PhD-FSTM-2023-063
The Faculty of Science, Technology and Communication

DISSERTATION

Defence held on 29/08/2023 in Esch-sur-Alzette

to obtain the degree of

DOCTEUR DE L'UNIVERSITÉ DU LUXEMBOURG

EN INFORMATIQUE

by

Douglas SIMÕES SILVA
Born on 05 October 1990 in Florianópolis, Brazil

RESILIENT THREAT-ADAPTIVE CONSENSUS

Dissertation defence committee

Dr Marcus Völp, dissertation supervisor
Professor, Université du Luxembourg

Dr Yair Amir, Member
Professor, Johns Hopkins University

Dr Nuno Ferreira Neves, Member
Professor, Universidade de Lisboa

Dr Gilbert Fridgen, Chairman
Professor, Université du Luxembourg

Dr Peter Y. A. Ryan, Vice Chairman
Professor, Université du Luxembourg

Acknowledgements

First, I would like to thank my family for all their effort in providing me a good education, which enabled me to achieve so many things in life that I am grateful for, all their emotional help and their support. I am extremely grateful for having such a special family always by my side.

Secondly, I would like to thank my wife for all her love, support, and patience with me. Thank you for being my test audience for all my presentations and always believing in me and my work.

I would like to thank Professor Paulo Esteves-Veríssimo for all the knowledge taught during meetings, seminars, and courses. A huge thanks to Professor Jérémie Decouchant for the countless hours of engaging discussions regarding ideas, concepts, and solutions throughout these years. I would like to thank Dr. Rafal, my co-supervisor, for consistently being receptive to discussions and for firmly believing in my capabilities from the beginning.

Finally, I would like to thank my supervisor Professor Marcus Völp for all his trust in my work, for showing me all I am capable of, and for all his constructive comments on my texts, presentations and papers written. I would like to thank all the members of the CritiX group for being part of my family during these PhD years.

Declaration

I, Douglas Simões Silva, declare that this thesis entitled, "Resilient Threat-Adaptive Consensus" and the work presented therein are my own. I confirm that:

- this work was done wholly or mainly while in candidature for the degree Docteur de l'Université du Luxembourg;
- where any part of this thesis has previously been submitted for a degree or any other qualification at this university or any other institution, this has been clearly stated;
- where I have consulted the published works of others, these are clearly attributed;
- where I have quoted from the works of others, the sources are always given;
- where the work presented in this thesis is based on work done by myself jointly with others, I have clearly outlined what was done by others and what I contributed;
- with the exception of such quotations, this is entirely my own work; and
- I have acknowledged all main sources of help.

Signed:

Date:

Abstract

Malicious and coordinated attacks are happening increasingly often, and have targeted critical systems such as nuclear plants, public transportation systems, hospitals and governments. Because critical infrastructures must be resilient against advanced and persistent threats, a common architecture of choice to mitigate those hazards are distributed systems, more specifically Byzantine fault-tolerant state-machine replicated(BFT-SMR) systems. In this PhD thesis, we propose solutions to critical challenges in the field of distributed systems, focusing on creating adaptive algorithms and protocols to strengthen the resilience state-of-the-art systems. The first challenge is how to ensure the security and reliability of critical infrastructures against advanced and persistent attacks at various threat levels. To address this, we present ThreatAdaptive, a novel BFT-SMR protocol that automatically adapts to changes in the anticipated and observed threats in an unattended manner. ThreatAdaptive proactively reconfigures the system to cope with the faults that one needs to expect given the imminent threats. It threreby avoids the limitations of traditional BFT-SMR protocols that require either by design a high fault threshold or a trusted external reconfiguration entity. Our results show that ThreatAdaptive meets the latency and throughput of BFT baselines while adapting 30% faster than previous methods, providing a more efficient and secure solution for critical infrastructures. The second challenge is how to optimize the performance of a distributed system in the presence of unreliable nodes. To address this, we propose a method for automatic reconfiguration based on a 3D virtual coordinate system (VCS) that allows correct nodes to detect and eliminate inconsistent latencies and protect system performance against Byzantine attacks. We evaluate our reconfiguration baseline, Geometric, on three real-world networking datasets and show that it protects performance up to 78% better than previous solutions and provides the closest representation of real-world connections. Our proposed solutions provide a more reliable and secure approach to automatic reconfiguration in distributed systems. Overall, this thesis makes a significant contribution to the field of distributed systems by proposing novel solutions to two critical challenges: ensuring the security and reliability of critical infrastructures and optimizing the performance of distributed systems in the presence of unreliable nodes.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Malicious and coordinated attacks are happening increasingly over the last decades, and have targeted critical systems such as nuclear plants [FMC11], public transportation systems [MP17], hospitals [SS16] and governments [Tan11]. Many of these attacks compromised a substantial proportion of the machines within the target infrastructure. It is, therefore, necessary for organizations to regularly check the integrity of their system and implement protective measures. However, as the number of infrastructures rises, it also increases the stack of software and the complexity of code executed at these instances, making surveillance and manual mitigation of attacks an extremely difficult task [Acc20]. Instead, systems should be able to function independently, in a distributed manner, and without constant oversight during periods of persistent and ongoing incidents [Gor+11].

