Fault Tolerant Broadcast in Bandwidth-Constrained Networks

by

Ioannis Kaklamanis

S.B., Computer Science and Engineering and Mathematics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2023

Authored By:	Ioannis Kaklamanis Department of Electrical Engineering and Computer Science May 12, 2023
Certified by:	Mohammad Alizadeh Associate Professor Thesis Supervisor
Certified by:	Lei Yang PhD Candidate Thesis Supervisor
Accepted by:	Katrina LaCurts Chair, Master of Engineering Thesis Committee

Fault Tolerant Broadcast in Bandwidth-Constrained Networks

by

Ioannis Kaklamanis

Submitted to the Department of Electrical Engineering and Computer Science
on May 12, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

This thesis addresses the problem of achieving scalable fault-tolerant broadcast in networks with limited bandwidth. We begin by examining the limitations of leader-based protocols, such as HotStuff, which suffer from a leader bottleneck and reduced system throughput as the number of servers increases. To mitigate this, we propose CodedBcaster and Coded HotStuff, a Byzantine Fault Tolerant (BFT) broadcast scheme based on erasure coding, demonstrating a significant improvement in throughput. We further explore the problem of optimal rate allocation in heterogeneous node-constrained networks and provide concrete theoretical results for determining the optimal system throughput rate. Additionally, we propose the MaxMin Rate Controller (MaxMin-RC) protocol as a feedback-based solution to optimize broadcast throughput in non-BFT settings, achieving close alignment with the optimal throughput rate. Through extensive simulations and evaluations, we demonstrate the effectiveness of our proposed solutions.

Thesis Supervisor: Mohammad Alizadeh
Title: Associate Professor

Thesis Supervisor: Lei Yang
Title: PhD Candidate

Acknowledgments

I would like to express my utmost gratitude to my thesis supervisor, Prof. Mohammad Alizadeh. His expertise, important contributions, insightful feedback, and overall guidance and continuous support have been invaluable in shaping the direction and quality of my work. I am truly fortunate to have had the opportunity to conduct research with him.

I would like to extend my sincere gratitude to my direct supervisor and mentor, Lei Yang. His exceptional guidance, dedication, and support have been instrumental in providing me with valuable insights and helping me navigate challenges throughout my research journey. I particularly appreciate his patience and willingness to discuss ideas, as well as his efforts to keep me updated on the latest advancements in the field. I am indebted to Lei for his continuous assistance with and concern for my overall research and academic success.

I am deeply grateful to my friends for their constant support, encouragement, and friendship throughout this MIT journey. Their presence has made this experience all the more meaningful and enjoyable. I wish to express my heartfelt appreciation to my friends Panagiotis, Antonis, Giorgos, William, Pawan, and Sagnik.

I would like to extend my special thanks to Savvina for her incredible support and encouragement throughout this journey. She has been a constant source of strength and inspiration. I am truly grateful for her presence in my life and for the love and care she has shown me.

I would like to express my heartfelt gratitude to my family, especially to my father Christos and my brother Petros. Their unwavering presence, love, and support have been a source of immense strength and motivation. I would also like to extend my deepest appreciation to my mother for her profound impact on my life and her boundless love and encouragement. I am certain she would be proud of my accomplishments.

I extend my gratitude to all those mentioned above, as well as everyone else who has supported me along the way during my time here at MIT. Thank you all.

Contents

List of Figures

Chapter 1

Introduction

With the success of blockchains and cryptocurrencies, Byzantine Fault Tolerant (BFT) state machine replication protocols have attracted considerable interest. Most deployed BFT protocols are *leader-based*, i.e, a designated server, known as the leader, is responsible for proposing and broadcasting blocks of new data to be applied to the system's state machine. However, the simple implementation of broadcast in leader-based protocols often leads to a network bottleneck at the leader, resulting in reduced system throughput as the number of servers scales.

In this thesis, we address the challenge of fault-tolerant broadcast in bandwidth-constrained networks. We begin by examining the limitations of leader-based protocols, focusing on the state-of-the-art HotStuff [21] protocol. HotStuff suffers from reduced performance due to the centralized nature of block broadcasting, where the leader is required to send entire blocks to all other servers. As a result, the leader becomes a performance bottleneck, leading to decreased system throughput.

To mitigate the bottleneck issue in leader-based protocols, we propose the use of erasure codes. The core idea is to let the leader encode the block into smaller "chunks" (coded shares) and assign each server the task of broadcasting a chunk. This approach leverages the bandwidth of all servers, improving overall system throughput.

While we focus on HotStuff, we believe that coded broadcast can benefit other leader-based BFT protocols, as long as there is a broadcast component on the critical path. This motivates the design and implementation of CodedBcaster, an erasure

code-based module that isolates and handles the broadcast functionality within a distributed protocol. We carefully prove that CodedBcaster is secure against Byzantine attacks and experimentally confirm its theoretical performance guarantees.

While our initial analysis assumes equal bandwidth capacities among all servers, real-world scenarios often involve variations in bandwidth availability across nodes and over time. In such cases, the previously proposed coding-based solution may not achieve optimal throughput. Therefore, we explore the challenge of broadcast in a heterogeneous node-constrained network, where different servers possess varying bandwidth capacities.

In particular, we investigate the challenge of determining the optimal rate allocation in heterogeneous node-constrained networks, assuming the absence of Byzantine faults. By "optimal rate allocation," we refer to the rate allocation that maximizes the system throughput, which is in turn defined as the minimum receiving rate across follower nodes. Our research yields two significant theoretical contributions. Firstly, we establish that the optimal system throughput rate (r_{OPT}) can be determined as a simple function of the ingress and egress capacities of the nodes. Notably, our proof is constructive, i.e., we describe how to construct a rate allocation scheme that achieves the r_{OPT} throughput rate. Secondly, we investigate the network coding variant of our optimization problem, formulated as a Linear Program. We prove that the optimal throughput rate of the network coding variant equals r_{OPT}, highlighting that achieving the optimal throughput rate does not require network coding.

Building upon these theoretical findings, we introduce the MaxMin Rate Controller (MaxMin-RC) protocol, a simple yet powerful feedback-based solution for optimizing broadcast rates in (non-BFT) heterogeneous node-constrained networks. The MaxMin-RC protocol aims to closely match the r_{OPT} throughput rate. We present the system design of MaxMin-RC, implement it in the OMNeT++ simulation framework, and conduct extensive simulations to validate its efficiency.

In summary, this thesis focuses on addressing the challenges of fault-tolerant broadcast in bandwidth-constrained networks. We propose CodedBcaster and Coded HotStuff as novel solutions to improve system performance and scalability in leader-

based BFT protocols. We also contribute to the understanding of optimal rate determination in heterogeneous node-constrained networks and introduce the MaxMin-RC protocol for efficient rate control.

Part of this thesis is published in our poster titled "Coded Broadcast for Scalable Leader-Based BFT Consensus" [11].

1.1 Contributions

Below are the four main contributions in this thesis:

- **CodedBcaster**: We designed and implemented CodedBcaster, an erasure code-based module that isolates and handles the broadcast functionality within a distributed protocol. We proved its security and conducted experiments to validate its theoretical performance guarantees.

- **Coded HotStuff:** We used CodedBcaster to address the leader bottleneck issue in HotStuff [21]. We proved Coded HotStuff is Byzantine Fault Tolerant, and provided atheoretical analysis of the improved performance backed by experiments.

- **Optimal Throughput Rate in Heterogeneous Node-Constrained Networks:** We conducted theoretical research and constructively proved that the optimal throughput rate (r_{OPT}) can be determined as a simple function of the ingress and egress capacities of the nodes. Further, we demonstrated that network coding is not necessary for achieving the optimal rate.

- **MaxMin-RC Protocol:** We designed the MaxMin-RC controller, a simple yet powerful solution for optimizing broadcast throughput rate in variable bandwidth, non-BFT settings. The protocol's implementation in OMNeT++ and extensive simulations validate its efficiency.

Chapter 2

Byzantine Broadcast in a Homogeneous Node-Constrained Network

2.1 Introduction

Byzantine fault-tolerant state machine replication protocols (*BFT protocols*) allow a group of N servers to agree on an ordered log of *transactions*, where up to f servers may be *Byzantine* and deviate from the protocol arbitrarily. Leader-based protocols, such as HotStuff [21], have gained popularity in the field. In these protocols, a leader server is elected for each epoch and is responsible for proposing and broadcasting batches of new transactions to the follower servers. However, the straightforward implementation of broadcasting in leader-based protocols leads to a network bottleneck at the leader, limiting the system throughput as the number of servers scales.

The imbalance in network bandwidth usage arises because the leader sends complete proposals to each follower, while a follower only needs to download one copy. As a result, the leader's uplink becomes a bottleneck, hindering the scalability of leader-based protocols, especially when large-scale deployments are required, such as in public blockchains.

In this study, we focus on HotStuff [21], a state-of-the-art leader-based BFT protocol designed for the partially synchronous network model. We demonstrate the existence of the leader bottleneck through experiments on a testbed with simulated bandwidth limitations.

We propose a simple mitigation to the bottleneck, namely an efficient BFT broadcast protocol based on erasure coding. The key idea is to split a b-byte proposal into $(N-1)$ coded *shares*, each of $O(b/N)$ bytes, and task each server with broadcasting a share. Note that the communication cost is balanced across all servers. Although the leader still needs to deliver the $(N-1)$ shares to $N-1$ servers so that they can start broadcasting, it sends in total $(N-1) \times O(b/N) = O(b)$ bytes, a cost that is independent of N.[1]

To accommodate Byzantine servers who might ignore or corrupt the shares they are responsible for broadcasting, the leader adds redundancy among the shares with an erasure code [17]. In the general case, to accommodate up to f Byzantine servers out of N, our approach requires each node to send/receive $\frac{N-1}{N-f-1}\times$ the proposal size, and is therefore within 2/3 of optimal (assuming $N \geq 3f+1$). We apply this mitigation on HotStuff, and demonstrate that it alleviates the leader bottleneck and improves the throughput by 64% when $N = 9$.

While we focus on HotStuff in this part of the thesis, we believe that coded broadcast can benefit other leader-based BFT protocols, as long as there is a broadcast component on the critical path.

This motivates the design and implementation of CodedBcaster, a standalone module that isolates and handles the broadcast functionality within a distributed protocol, relying on erasure codes. We carefully prove that CodedBcaster is secure against Byzantine attacks and experimentally confirm its theoretical performance guarantees.

[1]For simplicity, the calculation does not consider cryptographic authenticators (signatures, hashes, etc.). Such costs can be hidden by increasing the size of the proposals, so that the cost of broadcasting stays dominant. Our mitigation does not increase the asymptotic communication cost of HotStuff.

2.2 Related Work

Most deployed BFT protocols, PBFT [6], HotStuff [21], and Tendermint [4], are *leader-based*, where a *leader* server is periodically elected and responsible to drive the protocol for one or a few *epochs*. In each epoch, the leader proposes a batch of new transactions to be appended to the log, and broadcasts the proposal to all other servers (the *followers*). All servers then exchange votes to collectively commit (or discard) the proposal. A server must download the proposal before voting for it, otherwise it cannot check the validity of the content.[2] To ensure agreement despite Byzantine servers, the protocol must wait for enough followers to cast votes (usually $2f + 1$) before committing a proposal. As a result, broadcasting the proposal is on the critical path of the protocol and upper bounds the throughput of the system.

Existing protocols implement broadcasting in a simple way: the leader sends entire proposals directly to each follower [6, 21]. The problem with this simple mechanism is that the network bandwidth usage of the leader and the followers is *imbalanced*. Specifically, the leader needs to upload $N - 1$ copies of the proposal (one for each follower), while a follower downloads only one copy. As a result, the leader, specifically its network uplink, becomes the bottleneck during broadcasting.[3]

To allow leader-based protocols to scale, it is necessary to remove the leader as the communication hub during broadcast. While existing works on collective communication in high-performance computing and task-based distributed systems [22, 9] have studied this problem, their solutions do not apply to BFT protocols because of Byzantine servers.

On the other hand, recent progress on balanced BFT reliable broadcast (RBC) and verifiable information dispersal (VID) protocols [20, 2, 5] satisfies the need, but the guarantees from such protocols are too strong. Specifically, they guarantee that

[2]Some protocols, such as DispersedLedger [20, 7] forgo the check on content validity by allowing invalid transactions into the committed log and deterministically removing them at a later point. Still, the protocol must ensure the content is durably stored among the servers [20]. Such protocols are not the focus of this paper.

[3]Assuming that the uplink of the leader does not have $N\times$ higher bandwidth than the downlinks of the followers. Note that the leader is frequently rotated among all servers [21] to prevent censorship, so one cannot designate one server as the leader and provision it with extra bandwidth.

all honest servers agree on the content they receive from a broadcast. In comparison, the simple implementation of broadcast in existing protocols does not guarantee agreement[4], suggesting that this property is unnecessary in this context and may introduce extra overhead. Our approach draws inspiration from these protocols but specifically tailors the solution to minimize communication complexity in leader-based BFT protocols.

Kauri [15] leverages dissemination/aggregation trees to achieve load balancing and scalability while overcoming the drawbacks of previous tree-based solutions. The protocol introduces novel pipelining techniques that sustain high throughput as the system size grows, allowing parallel processing across different stages of the tree. Despite its novel reconfiguration strategy, the Kauri's tree-based approach (as all tree-based approaches) still suffers when one of the internal tree nodes is Byzantine. Additionally, while Kauri applies its dissemination/aggregation tree to all consensus phases, our work specifically focuses on abstracting the broadcast functionality.

Our study of Byzantine broadcast in homogeneous node-constrained networks extends our work in our published poster "Coded Broadcast for Scalable Leader-Based BFT Consensus" [11].

2.3 Coded HotStuff

In this section, we provide an overview of the Hotstuff [21] protocol, emphasizing the impact of the leader bottleneck on throughput in bandwidth-constrained networks. To address this challenge, we introduce our solution called "Coded HotStuff," which leverages erasure codes. We demonstrate the effectiveness of our coded broadcast solution in mitigating the leader bottleneck through both theoretical analysis and experimental evaluations. Furthermore, we establish the resilience of Coded HotStuff against Byzantine nodes by verifying that it satisfies the essential safety and liveness properties.

[4]For example, a Byzantine leader may send different proposals to different followers.

2.3.1 Security Model

We assume the same security model as HotStuff [21, §3]. Namely, the network is partially synchronous [6], and servers have direct network connections to each other. Among the $N = 3f + 1$ servers in the system, at most f are Byzantine.

2.3.2 HotStuff Overview

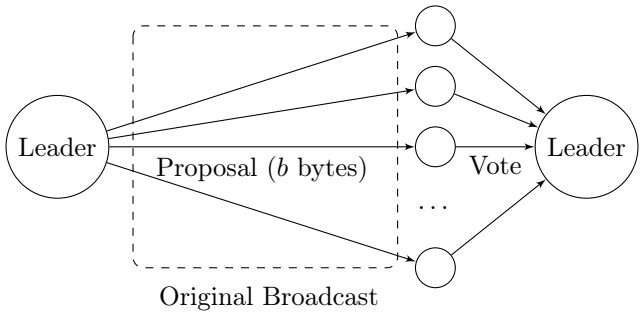


Figure 2-1: The PREPARE phase of the original HotStuff protocol. The leader sends a copy of the entire proposal to each follower, then waits for the votes.

HotStuff [21] is a leader-based BFT protocol that operates in the partially synchronous model and in a permissioned blockchain setting. The set of replicas participating is fixed, and nodes become leaders in a round-robin fashion. Unlike other efficient leader-based BFT protocols [3, 6, 4], HotStuff is built around a three-phase paradigm, which makes the leader replacement protocol simpler and allows for pipelining.

