B.Comp. Dissertation

Analytical Performance Comparison of Byzantine Consensus Protocols

By
Chan Jun Da

Department of Computer Science

School of Computing

National University of Singapore

2023

B.Comp. Dissertation

Analytical Performance Comparison of Byzantine Consensus Protocols

By
Chan Jun Da

Department of Computer Science

School of Computing

National University of Singapore

2023

Project No: H091970
Supervisor: Dr. Tay Yong Chiang & Dr. Zhang Louxin

Deliverables:
Report: 1 Volume
Source Code: 1 Zip File

Abstract

The decentralized nature of distributed computing systems makes it inherently difficult for nodes to come to an agreement over a specific task, action or value. In real-world applications, this problem is made more challenging by asynchronous communication and often the need to handle Byzantine faults in order to establish a trustless system. Byzantine Fault Tolerant (BFT) protocols are built to handle such situations and tolerate up to a fraction of faulty nodes in the system to provide some guarantee on the nodes being able to reach a consensus. Recently, interest in such BFT protocols has increased due to the popularity of blockchains and blockchain-related technologies that rely on BFT protocols as the basis for trustless services such as transactions and storage of data. In our work, we focused on a class of BFT protocols that work in partially-synchronous environments, involving two popular BFT protocols, the Istanbul Byzantine Fault Tolerant (IBFT) protocol and HotStuff. We simulate the performance of these protocols on various network topologies and under various conditions. After which, we propose a general methodology for developing analytical models for analyzing these protocols and applied them. We further verified the models via newer simulations and came to some general conclusions about these protocols.

Subject Descriptors:

• *Networks∼Network performance evaluation∼Network performance modeling* • **Computing methodologies∼Modeling and simulation∼Model development and analysis∼Modeling methodologies** • Computing methodologies∼Modeling and simulation∼Model development and analysis∼Model verification and validation • **Computer systems organization∼Dependable and fault-tolerant systems and networks∼Reliability**

Keywords:

Consensus protocol, distributed systems, Byzantine faults, network topology, analytical modeling

Implementation Software:

Windows 10, JDK 11.0.11, JVM 11.0.11, Gradle 7.5.1, Python 3.7

Acknowledgements

I would like to express my sincere gratitude to Prof. Tay, who has provided me with unending support to enable the completion of this project. His extensive knowledge in a broad array of topics relating to the project has been invaluable in guiding my work. The constant support throughout the year has made for an incredibly enriching experience.

I would also like to express thanks to Prof. Zhang who has made himself readily available to guide me when I needed help, without which, this project might not have come to fruition.

List of Figures

2.1 IBFT Normal Operations . . . 6

2.2 IBFT Round Change Operations . . . 7

2.3 HotStuff General Operations . . . 9

2.4 Architecture diagram of the main components of the simulator program. . . 10

3.1 Comparison of simulation results and model for no round change on clique for a) HotStuff and b) IBFT. . . . 14

3.2 HotStuff simulation on clique with faults. a) $n = 16$, b) $n = 32$. . . . 16

3.3 Comparison of HotStuff model with simulation results with $0, 2$ faults. a) $n = 16$, b) $n = 32$. Vertical lines give recommended *btl*. . . . 20

3.4 IBFT simulation on clique with faults. a) $n = 16$, b) $n = 32$. . . . 20

3.5 Comparison of IBFT model with simulation results. a) $0, 2$ faults for $n = 16$ b) 2 faults for $n = 32$. Vertical lines give recommended *btl*. . . . 25

4.1 Examples of various topology configurations used. Switches are labeled with (level/group, index). a) Folded Clos ($p = 8, q = 4$) b) Folded Clos ($p = 9, q = 3$) c) Dragonfly ($a = 3$) d) Dragonfly ($a = 4$). . . . 29

4.2 Various simulation experiments on a network topology. a), b) compares both protocols with changing λ on different topologies. c), d) compares both protocols with changing n on different topologies. e), f) compares topologies with changing n with different protocols. . . . 31

4.3 HotStuff model comparison. a) and b) compare both models with simulation results at $n = 31$, folded Clos $(8, 4)$. c) and d) compare both models for $\lambda = 9, 30$. . . . 35

4.4 IBFT folded Clos model comparison. a) compares both models with varying λ. b) compares improved model with varying n. . . . 37

4.5 IBFT dragonfly model comparison. a) compares both models with varying λ. b) compares improved model with varying n. 38
4.6 Comparing simulated crossing points with model crossing points for both folded Clos and dragonfly topology. 40
4.7 a) and c) compare the model for both topologies with simulation results for varying s and fixed k. Plots b) and d) are obtained from interpolating the results in a) and c) and taking their ratio. 42
4.8 Comparison of simulation results with $c = 1, 2, 3$ consensus instances running to model with adjusted rate parameters. 43
4.9 Simulation results of HotStuff with faults on dragonfly topology ($a = 4$). . . 44
4.10 Comparison of HotStuff model and simulation on dragonfly (a=4) with $n_f = 0, 4$. 47

B.1 Various simulation experiments with varying n on a network topology for different fixed values of λ. xv
B.2 Comparison of IBFT folded Clos model for scenario 3 with varying n for different fixed values of λ. xvi
B.3 Comparison of IBFT folded Clos model for scenario 3 with varying n for different fixed values of λ. xvii

Contents

Abstract **i**

Acknowledgements **ii**

List of Figures **iv**

1 Introduction **1**

1.1 Overview 1

1.2 Objectives 1

1.3 Literature Review 2

1.3.1 Performance Analysis 2

1.3.2 Predecessor's Work 3

1.3.3 Investigating Network Topology Effects on Consensus Behavior 3

2 Background Information **4**

2.1 Byzantine Protocols 4

2.1.1 Notation and Terminology 4

2.1.2 IBFT 6

2.1.3 HotStuff 8

2.2 Simulator Overview 10

2.2.1 Architecture 10

2.2.2 Node Modeling 11

2.2.3 Routing 12

3 BFT Protocols on a Clique **13**

3.1 Scenario 1: No round change 13

3.1.1 HotStuff Model 13

3.1.2 IBFT Model 14
3.1.3 Comparing IBFT and HotStuff Models 15
3.2 Scenario 2: Varying base time limits with faults 15
3.2.1 Model Methodology 15
3.2.2 HotStuff Clique Model 17
3.2.3 Analysis of HotStuff Clique Model 19
3.2.4 IBFT Clique Model 21
3.2.5 Analysis of IBFT Clique Model 24

4 BFT Protocols on a Network 26
4.1 Motivation 26
4.2 Network Topologies 26
4.2.1 Notations and Terminologies 26
4.2.2 Folded Clos Topology 27
4.2.3 Dragonfly Topology 28
4.2.4 Choice of Topologies 30
4.3 Scenario 3: Topology with no round change 30
4.3.1 Model Methodology 32
4.3.2 HotStuff Topology Model 32
4.3.3 IBFT Topology Model 36
4.4 Analysis of No Round Change Models 39
4.4.1 Finding the crossing point 39
4.4.2 Comparing Topologies (for IBFT) 41
4.4.3 Comment about switch processing rates 43
4.5 Scenario 4: Topology with round change 44
4.5.1 Model Methodology 44
4.5.2 HotStuff Model Details 45

5 Summary, Discussion and Future Work 48
5.1 Summary of Results 48
5.1.1 Scenario 1: Clique without round change 48
5.1.2 Scenario 2: Clique with round change 48
5.1.3 Scenario 3: Topology with no round change 49
5.2 Discussion 50
5.2.1 Comparing Consensus Protocols 50

5.2.2 Considerations for Topology 51
5.3 Future Work 52
5.3.1 Extending the Topology Fault Model 52
5.3.2 Investigating other protocols 52
5.3.3 Including other forms of delay 52

References **viii**

Appendices **x**

A Proofs and Derivations **xi**
A.1 Derivation of Clique IBFT model without Round Change xi
A.2 Conditions for comparing IBFT and HotStuff topology formula xii

B Additional Figures **xiv**
B.1 Scenario 3: Topology with no round change xiv
B.2 Comparison of folded Clos model xiv
B.3 Comparison of dragonfly model xiv

Chapter 1

Introduction

1.1 Overview

Distributed computation is common among large scale computing networks. Due to the decentralized nature of distributed systems, there was a need to handle faulty or compromised nodes that can behave in any arbitrary manner, also known as Byzantine nodes. Byzantine agreement protocols were developed, which are algorithms meant to handle Byzantine nodes in the network, such that the working nodes eventually agree on some value. These protocols vary in the requirement (synchronous vs partially synchronous vs asynchronous) and guarantees they provide (maximum number of faulty nodes, probabilistic guarantee, etc.).

The rise of Web3.0 has created renewed interest in these Byzantine agreement protocols. Web3.0 technology and services are built onto the guarantees of blockchain technology that themselves rely on these Byzantine agreement protocols to function correctly. The goal of this project is to create and validate analytical models for various Byzantine agreement protocols on various network topologies with the help of simulations.

1.2 Objectives

Many Byzantine agreement protocols or Byzantine Fault Tolerant (BFT) protocols have been analyzed from the perspective of complexity (number of rounds and messages) and micro-benchmarks. Thanks to the work of the predecessor to this project [3], they have developed analytical models for the performance of two BFT protocols, Istanbul Byzantine Fault Tolerant (IBFT) [6] and HotStuff [13] over a clique network under a no round-change scenario.

The objective of this project is to extend on the work done by

1. creating a more accurate analytical model for the round-change scenario on a clique,
2. include the considerations of more complex topologies and its impact on the protocols,
3. create a generalized methodology to analyze such scenarios.

The protocols are first simulated to understand the trends of the protocol performance given certain input parameters to develop an intuition of the factors that might affect the protocol performance. After which, an analytical model is developed, validated against further simulation and used to analyze the behavior of the protocol and the topologies used.

1.3 Literature Review

1.3.1 Performance Analysis

There has been a body of work attempting to analyze the performance of various BFT protocols via metrics like message complexity and microbenchmarks. A protocol that relies on each node in the system broadcasting messages to all nodes are often referred to as all-to-all BFTs. Such BFTs like IBFT generally have a $O(n^2)$ message complexity where n is the number of nodes involved in the protocol. More recent protocols such as the HotStuff protocol managed to achieve an $O(n)$ message complexity [4]. HotStuff requires 3 sets of round-trip communication (known as phases) between the leader and other nodes to complete a consensus instance. Research has been done to reduce the number of phases while retaining the linear message complexity and recently, Marlin [10] achieved a two-phase BFT, which is theoretically optimal in the number of phases.

Besides message complexity analysis, there has also been various microbenchmarking works aimed at quantifying the performance of these BFT protocols. One such work [5] attempted to develop a methodology known as HyPerf to evaluate the performance of the PBFT protocol (a predecessor of the IBFT protocol). This was checked against microbenchmarks of the PBFT protocol to some degree of success. Other works such as [2] compared some of the older BFT protocols like PBFT, HQ and Zyzzyva via microbenchmarks under various adverse scenarios.

1.3.2 Predecessor's Work

Yen's work in [3] investigates the performance of IBFT and HotStuff on a clique topology by building a simulator and subsequently developing an analytical model which fits closely with the simulated results for one of the scenarios they investigated. In particular, they created an analytical model for the time spent in each phase for both HotStuff and IBFT protocols under no round-change scenarios. While work was done for the round-change scenario, the model does not provide a good fit for the simulated results.

1.3.3 Investigating Network Topology Effects on Consensus Behavior

Of particular interest in the continuation of this project is to investigate how underlying network topologies might affect the effectiveness of a consensus protocol. On this, there seems to be comparatively less work done. The work that is done differs greatly on the conditions that are investigated.

[12] measured the proof-of-work, proof-of-stake and delegated proof-of-stake protocols (which are more general blockchain-specific protocols) over random, small-world and scale-free topologies. [8] modeled the performance of Practical Byzantine Fault Tolerance (PBFT) protocol, an earlier version of IBFT, on a 6G network. The analysis goes into great detail on various sources of delays at an individual node and various unique aspects of the 6G network. However, they also use a constant delay term to account for the impact of the network has on the protocol. [9] models the performance of HotStuff assuming the transmission delay on the network is much greater than the message processing time and thus considers the message processing time to be neglible. Our work focuses more on scenarios where the network delay is of a similar magnitude to the message processing delay and the focus is on how protocol behavior and topology characteristics can affect performance of the protocol.

Chapter 2

Background Information

2.1 Byzantine Protocols

Of particular interest in this report are the partially-synchronous class of Byzantine Fault Tolerant protocols. In particular, we investigate two protocols.

- IBFT - The Istanbul Byzantine Fault Tolerant (IBFT) protocol, which has quadratic message complexity and is a blockchain-adapted variant of the Practical Byzantine Fault Tolerance (PBFT) protocol, the first polynomial message complexity BFT protocol.
- HotStuff - The first linear message complexity BFT protocol in this domain.

2.1.1 Notation and Terminology

We first include some general notation for both protocols.

- $n =$ number of validators
- $f =$ maximum tolerable number of Byzantine faults. Generally, $f = \left\lfloor \frac{n-1}{3} \right\rfloor$
- $\mu =$ processing rate of nodes

As different protocols term the same objects in their protocol differently, we define a standardized terminology that will be used throughout this report.

- *System* - A set of n processes that are running the consensus protocol we are interested in.

- *Validator* - A process in the system that takes part in the consensus protocol being run on the network. Other commonly used terms are *replica* and *node*.
- *Block* - Collection of data (e.g. client information, requests, etc.). The blockchain consists of a series of blocks arranged in the order in which they were added. A leader will generally propose a new block to be added to the blockchain.
- *Consensus instance* - The xth consensus instance refers to the state in which the $(x-1)$th block has been added to the blockchain and the replicas are attempting to add the xth block to the blockchain via use of a consensus protocol.
- *Round* - A single iteration of the consensus protocol for a consensus instance. There is generally a fixed leader for each round and smooth operations of the protocol results in validators reaching a consensus in 1 round. The protocols generally define a round-change operation in the event that consensus is not reached by some time limit. IBFT refers to this as a *round* while HotStuff refers to this as a *view*.
- *Round change* - A validator experiencing a timeout and undergoing the steps required by the protocol during such an event.
- *Full round change* - All validators undergo a round change for a given round. This differs from a round change as it is possible for multiple validators to undergo round change while the remainder reach a consensus.
- *Leader* - In a round, 1 validator is usually designated a special role of the leader that drives the consensus instance in that round. The leader generally changes after every round and the leader for a certain round is known to all validators beforehand.
- *Base time limit* - The initial time limit a protocol sets to complete a round or part of a round. Most protocols suggest an exponential back-off for the timer, doubling each time consensus fails to be reached. Thus, a base time limit has to be defined which will often show up in equations as btl.
- *Quorum* - Generally refers to the quantity $n - f$ and is generally used as a threshold to progress the protocol.

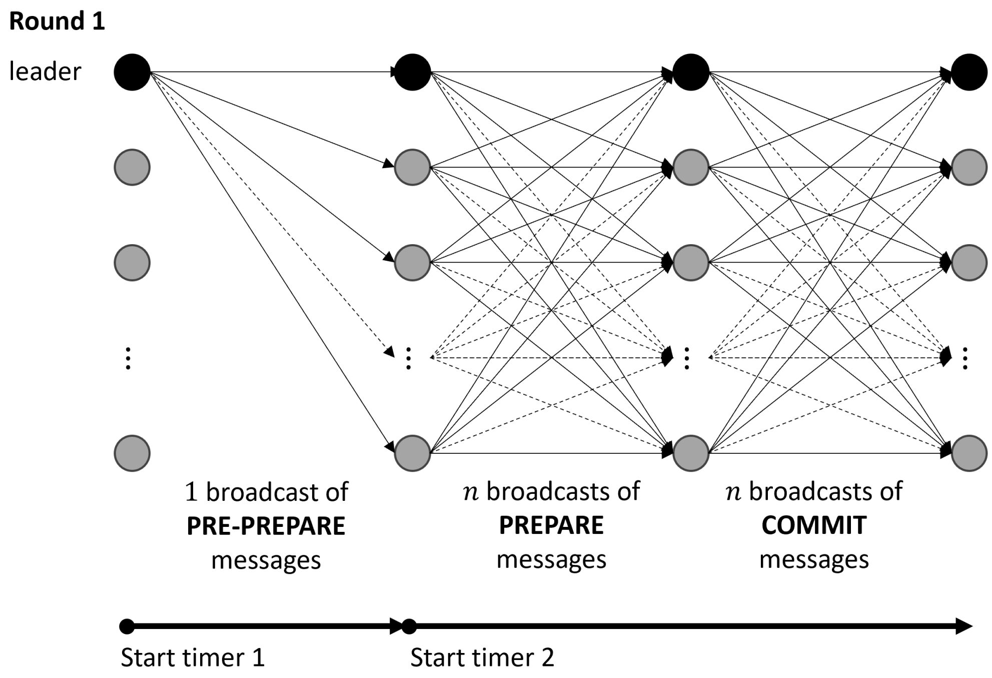

Round 1
leader
1 broadcast of PRE-PREPARE messages
n broadcasts of PREPARE messages
n broadcasts of COMMIT messages
Start timer 1
Start timer 2

Figure 2.1: IBFT Normal Operations

2.1.2 IBFT

We will only give an overview of the protocol described in [6], mainly relating to what messages are passed around. Justification and procedure details that validators would normally be required to perform will be left out. A normal IBFT consensus instance where no timer times out and thus no round change occurs can be seen in Fig 2.1.

IBFT - Normal Operations

In normal operations, the IBFT protocol is driven by a series of **upon** steps which generally occurs sequentially but is not required.

1. Timer starts. The leader broadcasts messages to all other validators including itself with a message of type *PRE-PREPARE*. This contains the leader's proposed value to be agreed upon.

2. Upon receiving a *PRE-PREPARE* message and verifying its sender, the timer is reset. A validator will then broadcast *PREPARE* messages with the value proposed by the leader.

