial Testbed

This experiment was performed in a single channel of the clinical trial testbed network, which consists of 4 organizations with 6 endorser peer nodes in total. Two different network configurations were set, one is the baseline network, and another one is the network using optimized parameters according to the results of the evaluation in Chapter 3. All the configurable parameters of these two networks are described, as shown in Table 15. We modified the benchmark and network configuration files provided by the Hyperledger Caliper to test our solution. To accurately evaluate the transaction processing capability of the clinical trial testbed, scripts used for the experiment were specified to target one function, which is the CreateECRFpillbox transaction. This function generates biomedical data collected from the pillbox and stores the data into the state database.

Table 15: Experiment parameter configurations

Component	Baseline	Optimized Network
Block Size	10 transactions per block	30 transactions per block
Block Frequency	250ms	250ms
Ordering Service	Solo	Solo
Number of Endorsers	6	6
Ledger Database	LevelDB	LevelDB
Programming Language of Smart Contract	Node.js	Node.js
Use of TLS	No	No
Number of Clients	1	5
Endorsement Policy	4 of 4	1 of 4
Send Rate	25-200 tps	25-200 tps
Number of Transactions	1000	1000
Target Function	CreateECRFpillbox	CreateECRFpillbox

To configure and launch the clinical trial network, the following three configuration files are required: configtx.yaml, crypto-config.yaml, and docker-compose.yaml. Configtx yaml file is used to specify network entities to use within the clinical trial network, and these definitions are

passed to create genesis block and channel artifacts. As shown in Figure 96, the main component of configtx yaml is the organizations block. This block sets a directory of certificates of each organization and specifies the customized policy for the organization in the channel.

Organizations:

```
# SampleOrg defines an MSP using the sampleconfig. It should never be used
# in production but may be used as a template for other definitions
- &OrdererOrg
  # DefaultOrg defines the organization which is used in the sampleconfig
  # of the fabric.git development environment
  Name: OrdererMSP

  # ID to load the MSP definition as
  ID: OrdererMSP

  # MSPDir is the filesystem path which contains the MSP configuration
  MSPDir: crypto-config/ordererOrganizations/example.com/msp

  AdminPrincipal: Role.MEMBER

- &Org0
  # DefaultOrg defines the organization which is used in the sampleconfig
  # of the fabric.git development environment
  Name: Org1MSP

  # ID to load the MSP definition as
  ID: Org1MSP

  MSPDir: crypto-config/peerOrganizations/org1.example.com/msp

  AdminPrincipal: Role.ADMIN

  AnchorPeers:
    # AnchorPeers defines the location of peers which can be used
    # for cross org gossip communication. Note, this value is only
    # encoded in the genesis block in the Application section context
    - Host: peer0.org1.example.com
      Port: 7051
    - Host: peer1.org1.example.com
      Port: 7051
    - Host: peer2.org1.example.com
      Port: 7051

- &Org1
  # DefaultOrg defines the organization which is used in the sampleconfig
  # of the fabric.git development environment
  Name: Org2MSP
```

Figure 96. Blockchain network configuration in clinical trial service (configtx.yaml)

The crypto-config yaml file is used to specify all network entities and for generating all required credentials for each peer node. As shown in Figure 97, this file mainly consists of the

OrdererOrgs block and PeerOrgs block. It is worth noting that the name and domain of order and each peer must be consistent with the definitions in the configtx yaml file.

```
OrdererOrgs:
# -----
# Orderer
# -----
- Name: Orderer
  Domain: example.com
# -----
# "Specs" - See PeerOrgs below for complete description
# -----
  Specs:
    - Hostname: orderer
# -----
# "PeerOrgs" - Definition of organizations managing peer nodes
# -----
PeerOrgs:
# -----
# Org1
# -----
- Name: Org1
  Domain: org1.example.com
  Template:
    Count: 3
  Users:
    Count: 1
# -----
# Org2: See "Org1" for full specification
# -----
- Name: Org2
  Domain: org2.example.com
  Template:
    Count: 3
  Users:
    Count: 1
# -----
# Org3: See "Org1" for full specification
# -----
- Name: Org3
  Domain: org3.example.com
  Template:
    Count: 3
  Users:
    Count: 1
```

Figure 97. Blockchain network credentials configuration in clinical trial service (crypto-config yaml)

As shown in Figure 98, the docker-compose yaml file defines the CA, peer container, and the orderer container. The CA is used for issuing and verifying the identities of participants in the

network. The peer container specifies the running environment for each peer node while the orderer container specifies the running environment for the orderer node.

```
peer0.org1.example.com:
  container_name: peer0.org1.example.com
  image: hyperledger/fabric-peer:${FABRIC_VERSION}
  environment:
    - CORE_LOGGING_PEER=debug
    - CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
    - CORE_CHAINCODE_EXECUTETIMEOUT=9999999
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    - CORE_PEER_ID=peer0.org1.example.com
    - CORE_PEER_ENDORSER_ENABLED=true
    - CORE_PEER_ADDRESS=peer0.org1.example.com:7051
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=clinicaltrialtestbed_default
    - CORE_PEER_LOCALMSPID=Org1MSP
    - CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.org1.example.com:7051
  working_dir: /opt/gopath/src/github.com/hyperledger/fabric
  command: peer node start
  ports:
    - 7051:7051
    - 7053:7053
  volumes:
    - /var/run:/host/var/run/
    - ../../config_solo_custom/mychannel.tx:/etc/hyperledger/configtx/mychannel.tx
    - ../../config_solo_custom/crypto-config/peerOrganizations/org1.example.com/peer0/
    - ../../config_solo_custom/crypto-config/peerOrganizations/org1.example.com/use
  depends_on:
    - orderer.example.com
```

Figure 98. Blockchain network container configurations in clinical trial service (docker-compose-cli yaml)

Figure 99 represents part of the benchmark configuration file used for testing the clinical trial network. The rounds block describes the settings of a round; the txNumber specifies the number of transactions should be submitted in a single round, the rateControl defines the type of rate controller, and the opts defines the tps of send rate. For this experiment, the benchmark contains eight rounds with varied send rate, which ranged from 25 tps to 200 tps. The callback function used in this experiment will construct the eCRF pillbox data and submit the transaction to the blockchain. The details of the eCRF pillbox data are defined under the arguments block that will be passed to the Caliper as the benchmark configuration.

```

rounds:
- label: open
  description: Test description for the opening of an account
  txNumber:
  - 75
  - 150
  - 225
  - 300
  - 375
  - 450
  - 525
  - 600
rateControl:
- type: fixed-rate
  opts:
    tps: 25
- type: fixed-rate
  opts:
    tps: 50
- type: fixed-rate
  opts:
    tps: 75
- type: fixed-rate
  opts:
    tps: 100
- type: fixed-rate
  opts:
    tps: 125
- type: fixed-rate
  opts:
    tps: 150
- type: fixed-rate
  opts:
    tps: 175
- type: fixed-rate
  opts:
    tps: 200
arguments:
  # money: 10000
  subject_id: SUBJECT1
  ct_cd: ct_cd1
  pillbox_id: PILLBOX1
  dt_taken: 20200630
  taken_status: True
callback: benchmarks/scenario/simple/open_clinical_trial.js

```

Figure 99. Blockchain network benchmark configuration in clinical trial service

The execution results of network initialization and performance testing in the console are represented in Figure 100 and Figure 101, respectively. The script creates the channel, joins peers to have communication in the network, installs the smart contract on each peer node, and instantiates the smart contract in the channel according to the defined network configuration. Afterward, client workers are initialized to generate workload to the blockchain network according to the defined benchmark configuration. Lastly, a result report is printed in the console, indicating that the evaluation test performed successfully. This report provides benchmark results such as the number of successful transactions, the number of failed transactions, send rate, transaction latency, and transaction throughput.

```

hanglei@hanglei-VirtualBox:~/caliper$ node ./packages/caliper-cli/caliper.js benchmark run --caliper-workspace ca
fig.yaml --caliper-networkconfig networks/fabric/fabric-v1.4.1/clinicalTrialTestbed/fabric-node.yaml
Benchmark for target Blockchain type fabric about to start
2020.07.06-17:32:57.860 info [caliper] [caliper-flow] ##### Caliper Test #####
2020.07.06-17:32:57.866 info [caliper] [local-observer] Observer interval set to 1 seconds
2020.07.06-17:32:57.870 info [caliper] [caliper-utils] Executing command: cd /home/hanglei/caliper/caliper
fabric/docker-compose/clinicalTrialTestbed/docker-compose.yaml up -d;sleep 3s
Creating orderer.example.com ... done
Creating ca.org1.example.com ... done
Creating ca.org4.example.com ... done
Creating ca.org3.example.com ... done
Creating ca.org2.example.com ... done
Creating peer0.org1.example.com ... done
Creating peer0.org3.example.com ... done
Creating peer1.org3.example.com ... done
Creating peer1.org2.example.com ... done
Creating peer0.org2.example.com ... done
Creating peer0.org4.example.com ... done
2020.07.06-17:33:07.237 info [caliper] [adapters/fabric] Fabric SDK version: 1.4.8; TLS: none
2020.07.06-17:33:07.554 warn [caliper] [adapters/fabric] Org1's registrar's materials found locally in fi
2020.07.06-17:33:07.571 warn [caliper] [adapters/fabric] Org2's registrar's materials found locally in fi
2020.07.06-17:33:07.583 warn [caliper] [adapters/fabric] Org3's registrar's materials found locally in fi
2020.07.06-17:33:07.595 warn [caliper] [adapters/fabric] Org4's registrar's materials found locally in fi
2020.07.06-17:33:07.615 warn [caliper] [adapters/fabric] Org1's admin's materials found locally in file s
2020.07.06-17:33:07.625 warn [caliper] [adapters/fabric] Org2's admin's materials found locally in file s
2020.07.06-17:33:07.634 warn [caliper] [adapters/fabric] Org3's admin's materials found locally in file s
2020.07.06-17:33:07.642 warn [caliper] [adapters/fabric] Org4's admin's materials found locally in file s
2020.07.06-17:33:07.652 warn [caliper] [adapters/fabric] client0.org1.example.com's materials found local
2020.07.06-17:33:07.661 warn [caliper] [adapters/fabric] client0.org2.example.com's materials found local
2020.07.06-17:33:07.670 warn [caliper] [adapters/fabric] client0.org3.example.com's materials found local
2020.07.06-17:33:07.678 warn [caliper] [adapters/fabric] client0.org4.example.com's materials found local
2020.07.06-17:33:07.679 info [caliper] [adapters/fabric] Channel 'mychannel' definiton being generated fr
2020.07.06-17:33:07.773 info [caliper] [adapters/fabric] Channel 'mychannel' successfully created
2020.07.06-17:33:07.774 info [caliper] [adapters/fabric] Sleeping 5s...
2020.07.06-17:33:12.844 info [caliper] [adapters/fabric] Org1's peers successfully joined mychannel: peer
2020.07.06-17:33:12.918 info [caliper] [adapters/fabric] Org2's peers successfully joined mychannel: peer
2020.07.06-17:33:12.989 info [caliper] [adapters/fabric] Org3's peers successfully joined mychannel: peer
2020.07.06-17:33:13.040 info [caliper] [adapters/fabric] Org4's peers successfully joined mychannel: peer
2020.07.06-17:33:13.040 info [caliper] [adapters/fabric] Sleeping 5s...
2020.07.06-17:33:18.043 info [caliper] [adapters/fabric] Installing chaincodes for mychannel...
2020.07.06-17:33:18.159 info [caliper] [adapters/fabric] simple@v0 successfully installed on Org1's peers
2020.07.06-17:33:18.207 info [caliper] [adapters/fabric] simple@v0 successfully installed on Org2's peers
2020.07.06-17:33:18.322 info [caliper] [adapters/fabric] simple@v0 successfully installed on Org3's peers
2020.07.06-17:33:18.363 info [caliper] [adapters/fabric] simple@v0 successfully installed on Org4's peers
2020.07.06-17:33:18.364 info [caliper] [adapters/fabric] Instantiating simple@v0 in mychannel. This might
2020.07.06-17:35:02.771 info [caliper] [adapters/fabric] Successfully instantiated simple@v0 in mychannel
2020.07.06-17:35:02.774 info [caliper] [adapters/fabric] Sleeping 5s...