- In the PREPARE phase, the new leader collects *new-view* messages from replicas, and it chooses the branch led by the quorum certificate (QC) with the highest view number. The leader sends a PREPARE message to all replicas, which in turn check whether to accept the current view received from the leader; if so, they send PREPARE votes on the proposal to the leader.
- In the PRE-COMMIT phase, the leader combines the PREPARE votes to create *prepareQC*, which it sends to all replicas. Each replica in turn responds with a PRE-COMMIT vote.

- In the COMMIT phase, the leader combines the PRE-COMMIT votes to create *precommitQC*, which it sends to all replicas. Each replica in turn responds with a COMMIT vote and locks its current state to this *precommitQC*.

There is also a DECIDE phase, which is not technically included in [21] as one of the three core phases. In this phase, the leader combines COMMIT votes into a *commitQC*. It then broadcasts *commitQC* to all replicas, which in turn execute the corresponding commands.

HotStuff achieves comparable latency and better throughput than its BFT-SMaRt [3] counterpart. Further, the number of authenticators needed to reach consensus is linear in the number of replicas, regardless of whether the leader stays the same or not. In short, HotStuff is a state-of-the-art BFT protocol which enjoys efficiency as well as simplicity through its three-phase core.

2.3.3 Leader Bottleneck in HotStuff

At any given time, HotStuff has one leader coordinating the consensus. Fig. 2-1 shows the PREPARE phase [21, §4.1] of the protocol, where the leader broadcasts the proposal (block) to all followers. Suppose there are $N \geq 4$ servers, the block size is b bytes, and the ingress/egress bandwidth of each server is c bytes per second. The leader needs to send $(N-1)b$ bytes, while each follower receives b bytes. As a result, the egress bandwidth of the leader is the bottleneck, and it takes $\frac{(N-1)b}{c}$ seconds to finish the broadcast. As N increases, the duration increases linearly, causing the system throughput to decrease.

The key issue here is the under-utilization of the followers' bandwidth, caused by the imbalanced network load. Fig. 2-3 shows the trace of the egress bandwidth usage on one server in a deployment of the original HotStuff protocol for $N = 4, f = 1$. The server only utilizes its egress bandwidth when it is the leader.

2.3.4 Resolving the Leader Bottleneck

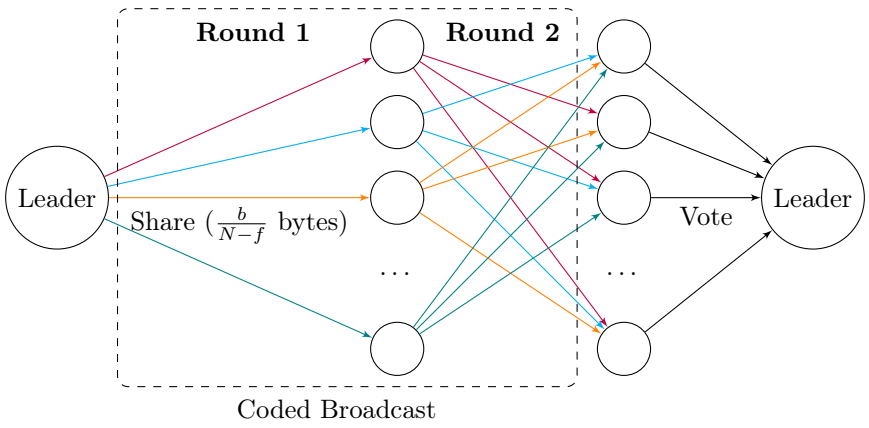


Figure 2-2: The PREPARE phase of HotStuff, modified to use our coded broadcast protocol (the inner box). In round 1, the leader sends a unique chunk to each follower. In round 2, each follower broadcasts the received chunk to all other followers.

Error Correcting Codes. Error correcting codes (ECC) are used to transmit data over unreliable communication channels. The sender encodes the message with redundant information, which allows the receiver to correct a limited number of errors and reconstruct the original message. Reed–Solomon (RS) codes [17] are a family of ECC commonly used to correct *erasure* errors, which corrupts a part of the message and make them unavailable to the receiver (the receiver still knows *which* part is corrupted). An RS(n, k) is a Reed-Solomon code which takes a message of k data symbols and adds parity symbols to make a codeword of $n > k$ symbols (shares). The original message can be recovered from any k-sized subset of the n shares.

Coded HotStuff Overview

Our design modifies the PREPARE phase of the HotStuff protocol, leaving the remaining phases unchanged. Instead of a one-round leader-to-all broadcast, our broadcast protocol has two rounds, as illustrated in Fig. 2-2.

When ready to broadcast a proposal, the leader uses an RS$(N, N-1-f)$ code to encode the proposal into $(N-1)$ shares, with the property that the original proposal can be decoded given any $(N-1-f)$ correct shares.

- In the first round, the leader sends the i-th coded share to the i-th server ($1 \leq i \leq N-1$).
- In the second round, each server broadcasts its share to all other servers.

If the leader is honest, then at the end of the second round, each honest follower must have received $N-1-f$ correct shares, so that it can decode the original proposal and proceed with the next steps per the original HotStuff protocol. If the leader is dishonest, then honest followers may fail to reconstruct a block. This is equivalent to not receiving the block during PREPARE in the original HotStuff protocol, and such followers should act accordingly.

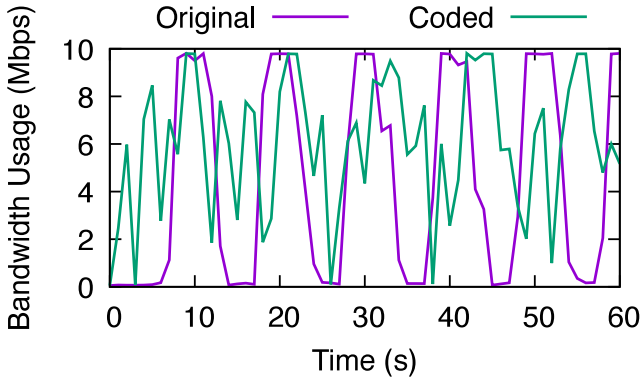


Figure 2-3: Trace of the egress bandwidth usage at one of the four servers running the original and the coded HotStuff protocol. In the original version, the server's egress bandwidth usage peaks when the server is the leader broadcasting proposed blocks. In the coded version, the server's egress bandwidth usage is more balanced, since bandwidth is being consumed both when the server is the leader and when the server forwards shares as a follower.

We now explain how coded broadcast alleviates the leader bottleneck. Consider the same setup as in §2.3.3. Each share is $b/(N-f-1)$ bytes in size, and the leader sends a total of $N-1$ shares to the followers. In the second round, each follower server broadcasts the share, in total sending $N-2$ shares. As a result, the first round takes $\frac{b}{c} \times \frac{N-1}{N-1-f}$ seconds, and the second round takes $\frac{b}{c} \times \frac{N-2}{N-1-f}$ seconds. The duration of the entire broadcast protocol upper bounded by $2\times$ the duration of the first round, i.e., $2 \cdot \frac{b(N-1)}{c(N-1-f)}$. Recall that $N \geq 3f+1$, so the duration is upper-bounded by $3b/c$, which is independent of N. Fig. 2-3 shows the trace of the egress bandwidth usage

on one server in a deployment of the coded HotStuff protocol for $N = 4, f = 1$. The server actively uses its egress bandwidth throughout the execution, both when proposing blocks as a leader and when forwarding shares as a follower.

2.3.5 Experiments and Results

Experimental Setup. [5] Our implementation extends an open-source HotStuff state machine written in Golang [18]. We modify it to replace the direct broadcast with our coded broadcast protocol. The testbed runs on an AWS EC2 `c5.xlarge` VM. We use Linux network namespace to create isolated network stacks for each HotStuff server process [6], and `qdisc` to limit the ingress and egress bandwidth of each server to 10 Mbps, respectively. We also inject a 100 ms round-trip propagation delay between each pair of nodes. Each block (proposal) is 1 MB and is filled with random data.

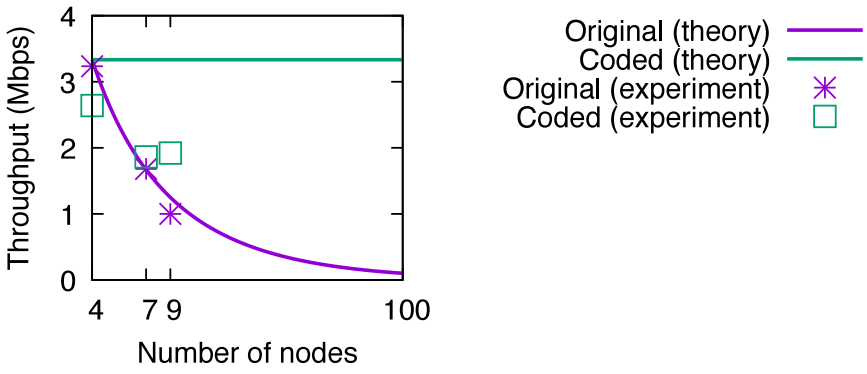


Figure 2-4: Throughput of the original HotStuff protocol and our modified version with coded broadcast. Solid lines are theoretical numbers (see §2.3.3 and §2.3.4) assuming that block broadcast dominates the overall cost and ignores other steps of the protocol. Dots are experimental results.

To demonstrate the effectiveness of coded broadcast, we vary N and compare the throughput of HotStuff using the original and coded broadcast. Results in Fig. 2-4 show that the throughput of the original HotStuff protocol decreases quickly as we increase N from 4 to 7, closely matching the theoretical calculation in §2.3.3. The

[5] Figures 2-3 and 2-4 are from our earlier variant of Coded HotStuff. In the earlier variant, the leader used an RS$(N, N - f)$ to encode the proposal into N shares. The leader also participated in the second round of coded broadcast to forward its share to all followers.

[6] `https://github.com/yangl1996/nanonet.git`

effectiveness of coded broadcast is weaker than expected in theory (it reduces the throughput when $N = 4$). We conjecture that the difference is due to the higher overhead of concurrently sending many small shares, as well as due to relying on an implementation of HotStuff that is not particularly robust. Nevertheless, coded broadcast improves the throughput by 64% when $N = 9$.

2.3.6 Modified PREPARE Phase

We describe how the Coded HotStuff PREPARE phase works and how it differs from the original HotStuff one. To that end, we use the CodedBcaster module (fully described in §2.4.2) as a black box for the two-round coded broadcast scheme introduced above. For the sake of simplicity, assume v_0 is the leader for this view. Each node initializes a new CodedBcaster instance to use for this view.

The leader v_0 selects $highQC$ the same way as in the original HotStuff PREPARE phase. Let p be the proposal that the leader v_0 is ready to broadcast. The leader v_0 uses [7] CodedBcaster to initiate the broadcast process. The leader accompanies each coded share with the proper identifiers, i.e., the view number and the digest of the parent node. Then all nodes follow the 2-round CodedBcaster protocol in order to receive the proposal p. After receiving the new proposal p, each node uses the CREATELEAF method to extend the tail of $highQC.node$ with p. Then each follower v_i proceeds as if it had received an (original) PREPARE message. That is, each v_i uses the SAFENODE predicate to determine whether to accept p. If it is accepted, the node sends a PREPARE vote with a partial signature for the proposal to the leader.

2.3.7 Security: Safety and Liveness

A BFT consensus algorithm must satisfy *safety* and *liveness* properties.

- **Safety**. No two non-faulty replicas agree differently on a total order for the execution of requests.

[7] In particular, it places p on the Broadcast channel of CodedBcaster. This channel serves to initiate the broadcast process on the leader side.

- **Liveness**: An execution will terminate if its input is correct.

We show that Coded HotStuff satisfies the safety and liveness properties.

Let e_{O} be an execution of the original HotStuff PREPARE phase and let e_{C} be an execution of the Coded HotStuff PREPARE phase. Suppose e_{O} and e_{C} both have the same start state [8], and assume the leader v_0 is honest.

Suppose the leader v_0 is ready to broadcast a new proposal p.

- Under the e_{O} execution, HotStuff [21] guarantees that all honest followers will receive p.

- Under the e_{C} execution, since the leader v_0 is honest, Lemma 2.4.2 guarantees that all honest followers will receive p.

In HotStuff, the SAFENODE predicate evaluates to true as long as either the safety or the liveness rule holds. Quoting HotStuff [21, §4.1], the *safety* rule to accept p is if the branch of $p.node$ extends from the currently locked node $lockedQC.node$. On the other hand, the *liveness* rule is the replica will accept p if $p.justify$ has a higher view than the current $lockedQC$. Notice that since all honest followers receive p under both e_{O} and e_{C}, the objects and attributes referred to above all have the same state under the two executions.

Therefore, for all honest followers, the SAFENODE predicate is True under the e_{C} execution if and only if the SAFENODE predicate is True under the e_{O} execution. Since the remaining phases and processes of Coded HotStuff are unchanged (w.r.t original HotStuff), and since Original HotStuff is proved to satisfy safety and liveness, we conclude that Coded HotStuff satisfies safety and liveness, too.

2.4 Coded Broadcaster

[8] Here we (loosely) define *state* to capture all relevant system properties, such as the current view number, the current *highQC*, the current *tree* of pending commands (locally stored in each replica per [21]), and the *branch* led by a given node.

CodedBcaster is a module that isolates and handles the broadcast functionality in a distributed application/protocol, using erasure codes. As such, it is intended to be run as a internal module by the node participating in the system. For example, we use CodedBcaster as an internal module in each Coded HotStuff instance. We designed and implemented CodedBcaster as a standalone module in Go.[9]

2.4.1 Setup and Assumptions

We assume there are N nodes $v_0, \ldots, v_{N-1}$ running a distributed protocol or application. We assume there is an established, bidirectional TCP/IP authenticated connection channel between all pairs of nodes (v_i, v_j), for $i \neq j$. Further, we assume that each piece of data to be broadcasted takes the form of a block.

Cryptographic Assumptions. We assume the existence of a Public Key Infrastructure and a Digital Signature scheme [12]. Each node v_i has a public key pk_i and a private key sk_i, which can be used for signing and verifying messages. In particular, assume there exist PPT algorithms $\mathsf{Sign}(\cdot)$ and $\mathsf{Verify}(\cdot)$. For any message m, node v_i can produce a signature $\sigma = \mathsf{Sign}(sk_i, m)$. Also, any node v_j can verify whether a signature σ is a valid signature from node v_i on a message m by invoking $Acc(1)/Rej(0) \leftarrow \mathsf{Verify}(pk_i, m, \sigma)$. We assume the standard correctness and EUF-CMA security definitions [12] for this digital signature scheme.

2.4.2 Module Design

The CodedBcaster module has a simple interface which can interact with the application layer of the node. The interface consists of two input channels (Broadcast, Receive) and two output channel (Send, Blocks). CodedBcaster listens on the Broadcast and Receive channels, while it writes to the Send and Blocks channels. Thus, a node using CodedBcaster is expected to write to the Broadcast and Receive channels, as well as listen to the Send and Blocks channels.

[9] `https://github.com/yangl1996/bftmpi.git`

For the sake of simplicity, we assume that each instance of CodedBcaster is used to broadcast a single block of data. Also, we assume that each CodedBcaster instance has a designated broadcaster (leader) node. The application layer using CodedBcaster is responsible for associating CodedBcaster instances with block identifiers, and for achieving consensus for the leader node. When the broadcasting node wishes to broadcast a block b, it should place b on the Broadcast channel, so that CodedBcaster can process it and initiate the broadcasting process.