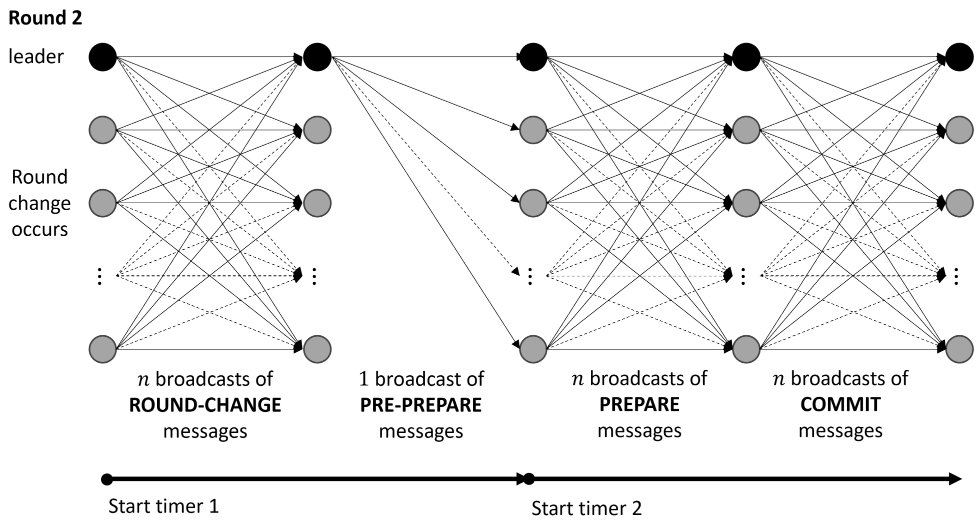

Round 2
leader
Round change occurs
n broadcasts of ROUND-CHANGE messages
1 broadcast of PRE-PREPARE messages
n broadcasts of PREPARE messages
n broadcasts of COMMIT messages
Start timer 1
Start timer 2

Figure 2.2: IBFT Round Change Operations

3. Upon receiving a quorum of *PREPARE* messages, a validator will broadcast *COMMIT* messages.

4. Upon receiving a quorum of *COMMIT* messages, a validator will stop its timer and commit a value based on the quorum of *COMMIT* messages it received. This will complete the current consensus instance.

IBFT - Round Change

A round change procedure is shown in Fig 2.2.

1. Upon a time out, the validator will increase round count and start a new timer, usually with an increased time limit. The validator will then broadcast *ROUND-CHANGE* messages.

2. For the new leader of the next round, upon receiving a quorum of valid *ROUND-CHANGE* messages, will broadcast *PRE-PREPARE* messages.

3. The consensus protocol then continues from Step 2 of the previous list.

Two less significant parts of the protocol are not illustrated to simplify the explanation which are the following.

- If fewer than f validators time out and the rest manage to come to a consensus, a catch-up mechanism is required. The protocol itself does not dictate a specific type of catch-up mechanism but suggests that a way to do it could be for validators to perform the following. Suppose validator v_1 receives a *ROUND-CHANGE* message from v_2 from a previous consensus instance, it can send a message containing the quorum of *COMMIT* messages for that consensus instance to v_2. Upon receiving this message and the piggybacked messages, v_2 will immediately complete that consensus instance.

- If a validator v receives more than $f + 1$ *ROUND-CHANGE* messages from rounds strictly greater than the round it is currently in, it will immediately jump to the minimum of those rounds and broadcast a new set of *ROUND-CHANGE* messages for that round.

We will informally refer to the various message steps as phases, with IBFT going through *PRE-PREPARE*, *PREPARE* and *COMMIT* phases. Note that the protocol itself does not require the idea of phases.

2.1.3 HotStuff

Again, we will mainly cover the messages sent in the protocol in [9] without going over the operations that validators are normally required to perform for the messages they receive. In contrast to IBFT, there are distinct phases defined for HotStuff which designate exactly what type of messages it is waiting for.

Additionally, HotStuff does not have the concept of rounds and consensus instances. It only keeps track of one value which are *views*. Regardless of whether a timeout occurs or consensus is reached, the *view* count is increased by 1. Hotstuff also distinguishes by name certain types of messages in the protocol as *votes*. This is mostly in name without any notable differences in behavior.

For our explanation of the HotStuff protocol, we will start at an arbitrary round r as the first *view* behaves somewhat differently from the rest. An illustrated HotStuff *view* can be found in Fig 2.3.

1. **PREPARE** phase. Timer starts.
    (a) Leader waits for a quorum of *NEW-VIEW* messages. After which the leader prepares a proposal (value to be committed) and broadcasts *PREPARE* messages.
    (b) Validator waits for a *PREPARE* message from the leader. Sends its *PREPARE* vote to the leader.

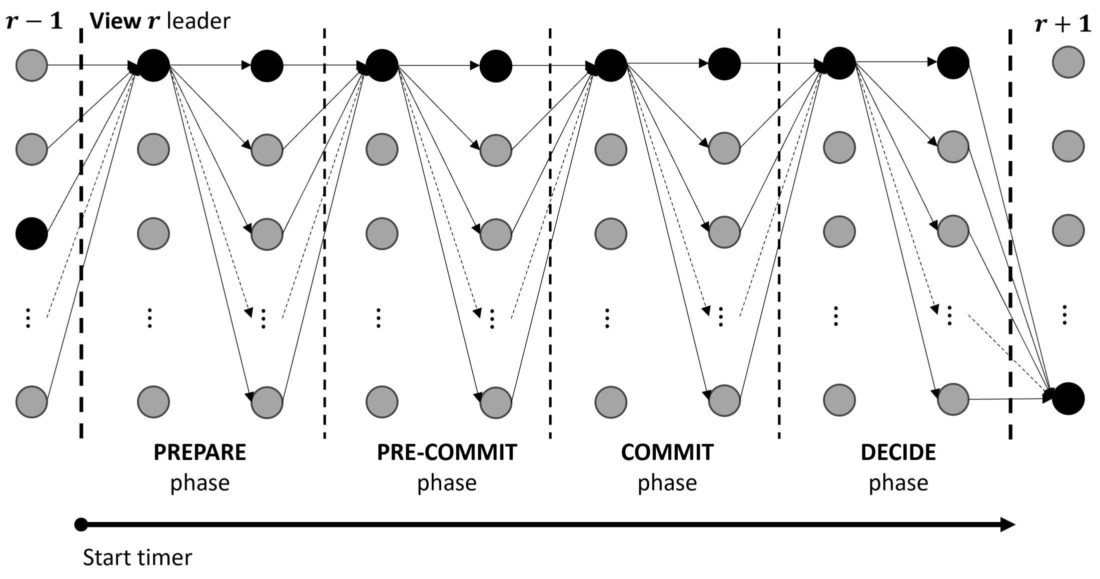


Figure 2.3: HotStuff General Operations

2. **PRE-COMMIT** phase.

   (a) Leader waits for a quorum of *PREPARE* votes. The leader then broadcasts *PRE-COMMIT* messages.

   (b) Validator waits for a *PRE-COMMIT* message from the leader. Sends its *PRE-COMMIT* vote to the leader.

3. **COMMIT** phase.

   (a) Leader waits for a quorum of *PRE-COMMIT* votes. The leader then broadcasts *COMMIT* messages.

   (b) Validator waits for a *COMMIT* message from the leader. Sends its *COMMIT* vote to the leader.

4. **DECIDE** phase.

   (a) Leader waits for a quorum of *COMMIT* votes. The leader then broadcasts *DECIDE* messages.

   (b) Validator waits for a *DECIDE* message from the leader and accepts the proposal contained in the *DECIDE* message.

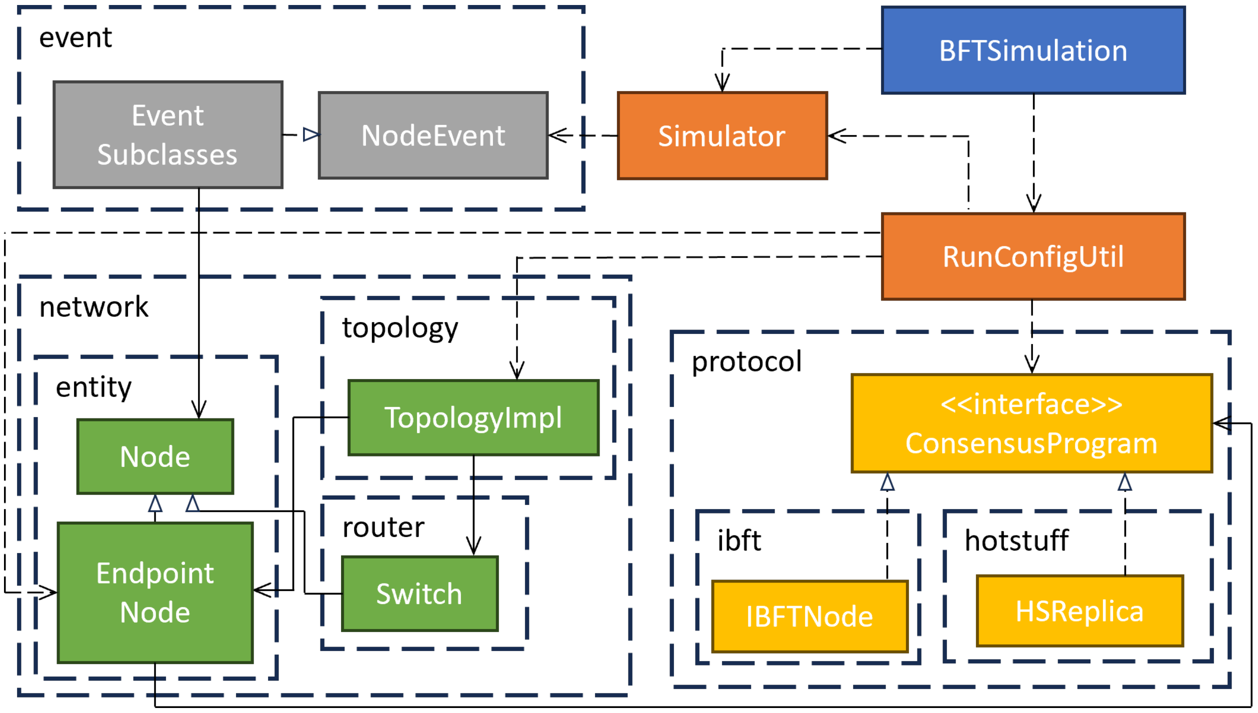


Figure 2.4: Architecture diagram of the main components of the simulator program.

5. Validator sends a *NEW-VIEW* message to the leader of the next view. Increase view count and restarts back at step 1 for the next view.

When a validator experiences timeout, it immediately skips to step 5.

Most of the details of the protocol are in handling and processing exactly what types of proposals can be accepted at each phase which are left out in the explanation above. Note that for future references, a view will be referred to as a round, and a view change due to timeout will be referred to as a round change.

2.2 Simulator Overview

2.2.1 Architecture

Fig 2.4 provides a high-level architecture diagram of the simulation program.

Overview

1. *BFTSimulation* class:

   Main entry class for the program.

2. *network* package:
   Defines various network entities that are used such as terminal nodes, switches, etc. It also provides implementation of various common topologies that connect nodes and handle routing logic of packets being transferred through the network.

3. *protocol* package:
   Contains the two main consensus protocol implementations IBFT and HotStuff as *programs* and necessary helper classes. These *programs* are then run on *EndpointNodes*.

4. *event* package:
   Contains implementation of various types of events that occur in a simulation of the protocol. Such events include queuing, processing and timeout events.

5. *RunConfigUtil* class:
   Takes in a run configuration in the form of a JSON file and sets up the necessary protocol, topology and various other configurations for a simulation run.

6. *Simulator* class:
   Driving class for the simulation. It runs similar to a discrete event simulator.

7. *statistics* package (not shown):
   Handles the tracking and calculation of various values of interest such as the average message queue time, message arrival rate, etc.

8. *json* package (not shown):
   Specifies and handles the format of input files that setup the run configuration and output files for results.

Details of how to run the program can be found in the README in the program files.

The simulator can be easily extended to run different consensus protocols by implementing the appropriate *ConsensusProgram* interface. Adding different topologies can also be done by defining the connection between the switches programmatically.

2.2.2 Node Modeling

Network nodes are treated as separate from the programs being run on the node itself. To simplify the model, of the four type of computing network delays, we have chosen transmission and propagation delay to be negligible. The message arrival distribution is unknown since it is determined by the behavior of the protocol used. The service rate is chosen to be exponentially distributed with 1 server.

For simplicity and for comparison with Yen's work, terminal nodes have processing rate $\mu = 3s^{-1}$. We are however free to vary the processing rate of switches, λ, which are usually greater than the processing rate of the terminal nodes.

2.2.3 Routing

Currently, for simplicity, we have chosen the Randomized, Oblivious Multi-phase Minimal (ROMM) routing where at each hop [1], we randomly select one node which still leads to a minimal hop count path.

Chapter 3

BFT Protocols on a Clique

The simplest topology to study these protocols would be a clique topology where every terminal node has direct connection to every other terminal node with no switches necessary. While clique topologies are rarely used today, it serves a simple theoretical starting point to observe how each protocol behaves. Further, if network delays are significantly smaller than the time a terminal node takes to process a protocol message, then the impact of the network topology is minimized and can be approximated with the behavior of running the protocol on a clique.

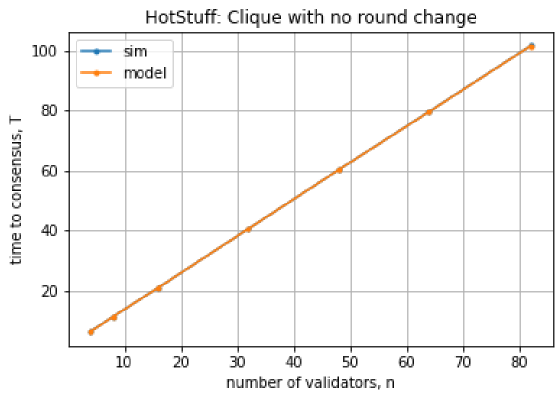
3.1 Scenario 1: No round change

This serves as a study of the performance of each protocol given that no round change occurs. This can be simulated by setting the base timer for both protocols to be arbitrarily large such that they will never time out. It further assumes that no validators are faulty.

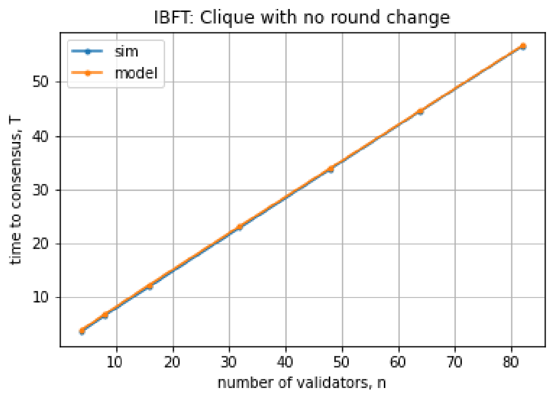
The simulation was done with terminal node processing rate $\mu = 3s^{-1}$ and number of validators, n, varying with $n = 4, 8, 16, 32, 48, 64, 82$. To avoid round change, the base time limit was set to $t = 10000s$. The results can be found in Figure 3.1 together with a comparison to the proposed models for each protocol.

3.1.1 HotStuff Model

We propose a model for the average time per consensus in this scenario as a function of the number of validators, n, and node processing rate, μ. The main bottleneck in the protocol is the number of messages the leader has to process in a single round. We can tally up these



a)



b)

Figure 3.1: Comparison of simulation results and model for no round change on clique for a) HotStuff and b) IBFT.

messages to get

$$T(n, \mu) = \frac{4n - f + 4}{\mu}. \tag{3.1}$$

Consider only the messages the leader has to process. $3 \cdot (n+1)$ messages from the n votes in the *PREPARE*, *PRE-COMMIT* and *COMMIT* phases and an additional one message the leader has to process in its role as a validator. For the last *DECIDE* phase, the leader only needs to process the first $n - f$ votes before broadcasting the *DECIDE* message that will complete that consensus instance, after which there is one last message every validator has to process. The need to process only $n - f$ votes does not occur for the previous phases as the leader has to process all n votes before beginning to process the votes from the next phase. This is the HotStuff model proposed by Yen in [3].

3.1.2 IBFT Model

For IBFT, the model is not quite as simple and we will quote the result done by Yen directly in [3]. A derivation of the model can be found in Appendix A.1. The model involves both message counting and some queuing theory, and we will be reusing this function later.

$$T(n, \mu) = \frac{(10n + 2)(2n + 1)}{\mu(7 + 11n - \sqrt{37 + 70n + n^2})}. \tag{3.2}$$

3.1.3 Comparing IBFT and HotStuff Models

From Figure 3.1, we see that IBFT takes almost half the time to reach consensus compared to HotStuff. This is reasonable to expect due to the bottleneck of IBFT only requiring to process $2n+1$ messages compared to the $4n-f+4$ messages of the HotStuff leader. In fact, we can obtain a simple asymptotic ratio of HotStuff performance to IBFT performance.

$$n \to \infty \Rightarrow \frac{T_h(n,\mu)}{T_i(n,\mu)} = \frac{4n-f+4}{\frac{(10n+2)(2n+1)}{(7+11n-\sqrt{37+70n+n^2})}} \to \frac{4n-n/3}{\frac{20n^2}{10n}} = \frac{11}{6} \approx 1.83.$$

3.2 Scenario 2: Varying base time limits with faults

Next, we investigate the behavior of both protocols under a varying base time limit, *btl*. Faults are also introduced to this simulation. Due to difficulty of categorizing and analyzing Byzantine faults, for simplicity, we only introduce unresponsive faults where a validator will not respond to any messages from other validators. We simulated $n = 16, 32$ over a wide range of btls. For both $n = 16, 32$, we simulated with number of faults $n_f = 0, 1, 2, 3$. Node processing rate is again fixed at $\mu = 3s^{-1}$.

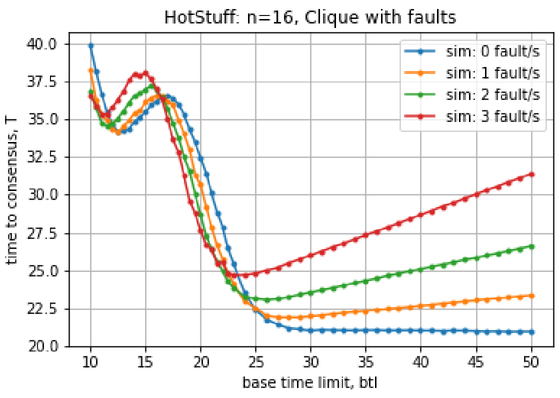
The goal of this scenario is to concretely show that given a fixed number of validators and number of faulty nodes $n_f > 0$, there exists an optimal *btl* to achieve the fastest time to consensus on average. We would also like to obtain a simple closed-form formula for an optimal or close-to-optimal *btl* to achieve the minimum time to consensus.