Distributed systems can be converted into resilient and available systems even if fully connected by potentially unreliable nodes. The consistency of correct decisions or progress obtained by systems like these can be attested as fault-tolerant when an agreement involving a majority of votes is reached. Majority voting is achieved in the system using a consensus protocol. By reaching an agreement, servers decide distributively on actions the system will execute.

Byzantine Fault-Tolerant State Machine Replication (BFT-SMR) allows the implementation of fault-tolerant distributed systems or services, by replicating data over multiple servers and coordinating client request-reply interactions with those servers [CL99]. BFT-SMR Protocols are usually configured with $n \geq 3f + 1$ servers to tolerate up to f Byzantine faulty servers. These protocols are widely used in blockchain systems, distributed databases, cloud computing, military and aerospace systems, and others. They can also incorporate different consensus mechanisms, such as majority voting or weighted voting, to reach an agreement among servers.

Weighted voting is a decision-making method in which each group member is assigned a certain number of votes, based on their level of authority, knowledge,

wealth, or expertise. This approach is commonly used in organizations and committees where specific individuals have more resources than others. The purpose of weighted voting is to ensure that the most informed and qualified members have a more significant say in the decision-making procedure, leading to more efficient and effective decision-making [Ber+20]. In distributed systems, it can be represented by servers with more processing power, better location, or better Internet connection [Gif79]. One potential drawback of weighted voting is that it can lead to uneven representation or discrepant voting power among group members. Therefore, it is essential to carefully consider the criteria used to assign weight and ensure that they are fair and reflective of the group's goals and objectives.

Static voting assignment algorithms [SB15b] use hard-coded predetermined information for all the sites connected to the system. With autonomous voting reassignment [BGS89], each server can change its voting power by requesting that to other servers, and they would, after enough votes, grant it the required voting power. There are dynamic weighted quorum majority systems where Byzantine servers can isolate themselves by partitioning the network or where one server can consolidate all voting power at once [Dav89; JM90] and become a single point of failure. Knowing that dynamically changing voting power is a task that involves many caveats, like by how much the voting power will be changed and by whom, still it was observed that it could enable systems to achieve better throughput and latency [Ber+20] when compared to other methods. Improvements were also seen when the variables that increase voting power are related to the reliability or availability of servers [CAA89]. Servers belonging to distributed systems will cast votes for requests usually issued by clients, but one of the most important request types, in order to adapt systems, is the reconfiguration request.

Reconfiguration is a mechanism used by BFT protocols to prevent malicious servers from manipulating systems decisions or when changing the set of servers that participate in a consensus protocol is needed [CL99]. This dynamic membership feature is important to adapt the system to network changes or to maintain high levels of availability and fault-tolerance by removing from the system faulty or compromised servers and it is supported by several BFT protocols like Hotstuff [Yin+19], Tendermint [BKM18] and BFT-SMaRt [BSA14b]. Distributed systems can operate reconfigurations unattended when built with a focus on the system's adaptability [Sil+21].

The primary contribution and overall hypothesis which this thesis answers is the following: by prioritizing adaptability during the design of BFT-SMR protocols, it is possible to enable their unattended operation to obtain improved performance, increased robustness and enhanced resilience even in the presence of powerful threats and coordinated attacks. To achieve this, the adaptation process itself must remain robust against attacks, to which this thesis contributes as well.

1.1 Thesis Outline

This thesis is organized as follows:

- Chapter 2 provides the background knowledge and motivation to support this thesis. We first introduce the building blocks concepts and works used as the basis for this work.