```

Figure 100. Blockchain network initialization in clinical trial service

```

2020.07.06-17:36:37.487 info [caliper] [report-builder]      ### All test results ###
2020.07.06-17:36:37.489 info [caliper] [report-builder]

```

Name	Succ	Fail	Send Rate (TPS)	Max Latency (s)	Min Latency (s)	Avg Latency (s)	Throughput (TPS)
open	75	0	25.3	0.94	0.08	0.44	21.9
open	150	0	50.3	0.44	0.09	0.26	45.6
open	225	0	75.1	1.01	0.16	0.49	69.5
open	300	0	93.4	1.13	0.16	0.75	85.3
open	375	0	123.2	0.75	0.26	0.47	105.2
open	450	0	117.4	2.20	0.15	1.00	108.6
open	525	0	123.8	1.27	0.14	0.58	104.1
open	600	0	130.3	1.29	0.17	0.61	110.7

```

2020.07.06-17:36:37.625 info [caliper] [caliper-flow] Generated report with path /home/hanglei/caliper/caliper-be
2020.07.06-17:36:37.626 info [caliper] [caliper-flow]

```

Figure 101. Performance testing results in clinical trial service

6.3 Evaluation Results of the Proposed Approach in Clinical Trial Testbed

6.3.1 Evaluation Results of the Optimized Network in Clinical Trial Testbed

This experiment evaluated the performance of the optimized network configurations with the baseline. We evaluated the impact of block size on the performance by varying the number of clients over different transaction send rate (range from 25 – 200 tps). Figure 102 plots the experimental results in terms of average transaction throughput with 1 client. The transaction throughput increased linearly with the increase in send rate until it reached around 100 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the optimized network and the baseline network was 118 tps, and 98.7 tps with a 19.6% increase of the transaction throughput. Figure 103 plots the experimental results in terms of average transaction latency with 1 client. It is observed that the baseline network generates more transaction latency than the optimized network, especially when the send rate was above the

saturation point. When the send rate was 200 tps, the transaction latency of the optimized network and the baseline was 520 ms and 650 ms, with a 20% reduction of transaction latency. Figure 104 plots the experimental results in terms of average transaction throughput with 5 clients. The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the throughput of the optimized network and the baseline network was 137.2 tps, and 125.5 tps with a 9.3% increase of the transaction throughput. Figure 105 plots the experimental results in terms of average transaction latency with 5 clients. It is observed that the baseline network generates more transaction latency than the optimized network, especially when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the optimized network and the baseline was 1940 ms and 2350 ms, with a 21.1% reduction of transaction latency.

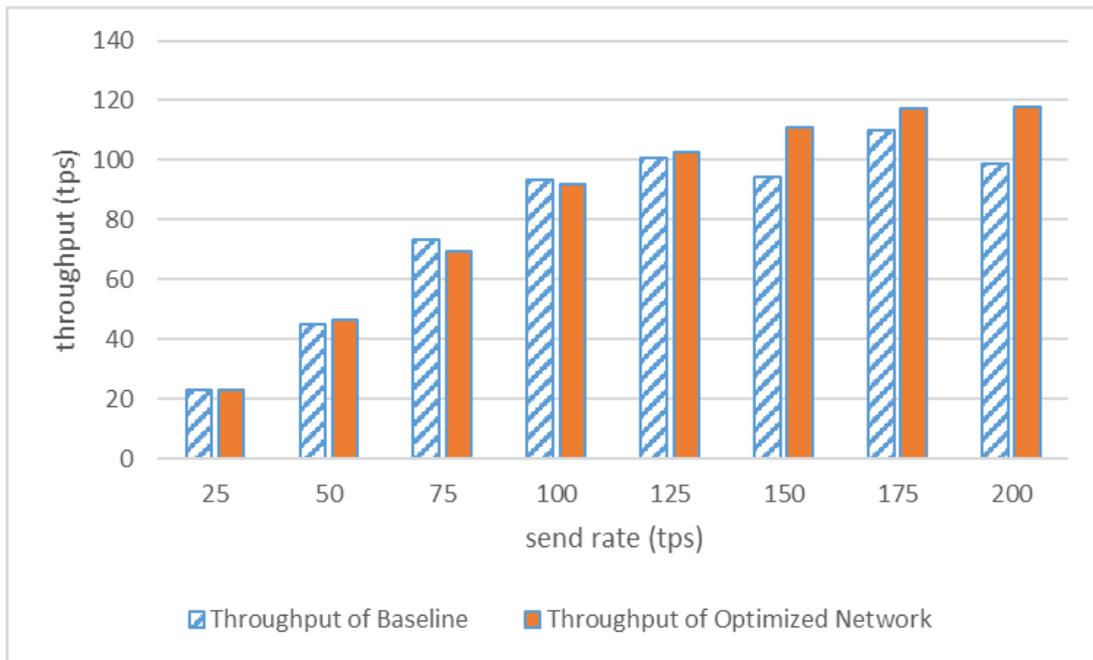


Figure 102: Evaluation of transaction throughput with one client (baseline and optimized network)

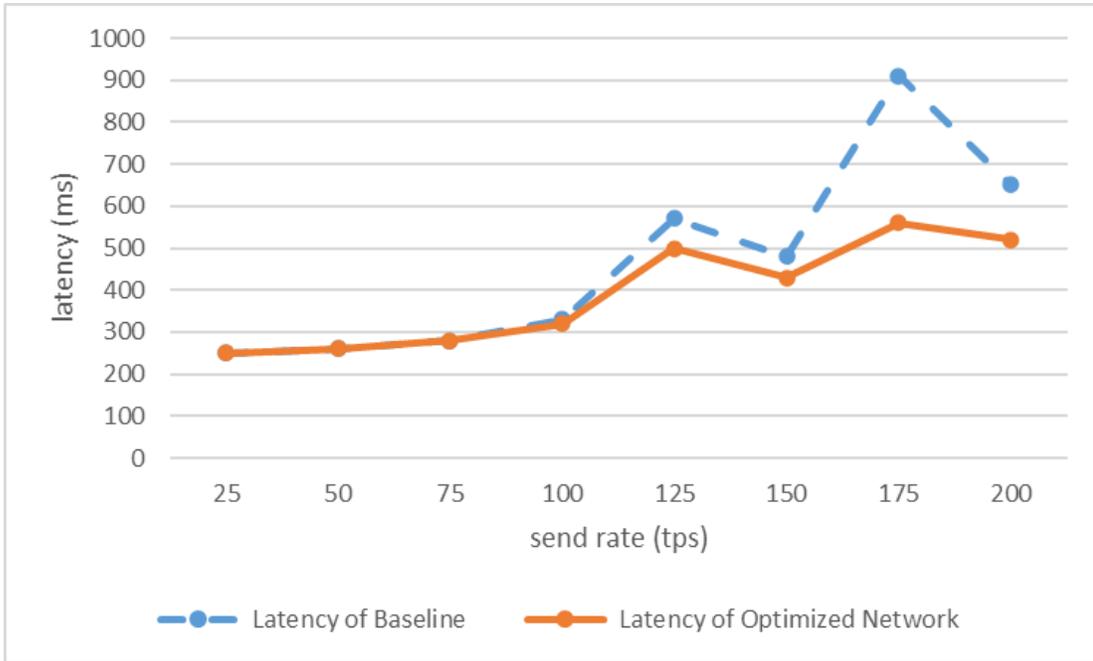


Figure 103: Evaluation of transaction latency with one client (baseline and optimized network)

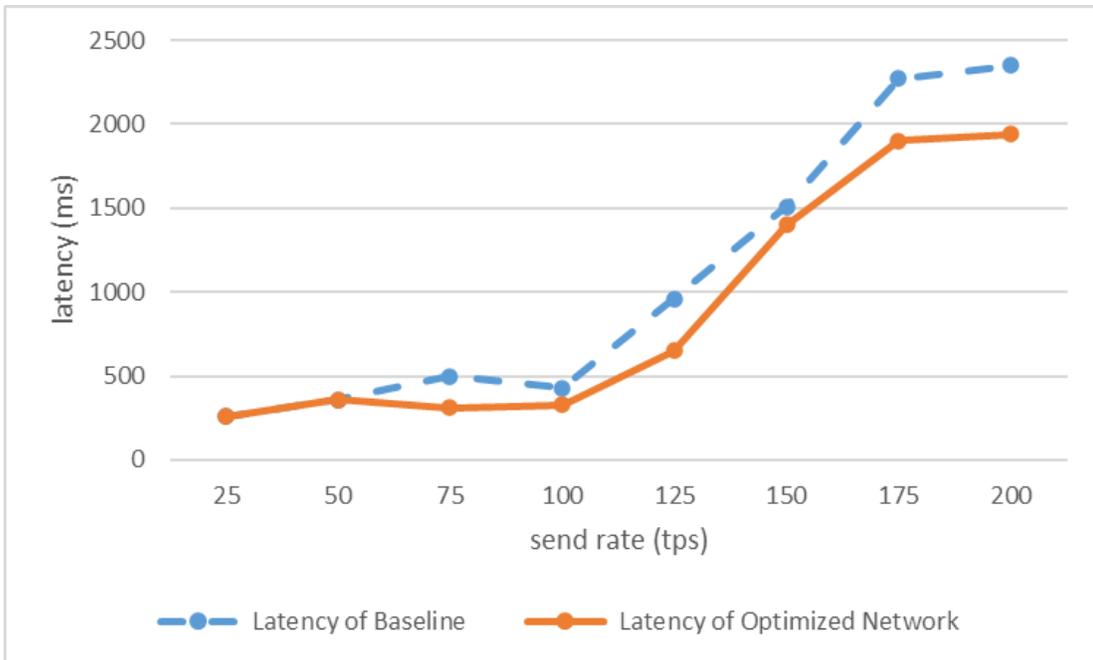


Figure 104: Evaluation of transaction throughput with five clients (baseline and optimized network)