The results for HotStuff simulation can be found in Figure 3.2. The results for IBFT can be found in Figure 3.4.

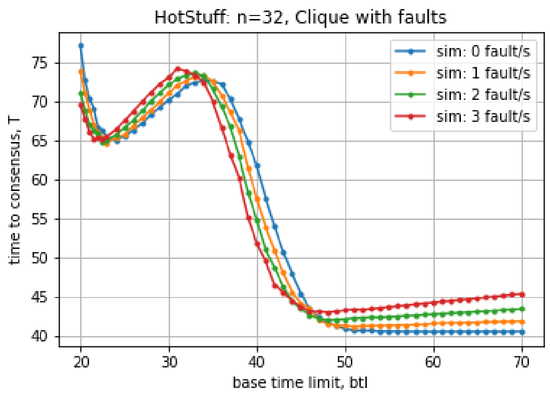
3.2.1 Model Methodology

As these full round change scenarios are extremely complex and can vary greatly with how the protocol handles timeouts, we propose a generalized scheme to develop analytical models for round change scenarios in these protocols.

The key is to consider how the protocol behaves as the base time limit *btl* decreases. Initially, if we set *btl* to be some arbitrarily large value, the performance of the protocol would be equivalent to that in scenario 1. However, as we decrease the *btl*, the probability of 1 full round change occurring begins to increase. Our goal is to find an event with the largest probability of occurring given a full round change occurs, at the region where the time to consensus begins to increase with decreasing *btl*. Let the event be E. It could be



a)



b)

Figure 3.2: HotStuff simulation on clique with faults. a) $n = 16$, b) $n = 32$.

that there exists multiple distinct events with similar probabilities but such considerations should be evaluated on a case-by-case basis. Then we assume that if a timeout occurs, it is due to event E. We can consider for a given *btl* the performance of the protocol first given E occurs and then given it does not.

To further simplify the problem, we only consider the cases of 0 or 1 full round change occurring. That is, the interval of *btl* values we are concerned with is such that if a timeout occurs at *btl*, then the next round the timer will be set to 2*btl* where given a working leader, the probability of another full round change occurring is 0. This greatly simplifies the problem while still allowing us to identify the minima point with varying *btl* but will fail to model the performance of the protocol at sufficiently low *btl*. However, intentionally setting extremely low *btl* values are unlikely to be encountered in practical applications as it sabotages the performance of the protocol with no benefit.

Thus, the approach goes like this.

1. Identify event E with probability $1 - p$. We define it this way such that p is the probability no full round change occurs.

2. Model the time to consensus given 1 round change occurs as a result of E.

3. Model the time to consensus given 0 round change occurs as E does not happen.

4. Model the distribution function of p by studying E.

3.2.2 HotStuff Clique Model

Next, we propose a model for the HotStuff time to consensus under varying btl. We begin with the following notations:

- $n =$ total number of validators in the networks (both faulty and working).
- $n_f =$ number of faulty validators.
- $n_w =$ number of working validators. $n_w = n - f$.
- $f =$ maximum tolerable number of Byzantine failures.
- $T =$ time for a consensus instance.

1. Identifying E

For HotStuff, the event is relatively easy to identify which is that the leader fails to process the required number of messages in *btl*.d Recall that HotStuff consists of 4 phases, each with the leader broadcasting a message and each working validator processing it and replying with 1 vote to the leader. For the last phase, only $n - f$ votes are required before the *DECIDE* messages are broadcasted. After which, the consensus instance is completed when the *DECIDE* message is processed. Define m to be the number of messages the leader needs to process, then

$$m = n_w + 1 + n_w + 1 + n_w + 1 + n - f + 1 = 3n_w + n - f + 4 = 4n - f - 3n_f + 4.$$

If $n_f = 0$ then the above reduces to the number of messages the leader needs to process in scenario 1. Event E occurs when the leader is unable to process $m - 1$ messages in *btl*. We consider $m - 1$ instead of m as consensus is generally reached the moment the leader broadcasts its *DECIDE* messages even if it does not finish processing its own. Other validators will reach consensus and the leader will eventually catch up.

2. and 3. Modeling time to consensus given p

We define $t_{e2e} = \frac{m}{\mu}$ to be the expected time to consensus given no round change. We can narrow down the outcome of the first round to the following three events. Note that we use a random leader selection protocol for each round.

- E_1 : Faulty validator as first leader.

- E_2 : Working validator as first leader but timed out. This is caused by event E.
- E_3 : Working validator as first leader and did not time out.

We can calculate the probabilities of each event outcome in a round. Define $r := \frac{n_f}{n}$, the probability that a randomly selected validator is faulty.

$$P(E_1) = r, \qquad P(E_2) = (1-r)\cdot(1-p), \qquad P(E_3) = (1-r)\cdot p.$$

Finally, we have the random variable T to be the time taken to reach consensus for a given consensus instance and calculate the conditional expectation of T given one of E_1, E_2, E_3 occurred.

$$E[T|E_1] = t_{e2e} + \sum_{i=1}^{\infty} r^{i-1}(btl \cdot 2^{i-1}) = t_{e2e} + btl \cdot \frac{1}{1-2r}.$$

The above equation assumes an exponential back off where the timer doubles each round. For each round, with probability r, the leader chosen is faulty, except the first round where by the conditions of E_1, the leader is faulty. The number of rounds has the form $1+X$ where $X \sim Geom(r)$, with the ith round that consensus is not reached lasting $2^{i-1} \cdot btl$ seconds. For the final round where consensus is reached, the round itself takes t_{e2e}.

For E_2, the scenario is identical to E_1 with the only difference being the failure of the first round is due to the leader timing out rather than having chosen a faulty leader first. Thus, $E[T|E_1] = E[T|E_2]$.

For E_3, the time taken is chosen to be t_{e2e}. More accurately, it should be $\min\{btl, t_{e2e}\}$ but taking it to just be t_{e2e} simplifies the expression and gives a reasonable approximation.

Thus our model for the average time to consensus T is given by

$$\begin{aligned} E[T] &= E[T|E_1]P(E_1) + E[T|E_2]P(E_2) + E[T|E_3]P(E_3) \\ &= \left(t_{e2e} + \frac{btl}{1-2r}\right)\cdot r + \left(t_{e2e} + \frac{btl}{1-2r}\right)\cdot(1-r)\cdot(1-p) + (1-r)\cdot p\cdot t_{e2e} \qquad (3.3) \\ &= t_{e2e} + \frac{btl}{1-2r}\cdot\left(r + (1-r)\cdot(1-p)\right). \end{aligned}$$

4. Modeling p

We can model this by first assuming that the time it takes to process the ith message is a random variable X_i with expectation x, variance σ^2. The time it takes to process $m-1$ messages is the sum of $m-1$ independent random variables X_i each with mean x and variance

σ^2. We can approximate this with the Central Limit Theorem with the time taken, T_m to process the $m-1$ messages to follow an approximate $\mathcal{N}((m-1)x, (m-1)\sigma^2)$ distribution with $p = P(T_m < btl)$. Comparison of this model to simulation results can be found in Figure 3.3.

3.2.3 Analysis of HotStuff Clique Model

We can consider the following three cases of values of p. Recall that p is the probability that the leader validator processes $m-1$ consecutive messages without timing out.

- $p = 1$: For sufficiently large btl, we should expect $p \approx 1$ which simplifies the equation to $T = t_{e2e} + \frac{r}{1-2r} \cdot btl$ which is linear in btl. It suggests that in the scenario of expecting to have at least one fault, we cannot simply set the btl to be arbitrarily large as we pay a penalty on the average time to consensus depending on the number of faults. Note the dependency of the gradient on n_f.

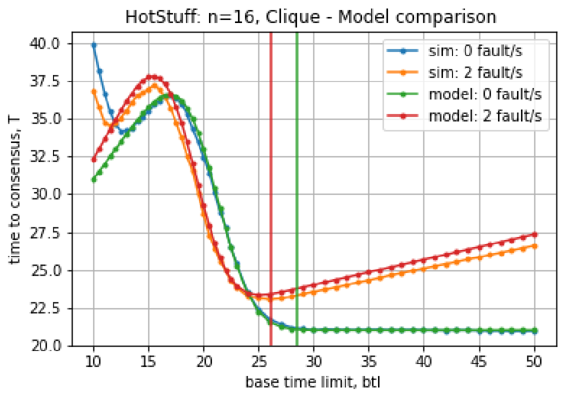
  Another interesting point is that $r < \frac{1}{3}$ as the maximum tolerable number of faulty validators is upper bounded by the nature of the protocol. If we take $r = \frac{1}{3}$, we have $\frac{r}{1-2r} = 1$ which gives us an upper bound on the gradient and thus penalty for choosing a btl that is larger than what is optimal.

- $p = 0$: The gradient becomes $\frac{r+(1-r)}{1-2r} = \frac{1}{1-2r}$. While this does simplify the equation to another linear equation with a different gradient, the region where this equation is applicable is quite small. As btl gets smaller, the probability of two or more round changes occurring increases which is not currently accounted for in our simplified model. Nevertheless, it does suggest the possibility that below the optimal btl, $E[T]$ is not strictly monotonic and we do in fact observe it in Fig. 3.2.

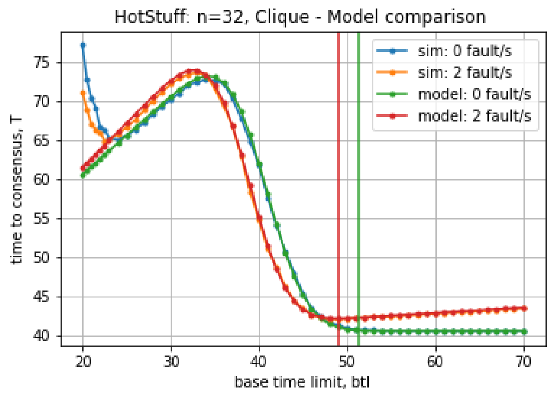
- $0 < p < 1$: This is the segment of the curve where T increases with decreasing btl due to p decreasing, causing more frequent round changes.

This gives us a full description of how the average time to consensus for HotStuff on a clique with faults behaves given at most 1 full round change occurs. Then given the expected time x and variance σ^2 to process a single message (via profiling or other means), the expected number of faults, and using our model, we can give a recommendation of an almost-optimal btl to set to be

$$btl_{rec} = (m-1)x + 3\sqrt{m-1} \cdot \sigma. \tag{3.4}$$

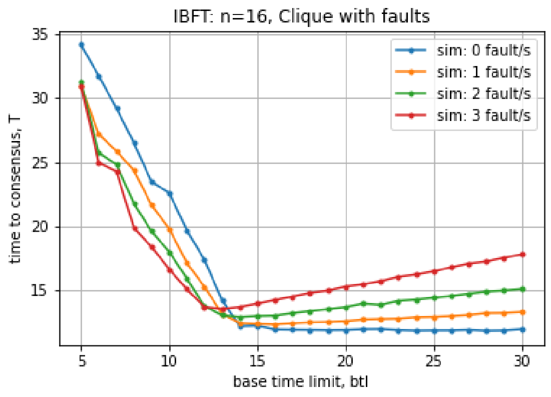


a)

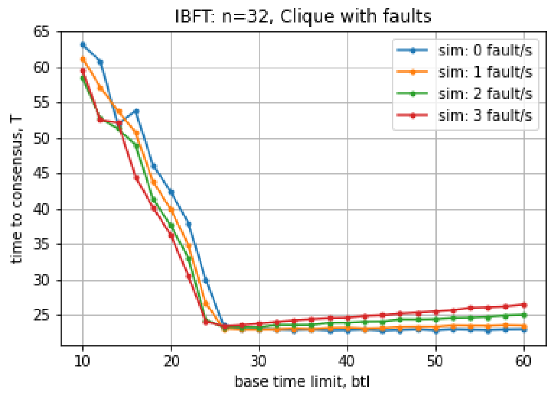


b)

Figure 3.3: Comparison of HotStuff model with simulation results with $0, 2$ faults. a) $n = 16$, b) $n = 32$. Vertical lines give recommended btl.



a)



b)

Figure 3.4: IBFT simulation on clique with faults. a) $n = 16$, b) $n = 32$.

3.2.4 IBFT Clique Model

With a similar approach to HotStuff, we can attempt to build a model for the performance of IBFT on a clique. However, as the message complexity of the protocol is quadratic, the behavior of the protocol is significantly more complex.

2. and 3. Modeling time to consensus given p

We will skip over step 1 initially as the event is not easy to identify, and move to approximating T given that we know p. Note that for IBFT, instead of a random leader being selected each round, at the start of a consensus instance, a random permutation of the validators is predetermined where the leader rotates following this permutation.

We define $t_{e2e} = \frac{(10n_w+2)(2n_w+1)}{\mu(7+11n_w-\sqrt{37+70n_w+(n_w)^2})}$ to be the time taken to reach consensus on the first round given no round change occurs. This is of a similar structure to our fault-less IBFT formula except n_w replacing n. Again, assuming at most one timeout, we can consider three similar events at the start of each consensus instance. Define ${}_nP_r(n,r) = \frac{n!}{(n-r)!}$ to be the permutation function.

- E_1 : Faulty validator as first leader.
- E_2 : Working validator as first leader but timed out.
- E_3 : Working validator as first leader and consensus is reached without round change.

$$E[T|E_1] = btl\left[\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f-1,i-1)}{{}_nP_r(n-1,i-1)}2^{i-1}\right] + \frac{2n_w+n-f+1}{\mu}.$$

To find E_1, the summation is similar to the analysis for HotStuff except with the slightly altered leader selection protocol. The probability that i validators in a row are faulty is $\frac{{}_nP_r(n_f-1,i-1)}{{}_nP_r(n-1,i-1)}$ instead of just $r^{i-1} = (\frac{n_f}{n})^{i-1}$ since the permutation guarantees that a previously selected faulty validator will not be selected again until all validators have become leader at least once. After a valid leader is chosen, n_w *ROUND-CHANGE*, 1 *PRE-PREPARE*, n_w *PREPARE* and $n-f$ *COMMIT* messages have to be processed to reach consensus. We do not use t_{e2e} as the queue for a validator is likely to be empty when a round is reached with a non-faulty leader. As such, we simply tally up the number of messags that are required to be processed, giving $\frac{2n_w+n-f+1}{\mu}$.

E_2 is significantly more complicated.

$$\begin{aligned} E[T|E_2] &= btl \sum_{i=2}^{n_f} \frac{P(n_f, i-1)}{P(n-1, i-1)} \cdot 2^{i-1} + \left(btl + \frac{f - n_f + 1}{\mu} \right) + \frac{n_w}{\mu} \cdot (1-r) + \frac{2n_w + n - f + 1}{\mu} \\ &= btl \sum_{i=1}^{n_f} \frac{{}_nP_r(n_f, i-1)}{{}_nP_r(n-1, i-1)} \cdot 2^{i-1} + \frac{1}{\mu} \left[n_w(4-r) + 2 \right]. \end{aligned}$$

The first summation (starting at index 2) is similar to E_1 where it is the penalty that a faulty leader is chosen in round $i \geq 2$. Since the first round has a non-faulty leader, there are still n_f faulty leaders left to be chosen starting from round 2.

$btl + \frac{f-n_f+1}{\mu}$: For the leader, $f - n_f$ messages come from the excess messages from the previous consensus instance and 1 *PRE-PREPARE* message. Note that IBFT has two timers, one for only the *PRE-PREPARE* phase and another to complete the *PREPARE* and *COMMIT* phases. Thus, we have to include this additional penalty on top of one *btl* in the first round.

$\frac{n_w}{\mu} \cdot (1-r)$: This is to cover the scenario where there are excess messages in the first round that slows down the processing in the second round given that the leader is not faulty. n_w is an approximation of the number of messages from round 1 in round 2. In reality, this number varies quite a bit depending on *btl*.

$\frac{2n_w+n-f+1}{\mu}$: This is similar to E_1 and measures the time taken to reach consensus on the next round with a non-faulty leader.

$$E[T|E_3] = t_{e2e}.$$

Lastly, for E_3, the time taken is relatively straightforward, similar to HotStuff. Define $r = \frac{n_f}{n}$. Putting it together, we have the time to consensus T to be

$$\begin{aligned} E[T] &= E[T|E_1]P(E_1) + E[T|E_2]P(E_2) + E[T|E_3]P(E_3) \\ &= E[T|E_1] \cdot r + E[T|E_2] \cdot (1-r)(1-p) + E[T|E_3] \cdot (1-r)p. \end{aligned} \tag{3.5}$$

1. Identifying E

Unlike HotStuff, we cannot simply consider the number of messages a validator has to process before a timeout. The communication in the protocol is all-to-all and a validator decides the moment it receives a quorum of *COMMIT* messages. Thus, for a full round change to occur, we require that more than f validators undergo round change before they broadcast their *COMMIT* messages. We can attempt to model this as a binomial random variable X giving the number of validators that underwent round change. The binomial distribution has parameters n, q where q is the probability that a validator experiences round change due to timeout. Then the probability of a full round change occurring is given by $P(X > f)$. Suppose we follow HotStuff and take q to be the probability that the time taken to process $2n + 1$ messages is greater than btl.

Unfortunately, in simulations, even when $P(X > f) \approx 0$, a significant fraction of consensus instances still undergo full round change, occurring at a rate much greater than would be expected if the above model or something similar were true. Thus, we propose an alternate hypothesis where a full round change occurs as a result of a slow leader, and further propose that the main event that causes a slow leader is that it underwent round change in a previous round.

To model this, consider the case where in round i, a full round change occurs. Assume in round $i-2$, the n_w validators were initially perfectly synchronized at the start of their *PREPARE* phase. A validator will need to process $m := n_w + n - f$ messages to complete their respective *PREPARE* and *COMMIT* phases. Let $S_{i,k}$ be the time taken to process m messages by validator k in round i. Let $X_k = S_{i-2,k} - \overline{S_{i-2}}$ where $\overline{S_{i-2}}$ is the mean time taken for all validators to process their respective m messages in round $i-2$. Then X_k is a measure of how much faster or slower validator k is relative to all other validators.