There are two types of messages that can be sent and received. When initiating a broadcast, a node sends shares by sending type 1 messages of the form $m^1 = (s, \sigma^1)$. Upon receiving a type 1 message, a node forwards the share it received to other nodes by sending type 2 messages of the form $m^2 = ((s, \sigma^1), \sigma^2)$.

When the application layer of node v_i reads a message m from the Send channel, it should send m to the intended recipient v_j. Similarly, upon receiving a message m (of type 1 or 2) from another node v_j, the application layer of v_i should place m on the Receive channel, so that it can be processed by CodedBcaster.

Initialization and Main Loop. Let v_L be the designated broadcasting node for this CodedBcaster instance. CodedBcaster initializes a Reed-Solomon encoder $RS(N-1, N-1-f)$. After initialization, CodedBcaster runs a main loop in the background, *listening* on the two input channels (Broadcast, Receive). Upon receiving a block b on the Broadcast channel, it invokes the handleBroadcast(b) method. Upon receiving a signed coded share message m on the Receive channel, it invokes the handleReceive(m) method. Below are the specifications for these two handler methods.

The handleBroadcast(b) method. Let v_i be the node on which the method is invoked. This means that v_i needs to broadcast block b to all other nodes. CodedBcaster uses the RS encoder to encode b into $(N-1)$ coded shares $\{s_{(i,j)} \mid 0 \leq j \leq N-1, j \neq i\}$, with the property that b can be reconstructed given any $(N-1-f)$ correct shares. In particular, each share $s_{(i,j)}$ contains two fields:

- $s_{(i,j)}.shareID$

- $s_{(i,j)}.shareData$

Now CodedBcaster produces signatures $\sigma^1_{(i,j)} = \mathsf{Sign}(sk_i, s_{(i,j)})$ and creates type 1 messages $m^1_{(i,j)} = (s_{(i,j)}, \sigma^1_{(i,j)})$. Then CodedBcaster places these $m^1_{(i,j)}$ messages on the Send output channel, to be accessed by the application level of v_i.

The handleReceive(m) method. Let v_j be the node on which the method is invoked, and let v_i be the sender of message m. We have two cases:

1. $m = m^1_{(i,j)} = (s_{(i,j)}, \sigma^1_{(i,j)})$ is a type 1 message. First, CodedBcaster checks if $v_i = v_L$ and whether v_i actually signed $s_{(i,j)}$, i.e., checks if $\mathsf{Verify}(pk_i, s_{(i,j)}, \sigma^1_{(i,j)}) = Acc(1)$ (otherwise, node v_j rejects this message). If verification succeeds, this means that node v_j is tasked (by node $v_i = v_L$) to forward the share $s_{(i,j)}$ to all other nodes. CodedBcaster produces a signature $\sigma^2_{(i,j)} = \mathsf{Sign}(sk_j, m^1_{(i,j)})$ and creates $(N-2)$ identical type 2 messages $\{m^2_{(j,k)} \mid 0 \leq k \leq N-1, k \neq i, k \neq j\}$. Each such message is defined as

$$m^2_{(j,k)} := (m^1_{(i,j)}, \sigma^2_{(i,j)}) = ((s_{(i,j)}, \sigma^1_{(i,j)}), \sigma^2_{(i,j)}).$$

   CodedBcaster sends type 2 messages to signal to each recipient v_k that it should not further forward this share. Similarly to earlier, CodedBcaster places these $m^2_{(j,k)}$ shares on the Send output channel, to be accessed by the application level of v_j.

2. $m = m^2_{(i,j)} = (m^1_{(k,i)}, \sigma^2_{(k,i)}) = ((s_{(k,i)}, \sigma^1_{(k,i)}), \sigma^2_{(k,i)})$ is a type 2 message.

   First, CodedBcaster checks if $v_i = v_L$ and whether v_i actually signed $m^1_{(k,i)}$, i.e., checks if $\mathsf{Verify}(pk_i, m^1_{(k,i)}), \sigma^2_{(k,i)}) = Acc(1)$. Second, it checks whether v_k actually signed $s_{(k,i)}$, i.e., checks if $\mathsf{Verify}(pk_k, s_{(k,i)}), \sigma^1_{(k,i)}) = Acc(1)$. If either of the checks fail, v_j rejects this message. If both checks succeed, this means that node v_k sent share $s_{(k,i)}$ to node v_i, which in turn forwarded it to node v_j. Now node v_j is *not* tasked with forwarding $s_{(k,i)}$, but still saves this share in its records.

Then, CodedBcaster checks whether it has already decoded the block. If it has, then it does not proceed further. If not, CodedBcaster adds share $s_{(k,i)}$ to the collection of shares. If it has collected at least $(N-1-f)$ shares (including the newly added share), CodedBcaster attempts to reconstruct the original block. To do so, it invokes the Reconstruct() and Verify() methods, which are assumed to be provided by the $RS(N-1, N-1-f)$ encoder. If reconstruction and verification are successful, let b' be the block reconstructed from the available shares.

Finally, CodedBcaster places the decoded block b' on the Blocks output channel, where it can be accessed by the application layer of v_j.

2.4.3 Security

We formally prove that CodedBcaster is resilient against Byzantine attacks.

Claim 2.4.1. *Suppose node v_i uses* CodedBcaster *to broadcast a block b, and assume v_i is honest. If the* CodedBcaster *module of an honest node v_j accepts a message m (of either type 1 or 2), then m contains a well-formed share.*

Proof. Assume to the contrary that m contains an ill-formed share. Message m can either be a type 1 or type 2 message.

- Case 1: $m = m^1_{(i',j)}$ is a type 1 message from node $v_{i'}$. Since v_j accepted $m^1_{(i',j)}$, we know where $i' = i$. This means v_i signed an ill-formed share and sent it to v_j, which contradicts the assumption that v_i is honest.

- Case 2: $m = m^2_{(k,j)}$ is a type 2 message from node v_k. Let $s^*_{(i',k)}$ be the ill-formed share contained in $m^2_{(k,j)}$. Since v_j accepts, it must be that $\mathsf{Verify}(pk_i, s^*_{(i',k)}), \sigma^1_{(i',k)}) = Acc(1)$, which implies $i' = i$. This means v_i signed the ill-formed share $s^*_{(i,k)}$ and sent it to node v_k. This contradicts the assumption that v_i is honest. [10]

□

[10] Note that we arrive at a contradiction without assuming anything about the honesty of the intermediate node v_k.

Lemma 2.4.2. *Suppose node v_i uses* CodedBcaster *to broadcast a block b. If v_i is honest, then all honest nodes will eventually receive b.*

Proof. Let h be the number of honest nodes (including v_i). Since at most f nodes are Byzantine, we know $h \geq N - f$. Also, by claim 2.4.1, we know that all messages accepted by honest nodes contain well-formed shares.

Since v_i is honest, it must be the designated broadcasting node v_L; it will also properly use the CodedBcaster interface. First, it will place b on the Broadcast channel. Then, v_i will send each well-formed message $m^1_{(i,j)}$ it reads from the Send channel to the corresponding node v_j.

Consider any honest node v_j, for $j \neq i$, which will also properly use the CodedBcaster interface. Node v_j will receive the well-formed message $m^1_{(i,j)}$ from $v_i = v_L$ and will place it on the Receive channel. CodedBcaster will accept $m^1_{(i,j)}$ and will save the well-formed share $s_{(i,j)}$. Thus, it will send each well-formed message $m^2_{(j,k)}$ it reads from the Send channel to the corresponding node v_k.

Honest node v_j will also receive a well-formed message $m^2_{(k,j)}$ from each honest node v_k, for $k \neq j, i$. Node v_j will place each $m^2_{(k,j)}$ on the Receive channel, CodedBcaster will accept it, and it will save the well-formed share $s_{(i,k)}$, for $k \neq j, i$. Thus, the CodedBcaster of honest node v_j will save a total of $h - 1 \geq N - 1 - f$ well-formed shares, and thus will be able to reconstruct the original block b. Finally, v_j will receive the reconstructed block b from the Blocks channel. This is true for all $h - 1$ honest nodes v_j, where $j \neq i$. Since honest node $v_i = v_L$ is trivially considered to have received b, we conclude that all $1 + (h - 1) = h$ honest nodes eventually receive b. □

2.4.4 Evaluation

To evaluate CodedBcaster, we designed and implemented a simple event-driven protocol centered around broadcasting data (blocks). For the sake of simplicity, we built a synchronizer on top of the event-driven protocol, so that it runs in synchronous rounds.

Simple application layer protocol using CodedBcaster In each round, the leader node - determined in a round-robin fashion - uses the CodedBcaster module to broadcast a new block of data. The round ends exactly when all other nodes have successfully received the broadcasted block. Below we describe the protocol in more detail.

In round r:

- Node $v_L = v_{r\%N}$ will serve as the leader for this round.
- The leader v_L initiates the broadcast of a new block by placing it in the Broadcast channel of the CodedBcaster.
- If a node v_i reads a share s from the Send channel of CodedBcaster, it sends s to node v_j where $(i, j) := s.shareID$.
- If a node v_i receives a coded share s from another node v_j, it places s on the Receive channel of CodedBcaster.
- If a node v_i reads a (decoded) block b from the Blocks channel of CodedBcaster, it sends a SAFE message to the leader v_L.
- Once the leader v_L receives $(N-1)$ SAFE messages, it broadcasts an ALLSAFE message to all nodes.
- If a node v_i receives an ALLSAFE message for round r, it proceeds to the next round $r+1$.

2.4.5 Experiments and Results

Similar to the setup for the Coded HotStuff experiments §2.3.5. We use Linux network namespace to create isolated network stacks for each CodedBcaster server process, and `qdisc` to limit the ingress and egress bandwidth of each server to 1Mbps, respectively. We also inject a 100 ms round-trip propagation delay between each pair of nodes. Each block (proposal) is 0.3 MB and is filled with random data. Given the block size and the bandwidth constraints, the expected (theoretical) throughput is 0.33 Mbps (regardless of N).

Each experiment in evaluating CodedBcaster involves selecting a value for N and executing the application layer protocol described above for a duration of 30 rounds. At the conclusion of each experiment, the experimental throughput is determined as the inverse of the average broadcast duration across all rounds. To thoroughly assess the performance of CodedBcaster, we conducted a total of 10 experiments, each with a distinct value for the number of nodes N, ranging from 4 to 46. Figure 2-5 shows that CodedBcaster achieves a broadcast throughput that is constant and independent of N, and (roughly) equal to the theoretically expected value.

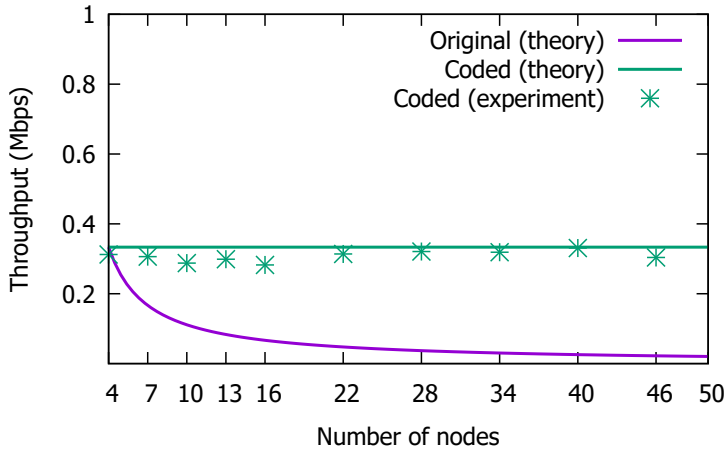


Figure 2-5: Throughput of the application layer running the CodedBcaster protocol. Solid lines are theoretical numbers which are calculated in a similar way as in Coded HotStuff (see §2.3.3 and §2.3.4). Dots are experimental results for CodedBcaster.

Figure 2-6 shows the trace of the egress bandwidth usage of $N = 6$ servers in

a single experiment. Each server actively uses its egress bandwidth throughout the execution, both when proposing blocks as a leader and when forwarding shares as a follower.

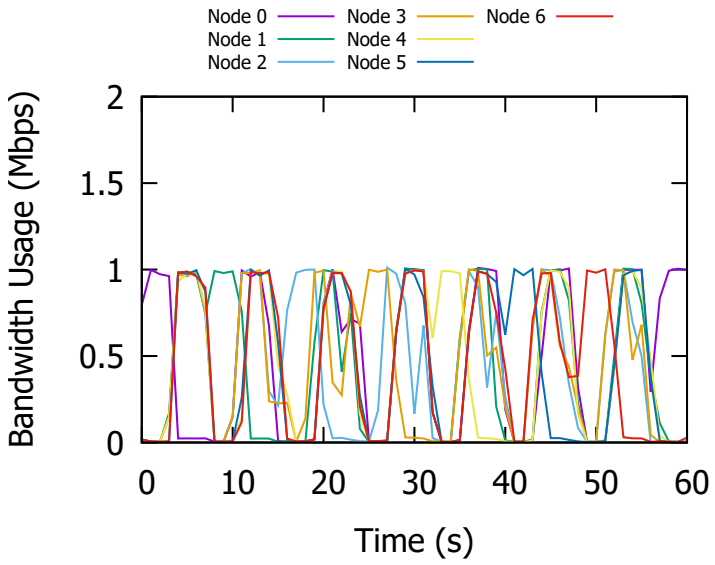


Figure 2-6: Trace of the egress bandwidth usage at each of the six servers running the simple application layer protocol using CodedBcaster. Each server's egress bandwidth usage is balanced, since bandwidth is being consumed both when the server is the leader and when the server forwards shares as a follower.

Chapter 3

Broadcast in a Heterogeneous Node-Constrained Network

3.1 Introduction

Up until now, our analysis has been based on the assumption of equal bandwidth capacities among all servers. However, in real-world scenarios, bandwidth availability can vary across nodes and over time. In such situations, the CodedBcaster solution described in §2.4 may not achieve optimal throughput. In this section, we provide an example of a bandwidth configuration that highlights this issue, along with insights on how to approach this problem.

3.1.1 Motivating Example

Suppose there are $N = 4$ nodes v_0, v_1, v_2, v_3, the block size is 60 Mbits, and the leader is v_0. Suppose the ingress and egress capacities (measured in Mbps) of each node are given by the arrays

$$in = [10, 10, 10, 10]$$
$$e = [10, 20, 4, 4].$$

For simplicity, assume all nodes are honest. We explain why CodedBcaster produces a sub-optimal solution in this simple setting. Since all nodes are honest, we can use a simpler variation of CodedBcaster which just splits the proposal into $N-1$ shares without any redundancy.

In the first round of this simpler CodedBcaster, the leader v_0 sends a unique share of size $\frac{60}{3} = 20$ Mbits to each of v_1, v_2, and v_3. The first round takes $\frac{3 \cdot 20}{10} = 6$ seconds. In the second round, each follower broadcasts its share. The second round takes $\frac{2 \cdot 20}{4} = 10$ seconds. The total duration is $6 + 10 = 16$ seconds. With pipelining of the two rounds, the throughput is $\frac{1}{10}$ blocks per second.

Instead of using CodedBcaster, the leader can instead send the entire block to node v_1, which in turn can forward it to nodes v_2 and v_3[1]. This simple solution takes a total of $6 + 6 = 12$ seconds. With pipelining of the two rounds, the throughput is $\frac{1}{6}$ blocks per second, which faster than the throughput achieved by CodedBcaster. In fact, Theorem 3.4.1 in §3.4 shows that the $\frac{1}{6}$ throughput rate is optimal.