- Chapter 3 introduces our Threat Adaptive BFT-SMR protocol. We draw attention to the existing issues that our protocol addresses and examine the current solutions available within the field of distributed systems. We finish this chapter by explaining in detail the benchmarking results achieved by our Threat-Adaptive protocol.

- Chapter 4 presents challenges and our solutions for latency-aware automatic reconfiguration protocols. We explain how the concept of a 3D virtual coordinate system (VCS) can improve systems with latency-based optimization. We also compare and evaluate the accuracy of our methods against state-of-art baselines using real-world datasets.

- Chapter 5 discusses integrating the solutions presented by Chapters 3 and 4 in one BFT protocol that optimizes resources dynamically at runtime using not only the threat level but also a trustworthy latency matrix as input. This protocol would also be able to return the system to safer configurations if needed without relying on consensus. By combining this dynamic optimization with the methods developed in Chapter 4 for creating more efficient system configurations, the protocol achieves robust self-regulation, efficient resource utilization, and improved performance.

- Chapter 6 complete this Thesis by providing a conclusion summary of the challenges encountered and our findings. We discuss the implications and potential impact of this research and we close this chapter by suggesting possible future works in the field of distributed systems.

1.2 Contributions

Contribution 1: The Development of a novel Threat-Adaptive BFT-SMR protocol that safely reconfigures itself to tolerate an increasingly powerful adversary.

- Key features: Our protocol enables servers to agree on more resilient fallback reconfigurations proactively and dynamically optimizes the system's performance by reducing the number of servers.

- Evaluation results: Our approach shows comparable throughput and latency to non-adaptive baselines in stable phases, with reconfigurations 30% faster than previous methods.

Contribution 2: The creation of new latency sanitization methods to mitigate the effect of coordinated latency collusion attacks in distributed systems.

- Key features: Our methods require reported latency information to be consistent in a multidimensional virtual coordinate space.
- Method 1: Geometric finds the new positions for each and every server by identifying the point of intersection between the references of where is the target node to all other servers.
- Method 2: RndSamp similar to Geometric finds the new positions for each and every node by identifying the point of intersection between the references of other servers, with the difference being that RndSamp generates 3d points and checks whether they intersect sufficient references.
- Method 3: Inspired by byzantine gradient averaging of MT-KRUM, our third method uses the makes a Byzantine 3D coordinate aggregation in order to mitigate the impact of faulty servers on the final latency matrix.
- Evaluation results: Our methods are able to protect latency after reconfiguration against Byzantine servers up to 95% better than state-of-the-art solutions.

1.3 Work submissions

This section summarizes the submission status of the work developed during this thesis:

- **Accepted** at Symposiun on Reliable Distributed Systems(SRDS 2021).
    1. Title: Threat Adaptive Byzantine Fault Tolerant State-Machine Replication
       Authors: DS Silva, R Graczyk, J Decouchant, P Esteves-Veríssimo, M Völp;
- **Ongoing**
    1. Title: Robust and Automatic Reconfiguration for BFT State-Machine Replication Systems;
    2. Title: Dynamic and Accountable Reconfiguration for BFT State-Machine Replication Systems;

Chapter 2

Related work

In this chapter, we will review works on which this thesis is based or which are related to it. The research scope of this work encompasses various areas, including Byzantine Fault-Tolerant State Machine Replication (BFT-SMR), Adaptive BFT-SMR protocols, Speculative Agreement protocols, Throughput-Based Optimization Protocols, Weighted Voting Consensus, Latency-Based Optimization Protocols, and Virtual Coordinate Systems. BFT-SMR protocols form a fundamental part of the discussion, focusing on fault-tolerant replication techniques that ensure the system's consistency and integrity in the presence of Byzantine faults. Adaptive BFT-SMR protocols are designed to dynamically adjust their features or configurations in order to achieve more performance or resilience. Speculative Agreement protocols explore the concept of optimistic confirmation and efficient consensus mechanisms that allow for faster transaction confirmation. Throughput-Based Optimization Protocols aim to maximize system throughput by optimizing resource allocation and coordination strategies. Weighted Voting Consensus techniques assign different weights to participants in the consensus process, providing flexibility and prioritization. Latency-Based Optimization Protocols focus on minimizing network latency to improve overall system performance. Lastly, Virtual Coordinate Systems leverage virtual coordinates to estimate network distances and enable efficient communication protocols. Valuable insights and knowledge that contribute to the development of this thesis topic will be clearer after examining these various works, the different approaches and techniques listed that address key challenges in the field of resilient distributed systems, and consensus algorithms.