Let x, σ^2 denote the mean and variance time taken to process one message by any validator. Then we can find X_k via

$$X_k = S_{i-2,k} - \frac{1}{n_w}\sum_{j=1}^{n_w} S_{i-2,j} \sim \mathcal{N}\left(0, m\left(1 + \frac{1}{n_w}\right)\sigma^2\right) \text{ approximately.}$$

When entering into round $i-1$, faster validators (with negative X_k values) will enter the *PREPARE* phase $-X_k$ time earlier waiting for the *PREPARE* and *COMMIT* messages of other, slower validators. Then for the leader of round i, call it validator c, to have underwent round change in round $i-1$, we require that in trying to process the m messages in round $i-1$, the time taken, $-X_c + S_{i-1,c} > btl$. We can approximate the distribution of $-X_c + S_{i-1,c} > btl$ to be

$$-X_c + S_{i-1,c} \sim \mathcal{N}\left(mx, m\left(2 + \frac{1}{n_w}\right)\sigma^2\right) \text{ approximately.}$$

Thus, our event E that causes a full round change is the event that $-X_c + S_{i-1,c} > btl$.

4. Modeling p

It follows from above that $p \approx P(-X_c + S_{i-1,c} < btl)$. The model gives a reasonable fit to the graph for both $n = 16$ and $n = 32$ as can be seen in Fig. 3.5.

3.2.5 Analysis of IBFT Clique Model

For the case of $p = 1$, the equation 3.5 reduces to

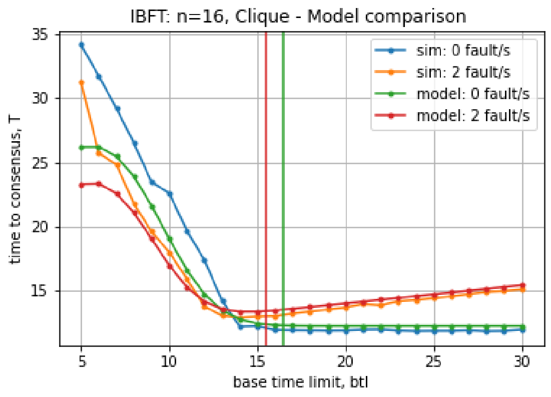
$$\begin{aligned}
E[T] &= E[T|E_1] \cdot r + E[T|E_3] \cdot (1-r) \\
&= r\left(btl\left[\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f-1, i-1)}{{}_nP_r(n-1, i-1)} 2^{i-1}\right] + \frac{2n_w + n - f + 1}{\mu}\right) + (1-r) \cdot t_{e2e} \\
&= r\left[\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f-1, i-1)}{{}_nP_r(n-1, i-1)} 2^{i-1}\right] \cdot btl + \left[r\left(\frac{2n_w + n - f + 1}{\mu} - t_{e2e}\right) + t_{e2e}\right].
\end{aligned}$$

For fixed n, n_f, the function above is linear in btl with gradient $r\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f-1,i-1)}{{}_nP_r(n-1,i-1)} 2^{i-1}$. For large n_f, we can approximate it by taking only the first three terms of the summation as later terms quickly approach 0.

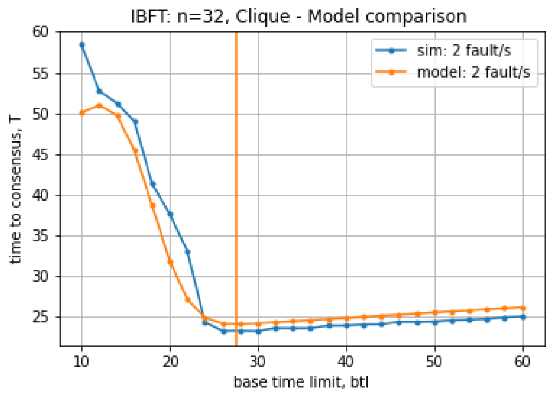
For $p = 0$, the simplified equation does not have much use as referencing Figure 3.4, we see that below the optimal btl, there does not appear to be any significant range of values where T decreases with decreasing btl.

We can again propose a recommended btl to set given the message processing rate profile x, σ^2 and expected number of faults.

$$btl_{rec} = mx + 3\sqrt{m \cdot \left(2 + \frac{1}{n_w}\right)} \cdot \sigma. \tag{3.6}$$



a)



b)

Figure 3.5: Comparison of IBFT model with simulation results. a) $0, 2$ faults for $n = 16$ b) 2 faults for $n = 32$. Vertical lines give recommended btl.

Chapter 4

BFT Protocols on a Network

We would like to observe the impact of running these protocols on more complex network topologies. Our work will mainly be using the folded Clos and dragonfly topology for analysis.

4.1 Motivation

Primarily, the main motivation is to demonstrate that the choice of topology impacts performance of a given protocol being run on the network differently. In particular, we observed in Chapter 3 that IBFT significantly outperforms HotStuff despite having a quadratic message complexity compared to the linear message complexity of HotStuff. A likely reason for this is that on a clique topology where every node is directly connected to every other node, there is no contention of network resources for messages to be delivered. In contrast, here we will see scenarios where HotStuff outperforms IBFT significantly due to its linear message complexity.

4.2 Network Topologies

We first introduce and properly define the network topologies we will be using. Note that for simplicity, the topologies are generally simplified, with fewer parameters that can be specified compared to what is possible, to reduce the complexity of the analysis.

4.2.1 Notations and Terminologies

We define some additional terms.

- *node* - Refers specifically to an endpoint/terminal node in a network that is capable of processing and sending messages. Note that this differs from a *validator* which is a process that the node runs. This distinction is only relevant for Sec 4.4.3 where simultaneous consensus protocols are run with the same set of nodes. Otherwise, one node runs one validator process.
- *edge switch* - Switch with terminal nodes directly connected to it.
- *leader switch* - The edge switch with the leader of the current round connected to it.
- *direct network* - A network topology consisting of only edge switches. All switches in the network are connected to some terminal node and may also help with routing messages between other switches.
- *indirect network* - A network topology consisting of both edge switches and non-edge switches. Non-edge switches facilitate message sending between edge-switches.

We also include some general notation for both topologies.

- $n =$ number of validators
- $f =$ maximum tolerable number of Byzantine faults. Generally, $f = \left\lfloor \frac{n-1}{3} \right\rfloor$
- $k =$ average number of terminal nodes per edge switch
- $\mu =$ processing rate of terminal nodes
- $\lambda =$ processing rate of switches.
- $s =$ number of switches

4.2.2 Folded Clos Topology

Clos networks are multistage indirect networks where messages are sent from terminal nodes through multiple stages of switches before arriving at its destination. The common Clos network described is a three stage network consisting of an ingress, middle and egress stage where messages from terminal nodes can take a unique path to any of the middle stage switches, before taking a unique path to its destination node.

A folded Clos network combines the egress and ingress stages into the same set of switches and making every connection bidirectional. This converts a three stage Clos network into a two level folded Clos network. This idea can be extended recursively to define folded Clos networks with more levels. For our work, we mostly use three level folded Clos networks with the following parameters.

- p: The number of switches in each level. For our implementation, we fix all levels to have the same number of switches though this is not necessarily required.
- q: The number of uplinks each switch will have from the first level to the second level. We require that q divides p. From the second level to the third, the number of uplinks depend on the value of p/q.
- Terminal node distribution over switches: Terminal nodes are connected only to first level switches in a way to minimize the maximum number of terminal nodes on a switch.

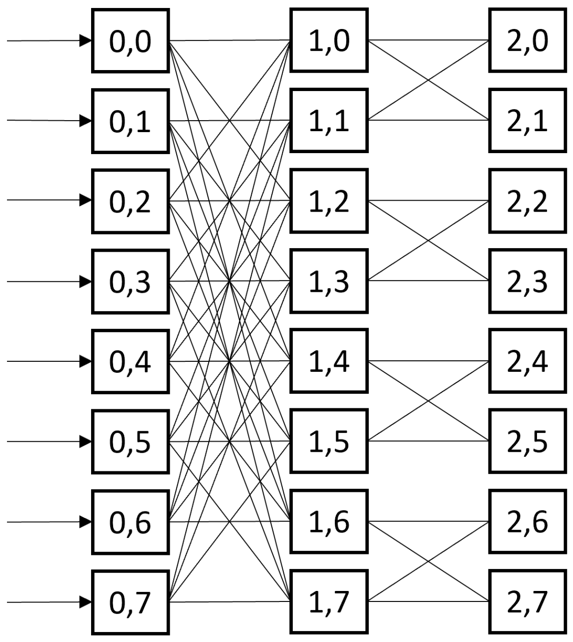
Examples can be found in Fig. 4.1 a) and b).

4.2.3 Dragonfly Topology

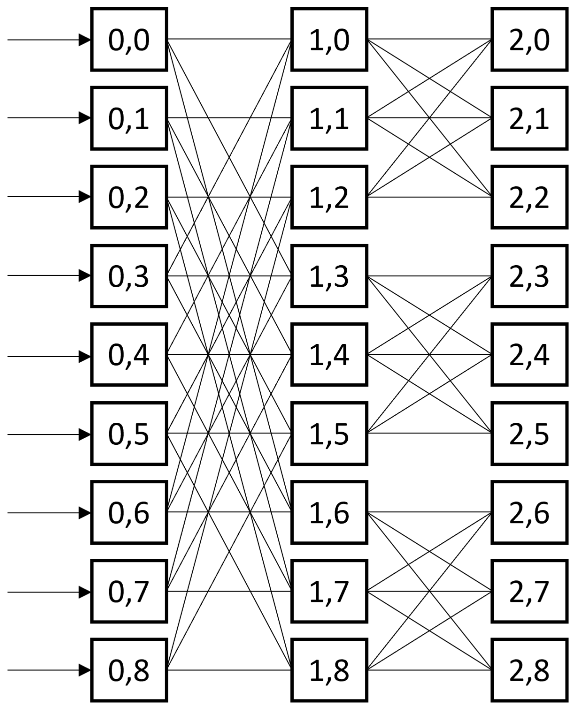
The dragonfly topology describes a general method which we can construct a larger network topology from smaller groups of switches, each potentially containing different topologies. It is considered a direct network as all switches are capable of being connected to one or more terminal nodes. For our work, we use a variant with the following parameters.

- a: Number of switches in a group. This is the only parameter that can be specified for our dragonfly implementation. A dragonfly topology is made up of multiple groups, each with a switches in them arranged in a predefined topology.
- Intra-group topology: Our implementation defines an intra-group topology to be a clique. In application, any intra-group topology can be used.
- g: Number of groups. We set $g = a + 1$ such that a group has all its nodes connected to exactly one node from a different group.
- Terminal node distribution over switches: As number of terminal nodes is likely to be larger than number of switches in the network, the terminal nodes are distributed by first minimizing the maximum number of nodes in each group, then minimizing the maximum number of nodes in each switch.

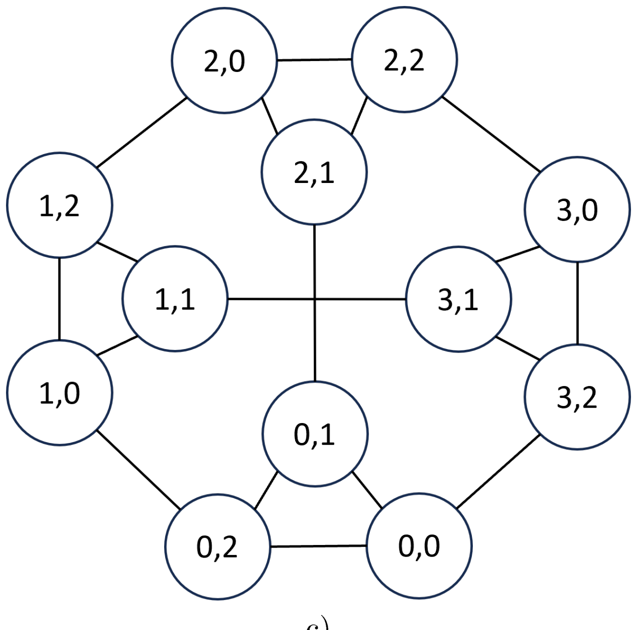
Examples can be found in Fig. 4.1 c) and d).



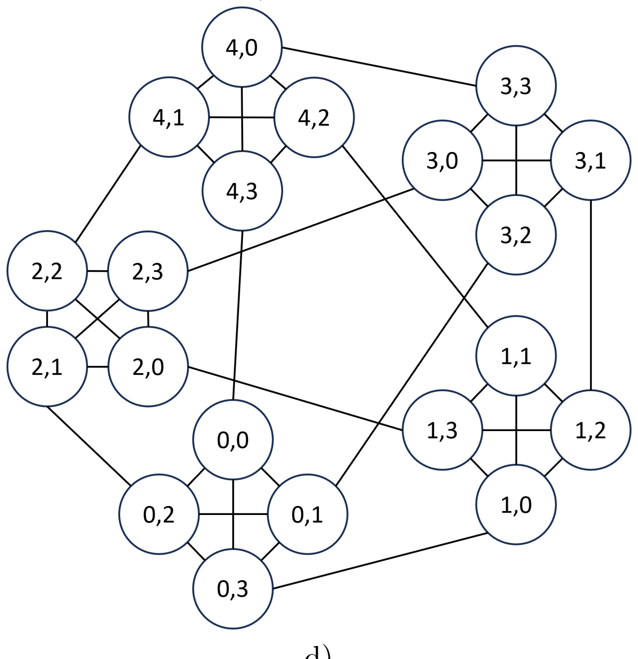
a)



b)



c)



d)

Figure 4.1: Examples of various topology configurations used. Switches are labeled with (level/group, index). a) Folded Clos ($p = 8, q = 4$) b) Folded Clos ($p = 9, q = 3$) c) Dragonfly ($a = 3$) d) Dragonfly ($a = 4$).

4.2.4 Choice of Topologies

The reason for the choice of these two topologies is that folded Clos is an indirect network while a dragonfly topology is a direct network. Edge switches on a folded Clos topology would only process and relay messages to and from the terminal nodes connected to it. In contrast, a dragonfly topology would have switches processing and relaying messages to and from its terminal nodes together with any traffic that is routed through the switch as an intermediate hop. We should expect a different impact in the performance of the consensus protocols on both topologies.

4.3 Scenario 3: Topology with no round change

There are many parameters that can be varied when attempting to measure the performance of a protocol on a topology. For this scenario, we attempt to measure the impact on time per consensus instance T by varying a) the number of validators n and b) the switch processing rate λ. We fix the topology together with the node processing rate $\mu = 3s^{-1}$. The *btl* is set arbitrarily large to avoid any round changes.

- Scenario 3a: Fixed n, varying λ.

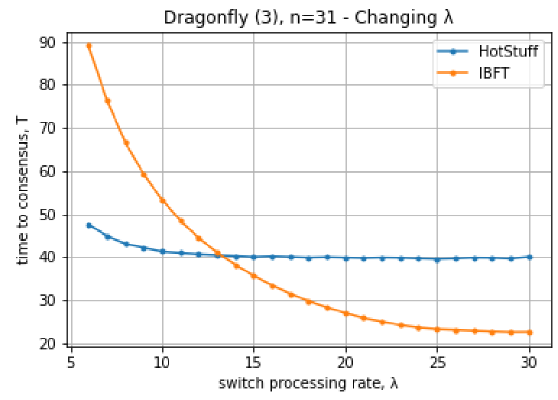
  For this scenario, we fix $n = 31$ and simulate both protocols on Dragonfly ($a = 3$) and Folded Clos ($p = 8, q = 4$). The results can be found in Fig. 4.2 a) and b).

- Scenario 3b: Fixed λ, varying n.

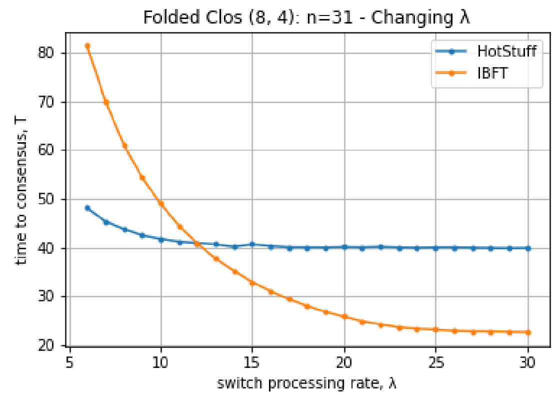
  Fix $\lambda = 9s^{-1}$ on a Dragonfly ($a = 5$) and Folded Clos ($p = 10, q = 5$) and vary the number of validators. The results can be found in Fig. 4.2 c) and d).

  This simulation is also repeated for $\lambda = 6, 15, 30$ such that we have covered $\frac{\lambda}{\mu} = 2, 3, 5, 10$ to observe how faster switches relative to node processing rates might affect the shape of the curve. Additional results can be found in Appendix Fig. B.1.

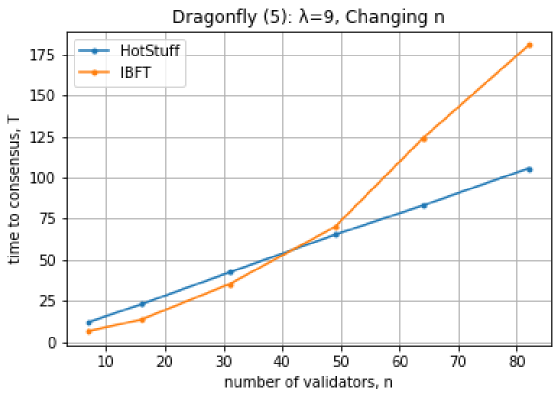
We can see the impact of the quadratic message complexity for IBFT here where settings which would increase congestion such as increasing n and decreasing λ both result in IBFT becoming slower than HotStuff which aligns with what we might expect due to their respective message complexities.