In essence, the CodedBcaster approach falls short of achieving optimality because it distributes the same load to all follower nodes. When there are variations in the ingress and egress capacities across nodes, the throughput rate of CodedBcaster is negatively affected by nodes with lower capacities.

It is important to note that the provided example serves as a motivation and does not encompass all possible scenarios of varying ingress/egress capacities among nodes. Nevertheless, it offers valuable insights on how to approach the heterogeneous node-constrained network setting. Based on the simple solution presented above, it seems reasonable that the leader should allocate a load [2] to each follower in proportion to their available bandwidth resources (both ingress and egress).

3.1.2 Focusing on Sending Rates

As shown in the example above, erasure codes with fixed code rates, such as Reed-Solomon codes [17], are not well-suited for the variable bandwidth setting. However,

[1] Note that this simple solution only works if node v_1 is honest.

[2] The term *load* can e.g. refer to the number of shares.

an alternative solution lies in utilizing *rateless* erasure codes. Rateless erasure codes, also known as fountain codes, offer flexibility and efficiency in error correction. They generate an infinite stream of independent encoded packets from the source data, enabling reliable data recovery in unreliable communication or storage systems. Notably, rateless erasure codes can adapt to changing conditions without prior knowledge of errors or lost packets, with the Luby Transform [14] (LT) code being a popular example.

In the context of broadcasting a proposal, the leader can encode it into LT coded shares and distribute these shares to the followers, who further forward them to one another. Each follower receives unique coded shares from the leader, and any sufficiently large subset of these shares can be used to reconstruct the original proposal. Consequently, the leader's responsibility shifts to determining the quantity of shares to send to each node, rather than selecting specific shares. Assuming sufficiently large blocks and small shares, the leader's task boils down to deciding *the rate at which it should send shares to each follower node*. This problem forms the focal point of the second part of this thesis.

3.1.3 Focusing on a Non-BFT Setting

The motivating example clearly demonstrates the complexity of finding an optimal leader-to-follower rate in a BFT protocol. The suggested simple solution implies that if the leader possesses knowledge of each node's ingress/egress resources (which it does not), it can allocate rates to followers proportionate to their bandwidth capacities. However, such an approach is vulnerable to Byzantine attacks. In the given example, if v_1 behaves in a Byzantine manner and refuses to forward the received data to v_2 and v_3, the broadcast operation fails.

To gain a deeper understanding of and to address this intricate problem, it is reasonable to first address the simplified version that assumes the *absence of Byzantine faults*. To the best of our knowledge, existing literature does not offer a comprehensive solution to either version of the problem (i.e., with or without Byzantine faults).

3.1.4 Maximizing the Minimum Receiving Rate

Our focus lies in investigating the decision-making process of the leader for determining sending rates to each follower, i.e., for allocating its egress bandwidth. Additionally, we aim to examine how each follower node, in turn, can decide on appropriate sending rates to each of its peers. The ultimate goal is to optimize a specific "system objective" that pertains to the overall performance of the system. This prompts the question of how to precisely define this system objective.

A scalable leader-based broadcast solution is motivated by its potential integration within a distributed protocol. This protocol typically consists of several phases which rely on the efficient operation of the broadcast component (such as in the case of HotStuff [21]). In most cases, the system can only proceed to subsequent phases once the broadcast is complete, ensuring that all nodes have received the data. Consequently, the node with the slowest receiving rate becomes the bottleneck for both the broadcast component and the entire system. Thus, a natural objective is to *maximize the minimum receiving rate among all follower nodes* [3]. We formally define this objective in §3.2 as part of our problem statement.

In §3.4 and §3.5, we provide a comprehensive theoretical analysis of the optimization problem defined in §3.2. In §3.6 and §3.7, we present MaxMin-RC, a simple MaxMin rate controller protocol, which aims to optimize broadcast throughput in non-BFT settings, achieving close alignment with the optimal throughput rate. Finally, in §3.8 and §3.9, we present simulation results which demonstrate that MaxMin-RC converges to a rate allocation scheme which yields a close-to-optimal throughput rate.

[3] This problem statement closely captures the intended use case described in §3.1. The leader can use LT coded shares to encode the data it wishes to broadcast. These shares are small in size (relative to the bandwidth capacities), and are all equally useful in decoding the original data.

3.2 Problem Statement

The latter part of this thesis delves into the problem of scalable leader-based broadcast in a *heterogeneous node-constrained network*, assuming the *absence of Byzantine faults*. Below we carefully articulate the problem statement that we aim to address.

Let $v_0, v_1, \ldots, v_{N-1}$ be the N nodes in the network. We assume there is a bidirectional communication link between each pair of nodes v_i, v_j. Let $\mathsf{in} = [\mathsf{in}_i]$ and $\mathsf{e} = [\mathsf{e}_i]$ be the arrays holding the ingress and egress capacities of all N nodes, respectively.

Revised Security Model. For the scope of this work, we assume that all nodes are honest, i.e., there are no Byzantine nodes ($f = 0$).

The leader v_0 wishes to broadcast data to all other nodes. Concretely, we assume the leader v_0 has access to an *infinite stream* of small, equally sized messages, which are all useful for the broadcasting process.

Both the leader and follower nodes possess the autonomy to determine the sending rate to each of their respective peers. However, certain constraints must be upheld. Specifically, the sum of ingoing rates for any given node must not surpass its ingress capacity, and the sum of outgoing rates must not exceed its egress capacity. Additionally, we establish the stipulation that *the leader exclusively serves as the source of data*[4]. This implies that a follower node cannot transmit data to any peer at a higher rate than the rate at which it receives data from the leader. We now formalize these statements.

Definition 3.2.1. *A* rate allocation scheme $S = \{r(i,j)\}$ *is a collection of peer-to-peer sending rates, where* $r(i,j)$ *is the rate at which node* v_i *is sending to node* v_j*.*

Let $S = \{r(i,j)\}$ be a rate allocation scheme. For all i, we define $r_{\mathsf{out}}(i)$ to be the total egress rate used by node v_i, and $r_{\mathsf{in}}(i)$ to be the total ingress rate used by node v_i. More precisely,

$$r_{\mathsf{out}}(i) := \sum_{j=0, j\neq i}^{N-1} r(i,j) \tag{3.1}$$

[4]In particular, we do not allow the use of network coding.

and

$$r_{\mathsf{in}}(i) := \sum_{j=0, j\neq i}^{N-1} r(j,i). \tag{3.2}$$

Now, we define the criteria for a rate allocation scheme to be *valid*.

Definition 3.2.2. *Let $S = \{r(i,j)\}$ be a rate allocation scheme. We say S is* valid *if it satisfies the ingress and egress capacity of each node, and if no follower transmits data to any peer at a higher rate than they receive from the leader. That is, for all i,*

$$\begin{aligned} r_{\mathsf{out}}(i) &\leq \mathsf{e}_i, \\ r_{\mathsf{in}}(i) &\leq \mathsf{in}_i, \quad \textit{and} \\ r(i,j) &\leq r(0,i) \quad \textit{for all } j \neq i. \end{aligned}$$

Let $\mathcal{S}$ be the set of all valid rate allocation schemes with respect to the ingress and egress capacities in_i and e_i.

Given a valid rate allocation scheme $S = \{r(i,j)\}$, let

$$r_{\mathsf{in}}^{\mathsf{min}}(S) := \min_i \ r_{\mathsf{in}}(i)$$

denote the minimum total receiving rate across all follower nodes, under S.

Next, we define r_{OPT} to be the maximum $r_{\mathsf{in}}^{\mathsf{min}}$ rate that can be achieved across all possible rate allocations. More formally, define

$$r_{\mathsf{OPT}} := \max_{S\in\mathcal{S}} \ r_{\mathsf{in}}^{\mathsf{min}}(S) \quad = \quad \max_{S\in\mathcal{S}} \ \min_i \sum_{j=0, j\neq i}^{N-1} r(j,i) \tag{3.3}$$

Note that r_{OPT} is a quantity defined with respect to a network with specific ingress and egress capacities.

3.3 Related Work

The "Network Information Flow" paper [1] addresses the problem of efficiently transmitting information over a network by introducing the concept of network coding. Traditional network communication protocols typically involve routing packets through the network without altering their content. However, the authors propose a different approach where intermediate nodes in the network can combine or code the incoming packets before forwarding them to their destinations. By allowing intermediate nodes to encode packets, the network can take advantage of the inherent redundancy in the transmitted information. This enables more efficient use of network resources and increases the overall throughput.

The authors in [1] establish fundamental theoretical results related to network coding. They show that in certain scenarios, network coding can achieve the maximum possible throughput and information dissemination rate. They also prove the capacity theorem for multicast networks, demonstrating the benefit of network coding for multicast communication.

Strokkur [19] is a preliminary version of a transaction broadcasting scheme based on LT codes [14]. LT coding produces codewords by taking a random subset of transactions and XORing them. Strokkur [19] is a novel transaction broadcasting scheme that, contrary to existing approaches, avoids any explicit coordination among peers to decide whether to forward individual transactions. In Strokkur, nodes encode new transactions they have received into codewords that they forward to their peers, who in turn use them to decode the original transactions.

The research project "Avalanche" [8] aims to revolutionize file distribution by providing a cost-effective, highly scalable, and remarkably fast solution for applications like TV on-demand, software distribution, and patches. It addresses the limitations of traditional swarming techniques used in Peer-Assisted file delivery systems, which struggle with optimal scheduling of file pieces as the number of receivers increases. Avalanche introduces network coding, where peers generate linear combinations of blocks they possess, allowing any uploaded piece to be valuable to other peers. This

approach ensures robustness, eliminates bottlenecks, and optimizes network bandwidth utilization. Notably, Avalanche achieves these benefits without relying on knowledge of network topology or available bandwidth, simplifying system design.

The literature review reveals that some existing works, such as [1], tackle the problem using network coding, but their proposed scheme can be challenging to implement and intrusive to the application layer. In contrast, our theoretical contribution demonstrates that the optimal rate achievable through network coding can be attained without the need for network coding itself. In comparison, Avalanche [8] does not specifically aim to compare its solution with the network coding optimal solution. Furthermore, their generic peer-to-peer network model differs from our assumption of a full mesh network configuration. Finally, Strokkur [19] primarily focuses on minimizing redundancy associated with LT codes, whereas our objective is centered around maximizing the overall system throughput.

The paper "Asymptotically Optimal Message Dissemination with Applications to Blockchains" [13] introduces novel flooding protocols that are proven to be asymptotically optimal in terms of communication complexity and per-party communication for sufficiently long messages. This work, similar to our research ([11]), achieves optimal throughput under equal bandwidth constraints among nodes. However, the protocols presented in [13] do not specifically tackle the problem of optimizing throughput in a heterogeneous node-constrained network, which constitutes the primary focus of the latter part of this thesis.

3.4 Optimal Broadcast Throughput in a Node-Constrained Network

In this section, we prove that the optimal throughput rate r_{OPT} can be expressed as a simple function of the ingress and egress capacities of the nodes. Notably, our proof is constructive, i.e., we describe how to construct a rate allocation scheme that achieves the r_{OPT} throughput rate. This section embodies the core theoretical contribution of

this thesis.

Let $\mathsf{in} = [\mathsf{in}_i]$ and $\mathsf{e} = [\mathsf{e}_i]$ be the arrays holding the ingress and egress capacities of all N nodes, respectively. That is, node v_i has ingress in_i and egress e_i. Recall that v_0 is the leader node. Also recall the definition of r_{OPT} from equation 3.3.

Let $\mathsf{in}_{\mathsf{min}}$ be the minimum ingress capacity value among all follower nodes, where ties are broken arbitrarily, i.e., define

$$\mathsf{in}_{\mathsf{min}} := \min_{i \geq 1} \{\mathsf{in}_i\}$$

Also, define $\mathsf{e}_{\mathsf{crit}}$ to be the average egress capacity of the system, normalized by the number of follower nodes, i.e.,

$$\mathsf{e}_{\mathsf{crit}} := \frac{1}{N-1} \sum_{i=0}^{N-1} \mathsf{e}_i.$$

Last, define $\mathsf{e}'_{\mathsf{crit}}$ to be the average egress capacity of all follower nodes, normalized by the number of follower nodes minus one, i.e.,

$$\mathsf{e}'_{\mathsf{crit}} := \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2}.$$

Theorem 3.4.1. *Define r_{OPT}, $\mathsf{in}_{\mathsf{min}}$, $\mathsf{e}_{\mathsf{crit}}$, and $\mathsf{e}'_{\mathsf{crit}}$ as above. Then*

$$r_{\mathsf{OPT}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}. \tag{3.4}$$

Proof. Follows by Theorem 3.4.2 and Theorem 3.4.3. □

Theorem 3.4.2. *Define r_{OPT}, $\mathsf{in}_{\mathsf{min}}$, $\mathsf{e}_{\mathsf{crit}}$, and $\mathsf{e}'_{\mathsf{crit}}$ as above. Then*

$$r_{\mathsf{OPT}} \leq \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}. \tag{3.5}$$

Proof. Let $S = \{r(i,j)\}$ be a rate allocation scheme which achieves r_{OPT}. We know $r_{\mathsf{in}}(i) \le \mathsf{in}_i$ and $r_{\mathsf{in}}(i) \le \mathsf{e}_0$ for all i. It follows that $r_{\mathsf{OPT}} \le \mathsf{in}_{\mathsf{min}}$ and $r_{\mathsf{OPT}} \le \mathsf{e}_0$. We only need to show that $r_{\mathsf{OPT}} \le \mathsf{e}_{\mathsf{crit}}$. By claim A.0.3 we have $\sum_{i=1}^{N-1} r_{\mathsf{in}}(i) \le \sum_{i=0}^{N-1} \mathsf{e}_i$. Also, by definition of r_{OPT}, we have $\sum_{i=1}^{N-1} r_{\mathsf{in}}(i) \ge \sum_{i=1}^{N-1} r_{\mathsf{OPT}} = (N-1) r_{\mathsf{OPT}}$. Combining these two, we get

$$(N-1) \cdot r_{\mathsf{OPT}} \le \sum_{j=0}^{N-1} \mathsf{e}_j.$$

Thus $r_{\mathsf{OPT}} \le \frac{1}{N-1} \sum_{j=0}^{N-1} \mathsf{e}_j = \mathsf{e}_{\mathsf{crit}}$, which completes the proof. □

Theorem 3.4.3. *Define* r_{OPT}, $\mathsf{in}_{\mathsf{min}}$, $\mathsf{e}_{\mathsf{crit}}$, *and* $\mathsf{e}'_{\mathsf{crit}}$ *as above. Then*

$$r_{\mathsf{OPT}} \ge \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}. \tag{3.6}$$

Proof. We consider two cases:

1. $\mathsf{e}'_{\mathsf{crit}} \le \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$.

   Then by Lemma 3.4.4, there exists a rate allocation which gives $r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \ge 1$. Also, by Claim A.0.1, we have $\mathsf{e}_0 \ge \mathsf{e}_{\mathsf{crit}} \ge \mathsf{e}'_{\mathsf{crit}}$.

   - If $\mathsf{in}_{\mathsf{min}} \le \mathsf{e}_{\mathsf{crit}}$, then $\mathsf{e}_0 \ge \mathsf{e}_{\mathsf{crit}} \ge \mathsf{in}_{\mathsf{min}}$ and thus $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \ge 1$.
   - Otherwise (if $\mathsf{e}_{\mathsf{crit}} < \mathsf{in}_{\mathsf{min}}$), and since we know $\mathsf{e}_0 \ge \mathsf{e}_{\mathsf{crit}}$, we have $r_{\mathsf{in}}(i) = \mathsf{e}_{\mathsf{crit}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \ge 1$.