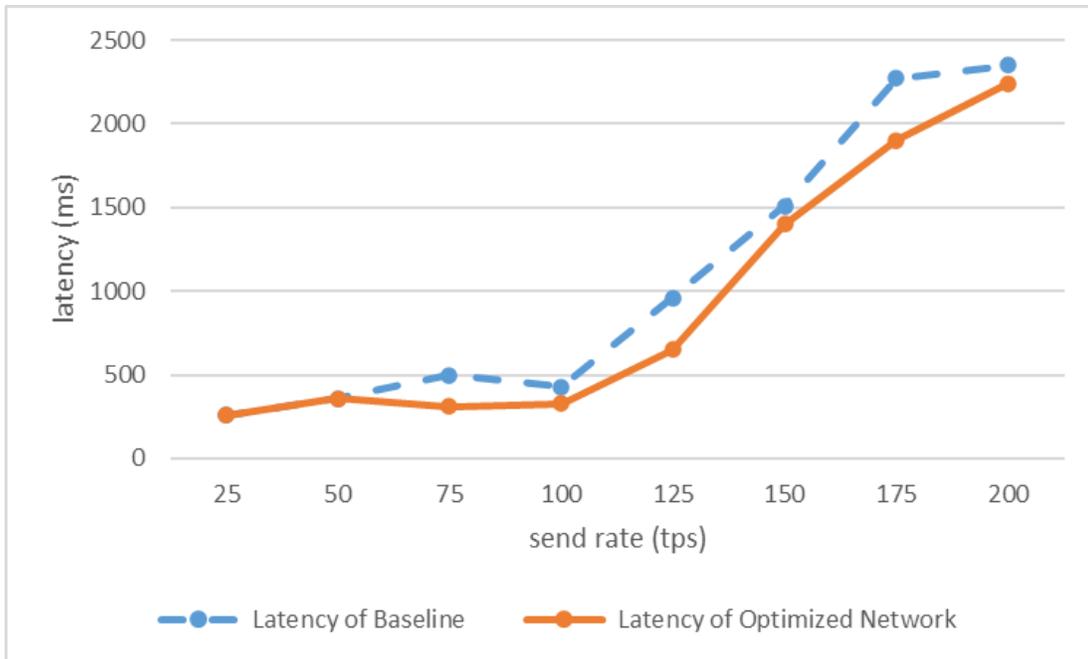


Figure 105: Evaluation of transaction latency with five clients (baseline and optimized network)

6.3.2 Evaluation Results of the Fuzzy Logic in Clinical Trial Testbed

This experiment evaluated the performance of the baseline, optimized network, and the fuzzy logic scheme. Four tests were performed by varying the number of clients over different transaction send rate (range from 25 – 200 tps). Figure 106 plots the experimental results of the baseline and the fuzzy logic scheme in terms of average transaction throughput with 1 client. The transaction throughput increased linearly with the increase in send rate until it reached around 100 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the fuzzy logic and the baseline network was 125.9 tps, and 98.7 tps with a 27.6% increase of the transaction throughput. Figure 107 plots the experimental results of the baseline and the fuzzy logic scheme in terms of average transaction latency with 1 client. When the send rate was 200

tps, the transaction latency of the fuzzy logic and the baseline was 380 ms and 650 ms, with a 41.5% reduction of transaction latency.

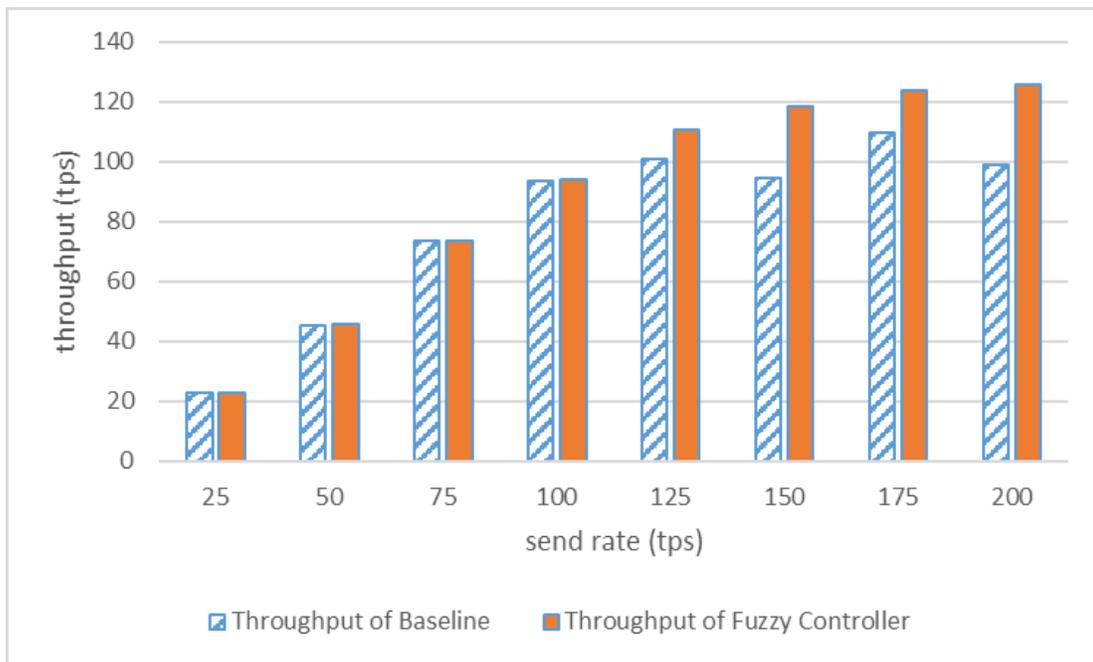


Figure 106: Evaluation of transaction throughput with 1 client (baseline and fuzzy logic)

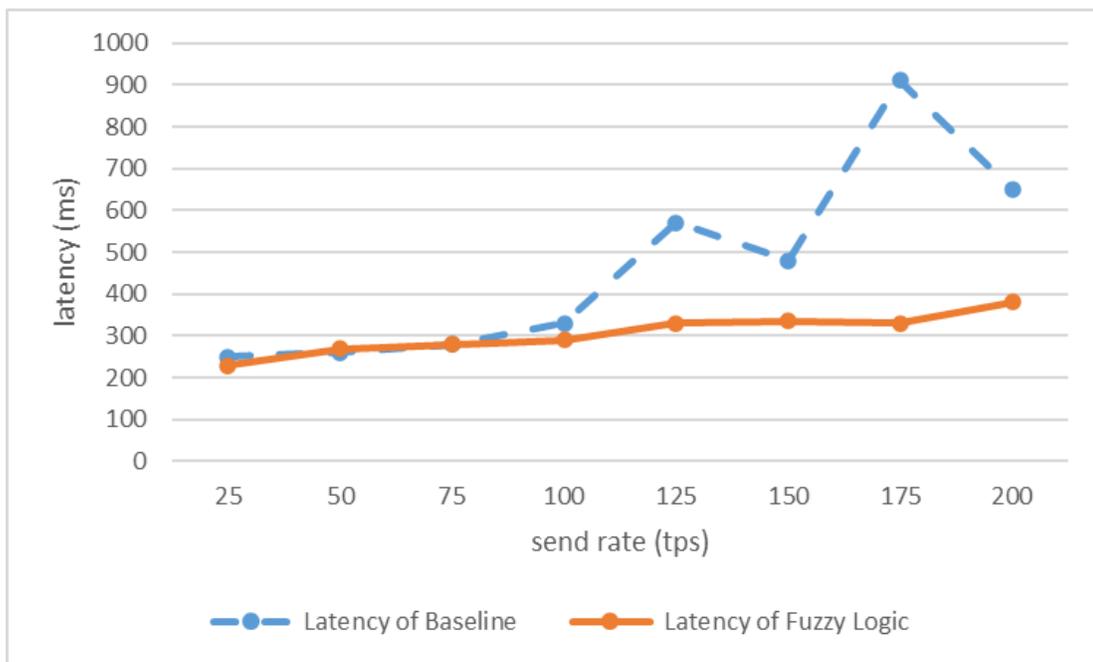


Figure 107: Evaluation of transaction latency with 1 client (baseline and fuzzy logic)

Figure 108 plots the experimental results of the baseline and the fuzzy logic scheme in terms of average transaction throughput with 5 clients. The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 150 tps, the transaction throughput of the fuzzy logic and the baseline was 152.9 tps, and 135.5 tps with a 12.8% increase of the transaction throughput. Figure 109 plots the experimental results of the baseline and the fuzzy logic scheme in terms of average transaction latency with 5 clients. It is observed that the baseline network generated more transaction latency than the fuzzy logic scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the fuzzy logic and the baseline was 1720 ms and 2350 ms, with a 36.6% reduction of transaction latency. This experiment results indicate that the fuzzy logic scheme shows better performance than the baseline concerning transaction latency and transaction throughput.

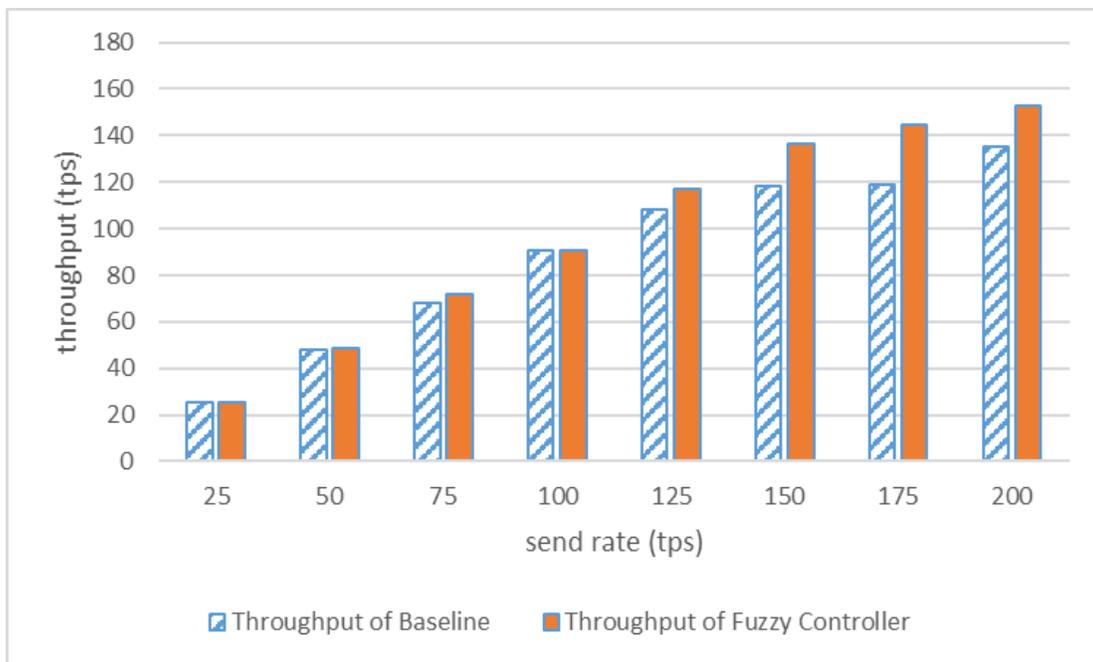


Figure 108: Evaluation of transaction throughput with 5 clients (baseline and fuzzy logic)