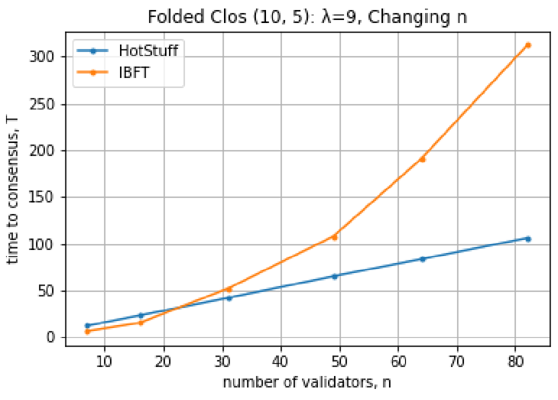
a)



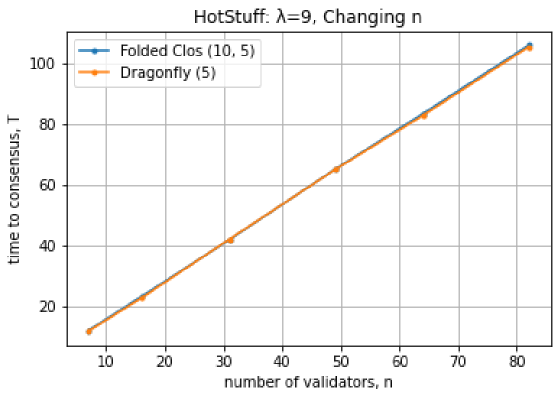
b)



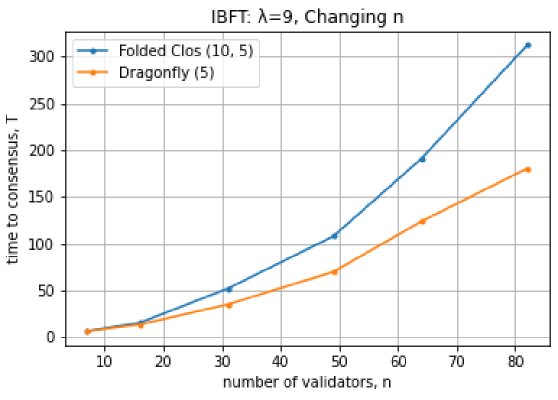
c)



d)



e)



f)

Figure 4.2: Various simulation experiments on a network topology. a), b) compares both protocols with changing λ on different topologies. c), d) compares both protocols with changing n on different topologies. e), f) compares topologies with changing n with different protocols.

4.3.1 Model Methodology

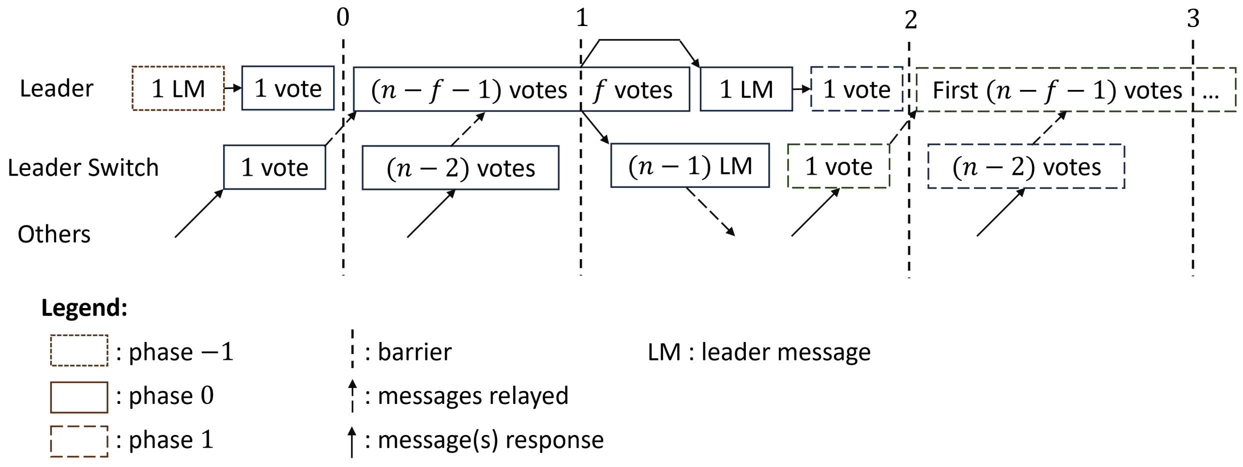
For this scenario, we look at the impact of the processing delay only at the most bottlenecked validator (usually the leader) and the most bottlenecked switch which is usually the leader switch. By comparing only these two bottlenecks, we can simplify the problem while still creating a reasonable model for the performance of both protocols on both topologies.

4.3.2 HotStuff Topology Model

Model Outline

From Fig. 4.2 e), we see that HotStuff seems largely unaffected by the topology, with almost identical performance on both the Folded Clos ($p = 10, q = 5$) and Dragonfly ($a = 5$). Our goal is to construct a model that explains and predict this behavior.

We can analyze the bottleneck for performance for HotStuff to see how it fares. Consider the figure below which attempts to illustrate how a *PREPARE* phase progresses.



For a specific round, we can take the *PREPARE* phase to be phase 0, *PRE-COMMIT* to be phase 1, etc. In this case, the *DECIDE* phase of the previous consensus instance can be considered to be phase -1.

Barrier 0 is characterized by the event that the leader is able to start processing more than one phase 0 vote. One vote is placed before barrier 0 as upon reading the leader message for the previous consensus instance, the leader would immediately send a vote to itself. Thus, the leader is guaranteed to have at least 1 of $n - f$ votes to process before the barrier.

Barrier 1 is characterized by the event that the leader switch starts relaying the leader's broadcast messages. For that to happen, it requires that 1) the leader has completed the processing of $n-f-1$ additional votes and performed a broadcast, and 2) the leader switch has its queue cleared (of $n-2$ votes) and is ready to relay messages from the leader. Only then can the leader switch begin relaying the $n-1$ broadcast messages. Note that only $n-1$ messages are sent out as 1 message is addressed to the leader itself.

Barrier 2 is similar to barrier 0 and is again characterized by the leader being able to process more than one phase 1 vote. It requires both that 1) the leader switch relayed out all $n-1$ outgoing messages from the leader and the first 1 incoming vote for phase 1, and 2) the leader has cleared all f remaining votes for phase 0 plus its own leader message and at least one vote. After which the leader can begin processing the $n-f-1$ votes required for phase 1 and the model repeats. In this case, barrier 3 is identical to barrier 1.

The above model repeats up till the *DECIDE* phase where only the portion between barrier 0 and 1 occurs, after which we only need to consider the expected time a validator takes to receive its *DECIDE* leader message to complete the consensus instance.

Model Details

A simplified model would look like

$$T(n,\mu,\lambda) = 4\max\left\{\frac{n-f-1}{\mu}, \frac{n-2}{\lambda}\right\} + 3\max\left\{\frac{f+2}{\mu}, \frac{n}{\lambda}\right\} + \frac{2h}{\lambda} + \frac{2}{\mu}. \tag{4.1}$$

The two max terms are explained by the model in the previous section. The term $\frac{2h}{\lambda}+\frac{2}{\mu}$ aims to measure the expected time between the leader switch beginning to relay the final *DECIDE* messages and the start of the next consensus instance. For the leader for the first round of the next consensus instance, even if it receives the *NEW-VIEW* messages from other validators before it received its own *DECIDE* message, it would still able to start processing those messages and adding them to a backlog. Let $h =$ average number of hops a message takes in a network from one terminal node to another. Then $\frac{h}{\lambda}+\frac{1}{\mu}$ is the average time taken for a message to travel the network and be processed at its destination. We double that for the expected time taken for the leader of the next consensus instance to receive its first *NEW-VIEW* message (regardless of if it has completed its current consensus instance) from the recipient of the first *DECIDE* message queued in the leader switch.

However, the simplified model significantly underestimates the time to consensus in the regions where $\frac{n-f-1}{\mu} \approx \frac{n-2}{\lambda}$ or $\frac{f+2}{\mu} \approx \frac{n}{\lambda}$. A more precise approximation of the time taken between each barrier would be required. Let X_1, X_2 be two normal random variables with means x_1, x_2 and standard deviations σ_1^2, σ_2^2. Let $X = \max\{X_1, X_2\}$ and $\theta(v_1, v_2) =$

$\sqrt{v_1 + v_2}$. We define the following function

$$G(u_1, v_1, u_2, v_2) = u_1 \Phi\left(\frac{u_1 - u_2}{\theta(v_1, v_2)}\right) + u_2 \Phi\left(\frac{u_2 - u_1}{\theta(v_1, v_2)}\right) + \theta(v_1, v_2)\phi\left(\frac{u_1 - u_2}{\theta(v_1, v_2)}\right), \tag{4.2}$$

where ϕ is the standard normal density function and Φ is the standard normal cumulative distribution function. Then $G(x_1, \sigma_1^2, x_2, \sigma_2^2) = E[X]$ [7], that is G is a function that gives the expectation of the maximum of two normal random variables.

Define $m_1 = n - f - 1, m_2 = n - 2, m_3 = f + 2, m_4 = n$. As our simulation uses an exponential distribution with parameter μ for validators and λ for switches, we have

$$T(n, \mu, \lambda) = 4 \cdot G\left(\frac{m_1}{\mu}, \frac{m_1}{\mu^2}, \frac{m_2}{\lambda}, \frac{m_2}{\lambda^2}\right) + 3 \cdot G\left(\frac{m_3}{\mu}, \frac{m_3}{\mu^2}, \frac{m_4}{\lambda}, \frac{m_4}{\lambda^2}\right) + \frac{2h}{\lambda} + \frac{2}{\mu}. \tag{4.3}$$

This technique is applicable to message processing times following other distributions. Suppose we know (by profiling or other means) the distribution of the time taken for a validator to process a message to be x_1, σ_1^2 mean and variance and the time taken for a switch to relay a message to be x_2, σ_2^2 mean and variance. We can then form a general model for HotStuff to be

$$\begin{aligned} T(n, x_1, \sigma_1^2, x_2, \sigma_2^2) = {} & 4 \cdot G(m_1 x_1, m_1 \sigma_1^2, m_2 x_2, m_2 \sigma_2^2) \\ & + 3 \cdot G(m_3 x_1, m_3 \sigma_1^2, m_4 x_2, m_4 \sigma_2^2) \\ & + 2h x_2 + 2x_1. \end{aligned} \tag{4.4}$$

The comparison of model fits can be found in Fig. 4.3. We see that as $\lambda \to \mu$ from the right, both models' begin to significantly underestimate T. However, within the region of interest where it is expected that $\lambda > \mu$, the improved model gives a good approximation for T.

For λ much larger than μ, both models approximate the simulated results reasonably well. But in c) and the left portion of a), we see that the simplified model begins to noticeably underestimate T compared to the simulated results.

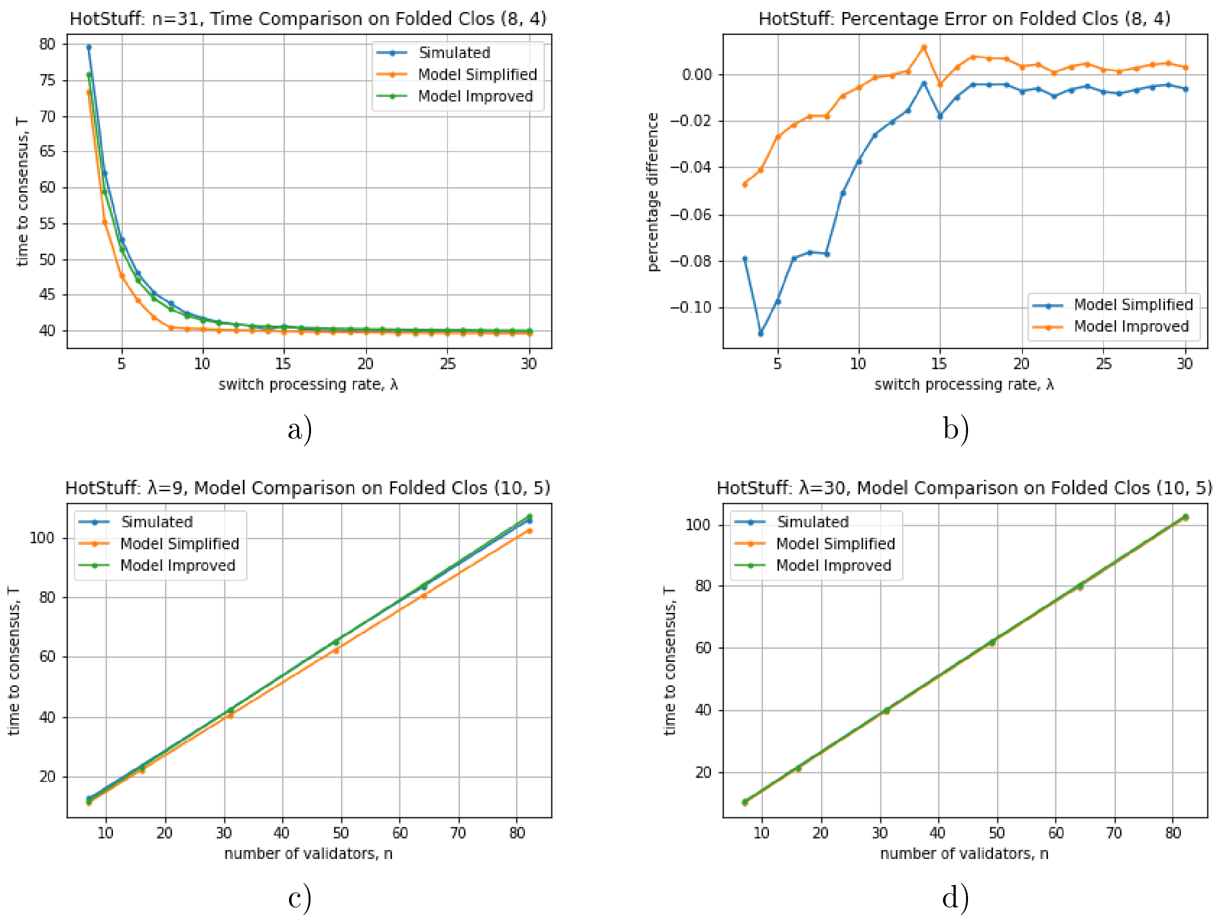


Figure 4.3: HotStuff model comparison. a) and b) compare both models with simulation results at $n = 31$, folded Clos $(8, 4)$. c) and d) compare both models for $\lambda = 9, 30$.

4.3.3 IBFT Topology Model

Generalized Model

IBFT's behavior on a network is far more complicated as a single consensus instance requires every validator to broadcast multiple messages. To simplify the problem, we consider the barriers in the progress of a consensus instance. Define m_1 to be the number of messages a validator has to process in a consensus instance. Next, define m_2 to be the number of messages processed by the most bottle-necked switch in the network. We can generally fix $m_1 = 2n + 1$.

Similar to what was done with HotStuff, we can then define a simplified model

$$T(n, \mu, \lambda) = \max\left\{\frac{m_1}{\mu}, \frac{m_2}{\lambda}\right\}. \tag{4.5}$$

And an improved model

$$T(n, \mu, \lambda) = G\left(\frac{m_1}{\mu}, \frac{m_1}{\mu^2}, \frac{m_2}{\lambda}, \frac{m_2}{\lambda^2}\right). \tag{4.6}$$

We can similarly generalize the findings here given any arbitrary message processing distribution for both validators and switches.

Next, we wish to find m_2 for each topology. To help with this, we define a function $k(n, s) = \frac{\lceil n/s \rceil + n/s}{2}$ (abbreviated as just k), where s is the number of edge switches. n/s is the average number of terminal nodes per switch. n/s is frequently a non-integer indicating that some edge switches contain more terminal nodes than others, causing a larger bottleneck at these switches. Empirically, the presence of edge switches with more validators disproportionately affects the performance of the protocol quite significantly at smaller values of n/s. Thus, we use k to be a measure of the average number of validators per edge switch that takes into account the impact of this phenomenon, which was found to have a better fit than using either n/s or $\lceil n/s \rceil$.

Folded Clos

As an indirect network, we consider the set of edges switches which only receive messages either addressed to its set of terminal nodes or were messages from its set of terminal nodes. It is straightforward to calculate the number of messages an edge switch has to handle. Consider a Folded Clos network with parameters p, q. p is the number of first-level (edge)

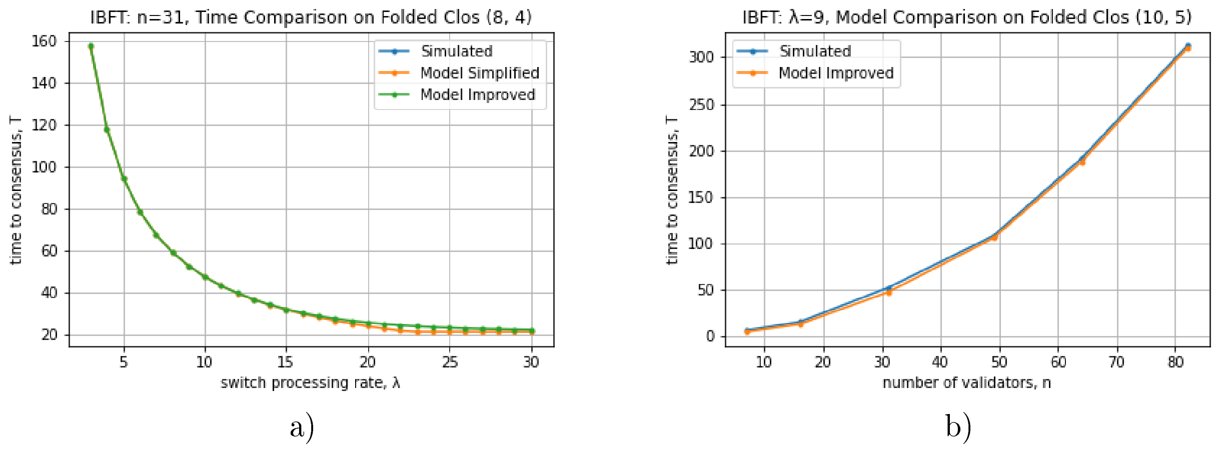


Figure 4.4: IBFT folded Clos model comparison. a) compares both models with varying λ. b) compares improved model with varying n.

switches. We have

$$m_2 = 4 \cdot k(n,p) \cdot (n-1). \tag{4.7}$$

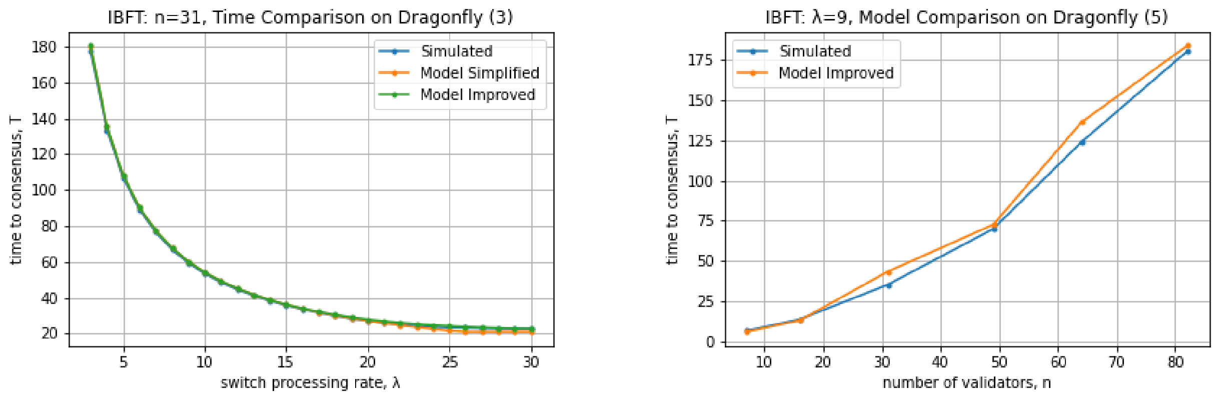
For the most bottle-necked switch, a switch has about $k(n,p)$ validators connected to it. Each of which sends $n-1$ messages and receives $n-1$ messages per broadcast instance (of which there are 2) in the protocol which gives $4 \cdot (n-1)$ messages. The broadcast of the $n-1$ *PRE-PREPARE* messages appear to have no effect on the performance empirically. It might be explained by this occurring while validators generally still having f extra *COMMIT* messages left to process and that the impact of this broadcast is localized to the leader switch.

