

# Measurements, Analysis and Modeling of Hyperledger Fabric

**Abstract**—Bitcoin network

**Index Terms**—Bitcoin Network, Mining Pools, Malthusian Trap, Incentive Mechanism

## 1 INTRODUCTION

**B**ITCOIN [1] is a decentralized peer to peer (P2P) cryptocurrency that was first proposed by Satoshi Nakamoto in 2008. Without resorting to any trusted third party, Bitcoin adapts a cryptographic proof mechanism that enables anonymous peers to complete transactions through the P2P network. Blockchain is the core mechanism of the Bitcoin system. It not only records historical transactions from Bitcoin clients, but also prevents the Bitcoin network from double spending attacks [2]. The Bitcoin network participants, who maintain and update the ongoing chain of blocks, are called miners. These miners compete in a mining race driven by an incentive mechanism [3], [4], where the one who first solves the Bitcoin cryptographic puzzle [5] has the right to collect unconfirmed transactions into a new block, append the new block to the main chain, i.e., the longest chain of blocks, and gain some BTCs [6] as a mining reward.

## 2 PRELIMINARY

### 2.1 Cluster Configuration

This is about cluster configuration

### 2.2 MSP

Certificates

### 2.3 TLS

TLS secure protocol description

## 3 ENVIRONMENT

Hyperledger Fabric version 1.4  
Fabric SDK v1.0.0  
Jmeter

## 4 BLOCKTIME SOLO ORDERER MODE

### 4.1 Model Definition

### 4.2 Model of Orderer's Configuration

The configuration file, i.e., configtx.yaml, configures the orderer service as follows.

TABLE 1  
Configuration Notations

Notations	Descriptions
$N_{msg}$	MaxMessageCount
$S_{max}$	AbsoluteMaxBytes
$S_{pref}$	PreferredMaxBytes
$T_{batch}$	Time to generate a block when MaxMessageCount is not satisfied

TABLE 2  
Customized Notations

Notations	Descriptions
$T_{\sigma}$	Time to generate a block when MaxMessageCount is satisfied
$TAR$	Transaction arrival rate
$BlockTime$	Time it takes to generate a block
$TxDelay$	Time it takes to issue a transaction
$S_{payload}$	Size of transaction payload
$Throughput$	Throughput of transactions

```

1 BatchTimeout: 2s
2 BatchSize:
3   MaxMessageCount: 10
4   AbsoluteMaxBytes: 98 MB
5   PreferredMaxBytes: 512 KB

```

Case Study 1 without considering block size, we have the following model, Given a  $TAR$  of which  $\Delta t = 1$ , the time it costs is given as follows, the average block time is,

$$BlkTime = \begin{cases} \infty & TAR = 0 \\ \frac{T_{batch} N_{msg}}{TAR} & 0 < TAR \leq \frac{N_{msg}}{T_{batch}} \\ \left\lfloor \frac{TAR}{N_{msg}} - 1 \right\rfloor \cdot T_{\sigma} & \frac{N_{msg}}{T_{batch}} < TAR \leq \frac{N_{msg}}{T_{\sigma}} \\ \left\lfloor \frac{TAR}{N_{msg}} - 1 \right\rfloor \cdot T_{\sigma} + T_{batch} & otherwise \end{cases} \quad (1)$$

If  $TAR = 0$ , then  $BlockTime = \infty$ . It means that if there are no transactions, there are no blocks.

If  $0 < TAR \leq \frac{MaxMessageCount}{BatchTimeout}$ . It means that the number of transactions are less than MaxMessageCount

given a BatchTimeout. Therefore, blocks are created for each BatchTimeout.

If  $\frac{MaxMessageCount}{BatchTimeout} < TAR$ . It means that the number of transactions are larger than MaxMessageCount in each BatchTimeout. Therefore, blocks are created as soon as possible and  $\sigma$  is a small value.

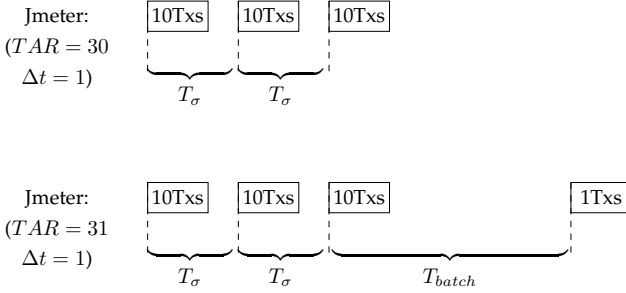


Fig. 1. The  $TAR$  Configurations

#### 4.2.1 Experiment 1: How $TAR$ affects $T_{sigma}$ and Throughput

In this section, we focus on how  $TAR$  affects the block time when MaxMessageCount is satisfied.

Experiment 1 – Setup: for normal  $TAR$  we can use workload generators “Thread Group” on Ubuntu 01; while for small  $TAR$  we can use other workload generators “Ultimate Thread Group” on Ubuntu 01.

Experiment 1 – Setup: for experiment analysis, we can check with logs in a peer. We have a program to analyze the log data, see *readdata.py* at Github benchmark module at the *q3v14* folder.

Experiment 1 – Observation 1: when  $TAR$  achieves a high value (e.g., 80 tps, 90 tps), then a block cannot be fully made used (e.g., 8 transactions per block + 2 transactions per block, instead of 10 transactions per block).