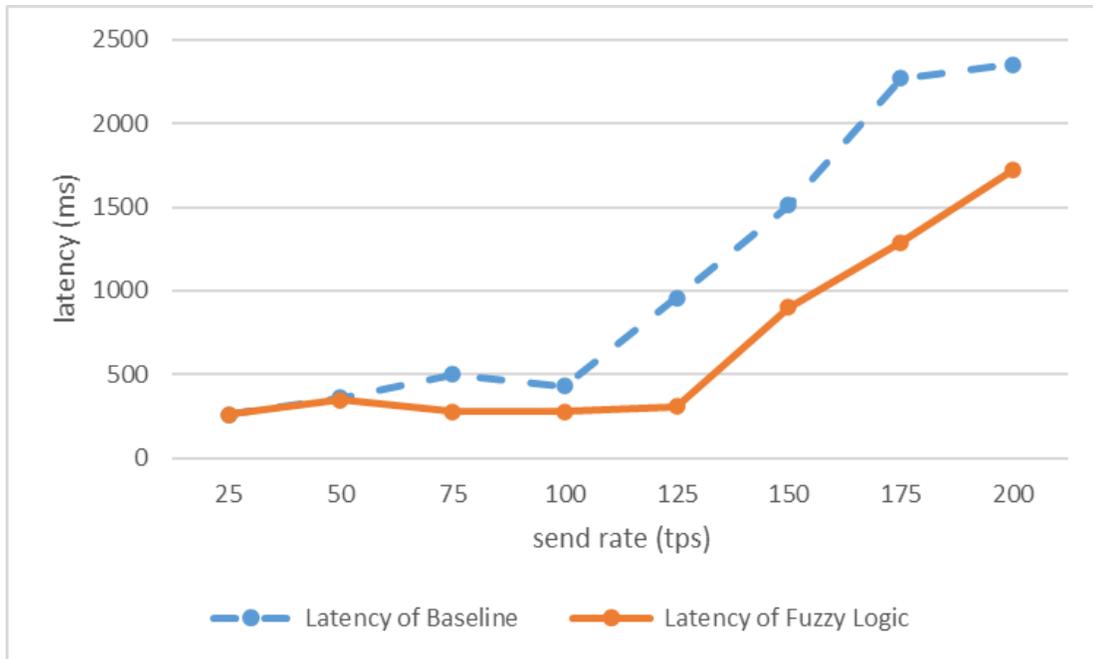


Figure 109: Evaluation of transaction latency with 5 clients (baseline and fuzzy logic)

Figure 110 plots the experimental results of the optimized network and the fuzzy logic scheme in terms of average transaction throughput with 1 client. The transaction throughput increased linearly with the increase in send rate until it flattened out at around 100 tps, as shown in Figure 110. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the throughput of the fuzzy logic and the optimized network was 130.9 tps, and 118 tps with a 10.9% increase of the transaction throughput. Figure 111 plots the experimental results of the optimized network and the fuzzy logic scheme in terms of average transaction latency with 1 client. For the transaction latency, it is observed that the optimized network generated more latency than the fuzzy logic scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the fuzzy logic and the optimized network was 310 ms and 520 ms, with a 40.4% reduction of transaction latency.

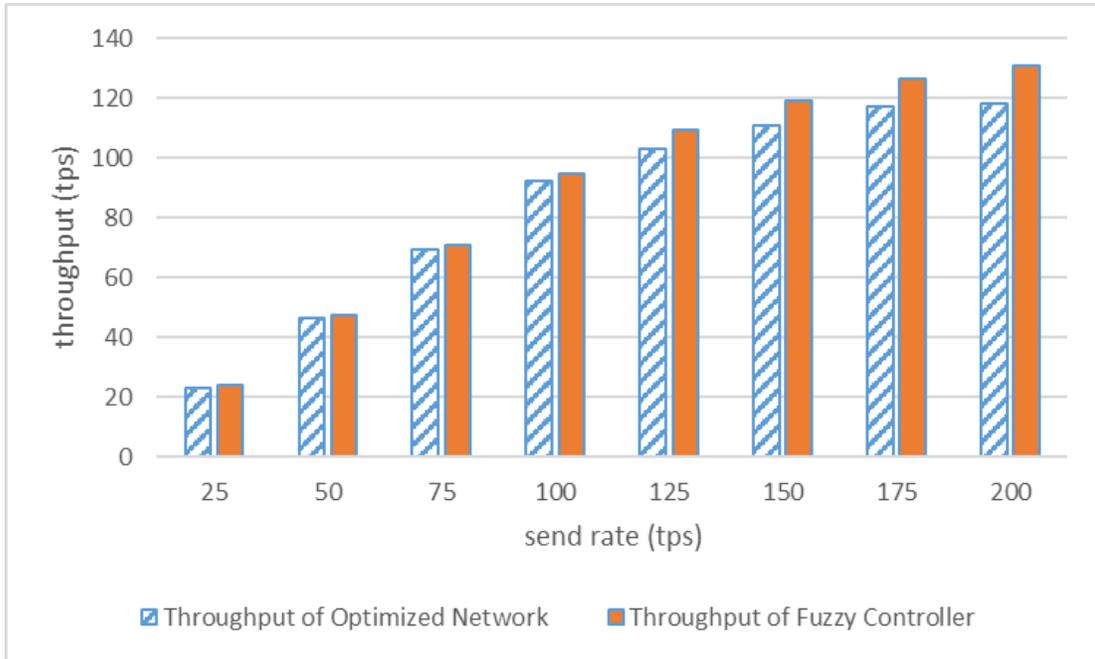


Figure 110: Evaluation of transaction throughput with one client (optimized network and fuzzy logic)

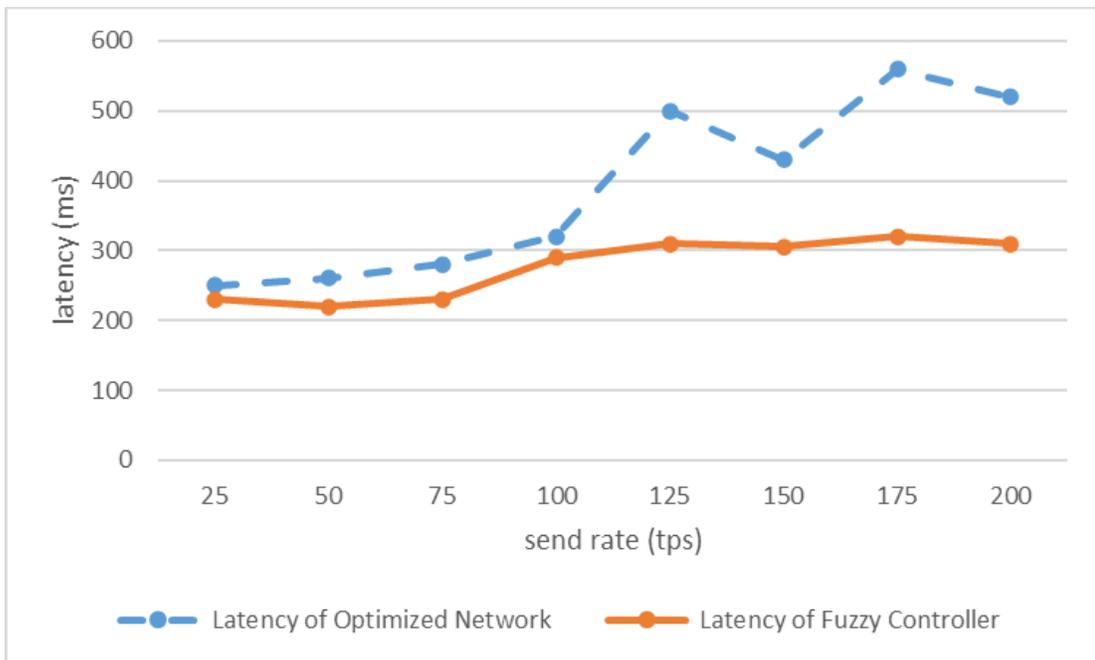


Figure 111: Evaluation of transaction latency with one client (optimized network and fuzzy logic)

Figure 112 plots the experimental results of the optimized network and the fuzzy logic scheme in terms of average transaction throughput with 5 clients. The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the throughput of the fuzzy logic and the optimized network was 156 tps and 137.2 tps, with a 13.7% increase of the transaction throughput. Figure 113 plots the experimental results of the optimized network and the fuzzy logic scheme in terms of average transaction latency with 5 clients. It is observed that the optimized network generated more transaction latency than the fuzzy logic scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the fuzzy logic and the optimized network was 1610 ms and 2240 ms, with a 28.1% reduction of transaction latency. This experiment results indicate that the fuzzy logic scheme can improve the performance concerning transaction latency and transaction throughput.

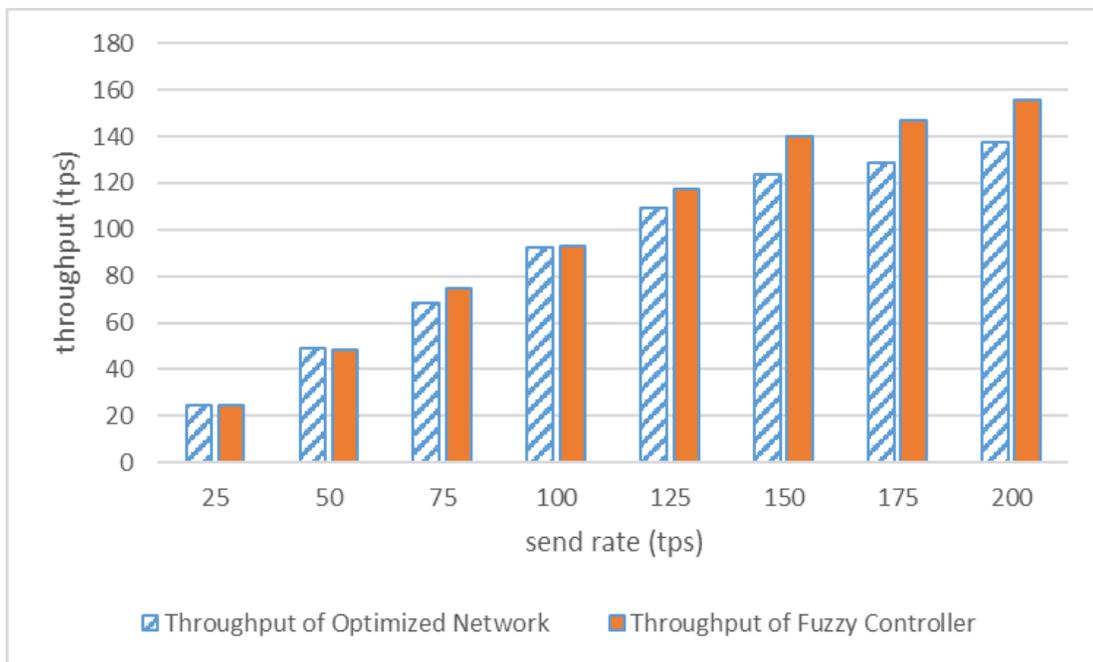


Figure 112: Evaluation of transaction throughput with five clients (optimized network and fuzzy logic)