Model comparisons can be found in Fig. 4.4. For varying λ, both models appear to perform similarly except for the range of $\lambda \in [20, 30]$ where the simplified model underestimates T. The improved model gives a decent fit for both scenarios.

Dragonfly

The dragonfly network is slightly more complex as every switch is an edge switch, serving as an intermediate hop for other switches to reach different parts of the network. This is in contrast to a folded Clos network where every message an edge switch processes is either directed to one of its nodes or was sent from it. However, the advantage that dragonfly has over a folded Clos is that because every switch is an edge switch, it has much smaller average number of validators per edge switch.

Applying a similar approach as with the folded Clos, take a dragonfly network with



a) b)

Figure 4.5: IBFT dragonfly model comparison. a) compares both models with varying λ. b) compares improved model with varying n.

parameter a. Denote number of switches $s = a(a+1)$. We have

$$m_2 = n - 1 + 4(k(n,s) \cdot (n-1) + k(n,s)^2 \cdot a(a-1)). \tag{4.8}$$

$4k(n,s) \cdot (n-1)$ comes from the total number of messages relayed by an edge switch similar to the analysis for the folded Clos topology. Empirically, it was found that adding $n-1$ messages to m_2 appears to give a better fit than if we were to ignore it like in IBFT. These could be the $n-1$ *PRE-PREPARE* at the leader switch where the impact of adding $n-1$ messages to a switch slows down the entire network as many messages might be using this switch to travel through the network.

The remainder $k(n,s)^2 \cdot a(a-1)$ comes from the relaying of messages not meant for validators directly connected to the switch. Consider a switch, c, connecting to its own group g_1 to another group g_2. Then every switch in g_1 that is not c, its broadcast will send 1 message to each of approximately ka validators in g_2. And there are approximately $k(a-1)$ such validators in g_1 which gives us the $k^2a(a-1)$ messages. Lastly, there are 2 such broadcasts and all messages are sent in both directions, giving the multiplier of 4.

Model comparisons can be found in Figure 4.5 for $\lambda = 9$. It fits the varying switch processing rate experiment in a) quite well. However, for the increasing n experiment in b), we see that there are points $n = 31, 64$ where the model overestimates the time to consensus. For comparisons of varying n at other fixed values of λ, go to Appendix Figure B.3.

4.4 Analysis of No Round Change Models

4.4.1 Finding the crossing point

Due to IBFT having $O(n^2)$ message complexity compared to $O(n)$ of HotStuff, we might expect that with decreasing switch processing rate, congestion in the network becomes significantly worse and HotStuff starts to outperform IBFT. As seen in Fig. 4.2 a) and b), we do observe this. A question might arise where we are interested in finding the crossing point λ where the change occurs.

For this analysis, we will be using the simplified model. Denote T_h to be the time to consensus for the HotStuff protocol for both topologies, $T_{i,f}$ for IBFT on a folded Clos network and $T_{i,d}$ for IBFT on a dragonfly network. Then recall that we have

$$\begin{aligned} T_h(n,\mu,\lambda) &= 4\max\left\{\frac{n-f-1}{\mu}, \frac{n-2}{\lambda}\right\} + 3\max\left\{\frac{f+2}{\mu}, \frac{n}{\lambda}\right\} + \frac{2h}{\lambda} + \frac{2}{\mu} \\ T_{i,f}(n,\mu,\lambda) &= \max\left\{\frac{2n+1}{\mu}, \frac{4\cdot k(n,p)\cdot(n-1)}{\lambda}\right\} \\ T_{i,d}(n,\mu,\lambda) &= \max\left\{\frac{2n+1}{\mu}, \frac{n-1+4(k(n,s)\cdot(n-1)+k(n,s)^2\cdot a(a-1))}{\lambda}\right\} \end{aligned}$$

Note further that in the region where this crossing occurs, for both IBFT formulas, it is the switch term that is dominant as $\frac{2n+1}{\mu}$ is a constant. For T_h, we can assume that the crossing generally occurs with relatively high λ such that we only consider the region where $\frac{f+2}{\mu} > \frac{n}{\lambda}$. A sufficient condition for this would be that if we have s to be the number of edge switches, then $\frac{n-1}{s} \geq 2$ (derivation in Appendix A.2). Let $m_f = 4\cdot k(n,p)\cdot(n-1), m_d = n-1+4(k(n,s)\cdot(n-1)+k(n,s)^2\cdot a(a-1))$. And refer to a general IBFT formula on a topology as T_i with numerator m. We can simplify the above equations to the following,

$$\begin{aligned} T_h(n,\mu,\lambda) &= \frac{4n-f+4}{\mu} + \frac{2h}{\lambda}, \\ T_i(n,\mu,\lambda) &= \frac{m}{\lambda}. \end{aligned}$$

And obtain the ratio of the switch processing rate to node processing rate which achieves the crossing point for both topologies.

$$T_h(n,\mu,\lambda) = T_i(n,\mu,\lambda) \Rightarrow \frac{\lambda}{\mu} = \frac{m-2h}{4n-f+4}. \tag{4.9}$$

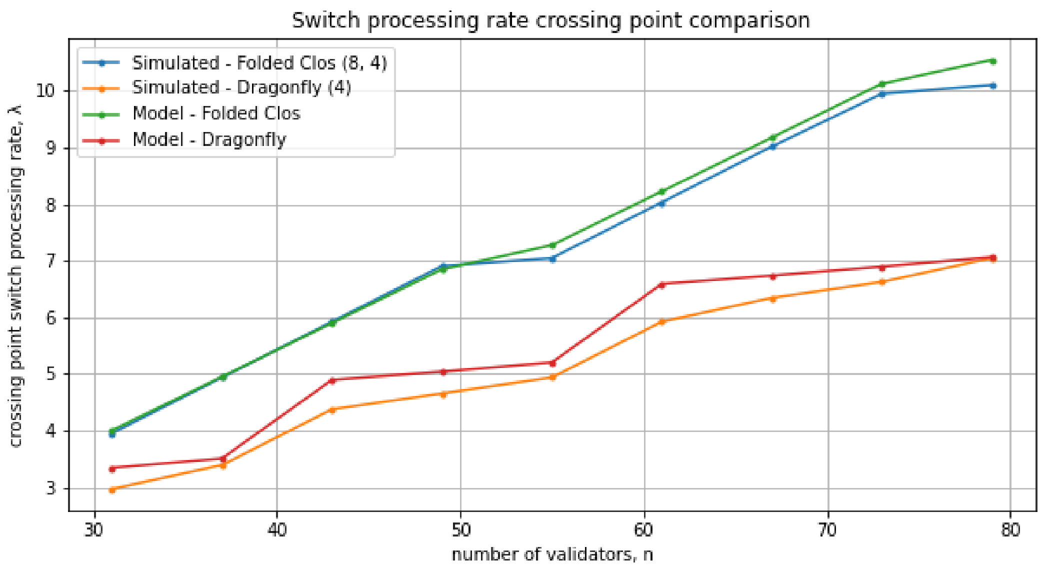


Figure 4.6: Comparing simulated crossing points with model crossing points for both folded Clos and dragonfly topology.

The h term in equation 4.9 is topology dependent as it is the average hop count a message takes to travel from one node to another. This term is generally much smaller than n and can usually be dropped without affecting the formula.

Thus, given a topology, we can analyze the switch bottlenecks in the topology for both IBFT and HotStuff to obtain estimates for h and m. After which, we can apply this formula to obtain the ratio of switch to node processing rates for which above that, IBFT performs better, and below that, HotStuff performs better. A comparison of the crossing point predicted by our formula (without h) can be found in Fig. 4.6. The model gives a reasonable approximation of the trends and actual values of the crossing points.

Of particular interest is the obvious step-like occurrences in the simulated results which gives reason to believe that a true formula for the performance involves the ceiling function in some manner. At the moment, the dragonfly overestimates the crossing point but if we were to set $k(n, s) = n/s$, it would be a linear curve that lies completely below the simulated results (not shown).

4.4.2 Comparing Topologies (for IBFT)

For HotStuff, the topology practically does not matter to the performance due to its linear message complexity. However, we are still interested in comparing the performance of IBFT on both a folded Clos network and a dragonfly network. With the formulas derived, we can calculate the ratio of time to consensus on a folded Clos network versus a dragonfly network.

Consider a scenario where $\frac{\lambda}{\mu}$ is sufficiently small such that the switch bottleneck term dominates for both topologies. We assume that both topologies have the same number of switches s. For a dragonfly, we assume its parameter, a follows the function $a(s) = \frac{-1+\sqrt{1+4s}}{2}$ such that $a(a+1) = s$. For folded Clos, we assume a three level topology, with $p = \frac{s}{3}$. It follows that $\frac{n}{p} = \frac{3n}{s}$.

$$\frac{T_{i,f}(n,\mu,\lambda)}{T_{i,d}(n,\mu,\lambda)} = \frac{4 \cdot k(n,p) \cdot (n-1)}{n-1+4(k(n,s)\cdot(n-1)+k(n,s)^2 \cdot a(a-1))}.$$

Assuming that $s|n$ and writing $k(n,s) = k = \frac{n}{s}$, we can rewrite the fraction above to be

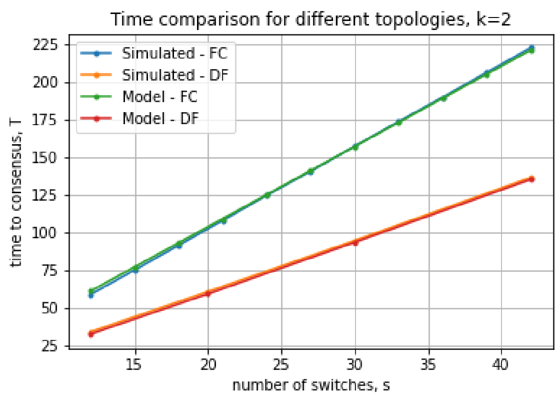
$$\begin{aligned}\frac{T_{i,f}(n,\mu,\lambda)}{T_{i,d}(n,\mu,\lambda)} &= \frac{4 \cdot 3k \cdot (n-1)}{n-1+4\left(k\cdot(n-1)+k^2(a(a-1))\right)} \\ &= \frac{12}{1/k+4+4k\frac{a(a-1)}{n-1}}, \qquad \text{factor out } k(n-1).\end{aligned}$$

An experiment we can perform would be to fix k instead and increase the number of switches, s. Then $n = ks$ and every term in the fraction becomes a function of s. We can test our models and the results of this experiment can be found in Fig. 4.7. Note that points are only taken when s is such that reasonable parameters can be set for the topology. Due to simulation limitations, s cannot be too large resulting in dragonfly only having 4 simulated results in the graph.

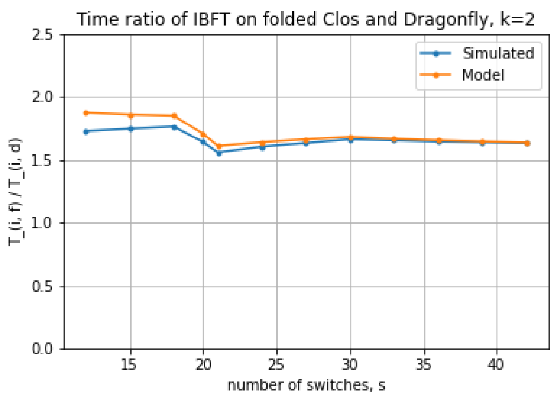
With fixed k, we can obtain some theoretical bounds for the relative performance of the topologies. Since $s = a(a+1)$ by definition of the topology, together with $s \to \infty \Rightarrow \frac{a(a-1)}{a(a+1)} \to 1$, we can establish that $\frac{a(a-1)}{n-1} = \frac{a(a-1)}{a(a+1)}\frac{s}{n-1} \to \frac{1}{k}$.

$$s \to \infty \Rightarrow \frac{T_{i,f}(n,\mu,\lambda)}{T_{i,d}(n,\mu,\lambda)} = \frac{12}{1/k+4+4k\frac{a(a-1)}{n-1}} \to \frac{12}{1/k+8} = 1.5 - \frac{3}{2+16k}.$$

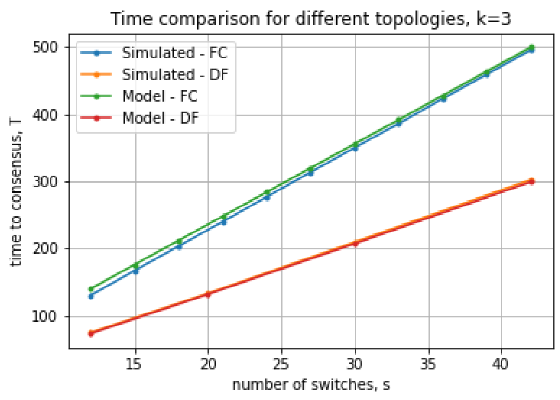
As $k \to \infty$, the fraction approaches 1.5. However, this is mostly of theoretical interest as the range in which the fraction in the model falls below 1.51 is in the order of $n > 1000$ which is a scale these consensus protocols are unlikely to run at. In simulations we ran, at range of $n \leq 100$, the ratio generally ranges from about 1.75 to slightly below 1.6.



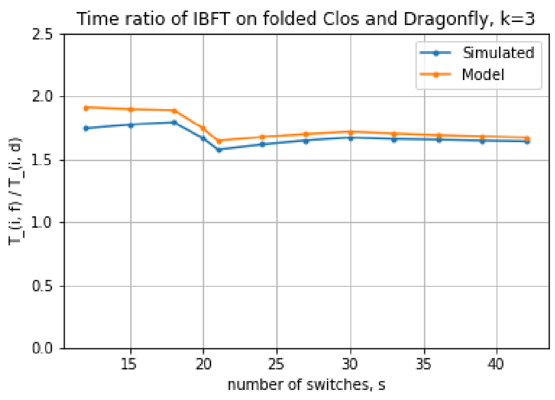
a)



b)



c)



d)

Figure 4.7: a) and c) compare the model for both topologies with simulation results for varying s and fixed k. Plots b) and d) are obtained from interpolating the results in a) and c) and taking their ratio.

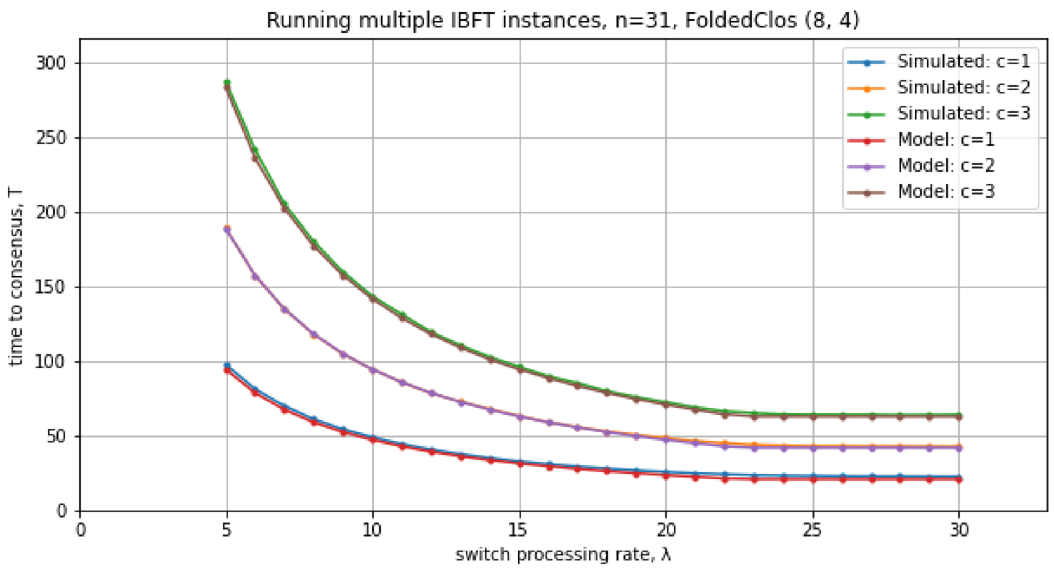


Figure 4.8: Comparison of simulation results with $c = 1, 2, 3$ consensus instances running to model with adjusted rate parameters.