Experiment 1 – Observation 2: when  $TAR$  achieve a very high value (e.g., 200 tps), then a peer cannot handle such a high frequent event, and the SDK client will be rejected by the peer. See more error about  $TAR=200$  per peer, Error “peer0.org2.example.com:7051, PKIdid:6830efc127d4818973edd435ee7df6fa6fdf3e23286479b1e34112292665591b5 isn’t responsive: EOF”. – To solve this problem, we need more distributed machines.

TO DO – why 70 tps to 80 tps, much not-full used blocks are created?

Experiment 2 – Observation 1: for small  $TAR$  (e.g., 20 tps, 30 tps, 40 tps, 60 tps), setting of  $TAR$  (e.g., 11 tps, 21 tps, 31 tps, 51 tps) highly affect the  $T_{sigma}$ .

TODO –: for large  $TAR$  (e.g., 80 tps, 90 tps, 140 tps), why experiment 2 is much lower than experiment 1?

TO DO –: why  $TAR=91$  results in fewer null-fully used blocks than  $TAR=90$ ?

#### 4.2.2 Endorsement Policy (OR) on Transaction Delay

Experiment 3 - Setup: Endorsement Policy (OR) the source code is at Ubuntu 01 - nodeTest invoke3.js.

Experiment 3 - Setup: we can calculate with timestamps at invoke3.js.

Experiment 3 - Setup: we save the timestamp results with Jmeter in a res.xml file.

Experiment 3 - Setup: we can summary the timestamp results with readxml.py at Github benchmark module of q1 transaction arrival rate.

Experiment 3 - Setup: get proposal approval from local-host, see invoke3.js nodeTest in ubuntu 01.

Experiment 3: Set up: get proposal approval from remote peer, see invoke3a.js nodeTest in ubuntu 01.

### 4.3 Transaction Delay (For Peer)

Here we need to discuss something about transaction delay in this section.

#### 4.3.1 Transaction Size

How transaction size affects transaction delay, transaction loss

#### 4.3.2 Experiment 1: Transaction Size 1 byte

#### 4.3.3 Experiment 2: Transaction Size 300 byte

#### 4.3.4 Experiment 3: Transaction Size 10 Mbyte

#### 4.3.5 Endorsement Policy

#### 4.3.6 Experiment 1: Policy a

#### 4.3.7 Experiment 2: Policy b

#### 4.3.8 Experiment 3: Policy c

Here we need to know how endorsement policy affect transaction delay

## 5 KAFKA ORDERER MODE

### 5.1 Model of Kafka Orderer’s Configuration

The configuration file of Kafka

## REFERENCES

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [2] G. O. Karame, E. Androulaki, M. Roeschlin, A. Gervais, and S. Čapkun, “Misbehavior in bitcoin: A study of double-spending and accountability,” *ACM Transactions on Information and System Security (TISSEC)*, vol. 18, no. 1, p. 2, 2015.
- [3] Y. Lewenberg, Y. Bachrach, Y. Sompolinsky, A. Zohar, and J. S. Rosenschein, “Bitcoin mining pools: A cooperative game theoretic analysis,” in *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2015, pp. 919–927.
- [4] O. Schrijvers, J. Bonneau, D. Boneh, and T. Roughgarden, “Incentive compatibility of bitcoin mining pool reward functions,” in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 477–498.
- [5] I. Giechaskiel, C. Cremers, and K. B. Rasmussen, “On bitcoin security in the presence of broken cryptographic primitives,” in *European Symposium on Research in Computer Security*. Springer, 2016, pp. 201–222.
- [6] BTC. [Online]. Available: <https://en.bitcoin.it/wiki/Bitcoin>, Accessed on 31 January 2019.

TABLE 3  
Experiment 1:  $TAR$  affects  $T_{sigma}$ ,  $numberoftransactionsofablock$  and  $numberoftransactionsrejected$

$TAR$	20	30	40	50	60	70	80	90	100	<u>150</u>
$BlkTime.AVG$	0.338	0.3828	0.4151	0.4035	0.5730	0.4866	<b>1.1803</b>	<b>1.1792</b>	<b>1.0968</b>	<b>1.0285</b>
$BlkTime.MAX$	0.325	0.4040	0.5727	0.5260	0.6412	0.5290	<b>1.5029</b>	<b>1.5668</b>	<b>1.3033</b>	<b>1.4904</b>
$BlkTime.MIN$	0.35	0.3715	0.3337	0.3362	0.4634	0.4492	0.6994	0.4832	0.7277	0.7244

TABLE 4  
Experiment 2: how batch of  $TAR$  affects  $T_{sigma}$

$TAR$	11	21	31	41	51	61	71	81	91	<u>141</u>
$BlkTime.AVG$	2.2673	1.2507	1.0408	0.9152	0.7564	0.7073	0.7243	<b>0.9656</b>	<b>0.7010</b>	<b>nll</b>
$BlkTime.MAX$	2.4900	1.2920	1.3040	1.1757	0.8044	0.8395	0.9021	<b>1.3219</b>	<b>0.8963</b>	<b>nll</b>
$BlkTime.MIN$	2.0030	1.1800	0.8203	0.7715	0.7222	0.6288	0.5626	0.5490	0.5187	<b>nll</b>