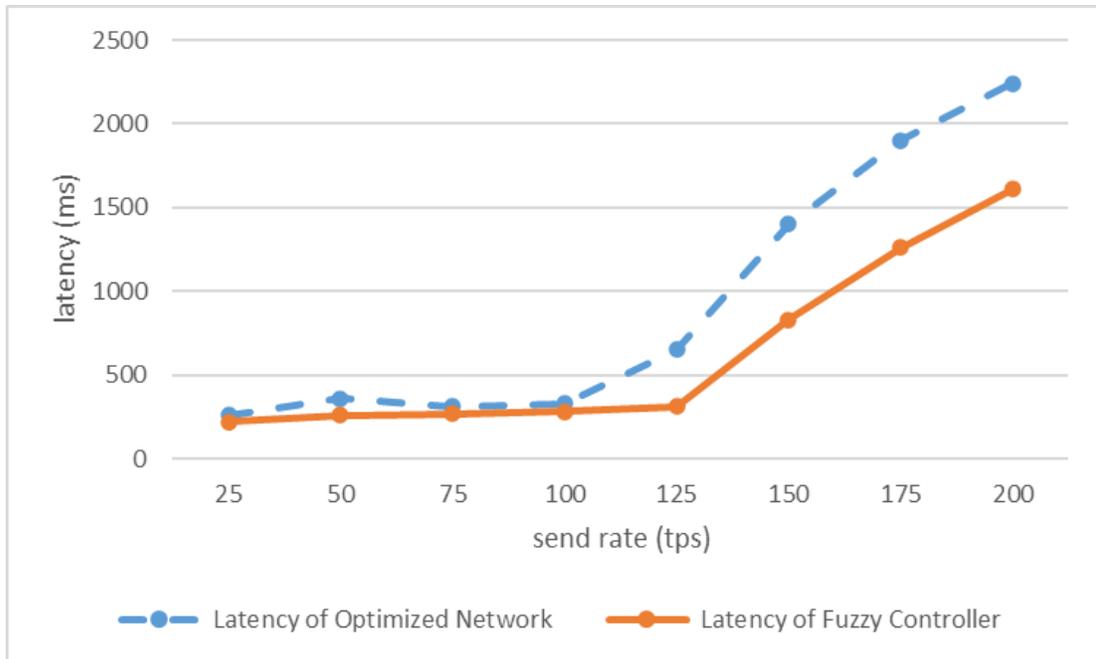


Figure 113: Evaluation of transaction latency with five clients (optimized network and fuzzy logic)

6.3.3 Evaluation Results of the Learning to Prediction in Clinical Trial Testbed

This experiment evaluated the performance of the baseline, optimized network, and the learning to prediction scheme. Four tests were performed by varying the number of clients over different transaction send rate (range from 25 – 200 tps). Figure 114 plots the experimental results of the baseline and the learning to prediction scheme in terms of average transaction throughput with 1 client. The transaction throughput increased linearly with the increase in send rate until it reached around 100 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the learning to prediction and the baseline set was 128 tps, and 98.7 tps with a 29.7% increase of the transaction throughput. Figure 115 plots the experimental results of the baseline and the learning to prediction scheme in terms of average transaction latency with 1

client. It is observed that the baseline network generated more transaction latency than the learning to prediction scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the learning to prediction and the baseline was 340 ms and 650 ms, with a 47.7% reduction of transaction latency.

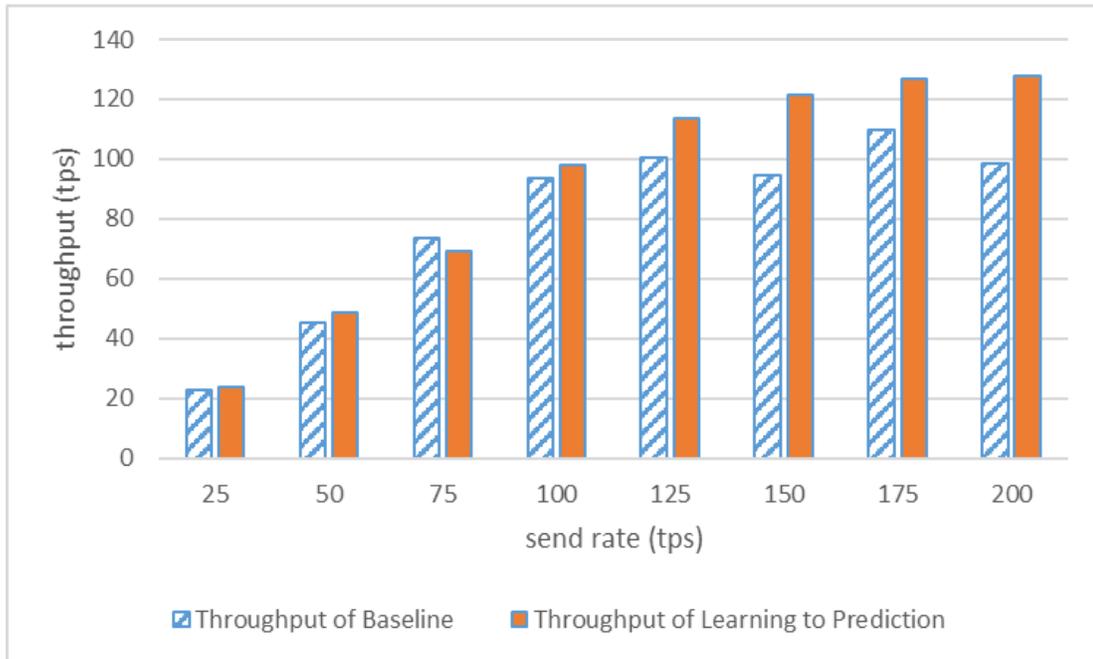


Figure 114: Evaluation of transaction throughput with one client (baseline and learning to prediction)

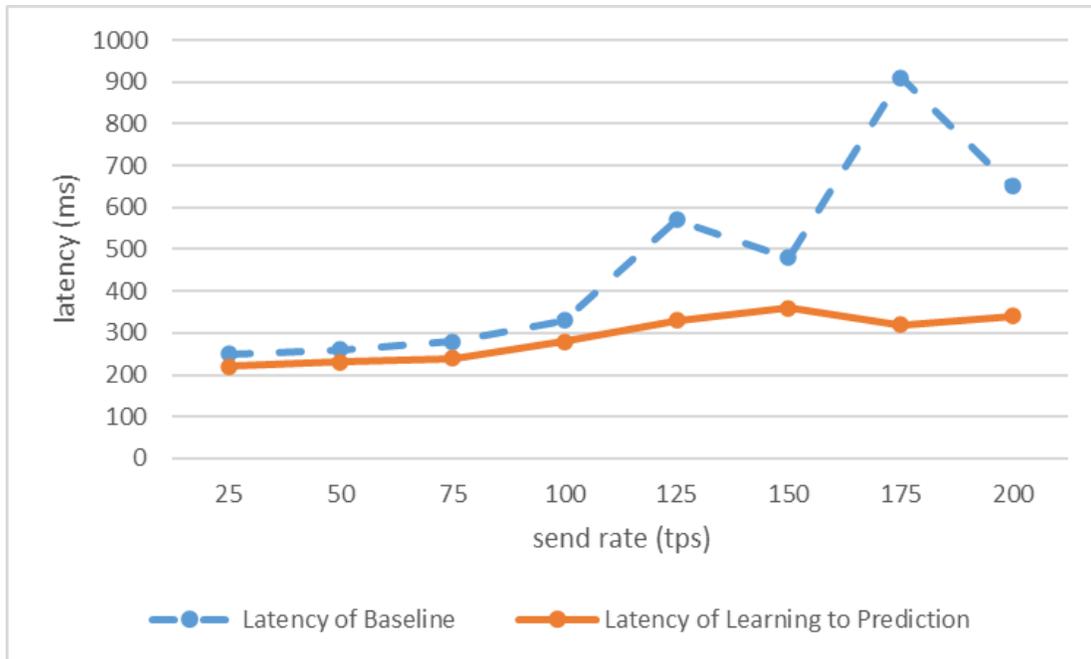


Figure 115: Evaluation of transaction latency with one client (baseline and learning to prediction)

Figure 116 plots the experimental results of the optimized network and the learning to prediction scheme in terms of average transaction throughput with 5 clients. The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the learning to prediction and the baseline was 158.8 tps, and 135.5 tps with a 17.2% increase of the transaction throughput. Figure 117 plots the experimental results of the optimized network and the learning to prediction scheme in terms of average transaction latency with 5 clients. It is observed that the baseline network generated more latency than the learning to prediction scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the learning to prediction and the baseline was 1410 ms and 2350 ms, with a 40% reduction of transaction latency.