4.4.3 Comment about switch processing rates

In a vacuum, individual switch processing rates are likely to be much faster than node processing rates. However, if we include other traffic within a network, from the perspective of the terminal nodes, switch processing rates would have slowed down due to dealing with other messages that are not part of the protocol. Depending on how much traffic is flowing through the network, it could easily slow down the switch processing rates to be of a similar order of magnitude to the node processing rates.

One way to verify such a phenomenon is to run multiple IBFT consensus protocols with the same set of terminal nodes. We use IBFT as the resulting traffic across the network should be significantly more uniform than HotStuff. Our hypothesis is that if we run two consensus instances, it would be almost equivalent to halving both the switch and node processing rates for a single consensus protocol run. And in general, if we run c simultaneous consensus instances, then it is equivalent to multiplying the processing rates by a factor of $1/c$. The results of the comparison can be found in Figure 4.8. The simulation results form a close fit with the model where we substituted the adjusted rate parameters.

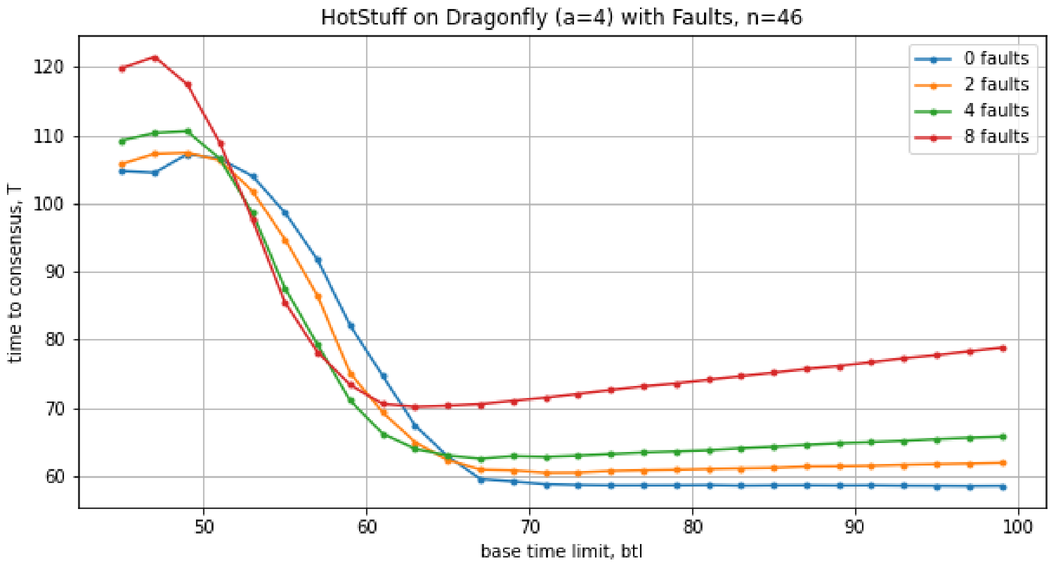


Figure 4.9: Simulation results of HotStuff with faults on dragonfly topology ($a = 4$).

4.5 Scenario 4: Topology with round change

For this section, we will only be looking at HotStuff on a dragonfly network and proposing a possible methodology to analyze BFT protocols on a more complex network topology.

For this scenario, we set number of validators, $n = 46$, using a dragonfly ($a = 4$) with fixed node processing rate $\mu = 3s^{-1}$, switch processing rate $\lambda = 15s^{-1}$. Number of faults is varied with $n_f = 0, 2, 4, 8$. Results of the simulation can be found in Figure 4.9.

4.5.1 Model Methodology

The approach for this scenario is to combine the methodology of both the approach to faults on a clique in Section 3.2.1 and the approach for a topology without round change in Section 4.3.1. We analyze how the presence of the topology affects the main timeout event E and the time to consensus given no timeout t_{e2e}, and combine both approaches to create our new model for HotStuff with faults on a topology.

4.5.2 HotStuff Model Details

For this model, we will start with equation 3.3, model for HotStuff with faults on a clique. T is the random variable for the time taken for a single consensus instance.

$$E[T] = t_{e2e} + \frac{btl}{1-2r} \cdot (r + (1-r)(1-p)).$$

We will only update the values for t_{e2e} and p.

Updating t_{e2e}

We will be using the simplified HotStuff model for this from equation 4.1. We had the following expression for time to consensus.

$$4\max\left\{\frac{n-f-1}{\mu}, \frac{n-2}{\lambda}\right\} + 3\max\left\{\frac{f+2}{\mu}, \frac{n}{\lambda}\right\} + \frac{2h}{\lambda} + \frac{2}{\mu}.$$

Each $\frac{1}{\mu}$ and $\frac{1}{\lambda}$ refers to the expected time taken to process a single message by a node or a switch respectively. Denote $X_{i,j}$ to be the time taken to process message i in phase j by a validator and $Y_{i,j}$ to be the time taken to process message i in phase j by a switch (if j is omitted, it does not belong to any phase), we can rewrite the above expression to be

$$E\left[\max\left\{\sum_{j=1}^{4}\sum_{i=1}^{n-f-1} X_{i,j}, \sum_{1}^{4}\sum_{1}^{n-2} Y_{i,j}\right\} + \max\left\{\sum_{j=1}^{3}\sum_{i=1}^{f+2} X_{i,j}, \sum_{1}^{3}\sum_{1}^{n} Y_{i,j}\right\} + 2X_i + 2hY_i\right].$$

Then we can update the bounds due to the faulty validators. For the first barrier, we compared the processing time of $n-f-1$ validator messages and $n-2$ switch messages. $n-f-1$ will not change as the leader will still need to process $n-f$ messages to broadcast the messages for the next phase. However, for $n-2$ which are meant to be all the messages the leader receives from other validators besides the first, it will be changed to $n-n_f-2$ as n_f validators will not respond to the leader due to being faulty. Similarly for barrier 2, the leader will still broadcast $n-1$ messages but will only have $f-n_f+2$ messages left to process since it received $n-n_f$ response messages in total for that phase. We can then update the bounds and define a random variable V to be

$$V := \max\left\{\sum_{j=1}^{4}\sum_{i=1}^{n-f-1} X_{i,j}, \sum_{1}^{4}\sum_{1}^{n-n_f-2} Y_{i,j}\right\} + \max\left\{\sum_{j=1}^{3}\sum_{i=1}^{f-n_f+2} X_{i,j}, \sum_{1}^{3}\sum_{1}^{n} Y_{i,j}\right\} + 2X_i + 2hY_i,$$

and set $t_{e2e} = E[V]$.

Updating E and p

Instead of event E being the time taken for the leader to process a certain amount of messages exceeding btl, we change E to be the event that $V > btl$. We can then get p using V.

A more accurate value of p can likely be obtained by considering the expectation and variance of the maximum of 2 normal random variables. However, we were not able to complete that analysis and we instead use a simple (but inaccurate) approximation for V by assuming it is a normal random variable with the following expectation and variance.

$$\begin{aligned} E[V] \approx\ & 4\max\{(n-f-1)E[X_i], (n-n_f-2)E[Y_i]\} \\ & + 3\max\{(f-n_f+2)E[X_i], nE[Y_i]\} + 2E[X_i] + 2hE[Y_i], \end{aligned}$$

and variance

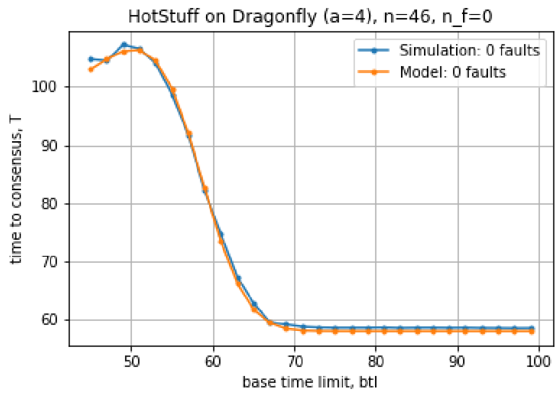
$$\begin{aligned} Var[V] \approx\ & 4\max\{(n-f-1)Var[X_i], (n-n_f-2)Var[Y_i]\} \\ & + 3\max\{(f-n_f+2)Var[X_i], nVar[Y_i]\} + 2Var[X_i] + 2hVar[Y_i]. \end{aligned}$$

assuming the time taken to process messages are independent.

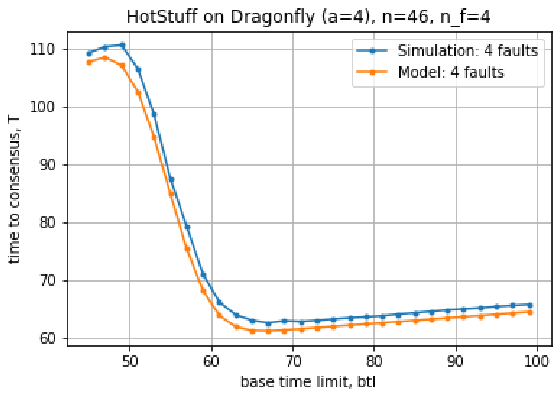
Then we assume

$$V \sim \mathcal{N}(E[V], Var[V]) \text{approximately}.$$

With this, we can approximate the value of $E[T]$ given the number of faults n_f and the btl. The results of this comparison can be found in Figure 4.10. From the figure, we see that if $n_f = 0$, we get a reasonably good fit for the performance of HotStuff on the topology. For $n_f = 4$, the model constantly underestimates the time to consensus but gets the general shape of the curve correct. This suggests that t_{e2e} is estimated to be too small when there are faults among the validators.



a)



b)

Figure 4.10: Comparison of HotStuff model and simulation on dragonfly (a=4) with $n_f = 0, 4$.

Chapter 5

Summary, Discussion and Future Work

5.1 Summary of Results

We give a summary of the results obtained from scenarios 1, 2 and 3. Scenario 4 is excluded as it is incomplete with significant room for improvement.

5.1.1 Scenario 1: Clique without round change

n = number of validators, $f = \left\lfloor \frac{n-1}{3} \right\rfloor$, μ = node processing rate.

- HotStuff (equation 3.1)

$$T(n, \mu) = \frac{4n - f + 4}{\mu}$$

- IBFT (equation 3.2)

$$T(n, \mu) = \frac{(10n + 2)(2n + 1)}{\mu(7 + 11n - \sqrt{37 + 70n + n^2})}$$

5.1.2 Scenario 2: Clique with round change

n_f = number of faulty validators, btl = base time limit for the protocol, $r = \frac{n_f}{n}$. Assume message processing time is profiled with mean x and variance σ^2.

- HotStuff (equation 3.3)

$$T(n, n_f, \mu, btl) = t_{e2e} + \frac{btl}{1 - 2r} \cdot (r + (1 - r) \cdot (1 - p)), \qquad p = P(T_m < btl)$$

where $m = 4n - f - 3n_f + 4$, $t_{e2e} = \frac{m}{\mu}$, $T_m \sim \mathcal{N}((m-1)x, (m-1)\sigma^2)$.

- IBFT (equation 3.5)

$$\begin{aligned}
T(n, n_f, \mu, btl) &= E[T|E_1] \cdot r + E[T|E_2] \cdot (1-r)(1-p) + E[T|E_3] \cdot (1-r)p, \\
E[T|E_1] &= btl \left[\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f - 1, i-1)}{{}_nP_r(n-1, i-1)} 2^{i-1} \right] + \frac{2n_w + n - f + 1}{\mu}, \\
E[T|E_2] &= btl \left[\sum_{i=1}^{n_f} \frac{{}_nP_r(n_f, i-1)}{{}_nP_r(n-1, i-1)} 2^{i-1} \right] + \frac{1}{\mu} \left[n_w(4-r) + 2 \right], \\
E[T|E_3] &= t_{e2e}, \\
p &= P(T_m < btl)
\end{aligned}$$

where ${}_nP_r = \frac{n!}{(n-r)!}$, $m = n_w + n - f$, $T_m \sim \mathcal{N}\left(mx, m\left(2 + \frac{1}{n_w}\right)\sigma^2\right)$ approximately and

$$t_{e2e} = \frac{(10n_w + 2)(2n_w + 1)}{\mu(7 + 11n_w - \sqrt{37 + 70n_w + (n_w)^2})}.$$

Recommendations for *btl* to set.

- HotStuff (equation 3.4)

$$btl_{rec} = (m-1)x + 3\sqrt{m-1} \cdot \sigma.$$

- IBFT (equation 3.6)

$$btl_{rec} = mx + 3\sqrt{m \cdot \left(2 + \frac{1}{n_w}\right)} \cdot \sigma.$$

5.1.3 Scenario 3: Topology with no round change

$\lambda =$ switch processing rate. Folded Clos topology are parameterized by p, q and dragonfly topologies are parameterized by a. Define the function G

$$G(u_1, v_1, u_2, v_2) = u_1 \Phi\left(\frac{u_1 - u_2}{\theta(v_1, v_2)}\right) + u_2 \Phi\left(\frac{u_2 - u_1}{\theta(v_1, v_2)}\right) + \theta(v_1, v_2) \phi\left(\frac{u_1 - u_2}{\theta(v_1, v_2)}\right)$$

- HotStuff - both topologies (equation 4.3).

$$T(n, \mu, \lambda) = 4G\left(\frac{m_1}{\mu}, \frac{m_1}{\mu^2}, \frac{m_2}{\lambda}, \frac{m_2}{\lambda^2}\right) + 3G\left(\frac{m_3}{\mu}, \frac{m_3}{\mu^2}, \frac{m_4}{\lambda}, \frac{m_4}{\lambda^2}\right) + \frac{2h}{\lambda} + \frac{2}{\mu}.$$

  where $m_1 = n - f - 1, m_2 = n - 2, m_3 = f + 2, m_4 = n$ and $h =$ average number of intermediate hops a message has to take to reach another node.

- IBFT - generalized (equation 4.6)

$$T(n, \mu, \lambda) = G\left(\frac{m_1}{\mu}, \frac{m_1}{\mu^2}, \frac{m_2}{\lambda}, \frac{m_2}{\lambda^2}\right).$$

  Where $m_1 = 2n + 1$. Define $k(n, s) = \frac{\lceil n/s \rceil + n/s}{2}$. Then m_2 is given by each topology to be

  - Folded Clos (equation 4.7)

$$m_2 = 4 \cdot k(n, p) \cdot (n - 1).$$

  - Dragonfly (equation 4.8)

$$m_2 = n - 1 + 4(k(n, s) \cdot (n - 1) + k(n, s)^2 \cdot a(a - 1)).$$

5.2 Discussion

5.2.1 Comparing Consensus Protocols

In Chapter 3, we saw that IBFT on a clique outperforms HotStuff significantly given the same node processing time. And the reason for that is that the most bottlenecked validator only needs to process $2n+1$ messages compared to the $4n+f-4$ messages of the HotStuff leader. However, in Chapter 4, we saw the impact of the quadratic message complexity of IBFT. At smaller values of *btl* and larger values of n, we often see IBFT perform significantly worse than HotStuff. There is contention with the usage of switches to relay messages through the network and the quadratic message complexity causes IBFT to suffer from it significantly more than HotStuff.

The main feature that causes loss of performance in HotStuff is the large number of messages the leader has to process. And its main strength against IBFT is its linear message complexity. Thus, ideally, we would like to have a protocol with linear message complexity and fewer phases than HotStuff. The pipeline variant of HotStuff mentioned in [9] would still require three sets of round-trip communication to agree on a value but is able to run multiple consensus instances in parallel by the usage of pipelining as opposed to the simple variant that we analyzed. Such a protocol might be able to outperform IBFT even on a clique scenario though it may be more adversely affected by faults in the network.

5.2.2 Considerations for Topology

We saw that in Section 4.4.2, with the same number of switches, the dragonfly topology outperforms folded Clos by having folded Clos time to consensus being about 1.5 times that of dragonfly. The reasoning behind this is relatively straightforward as the way we have configured our folded Clos topology is such that it only uses a third of its switches as edge switches that facilitate connections with terminal nodes. A result of this is that the average number of terminal node connections per edge switch is usually 3 times greater than that of a dragonfly topology.

We have seen from our analysis that a key factor in the performance of IBFT protocol on a topology is the average number of terminal node connections per edge switch which we denoted as the function k. For dragonfly, to have k be one third of that in folded Clos gives it that advantage and greatly offsets the performance hit as a result of having additional traffic that utilizes edge switches as an intermediate hop.

However, such analysis does not take into account the practical considerations of using these two topologies. For example, our configuration of dragonfly with number of groups $g = a + 1$ where a is the number of switches in a group, is such that every switch is vital as it is connected to some amount of terminal nodes. If a switch fails, all its terminal nodes would be taken off the network. Additionally, suppose a faulty switch is connected to another group g_1. Then switches in the same group with that faulty switch have to route their messages to other groups $g_k \neq g_1$ and then to g_1 which would take 2 inter-group hops that are considered to be more expensive than intra-group hops. In contrast, a non-first level switch can fail without affecting the connectivity of a folded Clos network. Folded Clos networks also have much greater path diversity of fewest hop paths compared to a dragonfly network. This greatly reduces the impact of a switch failure.

In general, there are many more practical considerations when selecting a topology for use. If we are only looking at the performance of a quadratic message complexity protocol, then we might consider using a topology that minimizes k. If we are more interested in resilience, then a topology like folded Clos having more built-in redundancy in the network might be a better choice.

5.3 Future Work

5.3.1 Extending the Topology Fault Model

For the scenario where we investigated a protocol with faults on a topology and attempt to find the optimal *btl*, we only looked at HotStuff and only at the dragonfly topology. The first extension would be to properly model the probability distribution of V by appropriately using the probability distribution of the maximum of two normal random variables.

We can also attempt to generalize a model building methodology for different topologies and try to extend it to IBFT as well.

5.3.2 Investigating other protocols

HotStuff as a protocol requires three sets of round-trip communication to come to a consensus on some value. The first extension to consider could be to implement the pipeline HotStuff variant described in [9]. Alternatively, we can investigate the Marlin protocol [10] which is a recent two-phase linear message complexity BFT that was developed. It would be interesting to investigate its performance and contrast it with the BFT protocols we have looked at.