2. $\mathsf{e}'_{\mathsf{crit}} > \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$.

   Then by Lemma 3.4.5, there exists a rate allocation which gives $r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$ for all $i \ge 1$.

   - If $\mathsf{in}_{\mathsf{min}} \le \mathsf{e}_0$, then $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \ge 1$.
   - Otherwise (if $\mathsf{e}_0 < \mathsf{in}_{\mathsf{min}}$), we have $r_{\mathsf{in}}(i) = \mathsf{e}_0 = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \ge 1$.

In all cases, there exists a rate allocation which gives $r_{\mathsf{in}}(i) = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \geq 1$. This implies that $r_{\mathsf{OPT}} \geq \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$, as desired. □

Below, we present Lemma 3.4.4 and Lemma 3.4.5, which play a crucial role in proving Theorem 3.4.3. The proofs of these lemmas can be found in Appendix §A.

Lemma 3.4.4. *Assume that* $\mathsf{e}'_{\mathsf{crit}} \leq \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$. *Suppose that the leader* v_0 *sends to each follower* v_i *at a rate*

$$r(0,i) = \frac{\mathsf{e}_i}{(N-2)} + \min\left\{\frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)}, \quad \mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}\right\}.$$

Also suppose that each follower v_i *sends to each other follower* v_j *at a rate*

$$r(i,j) = \frac{\mathsf{e}_i}{(N-2)}.$$

Then, for all $i \geq 1$,

$$r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}.$$

Lemma 3.4.5. *Assume that* $\mathsf{e}'_{\mathsf{crit}} \geq \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$. *Suppose that the leader* v_0 *sends to each follower* v_i *at a rate*

$$r(0,i) = \frac{\min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \frac{\mathsf{e}_i}{(N-2)}$$

Also suppose that each follower v_i *sends to each other follower* v_j *at a rate*

$$r(i,j) = r(0,i).$$

Then, for all $i \geq 1$,

$$r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}.$$

3.5 LP Formulation and Network Coding

In this section, we present the Linear Program (LP) formulation of our optimization problem, which aims to maximize the minimum receiving rate. Contrary to the assumptions made in §3.2, the LP optimization problem allows the use of *network coding*. We provide a proof demonstrating that the optimal rate obtained using network coding is in fact equal to the optimal rate (r_{OPT}) defined in §3.2. This implies that the utilization of network coding does not contribute to an improved throughput rate.

3.5.1 LP Formulation

Let s be the source, and $t_1, \ldots, t_L$ be the receivers. Let the network topology be a directed graph (V, E). Let $\delta^+(v)$ and $\delta^-(v)$ be the set of edges from and to node v, respectively.

Edge-capacity model. Each edge is given a bandwidth capacity. Multicast capacity h is

$$h = \min_{1 \le i \le L} \text{maxflow}\,(s \to t_i) \tag{3.7}$$

where maxflow $(s \to t_i)$ is the max-flow (equivalently the min-cut) from s to t_i.

Node-capacity model. Each node $v \in V$ is given an ingress capacity i_v and an egress capacity e_v. We can convert node constraints to edge constraints by assigning a capacity to each edge. Let $c(e)$ be the capacity assigned to edge e. Multicast capacity h is

$$h = \max_{c \text{ sat. node cap.}} \min_{1 \le i \le N-1} \text{maxflow}\,(s \to t_i) \tag{3.8}$$

where the max-flows are computed based on the edge constraints c, which must satisfy the node capacity constraints

$$\sum_{e \in \delta^+(v)} c(e) \leq e_v \text{ and } \sum_{e \in \delta^-(v)} c(e) \leq i_v$$

for $v \in V$.

To compute h in the node-capacity model, let $f^i(e)$ be the rate on edge e in the network flow $s \to t_i$, for $1 \leq i \leq L$ and $e \in E$. Here f^i must satisfy standard network flow constraints, namely

$$0 \leq f^i(e) \leq c(e)$$

for $e \in E$, and

$$\sum_{e \in \delta^+(v)} f^i(e) = \sum_{e \in \delta^-(v)} f^i(e)$$

for $v \in V \setminus \{s, t_i\}$.

Note that h is not larger than the value of each individual flow, i.e.,

$$h \leq \sum_{e \in \delta^+(s)} f^i(e)$$

for $1 \leq i \leq L$. Note that all the aforementioned constraints are linear constraints, and the optimization problem for h is a linear program where the objective function is h itself.

3.5.2 Connection With Network Coding

Let r_{LP} represent the optimal throughput rate achievable through network coding, denoted as h in the LP formulation mentioned above, which corresponds to the multicast capacity in the node-capacity model. In this section, we prove that $r_{\mathsf{LP}} = r_{\mathsf{OPT}}$, which implies that network coding is not needed to achieve the optimal throughput.

Since the construction in Theorem 3.4.3 is achieved without coding, we have $r_{\mathsf{LP}} \geq r_{\mathsf{OPT}}$. We now prove that $r_{\mathsf{LP}} \leq r_{\mathsf{OPT}}$.

Theorem 3.5.1. *Define* r_{LP}, r_{OPT}, $\mathsf{in}_{\mathsf{min}}$, $\mathsf{e}_{\mathsf{crit}}$, *and* $\mathsf{e}'_{\mathsf{crit}}$ *as before. Then*

$$r_{\mathsf{LP}} \leq r_{\mathsf{OPT}}.$$

Proof. Consider the assignment that achieves the r_{LP} rate, and let $\mathsf{edgecap}(i,j)$ denote the edge capacity from v_i to v_j under this assignment, for all $i \neq j$.

We know that r_{LP} must satisfy $r_{\mathsf{LP}} \leq \mathsf{in}_{\mathsf{min}}$ and $r_{\mathsf{LP}} \leq \mathsf{e}_0$. By Theorem 3.4.1, we know $r_{\mathsf{OPT}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$. To complete the proof, we need to show that $r_{\mathsf{LP}} \leq \mathsf{e}_{\mathsf{crit}}$. By the dual definition of r_{LP}, we have:

$$r_{\mathsf{LP}} = \max_{c \text{ sat. node cap.}} \min_{1 \leq i \leq N-1} \text{mincut}(v_0, v_i).$$

From this definition, we know that $r_{\mathsf{LP}} \leq \text{mincut}(v_0, v_i)$ for all $i \geq 1$, Note that $\text{mincut}(v_0, v_i)$ is defined under the $\mathsf{edgecap}(i,j)$ rates. Now, let's define $sc^*(v_0, v_i) := \sum_{j=0, j\neq i}^{N-1} \mathsf{edgecap}(j,i)$ for all $i \geq 1$. We assume We can then derive the following inequalities:

$$\begin{aligned}
\sum_{i=1}^{N-1} r_{\mathsf{LP}} &\leq \sum_{i=1}^{N-1} \text{mincut}\,(v_0, v_i) \\
&\leq \sum_{i=1}^{N-1} sc^*\,(v_0, v_i) \\
&= \sum_{i=1}^{N-1} \sum_{j=0, j\neq i}^{N-1} \mathsf{edgecap}(j,i) \\
&= \sum_{i=0}^{N-1} \sum_{j=1, j\neq i}^{N-1} \mathsf{edgecap}(i,j) \\
&\leq \sum_{i=0}^{N-1} \mathsf{e}_i.
\end{aligned}$$

Therefore, we have $r_{\mathsf{LP}} \leq \frac{1}{N-1} \sum_{i=0}^{N-1} \mathsf{e}_i = \mathsf{e}_{\mathsf{crit}}$, which completes the proof. □

We have proved that $r_{\mathsf{LP}} = r_{\mathsf{OPT}} = \min\{\mathsf{e}_0, \mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$. This theoretical result allows us to conclude that network coding is *not* needed to achieve the optimal multicast

throughput.

3.6 MaxMin-RC: A Simple Rate Controller Protocol

The previous sections (§3.4 and §3.5) provide a formal treatment of the optimization problem defined in §3.2, from a *global* perspective. However, the optimal rate allocation constructed by Theorem 3.4.3 cannot be applied in practice, since it requires *global* knowledge of all nodes' ingress and egress capacities. Equipped with this theoretical analysis, we are now interested in a *distributed* protocol which can achieve or at least approximate the r_{OPT} optimal throughput rate.

In this section, we present MaxMin-RC, a distributed broadcast protocol relying on a simple, feedback-based MaxMin rate controller to decide on sending rates to each peer. MaxMin-RC aims to achieve a rate allocation whose throughput is closely aligned with the globally optimal throughput rate r_{OPT}.

The MaxMin Rate Controller (MaxMin-RC) scheme is a two-hop solution that allows the leader to efficiently broadcast MaxMin messages in the network. The term "MaxMin message" is used in our system design to abstract the type of messages transmitted, such as the LT coded shares discussed in §3.1.2. This two-hop scheme involves the leader sending a MaxMin message to one follower node (called intermediate node), which in turn forwards it to other follower nodes (called destination nodes). The leader runs MaxMin-RC to decide on sending rates to each intermediate node. Intermediate nodes run a simple AIMD [10] rate controller to decide on sending rates to destination nodes. Whenever a follower node receives a MaxMin message (either as an intermediate or as a destination node), it sends an ACK to the leader. The leader uses these ACKs to update the utility scores of the intermediate nodes. These utility scores reflect the contribution of each follower node as an intermediate, and the leader uses them to determine the new sending rates to each intermediate node. In §3.7, we provide a comprehensive and detailed description of the protocol.

3.7 System Design

3.7.1 Setup and Assumptions

Let $v_0, v_1, \ldots, v_{N-1}$ be the N nodes (aka servers) in the network. We assume there is a bidirectional communication link between each pair of nodes v_i, v_j. As specified in the problem statement(§3.2), we assume all nodes are honest. For the sake of simplicity, assume that node v_0 is the leader and nodes $v_1, \ldots, v_{N-1}$ are the followers.

3.7.2 Initialization

Initially, all pairs of nodes participate in a handshake protocol to identify their peers. The leader initializes the utility scores of all followers: $U(v_i) = 1$ for all $i \geq 1$. Each follower v_i initializes all its sending rates: $r_{(i,j)} = 1$ for all $j \neq i$. The leader v_0 starts sending MaxMin messages at a rate of $|m|/\mathsf{e}_0$, where $|m|$ is the size of a MaxMin message.

3.7.3 Leader Schedule

Let $U(v_i)$ be the (nominal) utility score assigned by the leader v_0 to node v_i, for all $i \neq 0$. The $U(\cdot)$ utility scores assigned by the leader reflect the contribution of each node – as an intermediate – towards the main objective of this protocol, i.e., towards maximizing the system throughput. Since the $U(\cdot)$ utility scores can only increase by design, it makes sense for the leader to compare them relative to one another.

The leader will create a short-term schedule which will determine which nodes will serve as intermediates for the next $s = 100$ MaxMin messages. The main idea is that the leader should send to each follower node at a rate proportional to its utility score. We now elaborate on how the leader creates this short-term schedule.

For the next $s = 100$ MaxMin messages $m_0, m_1, \ldots, m_{99}$, the leader maps each message m_i to an intermediate node as follows. For each follower node v, let $U_{\mathsf{norm}}(v)$ be the normalized utility score of node v, rounded to the nearest 100dth. The leader

will send approximately $100 \cdot U_{\mathsf{norm}}(v)$ MaxMin messages to node v. Further, the leader's schedule will consist of *interleaving assignments*, so that each node serves as an intermediate at equally distanced intervals. In particular, node v will serve as the intermediate for the subset of messages $I_v = \{m_{i^*+k\cdot\Delta_v} \mid 0 \leq k \leq s/\Delta_v\}$, where i^* is the index of the first message sent to v, and $\Delta_v = 1/U_{\mathsf{norm}}(v)$ is the interval length between two consecutive assignments to node v.

3.7.4 Receiving a MaxMin Message

Suppose node v_i receives a MaxMin message m from node v_k, where $1 \leq i \leq N-1$ and $k \neq i$. First, node v_i sends an ACK of the form $\mathsf{ack}(m) = (m.\mathsf{id}, \mathsf{hash}(m))$ to node v_k. Now there are two cases. If $k = 0$, i.e., if the sender v_k is the leader node, then we say v_i is an *intermediate* node for message m and will serve as such. Otherwise, we say v_i is a *destination* node for message m.

As an Intermediate Node. If v_i serves as an intermediate node, it is tasked with broadcasting m to all other follower nodes.

Recall that v_i uses an AIMD controller to determine the rate $r_{(i,j)}$ at which it should send to each $v_j \neq v_i$. These $r_{(i,j)}$ rates are defined with respect to the messages for which v_i serves as an intermediate node. That is, if the leader sends MaxMin messages to node v_i at a rate of $r_{(0,i)}$, then v_i is effectively sending MaxMin messages to v_j at a rate of $r_{(0,i)} \cdot r_{(i,j)}$. To enforce the $r_{(i,j)}$ rate, node v_i keeps track of a *token bucket* associated with v_j, referred to as $v_i.\mathsf{tokenBucket}(v_j)$. For each follower node v_j, node v_i increments $v_i.\mathsf{tokenBucket}(v_j)$ by $r_{(i,j)}$. If $v_i.\mathsf{tokenBucket}(v_j) \geq 1$, then node v_i forwards m to node v_j and decrements $v_i.\mathsf{tokenBucket}(v_j)$ by 1. This approach guarantees that node v_i transmits MaxMin messages to node v_j at a rate of $r_{(i,j)}$, and enforces an interval of $1/r_{(i,j)}$ between two consecutive MaxMin messages sent to node v_j.

As a Destination Node. If v_i is a destination node, it is not tasked with broadcasting m; in this case, node v_k must be serving as the intermediate node. Instead,

node v_i is tasked with sending an ACK of the form $\mathsf{ack}(m) = (m.\mathsf{id}, \mathsf{hash}(m))$ to the leader node v_0. Note that v_i sends an ACK to the leader in *addition* to sending an ACK to the intermediate node v_k.

As will become clearer, ACKs from destination nodes crucially help the leader assess the contribution (utility) of intermediate nodes, and thus determine the rate at which it should send MaxMin messages to each.

3.7.5 Receiving an ACK for a MaxMin Message

Suppose node v_i receives an ACK for MaxMin message m of the form $\mathsf{ack}(m) = (m.\mathsf{id}, \mathsf{hash}(m))$ from node v_k, where $1 \leq k \leq N-1$ and $k \neq i$. Now there are two cases. If $i = 0$, i.e., if node $v_i = v_0$ is the leader node, then v_0 will use this ACK to (potentially) update the utility score of the intermediate node associated with m. Otherwise, i.e., if v_i is a follower (intermediate) node, it will use this ACK to inform the AIMD controller which in turn updates the sending rate $r_{(i,k)}$ of node v_i to node v_k.

As the Leader Node. If $v_i = v_0$ is the leader node, and if $m.\mathsf{timeout}$ has not elapsed, then v_0 records that node v_k has successfully received message m. Note that node v_k could either be the intermediate node for m or a destination node, since both intermediate and destination nodes are tasked with sending an ACK to the leader. In any case, let $v_j := m.\mathsf{intermNode}$ be the intermediate node tasked with broadcasting message m. Also, let $v_l := \mathsf{minRxNode}$ be the node which has received the least MaxMin messages out of the last k MaxMin messages sent out. If $v_k = v_l$, i.e., if the ACK was sent by the current $\mathsf{minRxNode}$, then the leader v_0 increments $v_j.\mathsf{utilScore}$ by 1.