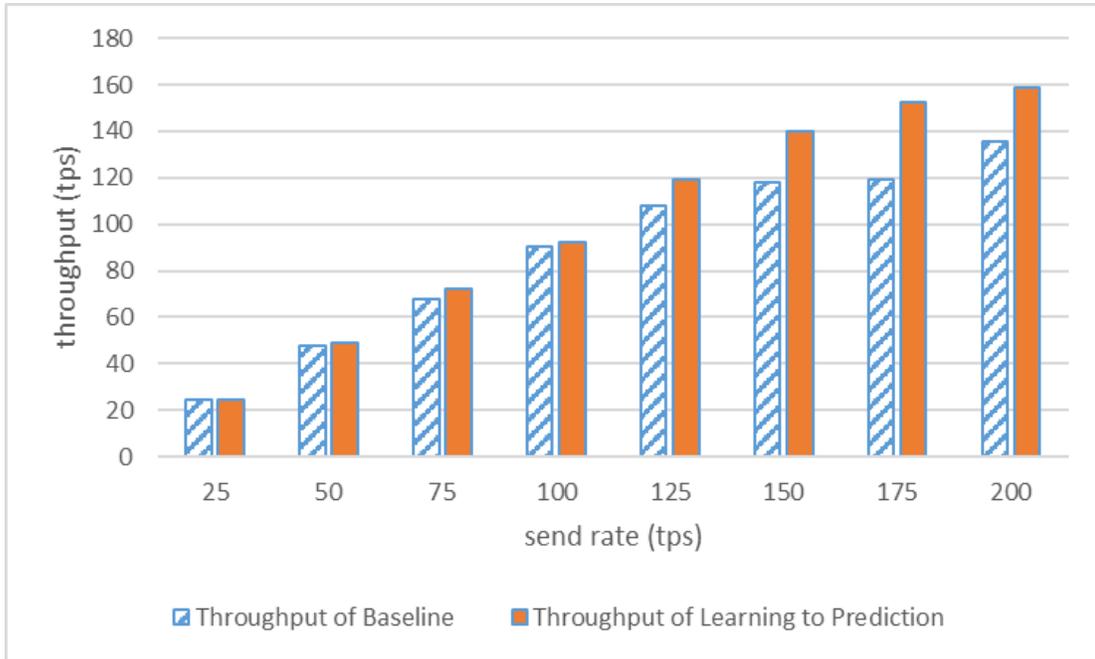


Figure 116: Evaluation of transaction throughput with five clients (baseline and learning to prediction)

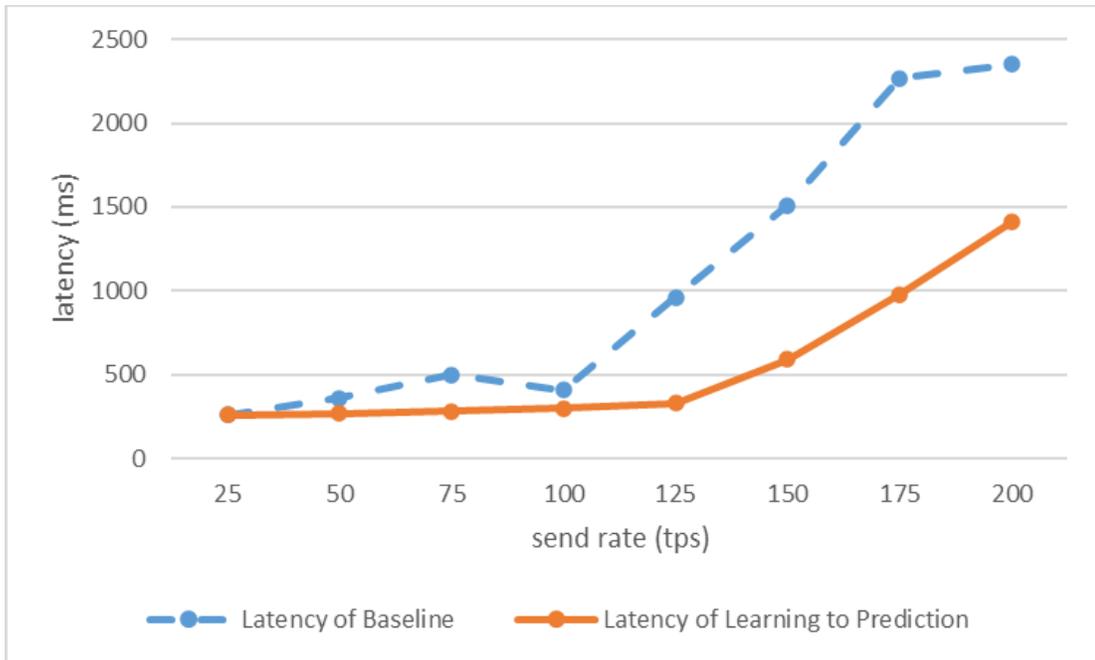


Figure 117: Evaluation of transaction latency with five clients (baseline and learning to prediction)

Figure 118 plots the experimental results of the baseline and the learning to prediction scheme in terms of average transaction throughput with 1 client. The transaction throughput increased linearly with the increase in send rate until it reached around 125 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the learning to prediction and the optimized network was 134.2 tps, and 118 tps with a 13.7% increase of the transaction throughput. Figure 119 plots the experimental results of the baseline and the learning to prediction scheme in terms of average transaction latency with 1 client. It is observed that the baseline network generated more latency than the learning to prediction scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the learning to prediction and the baseline was 280 ms and 520 ms, with a 46.2% reduction of transaction latency.

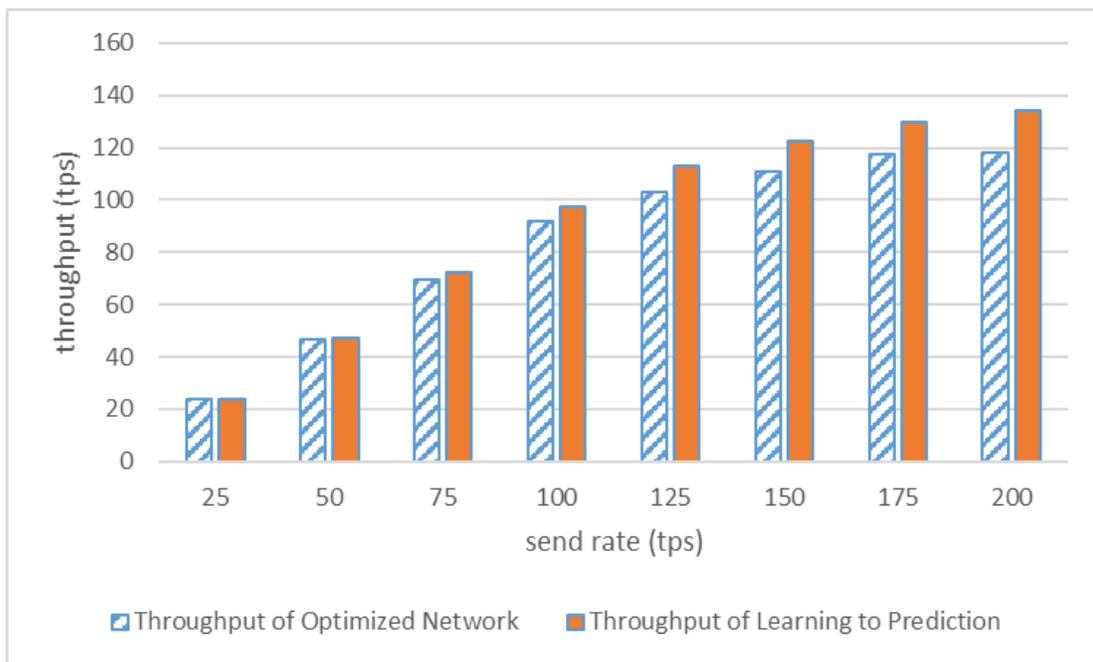


Figure 118: Evaluation of transaction throughput with 1 client (optimized network and learning to prediction)

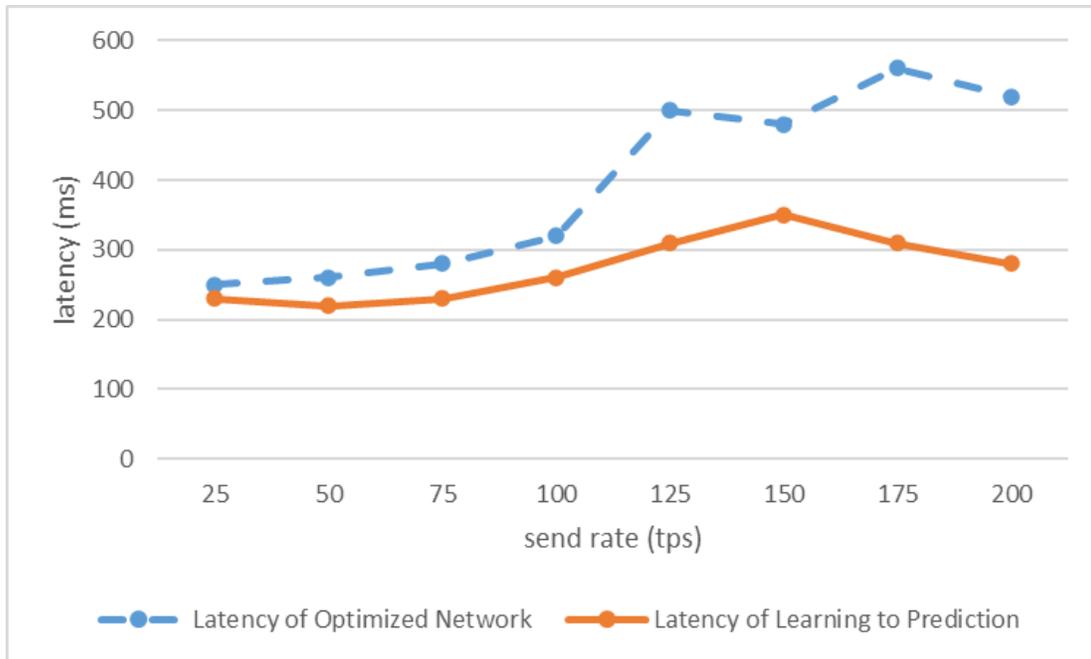


Figure 119: Evaluation of transaction latency with 1 client (optimized network and learning to prediction)

Figure 120 plots the experimental results of the optimized network and the learning to prediction scheme in terms of average transaction throughput with 5 clients. The transaction throughput increased linearly with the increase in send rate until it reached around 150 tps. The growth of transaction throughput decreased significantly and approached to a flat when the send rate was above this point. When the send rate was 200 tps, the transaction throughput of the learning to prediction and the optimized network was 162.5 tps, and 137.2 tps with an 18.4% increase of the transaction throughput. Figure 121 plots the experimental results of the optimized network and the learning to prediction scheme in terms of average transaction latency with 5 clients. It is observed that the baseline network generated more latency than the learning to prediction scheme when the send rate was above the saturation point. When the send rate was 200 tps, the transaction latency of the learning to prediction and the optimized network was 1230 ms and 2240 ms, with an 82.1% reduction of transaction latency.