5.3.3 Including other forms of delay

A more realistic simulation would require including transmission delay that we had omitted in our simulation. Additionally, we have simplified the time required for message processing to follow an exponential distribution with some predetermined rate. In practice, the need for public/private key authentication in the protocol and the difference in the overhead that each protocol message is required to carry might affect the rate the messages are processed, affecting the performance of a protocol.

References

[1] Dennis Abts and John Kim. *High Performance Datacenter Networks Architectures, Algorithms, and Oportunities*. Morgan & Claypool, 2011.

[2] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The Next 700 BFT Protocols. *ACM Trans. Comput. Syst.*, 32(4), jan 2015.

[3] Yen Ren Zuo Brian. Analytical Performance Comparison of Byzantine Consensus Protocols. *School of Computing Final Year Project*, 2022.

[4] Sisi Duan and Haibin Zhang. Recent progress on BFT in the era of blockchains. *National Science Review*, 9(10), 07 2022. nwac132.

[5] Raluca Halalai, Thomas A. Henzinger, and Vasu Singh. Quantitative Evaluation of BFT Protocols. In *2011 Eighth International Conference on Quantitative Evaluation of SysTems*, pages 255–264, 2011.

[6] Henrique Moniz. The Istanbul BFT Consensus Algorithm. *CoRR*, abs/2002.03613, 2020.

[7] Saralees Nadarajah and Samuel Kotz. Exact Distribution of the Max/Min of Two Gaussian Random Variables. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 16(2):210–212, 2008.

[8] Qinyin Ni, Zhang Linfeng, Xiaorong Zhu, and Inayat Ali. A Novel Design Method of High Throughput Blockchain for 6G Networks: Performance Analysis and Optimization Model. *IEEE Internet of Things Journal*, 9(24):25643–25659, 2022.

[9] Yahya Shahsavari, Kaiwen Zhang, and Chamseddine Talhi. Performance Modeling and Analysis of Hotstuff for Blockchain Consensus. In *2022 Fourth International Conference on Blockchain Computing and Applications (BCCA)*, pages 135–142, 2022.

[10] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-Phase BFT with Linearity. In *2022 52nd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 54–66, 2022.

[11] Y. C. Tay. *Analytical Performance Modeling for Computer Systems.* Morgan & Claypool Publishers, 2nd edition, 2013.

[12] Xianhua Wei, Aiya Li, and Zhou He. Impacts of consensus protocols and trade network topologies on blockchain system performance. *Journal of Artificial Societies and Social Simulation*, 23(3):2, 2020.

[13] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. Hotstuff: BFT Consensus in the Lens of Blockchain. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 347–356. ACM, 2019.

Appendices

Appendix A

Proofs and Derivations

A.1 Derivation of Clique IBFT model without Round Change

This derivation is taken from [3]. Let μ_T denote the rate of consensus instance of IBFT and μ to denote the rate a message is processed by a validator, and n to be the number of validators. Assume further that $n = 3f + 1$ for some integer f which is the maximum number of acceptable faulty validators.

Recall that for IBFT, a validator has to process $2n + 1$ messages per consensus instance. 1 *PRE-PREPARE* message, and n of *PREPARE* and *COMMIT* messages each from the broadcast step.

Next, we set up an expression for $T = \frac{1}{\mu_T}$. For the n *PREPARE* messages, a validator has to process all n of them before starting to process *COMMIT* messages. For the n *COMMIT* messages, only $n - f$ of them have to be processed before consensus is reached.

Lastly, to account for the 1 *PRE-PREPARE* message, we use the Pollaczek-Khinchine formula [11] to find the mean number of messages in a validator's queue. We introduce the following definitions:

- $L =$ average number of messages in the queue
- $\lambda =$ arrival rate of messages
- $\rho = \lambda/\mu$
- $S =$ service time distribution

- W = average waiting time of a message in the queue (including processing)

From here, we can obtain the time from completing a consensus instance to completing the processing of the *PRE-PREPARE* message for the next instance by using Little's Law. Note that S follows an exponential distribution with rate μ.

$$L = \rho + \frac{\rho^2 + \lambda^2 Var(S)}{2(1-\rho)}.$$

For an exponential distribution, $Var(S) = 1/\mu^2$ which gives

$$L = \rho + \frac{\rho^2 + \rho^2}{2(1-\rho)} = \frac{2\rho - 2\rho^2 + \rho^2 + \rho^2}{2(1-\rho)} = \frac{\rho}{1-\rho}.$$

Using Little's Law [11], $L = \lambda W$, we can obtain the time taken to process the *PRE-PREPARE* message which would be W.

$$W = \frac{L}{\lambda} = \frac{\rho}{\lambda(1-\rho)} = \frac{1}{\mu - \lambda}.$$

Putting it together, we have

$$T = \frac{1}{\mu-\lambda} + \frac{n}{\mu} + \frac{n-f}{\mu} = \frac{1}{\mu-\lambda} + \frac{2n-f}{\mu}$$

Lastly, note that λ and μ_T are related by the fact that a consensus instance is achieved once every $2n+1$ messages are received and processed. Thus,

$$\mu_T = \frac{1}{2n+1}\lambda \Rightarrow T = \frac{2n+1}{\lambda} \Rightarrow \frac{2n+1}{\lambda} = \frac{1}{\mu-\lambda} + \frac{2n-f}{\mu}.$$

From here, we can express λ in terms of n, f, μ, of which f is also a function of n. Solving for λ, and then finding $T = \frac{2n+1}{\lambda}$, we get

$$T = \frac{(10n+2)(2n+1)}{\mu(7 + 11n - \sqrt{37 + 70n + n^2})}.$$

A.2 Conditions for comparing IBFT and HotStuff topology formula

We wish to show the conditions for which $\frac{n}{\lambda} > \frac{f+2}{\mu} \Rightarrow T_i > T_h$, where T_i is an IBFT model on any topology. If $\frac{n}{\lambda} > \frac{f+2}{\mu}$, then

$$\begin{aligned}
T_h(n,\mu,\lambda) &= 4\max\left\{\frac{n-f-1}{\mu}, \frac{n-2}{\lambda}\right\} + \frac{3n}{\lambda} + \frac{2h}{\lambda} + \frac{2}{\mu} \\
&\le \frac{4n}{\lambda} + \frac{3n}{\lambda} + \frac{2h}{\lambda} + \frac{2}{\mu}, && \because \frac{n-f-1}{\mu} \le \frac{2(f+2)}{\mu} \le \frac{2n}{\lambda} \\
&\le \frac{8n}{\lambda}, && \text{if} \quad 2h < n.
\end{aligned}$$

Then let s be the number of edge switches. In our models, we always have $T_i \ge \frac{4(n/s)(n-1)}{\lambda}$ which comes from the number of broadcasted messages going through an edge switch for IBFT. Thus, we only require that

$$\frac{4(n-1)}{s} \ge 8 \Rightarrow \frac{n-1}{s} \ge 2$$

which is often the case for most real-world switch uses.

Appendix B

Additional Figures

B.1 Scenario 3: Topology with no round change

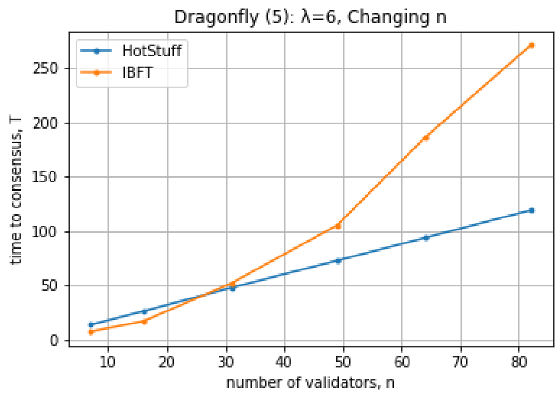
For scenario 3b, we simulated the results for varying n at $\lambda = 6, 9, 15, 30$. The additional results can be found in Fig. B.1.

B.2 Comparison of folded Clos model

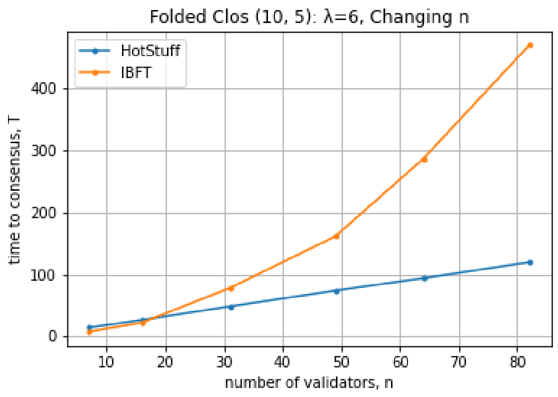
For comparison of folded Clos model in scenario 3, we plotted the results of varying n at $\lambda = 6, 9, 15, 30$. The additional results can be found in Fig. B.2.

B.3 Comparison of dragonfly model

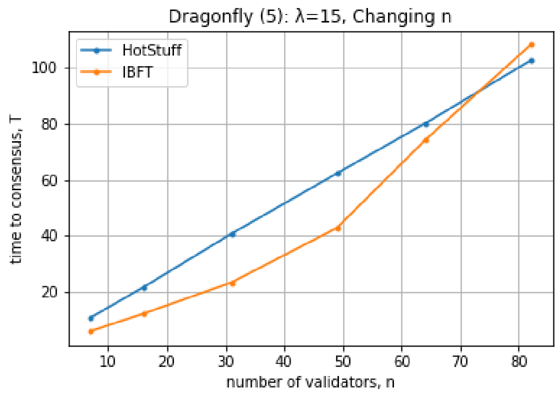
For comparison of dragonfly model in scenario 3, we plotted the results of varying n at $\lambda = 6, 9, 15, 30$. The additional results can be found in Fig. B.3.



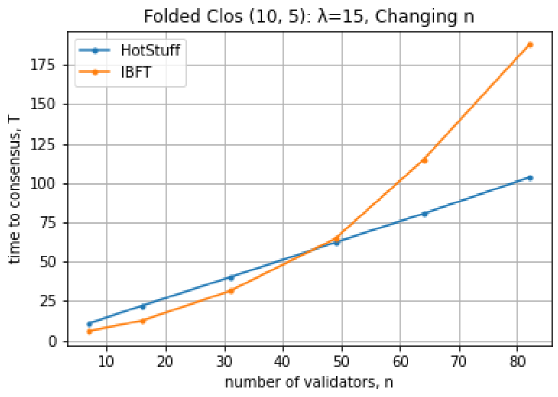
a)



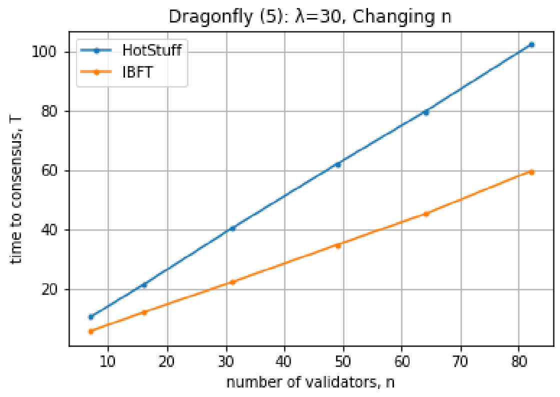
b)



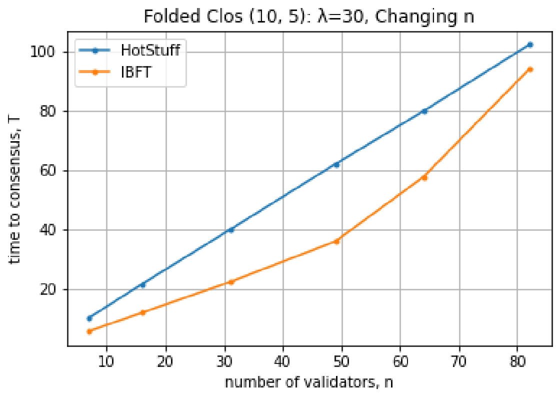
c)



d)

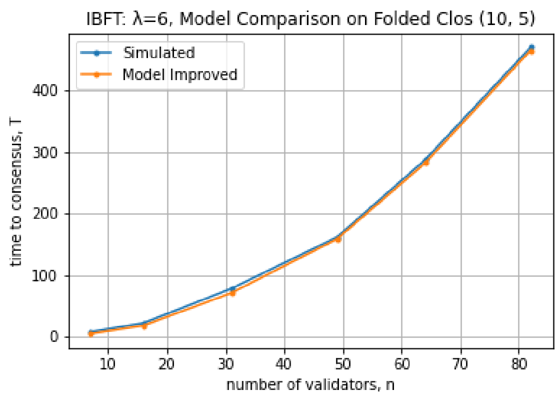


e)

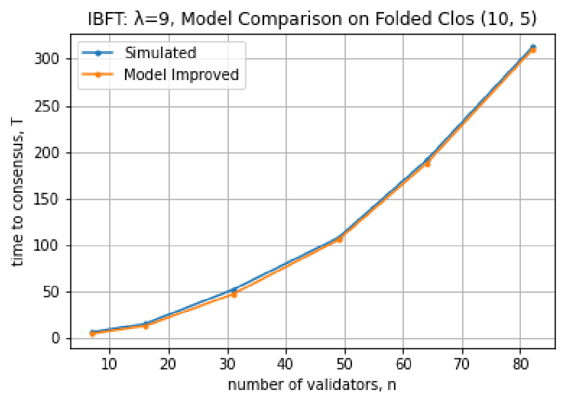


f)

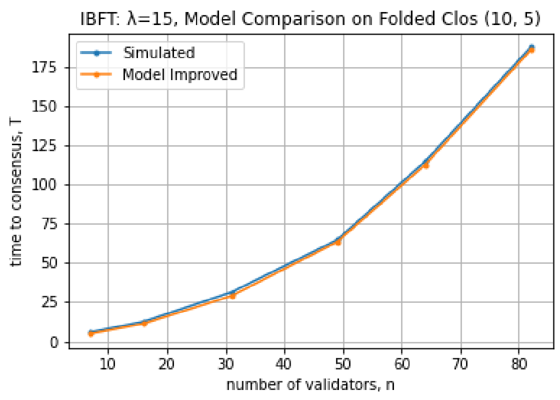
Figure B.1: Various simulation experiments with varying n on a network topology for different fixed values of λ.



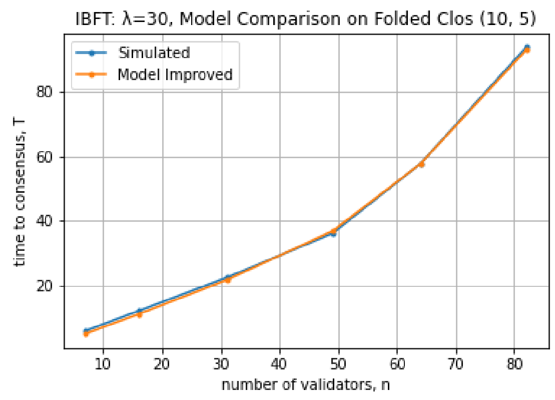
a)



b)

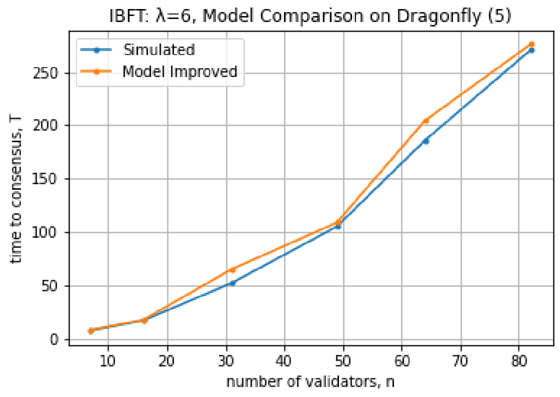


c)

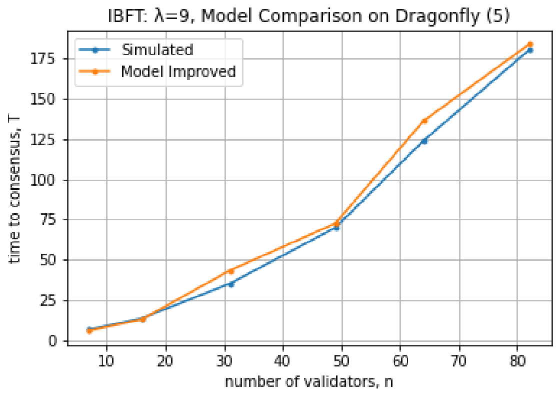


d)

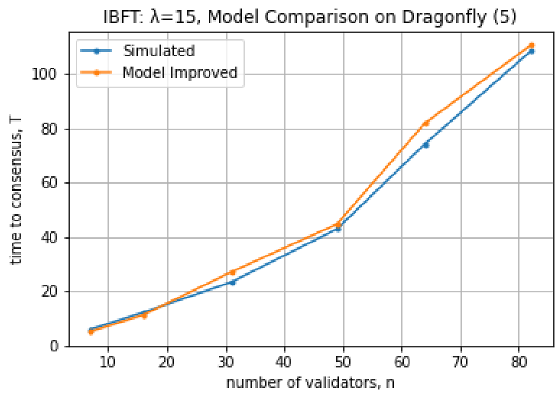
Figure B.2: Comparison of IBFT folded Clos model for scenario 3 with varying n for different fixed values of λ.



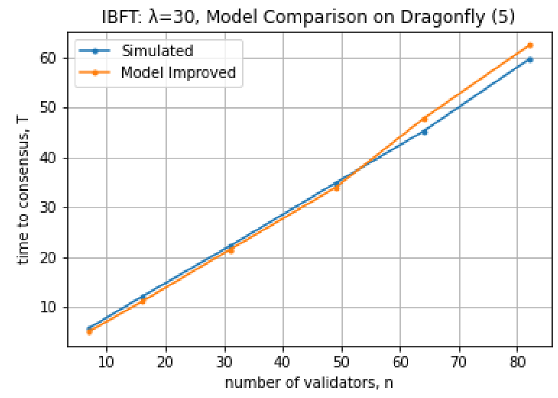
a)



b)



c)



d)

Figure B.3: Comparison of IBFT folded Clos model for scenario 3 with varying n for different fixed values of λ.