As an Intermediate Node. If v_i is a follower, then it must be the intermediate node for message m. If $m.\mathsf{timeout}$ has not elapsed, then v_i records that (destination node) v_k has successfully received message m. Now v_i will apply the simple AIMD update rule to update the rate $r_{(i,k)}$. Let m' be the last MaxMin message which node

v_i forwarded to node v_k. Note that m' might not be the same as the last message for which v_i served as an intermediate node, since $r_{(i,k)}$ is not necessarily equal to 1.

- If v_i has recorded that it received a timely ACK for m' from v_k, then it applies an additive increase to $r_{(i,k)}$. Per the AIMD logic, since v_k has successfully ACKEed two consecutive MaxMin messages, the link $v_i \rightarrow v_k$ can be utilized at a higher rate.

- If v_i has not recorded any timely ACK for m' from v_k, then it applies an multiplicative decrease to $r_{(i,k)}$. Per the AIMD logic, since there is a gap in the messages ACKed by v_k, the link $v_i \rightarrow v_k$ is oversubscribed and should be utilized at a lower rate. Note that this could be either because node v_i's egress capacity is oversubscribed, or because node v_k's ingress capacity is oversubscribed, or both.

Allowing for the MD update to take effect. Let m^* the first message that node v_i forwarded to v_k after the most recent multiplicative decrease (MD) update to rate $r_{(i,k)}$. In order to allow for that MD update to have an effect, node v_i will not proceed to any further MD updates to $r_{(i,k)}$ until it receives an ACK for m^*, or until m^*.timeout has elapsed.

Having explained this rule, let us revisit the second case above, in which v_i has not recorded any timely ACK for m' from v_k. If v_i has received an ACK for m^*, or if m^*.timeout has elapsed, then v_i proceeds to apply a new MD update to $r_{(i,k)}$. Otherwise, v_i leaves $r_{(i,k)}$ unchanged.

3.8 Protocol Simulation

To evaluate our MaxMin-RC protocol in practice, we used the OMNeT++ discrete-event simulation framework[5].

[5] `https://omnetpp.org`

3.8.1 Simulating the MaxMin Rate Controller in OMNeT++

OMNeT++ is a simulation environment that is widely used in the research community for building discrete event simulations of various types of networks and systems. It provides a modular, event-driven approach to simulation modeling, allowing researchers to define network topologies, node behaviors, message formats, and event scheduling.

Using OMNeT++, researchers can program these models in C++ or other supported programming languages and execute and analyze the simulations using the simulation environment. The simulation environment offers several features, such as a graphical user interface for model creation and visualization, a powerful result recording and analysis system, and support for parallel simulation execution.

We used C++ to implement the MaxMin Rate Controller as a standalone "RcNode" module in the OMNeT++ environment [6]. An RcNode instance is equipped with all attributes and methods necessary to run the MaxMin Rate Controller protocol, which is fully described in §3.7. We also defined and created the "RcNet" network. RcNet is a network module which consists of N RcNode instances and all-to-all connection channels among them.

Simulating Node Bandwidth Capacities Using Queues. OMNeT++ offers parameters such as the "data rate" parameter for its channel object. However, the OMNeT++ built-in data rate property is a per link rate limit; we are instead interested in per node capacity limits. We used C++ queues to simulate such network bottlenecks. Every message going out of/into a node must traverse though one such queue. The node simply takes the incoming messages, appends them to a local queue (simulating the network buffer), and drops excessive packets when the queue becomes full.

[6] `https://github.com/ikaklamanis/rc-simple-sim.git`

3.8.2 Simulation Experiments: Setup and Design

For each simulation, we initialized and used a new RcNet network instance consisting of $N = 4$ RcNode instances v_0, v_1, v_2, v_3. RcNode v_0 is the leader node in all simulations. Each MaxMin message has unit size; this means that a node which has an ingress (resp. egress) capacity of c can receive (resp. send) c MaxMin messages per second. We have also run a simulation with $N = 6$ nodes.

We present nine different simulations, each based on a different bandwidth configuration, i.e., collection of ingress/egress capacities. Simulations $1 - 4$ cover the casework developed in Theorem 3.4.3; each simulation is titled with the corresponding case (1(a), 1(b), 2(a), 2(b)) it belongs to. Simulations $5 - 8$ cover some additional interesting bandwidth configurations; each belongs in one of the four cases from Theorem 3.4.3 but is still worthy of presenting. Simulation 9 is run on a network with 6 nodes.

3.9 Simulation Results

In this section, we present the results from all simulations. For each simulation, we specify the chosen bandwidth configuration, i.e., the ingress and egress capacities of the nodes, and the corresponding r_{OPT} value – which can be calculated using Theorem 3.4.1. We also specify the case from Theorem 3.4.3 which the configuration belongs to.

In each simulation, we present two plots that effectively summarize the findings. The first plot illustrates the total receiving rate (measured in messages received per second) of each follower node over time. We also plot a horizontal line which corresponds to the optimal rate r_{OPT}, in order to facilitate comparison. The second plot depicts the utility scores assigned by the leader to each follower node, also plotted against time.

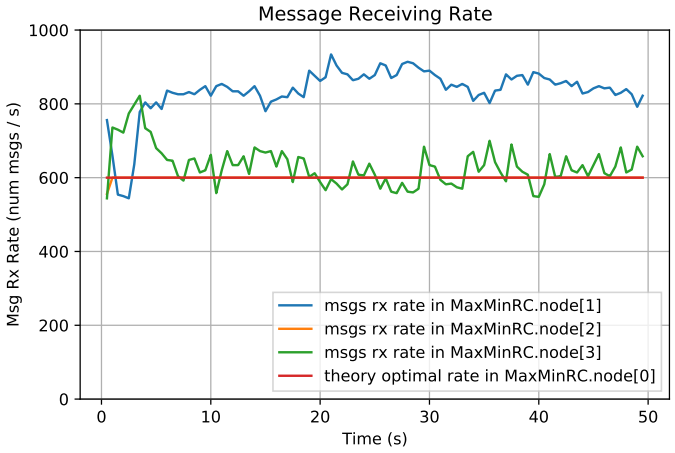
3.9.1 Summary of Results

The simulation results are very promising. In all simulations, we observe that MaxMin-RC manages to converge to a rate allocation scheme which yields a minimum receiving rate very close to r_{OPT}. In fact, in most cases, convergence occurs within $5-10$ seconds after the simulation starts. In certain cases where we have an excess of egress/ingress resources, our MaxMin-RC scheme performs even *better* than required because it is work-conserving. That is, after ensuring that all nodes receive a rate of r_{OPT}, it is able to allocate the remaining egress resources towards nodes with an excess ingress capacity.

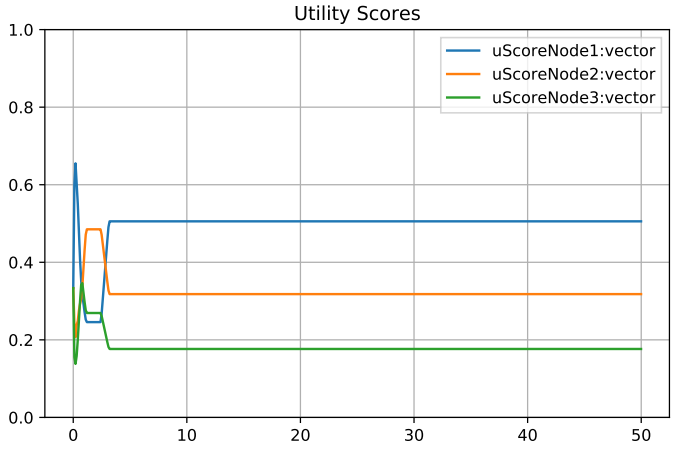
Simulation (1). Case 1(a): $\mathsf{e}'_{\mathsf{crit}} \leq \min(\mathsf{e}_0, \mathsf{in}_{\mathsf{min}})$ and $\mathsf{in}_{\mathsf{min}} \leq \mathsf{e}_{\mathsf{crit}}$

$$in = [1000, 1000, 600, 1000]$$
$$e = [1000, 500, 400, 200]$$
$$r_{\mathsf{OPT}} = 600$$

In this scenario, $\mathsf{in}_{\mathsf{min}} = \mathsf{in}_2 = 600$ is the bottleneck. As depicted in Figure 3-1, MaxMin-RC effectively distributes the surplus egress capacity to nodes with additional ingress capacity, specifically nodes v_1 and v_3.



(a) Receiving rates plotted over time.



(b) Utility Scores plotted over time.

Figure 3-1: Receiving rates and utility scores plotted over time.

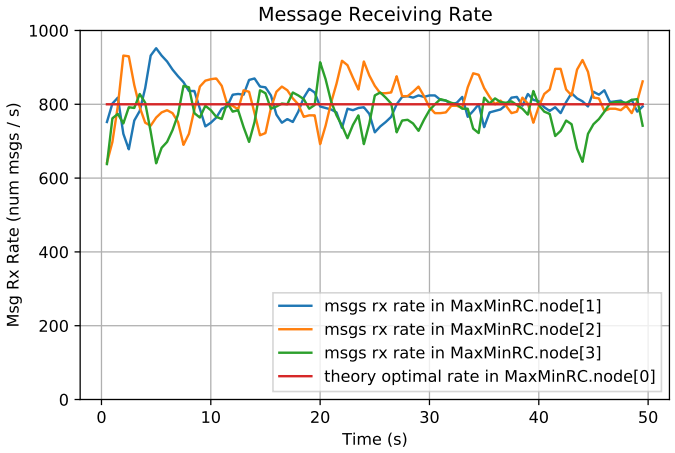
Simulation (2). Case 1(b): $\mathsf{e}'_{\mathsf{crit}} \leq \min(\mathsf{e}_0, \mathsf{in}_{\mathsf{min}})$ and $\mathsf{e}_{\mathsf{crit}} < \mathsf{in}_{\mathsf{min}}$

$$in = [1000, 1000, 1000, 1000]$$

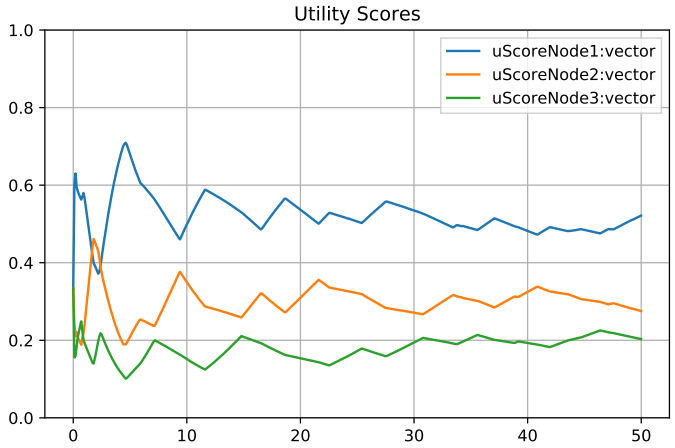
$$e = [1000, 800, 400, 200]$$

$$r_{\mathsf{OPT}} = 800$$

In this scenario, $\mathsf{e}_{\mathsf{crit}} = 800$ is the bottleneck.



(a) Receiving rates plotted over time.



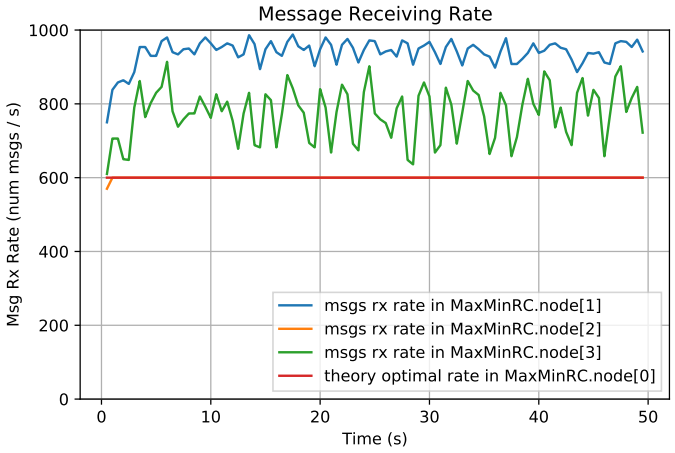
(b) Utility Scores plotted over time.

Figure 3-2: Receiving rates and utility scores plotted over time.

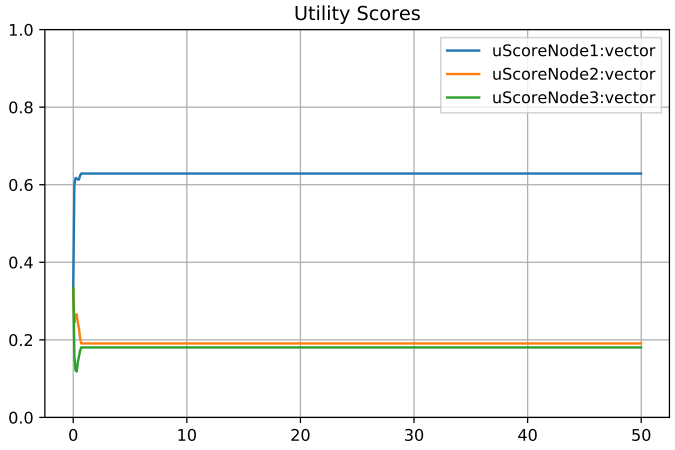
Simulation (3). Case 2(a): $\mathsf{e}'_{\mathsf{crit}} > \min(\mathsf{e}_0, \mathsf{in}_{\mathsf{min}})$ and $\mathsf{in}_{\mathsf{min}} \leq \mathsf{e}_0$

$$in = [1000, 1000, 600, 1000]$$
$$e = [1000, 800, 400, 200]$$
$$r_{\mathsf{OPT}} = 600$$

In this scenario, $\mathsf{in}_{\mathsf{min}} = \mathsf{in}_2 = 600$ is the bottleneck. As depicted in Figure 3-3, MaxMin-RC effectively distributes the surplus egress capacity to nodes with additional ingress capacity, specifically nodes v_1 and v_3.



(a) Receiving rates plotted over time.



(b) Utility Scores plotted over time.

Figure 3-3: Receiving rates and utility scores plotted over time.

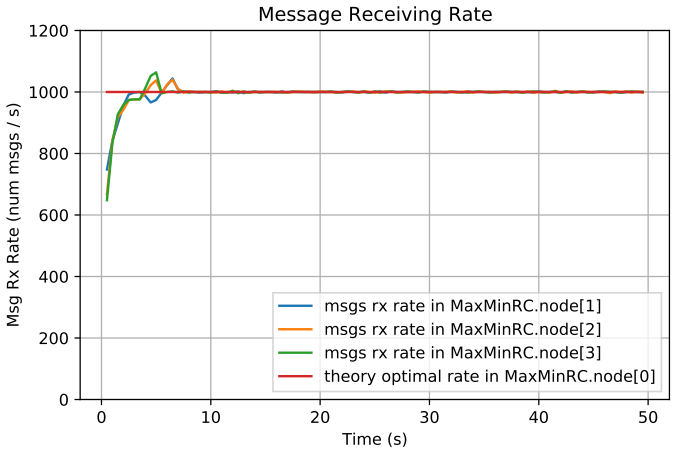
Simulation (4). Case 2(b): $\mathsf{e}'_{\mathsf{crit}} > \min(\mathsf{e}_0, \mathsf{in}_{\mathsf{min}})$ and $\mathsf{e}_0 < \mathsf{in}_{\mathsf{min}}$

$$in = [1000, 1500, 1200, 1500]$$

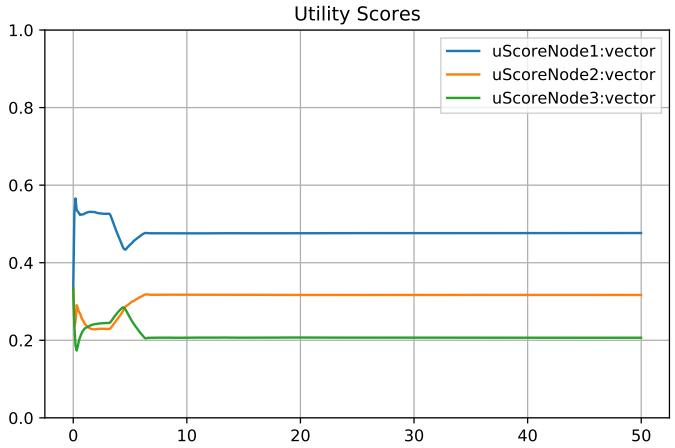
$$e = [1000, 1000, 800, 500]$$

$$r_{\mathsf{OPT}} = 1000$$

In this scenario, $\mathsf{e}_0 = 1000$ is the bottleneck.