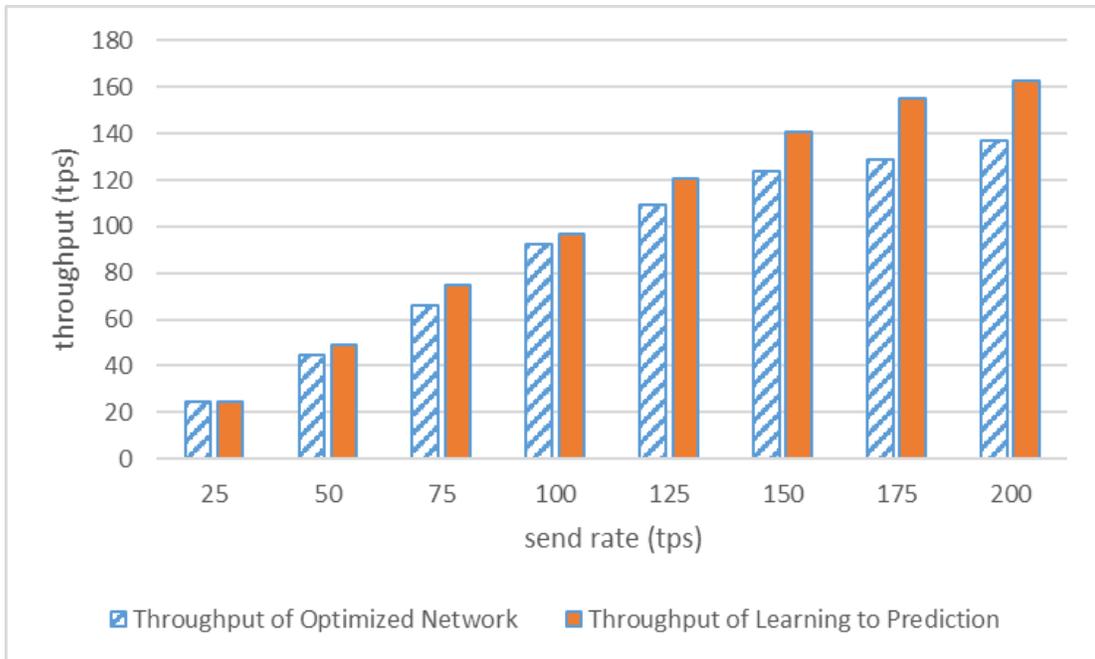


Figure 120: Evaluation of transaction throughput with 5 clients (optimized network and learning to prediction)

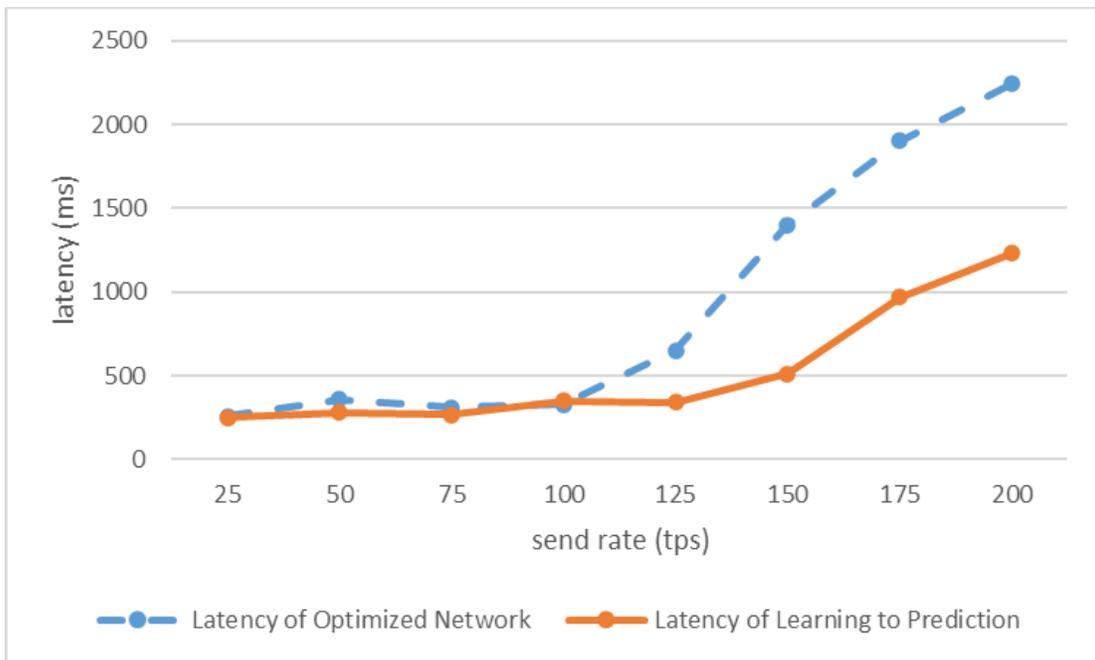


Figure 121: Evaluation of transaction latency with 5 clients (optimized network and learning to prediction)

Table 16 describes the analysis results by comparing the proposed two transaction traffic control mechanism according to the experiment results. For the case of the baseline network using one client, the network with optimized configurable parameters increases the transaction throughput by 19.6% and decrease the transaction latency by 20% compared to the baseline network. The transaction throughput is increased by 27.6%, and the transaction latency is decreased by 41.5% with the transaction traffic control mechanism based on fuzzy logic. The transaction throughput is increased by 29.7%, and the transaction latency is decreased by 47.7% with the transaction traffic control mechanism based on learning to prediction. For the case of the optimized network using one client, the transaction throughput is increased by 10.9%, and the transaction latency is decreased by 40.4% with the transaction traffic control mechanism based on fuzzy logic. The transaction throughput is increased by 13.7%, and the transaction latency is decreased by 46.2% with the transaction traffic control mechanism based on learning to prediction.

For the case of the baseline network using five clients, the network with optimized configurable parameters increases the transaction throughput by 9.3% and decrease the transaction latency by 17.4% compared to the baseline network. The transaction throughput is increased by 21.8%, and the transaction latency is decreased by 26.8% with the transaction traffic control mechanism based on fuzzy logic. The transaction throughput is increased by 26.5%, and the transaction latency is decreased by 40% with the transaction traffic control mechanism based on learning to prediction. For the case of the optimized network using five clients, the transaction throughput is increased by 13.7%, and the transaction latency is decreased by 17% with the transaction traffic control mechanism based on fuzzy logic. The transaction throughput is increased by 18.4%, and the transaction latency is decreased by 36.6% with the transaction traffic control mechanism based on learning to prediction.

Table 16: Comparison analysis of the performance evaluation

Number of Clients	Component	Baseline	Baseline with Fuzzy Logic	Baseline with Learning to Prediction	Optimized Network	Optimized Network with Fuzzy Logic	Optimized Network with Learning to Prediction
1	Avg Transaction Throughput (tps)	98.7	125.9	128	118	130.9	134.2
	Avg Transaction Latency (ms)	650	380	340	520	310	280
5	Avg Transaction Throughput (tps)	125.5	152.9	158.8	137.2	156	162.5
	Avg Transaction Latency (ms)	2350	1720	1410	1940	1610	1230

To prove the scalability of the designed approaches, we applied two proposed transaction traffic control mechanisms into the Accelerator [38], and the evaluation results are shown in Table 17. For the case of the baseline network using one client, the network with Accelerator increases the transaction throughput by 67.2% and decrease the transaction latency by 35.4% compared to the baseline network. The transaction throughput is increased by 77.7%, and the transaction latency is decreased by 52.3% with the fuzzy logic-based approach. The transaction throughput is

increased by 80.7%, and the transaction latency is decreased by 56.9% with the learning to prediction-based approach.

For the case of the baseline network using five clients, the network with Accelerator increases the transaction throughput by 82.1% and decrease the transaction latency by 20% compared to the baseline network. The transaction throughput is increased by 96.9%, and the transaction latency is decreased by 37.9% with the fuzzy logic-based approach. The transaction throughput is increased by 99.9%, and the transaction latency is decreased by 43.6% with the learning to prediction-based approach.

The experiment results indicate that the proposed transaction traffic control approaches can significantly improve the transaction transmission environment by enhancing transaction throughput and reducing transaction latency. This approach can also be applied to the existing blockchain performance-enhancing tool without modifying the original architecture. In this paper, we only tested the proposed approach in Hyperledger Fabric blockchain. However, existing smart contract enabled blockchain platforms can significantly benefit from the significance of the proposed approach to improve the transaction processing capability. This approach has the potential to be applied to existing enterprise use cases, which have a high demand for transaction throughput and transaction latency.

Table 17: Comparison analysis of the performance evaluation with Accelerator

Number of Clients	Component	Baseline	Baseline with Accelerator [38]	Baseline with Accelerator Using Fuzzy Logic	Baseline with Accelerator Using Learning to Prediction
1	Avg Transaction Throughput (tps)	98.7	165	175.4	178.4
	Avg Transaction Latency (ms)	650	420	310	280
5	Avg Transaction Throughput (tps)	135.5	246.7	266.8	270.8
	Avg Transaction Latency (ms)	2350	1880	1460	1325

7. Conclusion

The performance and scalability of Information Technology (IT) systems have always been a primary non-functional requirement used to measure the production readiness of an implementation project. Blockchain networks are the same, providing a decentralized model that enables peers to collaborate and build trust through business networks. Each peer node must perform operations and communicate with other peers to confirm transactions, reach consensus, and update the status of the shared ledger. To make informed architectural decisions about blockchain-based solutions, it is essential to be aware of factors and areas of improvement that affect blockchain performance. Several factors affect the performance of the blockchain network, so benchmarking or testing the performance of the blockchain is not a simple exercise. Besides, when processing end-to-end business transactions, the use cases being implemented and the off-chain component architecture, as well as the design of the blockchain-based solution, should also be considered.

This paper examines various configurable factors that can affect the Hyperledger Fabric blockchain performance. These configurable parameters can be mainly summarized into two categories: software and hardware. Software-based parameters include block size, block frequency, ledger database, ordering service, the programming language of smart contract, use of TLS, number of clients, number endorser peers, number of organizations, and the endorsement policy. Hardware-based parameters include the number of vCPUS, memory allocation, disk type and speed, network speed, and CPU speed. A comprehensive experiment is carried out to analyze the impact on network performance for each parameter.

As a consequence, an optimized network configuration is set up in terms of the experimental results. Besides, this paper proposes two transaction traffic control approaches using fuzzy logic to improve the blockchain performance. For the first approach, we implement the fuzzy controller

in the smart contract. Moreover, for the second approach, we implement additional learning to prediction module to enhance the performance of the fuzzy controller in the smart contract. Real-time network feedback is used as input parameters, and the fuzzy controller adjusts the transaction traffic across the whole network accordingly. A clinical trial testbed is used as the experiment environment to evaluate the performance of the proposed transaction traffic control mechanisms. We evaluate the performance of the optimized network by comparing it with the original network. Besides, we evaluate the performance of the proposed transaction traffic control mechanisms by applying them in the original network and optimized network. The experiment results indicate that the designed solutions can improve the network throughput while reducing the latency. The evaluation results show that the proposed approaches can enhance network performance compared to the baseline and optimized schemes. For the case of baseline scheme using one client, the network with optimized configurable parameters increases the transaction throughput by 19.6% and decrease the transaction latency by 20% compared to the baseline. The transaction throughput is increased by 27.6% and 29.7%; the transaction latency is decreased by 41.5% and 47.7% with the fuzzy logic and learning to prediction approaches, respectively.

For the case of the optimized scheme using one client, the transaction throughput is increased by 10.9% and 13.7%; the transaction latency is decreased by 40.4% and 46.2% with the fuzzy logic and learning to prediction approaches, respectively. For the case of baseline scheme using five clients, the network with optimized configurable parameters increases the transaction throughput by 9.3% and decrease the transaction latency by 17.4% compared to the baseline network. The transaction throughput is increased by 21.8% and 26.5%; the transaction latency is decreased by 26.8% and 40% with the fuzzy logic and learning to prediction approaches, respectively. For the case of the optimized network using five clients, the transaction throughput is increased by 13.7% and 18.4%; the transaction latency is decreased 17% and 36.6% with the fuzzy logic and learning to prediction approach.

Furthermore, the proposed approaches are deployed with one of the existing performance-enhancing tools, and the results indicate that the proposed approaches integrate with the existing performance-enhancing approach and further improve the blockchain performance. For the case of using one client, the network with Accelerator increases the transaction throughput by 67.2% and decrease the transaction latency by 35.4% compared to the baseline. The transaction throughput is increased by 77.7% and 80.7%; the transaction latency is decreased by 52.3% and 56.9% with the fuzzy logic and learning to prediction approaches, respectively. For the case of using five clients, the network with Accelerator increases the transaction throughput by 82.1% and decreases the transaction latency by 20% compared to the baseline. The transaction throughput is increased by 96.9% and 99.9%; the transaction latency is decreased by 37.9% and 43.6% with the fuzzy logic and learning to prediction approaches, respectively.

One limitation of this study is that all the benchmark experiments are performed in a single-host virtual machine, and we only consider software-based configurable parameters to simplify the implementation. Besides, the blockchain network is deployed in a Local Area Network (LAN) that is not suitable for the production environment. Future work will refine the prototype system, and we will replicate the results by using a cloud service such as AWS or IBM Blockchain to evaluate the impact of various hardware components such as memory allocation, disk type and speed, network speed, and CPU speed. Furthermore, we will deploy the proposed approaches into some existing smart contract enabled blockchain platforms such as Ethereum and Corda to test the applicability of the proposed approaches.

References

1. Maull, Roger, et al. "Distributed ledger technology: Applications and implications." *Strategic Change* 26.5 (2017): 481-489.
2. V. Espinel, D. O'Halloran, E. Brynjolfsson, and D. O'Sullivan, "Deep shift, technology tipping points and societal impact," in New York: World Economic Forum–Global Agenda Council on the Future of Software & Society (REF 310815), 2015.
3. Zheng, Zhibin, et al. "Blockchain challenges and opportunities: A survey." *International Journal of Web and Grid Services* 14.4 (2018): 352-375.
4. S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han and F. Wang, "Blockchain-Enabled Smart Contracts: Architecture, Applications, and Future Trends," in *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266-2277, Nov. 2019.
5. Omohundro, S. (2014) 'Cryptocurrencies, smart contracts, and artificial intelligence,' *AI Matters*, Vol. 1, No. 2, pp.19–21.
6. K. Salah, M. H. U. Rehman, N. Nizamuddin and A. Al-Fuqaha, "Blockchain for AI: Review and Open Research Challenges," in *IEEE Access*, vol. 7, pp. 10127-10149, 2019, doi: 10.1109/ACCESS.2018.2890507.
7. E. Osaba, E. Onieva, A. Moreno, P. Lopez-Garcia, A. Perallos and P. G. Bringas, "Decentralised intelligent transport system with distributed intelligence based on classification techniques", *IET Intell. Transp. Syst.*, vol. 10, no. 10, pp. 674-682, Dec. 2016.
8. Ai and Blockchain are Taking Root in the Global Agriculture Industry, 2018, [online] Available: <https://www.entefy.com/blog/post/570/ai-and-blockchain-are-taking-root-in-the-global-agriculture-industry/>.

9. Autonomous Supply Chain Will Soon be Empowered by IoT AI and Blockchain-Here's How, 2018, [online] Available: <https://iot.eetimes.com/autonomous-supply-chain-will-soon-be-empowered-by-iot-ai-and-blockchain-heres-how>.
10. E. C. Ferrer, The blockchain: A new framework for robotic swarm systems, 2016, [online] Available: <https://arxiv.org/abs/1608.00695>.
11. Hang, L.; Kim, D.-H. Reliable Task Management Based on a Smart Contract for Runtime Verification of Sensing and Actuating Tasks in IoT Environments. *Sensors* 2020, 20, 1207.
12. Hang, L.; Kim, D.-H. SLA-Based Sharing Economy Service with Smart Contract for Resource Integrity in the Internet of Things. *Appl. Sci.* 2019, 9, 3602.
13. Yuan, Y.; Wang, F.Y. Towards blockchain-based intelligent transportation systems. In *Proceedings of the 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, Rio de Janeiro, Brazil, 1–4 November 2016; pp. 2663–2668.
14. Gordon, W.J.; Catalini, C. Blockchain Technology for Healthcare: Facilitating the Transition to Patient-Driven Interoperability. *Comput. Struct. Biotechnol. J.* 2018, 16, 224–230.
15. Dubovitskaya, A.; Xu, Z.; Ryu, S.; Schumacher, M.; Wang, F. Secure and trustable electronic medical records sharing using blockchain. In *Proceedings of the AMIA 2017, American Medical Informatics Association Annual Symposium*, Washington, DC, USA, 4–8 November 2017.
16. Hang, L.; Choi, E.; Kim, D.-H. A Novel EMR Integrity Management Based on a Medical Blockchain Platform in Hospital. *Electronics* 2019, 8, 467.
17. Paralkar, K.; Yadav, S.; Kumari, S.; Kulkarni, A.; Pingat, S.P. Photogroup: Decentralized Web Application Using Ethereum Blockchain. *Int. Res. J. Eng. Technol.* 2018, 5, 489–492.
18. Raval, S. *Decentralized Applications: Harnessing Bitcoin's Blockchain Technology*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2016.

19. Brito, J.; Shadab, H.B.; Castillo O’Sullivan, A. Bitcoin financial regulation: Securities, derivatives, prediction markets, and gambling. *Columbia Sci. Technol. Law Rev.* 2014.
20. MacDonald, T.J.; Allen, D.W.E.; Potts, J. Blockchains and the Boundaries of Self-Organized Economies: Predictions for the Future of Banking. In *Banking Beyond Banks and Money*; Tasca, P., Aste, T., Pelizzon, L., Perony, N., Eds.; Springer: Cham, Switzerland, 2016; pp. 279–296.
21. Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564, doi: 10.1109/BigDataCongress.2017.85.
22. Nakamoto, Satoshi. "Bitcoin: A peer-to-peer electronic cash system.(2008)." (2008).
23. Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." *Ethereum project yellow paper 151.2014* (2014): 1-32.
24. Buterin, Vitalik. "A next-generation smart contract and decentralized application platform." *white paper 3.37* (2014).
25. Z. Li, J. Kang, R. Yu, D. Ye, Q. Deng and Y. Zhang, "Consortium Blockchain for Secure Energy Trading in Industrial Internet of Things," in *IEEE Transactions on Industrial Informatics*, vol. 14, no. 8, pp. 3690-3700, Aug. 2018, doi: 10.1109/TII.2017.2786307.
26. "Ripple", <https://ripple.com/>.
27. "Stellar", <https://www.stellar.org/>.
28. "Corda", <https://www.corda.net/>.
29. Cachin, Christian. "Architecture of the hyperledger blockchain fabric." *Workshop on distributed cryptocurrencies and consensus ledgers*. Vol. 310. 2016.
30. "MultiChain", <https://www.multichain.com/>.
31. "OpenChain", <https://www.openchain.org/>.

32. J. A. Garay, A. Kiayias and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications" in EUROCRYPT volume 9057 of LNCS, Springer, pp. 281-310, 2015.
33. Q. Nasir, I. A. Qasse, M. A. Talib, and A. B. Nassif, "Performance Analysis of Hyperledger Fabric Platforms," Security and Communication Networks, vol. 2018, Article ID 3976093, 14 pages, 2018. <https://doi.org/10.1155/2018/3976093>.
34. Zheng, Z.; Xie, S.; Dai, H.N.; Wang, H. Blockchain Challenges and Opportunities: A Survey. Available online: <http://inpluslab.sysu.edu.cn/files/blockchain/blockchain.pdf>.
35. L. Aniello, R. Baldoni, E. Gaetani, F. Lombardi, A. Margheri and V. Sassone, "A Prototype Evaluation of a Tamper-Resistant High Performance Blockchain-Based Transaction Log for a Distributed Database," 2017 13th European Dependable Computing Conference (EDCC), Geneva, 2017, pp. 151-154, doi: 10.1109/EDCC.2017.31.
36. Park Jin Hee, Choong Seon Hong. (2019). Blockchain Architecture Using Consensus Algorithm with Adjustable Validation and its Performance Improvement. The Korean Institute of Information Scientists and Engineers, (), 1218-1220.
37. Kwon, Minsu, and Heonchang Yu. "Performance Improvement of Ordering and Endorsement Phase in Hyperledger Fabric." 2019 Sixth International Conference on Internet of Things: Systems, Management and Security (IOTSMS). IEEE, 2019.
38. Lee, Hyojung, et al. "Multi-batch Scheduling for Improving Performance of Hyperledger Fabric based IoT Applications." 2019 IEEE Global Communications Conference (GLOBECOM). IEEE, 2019.
39. F. Lu, L. Gan, Z. Dong, W. Li, H. Jin and A. Y. Zomaya, "A Cache Enhanced Endorser Design for Mitigating Performance Degradation in Hyperledger Fabric," 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), Halifax, NS, Canada, 2018, pp. 1001-1006.

40. Sousa, Joao, Alysson Bessani, and Marko Vukolic. "A byzantine fault-tolerant ordering service for the hyperledger fabric blockchain platform." 2018 48th annual IEEE/IFIP international conference on dependable systems and networks (DSN). IEEE, 2018.
41. Dinh, Tien Tuan Anh, et al. "Blockbench: A framework for analyzing private blockchains." Proceedings of the 2017 ACM International Conference on Management of Data. 2017.
42. Miyamae, Takeshi, et al. "Performance improvement of the consortium blockchain for financial business applications." Journal of Digital Banking 2.4 (2018): 369-378.
43. Hyperledger Caliper—A Blockchain Benchmark Tool. Available online: <https://www.hyperledger.org/projects/caliper> (accessed on 25 May 2020).
44. Eyal, Ittay, et al. "Bitcoin-ng: A scalable blockchain protocol." 13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16). 2016.
45. Sukhwani, Harish. "Performance Modeling & Analysis of Hyperledger Fabric (Permissioned Blockchain Network)." Duke University: Duke, UK (2018).
46. P. Thakkar, S. Nathan and B. Viswanathan, "Performance Benchmarking and Optimizing Hyperledger Fabric Blockchain Platform," 2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS), Milwaukee, WI, 2018, pp. 264-276, doi: 10.1109/MASCOTS.2018.00034.
47. M. Kuzlu, M. Pipattanasomporn, L. Gurses and S. Rahman, "Performance Analysis of a Hyperledger Fabric Blockchain Framework: Throughput, Latency and Scalability," 2019 IEEE International Conference on Blockchain (Blockchain), Atlanta, GA, USA, 2019, pp. 536-540, doi: 10.1109/Blockchain.2019.00003.
48. Mamdani, E.H.; Assilian, S. An experiment in linguistic synthesis with a fuzzy logic controller. *Int. J. Man-Mach. Stud.* 1975, 7, 1–13.
49. Ullah, I.; Fayaz, M.; Kim, D. Improving Accuracy of the Kalman Filter Algorithm in Dynamic Conditions Using ANN-Based Learning Module. *Symmetry* 2019, 11, 94.

50. Ranganathan, A. The levenberg-marquardt algorithm. Tutor. LM Algorithm 2004, 11, 101–110.