(a) Receiving rates plotted over time.



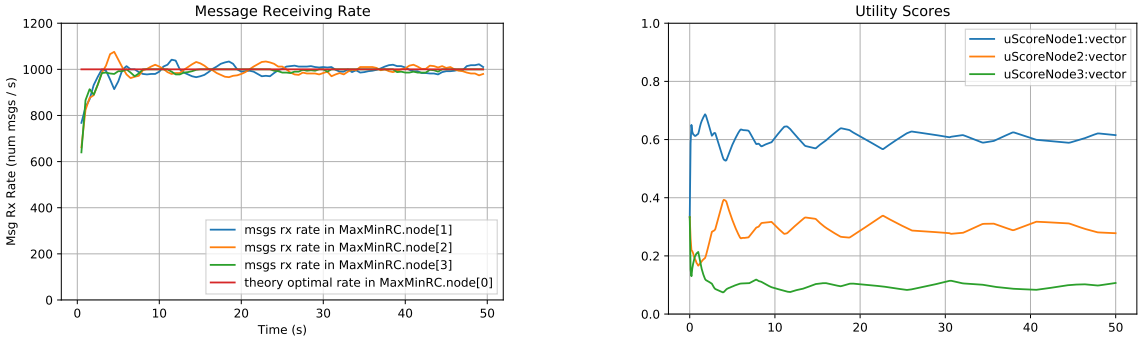
(b) Utility Scores plotted over time.

Figure 3-4: Receiving rates and utility scores plotted over time.

Simulation (5). Tight Configuration

$$in = [1000, 1500, 1500, 1000]$$
$$e = [1000, 1200, 600, 200]$$
$$r_{\mathsf{OPT}} = 1000$$

We refer to this particular configuration as "tight" due to the equality of all critical values, namely $\mathsf{e}_0 = \mathsf{in}_{\mathsf{min}} = \mathsf{e}_{\mathsf{crit}} = 1000$. It can be demonstrated, as a corollary of Theorem 3.4.3, that in a tight configuration, there exists a singular rate allocation scheme that achieves r_{OPT}. The presence of this unique solution implies that tight configurations can pose a considerable challenge for the effectiveness of a distributed rate controller. However, as depicted in Figure 3-5a, MaxMin-RC manages to successfully converge towards an optimal rate allocation.



(a) Receiving rates plotted over time.

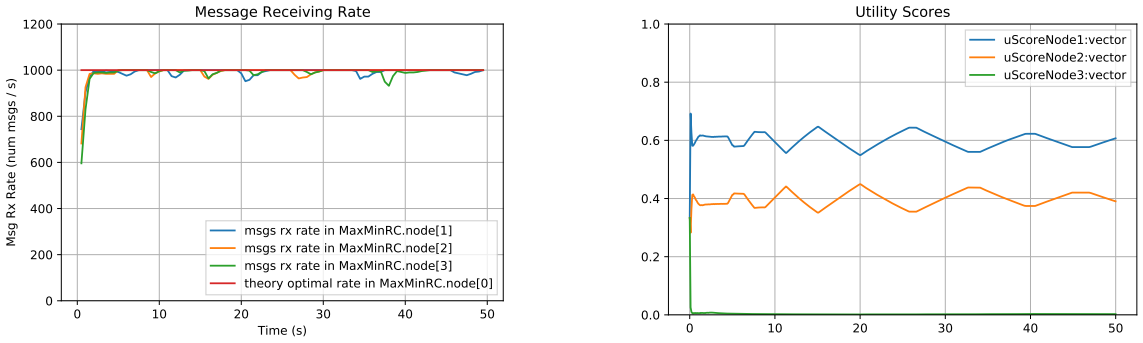
(b) Utility Scores plotted over time.

Figure 3-5: Receiving rates and utility scores plotted over time.

Simulation (6). Zero Contribution From a Node

$$in = [1000, 1000, 1000, 1000]$$
$$e = [1000, 1200, 800, 10]$$
$$r_{\mathsf{OPT}} = 1000$$

In this scenario, $\mathsf{in}_{\mathsf{min}} = 1000$ is the bottleneck. Notably, node v_3 possesses negligible egress capacity, making it incapable of serving as an intermediate node. Our feedback-based controller promptly identifies this limitation, leading the leader to assign a (relative) utility score of zero to v_3, as depicted in Figure 3-6b. Consequently, the leader allocates its entire egress bandwidth to nodes v_1 and v_2. Once again, as depicted in Figure 3-6a, MaxMin-RC achieves convergence towards an optimal rate allocation.



(a) Receiving rates plotted over time.

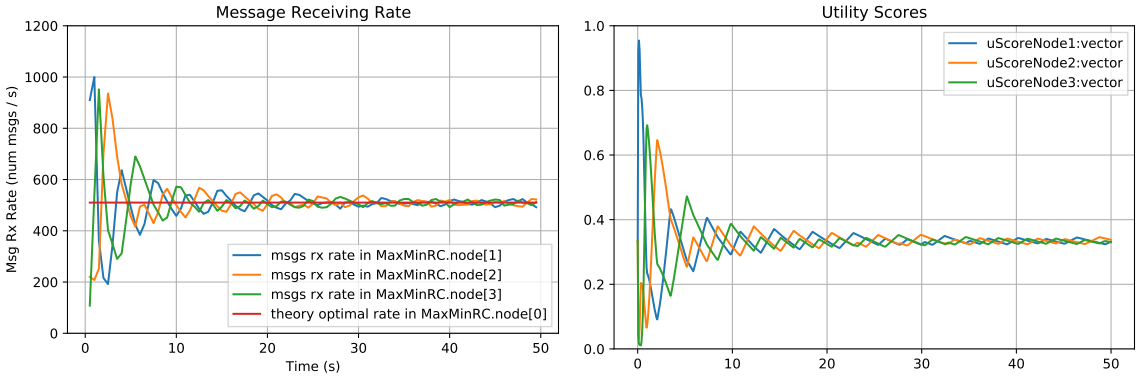
(b) Utility Scores plotted over time.

Figure 3-6: Receiving rates and utility scores plotted over time.

Simulation (7). One-hop Broadcast

$$in = [1000, 1000, 1000, 1000]$$
$$e = [1500, 10, 10, 10]$$
$$r_{\mathsf{OPT}} = 510$$

In this scenario, $\mathsf{e}_{\mathsf{crit}} = 510$ is the bottleneck. It is noteworthy that follower nodes v_1, v_2, v_3 have negligible egress capacity and are thus ineffective as intermediate nodes. Despite our feedback-based controller recognizing this limitation, one might question why all utility scores converge to $u = 0.33 >> 0$, as illustrated in Figure 3-7b. However, this outcome is expected; the lack of a follower node with a substantial egress capacity makes nodes v_1, v_2, v_3 equally unhelpful as intermediate nodes. As a result, the leader allocates equally its entire egress bandwidth to nodes v_1, v_2, v_3, essentially approximating a one-hop rate allocation scheme. Once again, as depicted in Figure 3-7a, MaxMin-RC achieves convergence towards an optimal rate allocation.



(a) Receiving rates plotted over time.

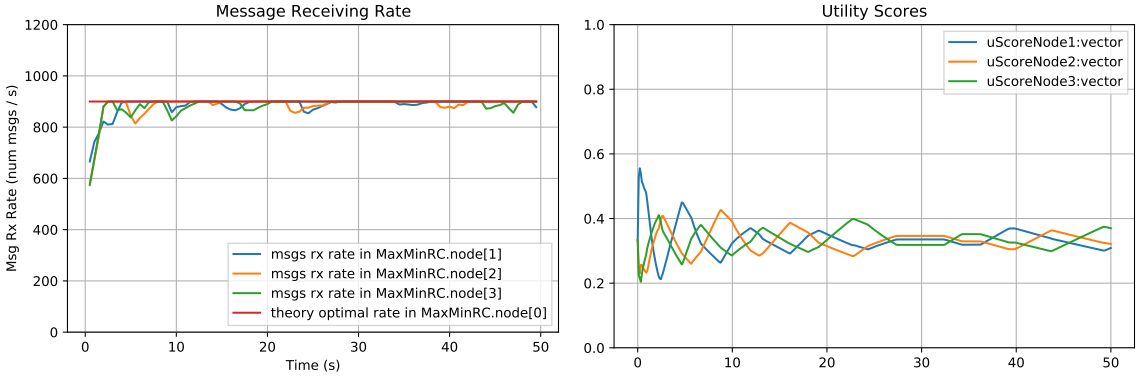
(b) Utility Scores plotted over time.

Figure 3-7: Receiving rates and utility scores plotted over time.

Simulation (8). Equal Rate Allocation – Back to CodedBcaster

$$in = [1000, 900, 900, 900]$$
$$e = [900, 600, 600, 600]$$
$$r_{\mathsf{OPT}} = 900$$

In this additional tight configuration scenario, it is worth noting that the optimal rate allocation, as determined by Theorem 3.4.3, aligns with the CodedBcaster solution (see §2.4). Specifically, the rate allocation that achieves r_{OPT} corresponds to the rate allocation employed by the CodedBcaster variant, which does not introduce redundancy to the shares.



(a) Receiving rates plotted over time.

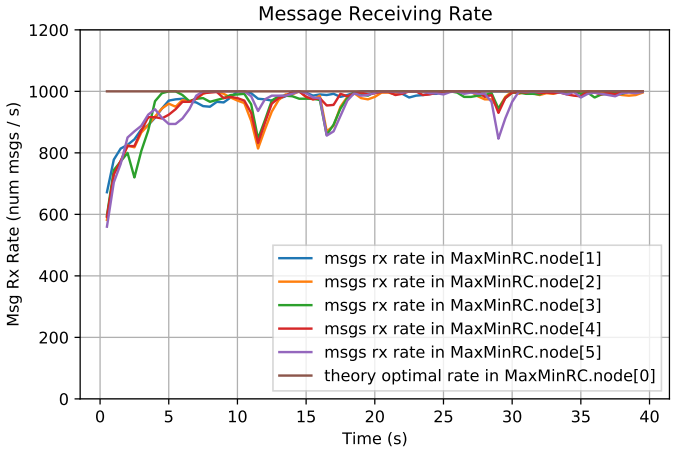
(b) Utility Scores plotted over time.

Figure 3-8: Receiving rates and utility scores plotted over time.

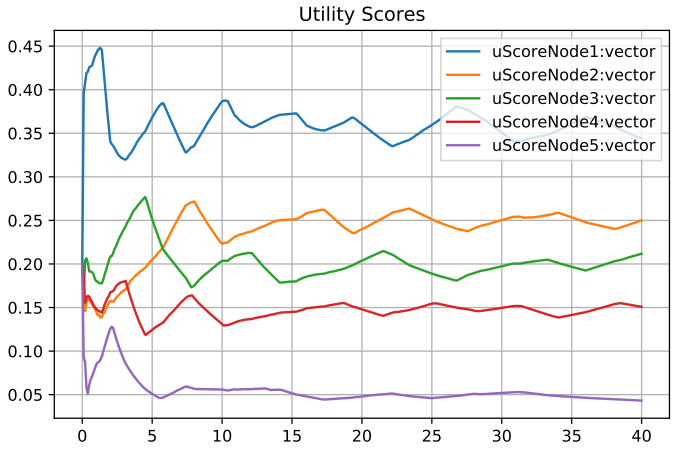
Simulation (9). Six Nodes

$$in = [1000, 1000, 1000, 1000, 1000, 1000]$$
$$e = [1000, 1400, 1000, 800, 600, 200]$$
$$r_{\mathsf{OPT}} = 1000$$

In this scenario, we have $N = 6$ nodes and a tight configuration. As depicted in Figure 3-9a, MaxMin-RC manages to successfully converge towards an optimal rate allocation.



(a) Receiving rates plotted over time.



(b) Utility Scores plotted over time.

Figure 3-9: Receiving rates and utility scores plotted over time.

Chapter 4

Discussion and Future Steps

Our MaxMin-RC protocol is inspired by the formulation of the Max-Min rate optimization problem. As explained in §3.7, the leader increases the utility score of an intermediate node each time it assists in delivering a MaxMin message to the minimum receiving node. It is worth noting that this definition of the utility function closely corresponds to the formulation of the optimization problem discussed in §3.2.

Furthermore, it is worth emphasizing the simplicity inherent in the design of MaxMin-RC. The leader relies solely on the received ACKs from the followers to determine the sending rates. Therefore, MaxMin-RC avoids explicit coordination among peers to yield an optimal allocation scheme.

While Byzantine faults are not the focus of this part of the thesis, it is important to note that the simplicity of our design reduces the threat model and attack space against MaxMin-RC. In fact, MaxMin-RC is already resilient to *passive* attacks (faults), which involves nodes not participating in the broadcast process. The leader's feedback-based controller treats such faulty nodes in the same manner as nodes with negligible egress bandwidth.

In pursuit of a practical and robust Byzantine Fault Tolerant (BFT) rate controller that achieves or closely approximates the optimal throughput, we identify several potential directions for future research:

- Prove theoretical guarantees for the convergence of the MaxMin-RC protocol

to the optimal r_{OPT} rate. That is, prove that MaxMin-RC eventually converges to r_{OPT}, and also prove that it converges relatively quickly.

- A different definition of optimality implies a different definition of fairness. It would be interesting to explore other forms of fairness, (beyond the current r_{OPT} definition), such as proportional fairness.

- Formally explore the threat model of Byzantine attacks against MaxMin-RC, and upper bound the disruption (i.e., deviation from r_{OPT}) that Byzantine nodes can cause.

- Enhance MaxMin-RC to mitigate against Byzantine nodes and render it BFT.

Appendix A

Supplementary Proofs

We prove lemma 3.4.4 and lemma 3.4.5.

Lemma 3.4.4. *Assume that* $\mathsf{e}'_{\mathsf{crit}} \leq \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$. *Suppose that the leader* v_0 *sends to each follower* v_i *at a rate*

$$r(0,i) = \frac{\mathsf{e}_i}{(N-2)} + \min\left\{\frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)}, \quad \mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}\right\}.$$

Also suppose that each follower v_i *sends to each other follower* v_j *at a rate*

$$r(i,j) = \frac{\mathsf{e}_i}{(N-2)}.$$

Then, for all $i \geq 1$,

$$r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}.$$

Proof. We consider cases:

1. $\mathsf{e}_{\mathsf{crit}} \leq \mathsf{in}_{\mathsf{min}}$.

   By claim A.0.2, we have $\frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)} \leq \mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}$. We show that each bandwidth constraint is satisfied and that $r_{\mathsf{in}}(i) = \mathsf{e}_{\mathsf{crit}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \geq 1$.

- e_0 **constraint**. We have

$$\begin{aligned} r_{\mathsf{out}}(0) &= \sum_{i=1}^{N-1} r(0,i) \\ &= \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2} + \sum_{i=1}^{N-1} \frac{(\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}})}{N-1} \\ &= \mathsf{e}'_{\mathsf{crit}} + (\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}) = \mathsf{e}_0. \end{aligned}$$

- e_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{out}}(i) &= \sum_{j=1, j\neq i}^{N-1} r(i,j) \\ &= \sum_{j=1, j\neq i}^{N-1} \frac{\mathsf{e}_i}{N-2} \\ &= (N-2) \cdot \frac{\mathsf{e}_i}{N-2} \\ &= \mathsf{e}_i. \end{aligned}$$

- in_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{in}}(i) &= r(0,i) + \sum_{j=1, j\neq i}^{N-1} r(j,i) \\ &= \left[\frac{\mathsf{e}_i}{N-2} + \frac{(\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}})}{N-1}\right] + \sum_{j=1, j\neq i}^{N-1} \frac{\mathsf{e}_j}{N-2} \\ &= \sum_{j=1}^{N-1} \frac{\mathsf{e}_j}{N-2} - \frac{1}{N-1} \sum_{j=1}^{N-1} \frac{\mathsf{e}_j}{N-2} + \frac{\mathsf{e}_0}{N-1} \\ &= \frac{1}{N-1} \sum_{j=1}^{N-1} \mathsf{e}_j + \frac{\mathsf{e}_0}{N-1} \\ &= \mathsf{e}_{\mathsf{crit}} \leq \mathsf{in}_{\mathsf{min}} \leq \mathsf{in}_i. \end{aligned}$$

By the in_i-constraint reasoning above, we also get that for all $i \geq 1$, $r_{\mathsf{in}}(i) = \mathsf{e}_{\mathsf{crit}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$.

2. $\mathsf{in}_{\mathsf{min}} < \mathsf{e}_{\mathsf{crit}}$.

By claim A.0.2, we have $\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}} < \frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)}$. We show that each bandwidth constraint is satisfied and that $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$ for all $i \geq 1$.

By claim A.0.1, we have $\mathsf{e}_0 \geq \mathsf{e}_{\mathsf{crit}} \geq \mathsf{e}'_{\mathsf{crit}}$, which we will use.

- e_0 **constraint**. We have

$$\begin{aligned} r_{\mathsf{out}}(0) &= \sum_{i=1}^{N-1} r(0,i) \\ &= \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2} + \sum_{i=1}^{N-1} (\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}) \\ &= \mathsf{e}'_{\mathsf{crit}} + (\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}) = \mathsf{in}_{\mathsf{min}} < \mathsf{e}'_{\mathsf{crit}} \leq \mathsf{e}_0. \end{aligned}$$

- e_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{out}}(i) &= \sum_{j=1, j\neq i}^{N-1} r(i,j) \\ &= \sum_{j=1, j\neq i}^{N-1} \frac{\mathsf{e}_i}{N-2} \\ &= (N-2) \cdot \frac{\mathsf{e}_i}{N-2} \\ &= \mathsf{e}_i. \end{aligned}$$

- in_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{in}}(i) &= r(0,i) + \sum_{j=1, j\neq i}^{N-1} r(j,i) \\ &= \left[\frac{\mathsf{e}_i}{N-2} + (\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}})\right] + \sum_{j=1, j\neq i}^{N-1} \frac{\mathsf{e}_j}{N-2} \\ &= \sum_{j=1}^{N-1} \frac{\mathsf{e}_j}{N-2} + (\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}) \\ &= \mathsf{e}'_{\mathsf{crit}} + (\mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}}) \\ &= \mathsf{in}_{\mathsf{min}} \leq \mathsf{in}_i. \end{aligned}$$

By the in_i-constraint reasoning above, we also get that for all $i \geq 1$, $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_{\mathsf{crit}}\}$.

□

Lemma 3.4.5. *Assume that* $\mathsf{e}'_{\mathsf{crit}} \geq \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$. *Suppose that the leader* v_0 *sends to each follower* v_i *at a rate*

$$r(0,i) = \frac{\min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \frac{\mathsf{e}_i}{(N-2)}$$

Also suppose that each follower v_i *sends to each other follower* v_j *at a rate*

$$r(i,j) = r(0,i).$$

Then, for all $i \geq 1$,

$$r_{\mathsf{in}}(i) = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}.$$

Proof. We consider cases:

1. $\mathsf{in}_{\mathsf{min}} \leq \mathsf{e}_0$.

   We show that each bandwidth constraint is satisfied and that $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$ for all $i \geq 1$.

   - e_0 **constraint**. We have

$$\begin{aligned} r_{\mathsf{out}}(0) &= \sum_{i=1}^{N-1} r(0,i) \\ &= \frac{\mathsf{in}_{\mathsf{min}}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{(N-2)} \\ &= \frac{\mathsf{in}_{\mathsf{min}}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \mathsf{e}'_{\mathsf{crit}} = \mathsf{in}_{\mathsf{min}} \leq \mathsf{e}_0. \end{aligned}$$

- e_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{out}}(i) &= \sum_{j=1, j\neq i}^{N-1} r(i,j) \\ &= (N-2) \cdot r(0,i) \\ &\leq (N-2)\frac{\mathsf{e}_i}{(N-2)} \\ &= \mathsf{e}_i. \end{aligned}$$

- in_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{in}}(i) &= r(0,i) + \sum_{j=1, j\neq i}^{N-1} r(j,i) \\ &= \frac{\mathsf{in}_{\mathsf{min}}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{(N-2)} \\ &= \frac{\mathsf{in}_{\mathsf{min}}}{\mathsf{e}'_{\mathsf{crit}}} \cdot \mathsf{e}'_{\mathsf{crit}} \\ &= \mathsf{in}_{\mathsf{min}} \leq \mathsf{in}_i. \end{aligned}$$

By the in_i-constraint reasoning above, we also get that for all $i \geq 1$, $r_{\mathsf{in}}(i) = \mathsf{in}_{\mathsf{min}} = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$.

2. $\mathsf{e}_0 < \mathsf{in}_{\mathsf{min}}$.

We show that each bandwidth constraint is satisfied and that $r_{\mathsf{in}}(i) = \mathsf{e}_0 = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$ for all $i \geq 1$.

- e_0 **constraint**. We have

$$\begin{aligned} r_{\mathsf{out}}(0) &= \sum_{i=1}^{N-1} r(0,i) \\ &= \frac{\mathsf{e}_0}{\mathsf{e}'_{\mathsf{crit}}} \cdot \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{(N-2)} \\ &= \frac{\mathsf{e}_0}{\mathsf{e}'_{\mathsf{crit}}} \cdot \mathsf{e}'_{\mathsf{crit}} = \mathsf{e}_0. \end{aligned}$$

- e_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{out}}(i) &= \sum_{j=1, j\neq i}^{N-1} r(i,j) \\ &= (N-2) \cdot r(0,i) \\ &\leq (N-2)\frac{\mathsf{e}_i}{(N-2)} \\ &= \mathsf{e}_i. \end{aligned}$$

- in_i **constraint**, $i \geq 1$. We have

$$\begin{aligned} r_{\mathsf{in}}(i) &= r(0,i) + \sum_{j=1, j\neq i}^{N-1} r(j,i) \\ &= \frac{\mathsf{e}_0}{\mathsf{e}'_{\mathsf{crit}}} \cdot \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{(N-2)} \\ &= \frac{\mathsf{e}_0}{\mathsf{e}'_{\mathsf{crit}}} \cdot \mathsf{e}'_{\mathsf{crit}} \\ &= \mathsf{e}_0 \\ &< \mathsf{in}_{\mathsf{min}} \leq \mathsf{in}_i. \end{aligned}$$

By the in_i-constraint reasoning above, we also get that for all $i \geq 1$, $r_{\mathsf{in}}(i) = \mathsf{e}_0 = \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$.

□

Claim A.0.1. *The following implications are true:*

1. *If* $\mathsf{e}_0 \geq \mathsf{e}'_{\mathsf{crit}}$*, then* $\mathsf{e}_0 \geq \mathsf{e}_{\mathsf{crit}} \geq \mathsf{e}'_{\mathsf{crit}}$.

2. *If* $\mathsf{e}'_{\mathsf{crit}} \geq \mathsf{e}_0$*, then* $\mathsf{e}'_{\mathsf{crit}} \geq \mathsf{e}_{\mathsf{crit}} \geq \mathsf{e}_0$.

Proof. Consider the difference

$$\begin{aligned}
\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}} &= \mathsf{e}_0 - \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2} \\
&= \mathsf{e}_0 + \frac{\mathsf{e}_0}{N-2} - \frac{\mathsf{e}_0}{N-2} - \sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2} \\
&= \frac{N-1}{N-2}\mathsf{e}_0 - \frac{1}{N-2}\sum_{i=0}^{N-1} \mathsf{e}_i \\
&= \frac{N-1}{N-2}\big(\mathsf{e}_0 - \frac{1}{N-1}\sum_{i=0}^{N-1} \mathsf{e}_i\big) \\
&= \frac{N-1}{N-2}\big(\mathsf{e}_0 - \mathsf{e}_{\mathsf{crit}}\big).
\end{aligned}$$

Also consider the difference

$$\begin{aligned}
\mathsf{e}_{\mathsf{crit}} - \mathsf{e}'_{\mathsf{crit}} &= \frac{1}{N-1}\sum_{i=0}^{N-1} \mathsf{e}_i - \frac{1}{N-2}\sum_{i=1}^{N-1} \mathsf{e}_i \\
&= \frac{\mathsf{e}_0}{N-1} + \frac{1}{N-1}\sum_{i=1}^{N-1} \mathsf{e}_i - \frac{1}{N-2}\sum_{i=1}^{N-1} \mathsf{e}_i \\
&= \frac{\mathsf{e}_0}{N-1} - \frac{1}{(N-1)(N-2)}\sum_{i=1}^{N-1} \mathsf{e}_i \\
&= \frac{\mathsf{e}_0}{N-1} - \frac{1}{N-1}\sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{N-2} \\
&= \frac{1}{N-1}\big(\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}\big).
\end{aligned}$$

Now implications (1) and (2) follow from these differences.

□

Claim A.0.2. *Assume that* $\mathsf{e}'_{\mathsf{crit}} \leq \min\{\mathsf{in}_{\mathsf{min}}, \mathsf{e}_0\}$. *Then*

$$\frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)} \leq \mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}} \quad \textit{if and only if} \quad \mathsf{e}_{\mathsf{crit}} \leq \mathsf{in}_{\mathsf{min}}.$$

Proof. We have

$$\begin{aligned}
&\frac{\mathsf{e}_0 - \mathsf{e}'_{\mathsf{crit}}}{(N-1)} \leq \mathsf{in}_{\mathsf{min}} - \mathsf{e}'_{\mathsf{crit}} \iff \\
&\mathsf{e}_0 + (N-2)\mathsf{e}'_{\mathsf{crit}} \leq (N-1)\mathsf{in}_{\mathsf{min}} \iff \\
&\mathsf{e}_0 + (N-2)\sum_{i=1}^{N-1} \frac{\mathsf{e}_i}{(N-2)} \leq (N-1)\mathsf{in}_{\mathsf{min}} \iff \\
&\sum_{i=0}^{N-1} \mathsf{e}_i \leq (N-1)\mathsf{in}_{\mathsf{min}} \iff \\
&\mathsf{e}_{\mathsf{crit}} \leq \mathsf{in}_{\mathsf{min}}.
\end{aligned}$$

□

Claim A.0.3. *For all* i, *the following inequality holds:*

$$\sum_{i=1}^{N-1} r_{\mathsf{in}}(i) \leq \sum_{i=0}^{N-1} \mathsf{e}_i.$$

Proof. We have

$$\begin{aligned}
\sum_{i=1}^{N-1} r_{\mathsf{in}}(i) &\leq \sum_{i=1}^{N-1} \sum_{j \neq i} r(j,i) \\
&= \sum_{j=0}^{N-1} \sum_{i \neq j} r(j,i) \\
&= \sum_{j=0}^{N-1} r_{\mathsf{out}}(j) \\
&\leq \sum_{i=0}^{N-1} \mathsf{e}_i.
\end{aligned}$$

□

Bibliography

[1] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *IEEE Transactions on Information Theory*, 46(4):1204–1216, 2000.

[2] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Mayank Varia, Zhuolun Xiang, and Haibin Zhang. Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, pages 399–417, 2022.

[3] Alysson Bessani, João Sousa, and Eduardo E.P. Alchieri. State machine replication for the masses with bft-smart. In *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 355–362, 2014.

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

[5] Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005.

[6] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OsDI*, volume 99, pages 173–186, 1999.

[7] George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. Narwhal and tusk: a dag-based mempool and efficient bft consensus. In *Proceedings of the Seventeenth European Conference on Computer Systems*, pages 34–50, 2022.

[8] Christos Gkantsidis, John L. Miller, and Pablo Rodriguez. Anatomy of a P2P content distribution system with network coding. In Emin Gün Sirer and Ben Y. Zhao, editors, *5th International workshop on Peer-To-Peer Systems, IPTPS 2006, Santa Barbara, CA, USA, February 27-28, 2006*, 2006.

[9] Richard L Graham, Timothy S Woodall, and Jeffrey M Squyres. Open mpi: A flexible high performance mpi. In *International Conference on Parallel Processing and Applied Mathematics*, pages 228–239. Springer, 2005.

[10] V. Jacobson. Congestion avoidance and control. In *Symposium Proceedings on Communications Architectures and Protocols*, SIGCOMM '88, page 314–329, New York, NY, USA, 1988. Association for Computing Machinery.

[11] Ioannis Kaklamanis, Lei Yang, and Mohammad Alizadeh. Poster: Coded broadcast for scalable leader-based bft consensus. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 3375–3377, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition*. Chapman & Hall/CRC, 2nd edition, 2014.

[13] Chen-Da Liu-Zhang, Christian Matt, and Søren Eller Thomsen. Asymptotically optimal message dissemination with applications to blockchains. Cryptology ePrint Archive, Paper 2022/1723, 2022. `https://eprint.iacr.org/2022/1723`.

[14] M. Luby. Lt codes. In *The 43rd Annual IEEE Symposium on Foundations of Computer Science, 2002. Proceedings.*, pages 271–280, 2002.

[15] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation. SOSP '21, page 35–48, New York, NY, USA, 2021. Association for Computing Machinery.

[16] OpenAI. ChatGPT (May 10 version) [large language model], May 2023.

[17] I. S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[18] Dmitry Shulyak. go-hotstuff. `https://github.com/dshulyak/go-hotstuff`.

[19] Lei Yang, Yossi Gilad, and Mohammad Alizadeh. Coded transaction broadcasting for high-throughput blockchains, 2022.

[20] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. Dispersedledger:high-throughput byzantine consensus on variable bandwidth networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 493–512, 2022.

[21] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[22] Siyuan Zhuang, Zhuohan Li, Danyang Zhuo, Stephanie Wang, Eric Liang, Robert Nishihara, Philipp Moritz, and Ion Stoica. Hoplite: efficient and fault-tolerant collective communication for task-based distributed systems. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*, pages 641–656, 2021.