2.1 Byzantine Fault-Tolerant State Machine Replication(BFT-SMR)

BFT-SMR protocols are commonly designed to be operated on systems with $n \geq 3f + 1$, where n is the total number of servers and f is the number of faulty replicas the system can cope with. The terminology for servers on BFT protocols can vary, the terms "nodes" or "replicas" are also used depending on the specific research paper or line of research being discussed. These protocols, when combined with reconfiguration techniques, can be used to tolerate powerful adversaries. The diversification of the software stack used by servers can also help the system to avoid single-point-of-failure problems, increasing the difficulty of exploiting software errors and reutilizing this knowledge to attack other parts of the system. Therefore, applying these techniques together would make BFT-SMR protocols more robust and resilient.

2.1.1 Practical Byzantine Fault Tolerance(PBFT)

Practical Byzantine Fault Tolerance (PBFT) [CL99] is considered to be the first practical BFT-SMR consensus algorithm designed to provide fault tolerance in environments with realistic system assumptions where nodes may exhibit arbitrary and malicious behavior. PBFT is designed to function effectively with asynchronous and unreliable networks, which means that a message can be delayed as an artifact of loaded networks, it can be dropped, duplicated, tampered with, or delivered out of order. PBFT ensures consensus among a group of replicated state machines by employing a three-phase protocol that consists of a pre-prepare, prepare and commit phase. During protocol execution, client requests are multicasted, assigned sequence numbers, validated by replicas, and eventually committed for execution. PBFT guarantees safety and liveness properties as long as the number of faulty nodes remains below one-third of the total network size. However, PBFT suffers from high communication complexity, as it requires $3f + 1$ replicas to tolerate f faulty nodes, resulting in increased message overhead. Despite this drawback, PBFT has had a significant impact on the field of distributed systems and has inspired subsequent research efforts to improve the scalability and performance of Byzantine fault-tolerant consensus algorithms. Its robust fault tolerance mechanisms and provable properties make PBFT a notable contribution to the field of distributed systems, serving as a foundational algorithm for designing secure and reliable consensus protocols.

PBFT ensures the integrity and reliability of its consensus protocol through safety and liveness properties. Safety in PBFT guarantees that all correct replicas agree on the same sequence of requests and the resulting state, even in the presence

of Byzantine faults. It also ensures that replicas do not diverge or reach conflicting conclusions. Liveness, on the other hand, ensures that the protocol continues to make progress and eventually reaches a decision, even in the face of potential faults or delays. In PBFT, liveness guarantees that client requests are eventually processed, responses are generated, and the system remains responsive despite the presence of faulty or slow nodes. By satisfying both safety and liveness properties, PBFT provides a robust and fault-tolerant consensus protocol that can withstand malicious behavior and maintain the consistency and progress of its replicated state machine infrastructure.

In addition to its support for deterministic state machine replication algorithms, PBFT incorporates an important mechanism called View Change. View Change is a process that enables the system to recover from faults or to change a faulty leader from its role, helping to maintain the safety and liveness properties of the protocol. In PBFT, a view is essentially a period identifier during which a specific primary replica is responsible for processing client requests and coordinating the consensus protocol. However, in the event that the primary replica becomes faulty or unresponsive, the system needs to transition to a new view with a different primary replica. This is where View Change comes into play when a replica suspects that the primary replica is faulty, it initiates a view change by broadcasting a view change message to other replicas. Upon receiving a sufficient number of view change messages, the replicas collaborate to establish a new view and select a new primary replica. This process ensures that the consensus protocol can continue even in the presence of faulty primaries, maintaining the desired properties of safety and liveness. By incorporating View Change, PBFT enhances the fault tolerance capabilities of the protocol and ensures the consistency of operations across all correct replicas.

2.1.2 Proactive Resilience through architectural hybridization

While PBFT addresses fault tolerance specifically within the consensus protocol, there are other ways of making distributed systems more resilient, considering the overall system architecture and proactive measures to prevent and mitigate faults. The proactive resilience design and methodology were built to achieve exhaustion-safe systems[SNV06]. Every distributed system that has by design a finite amount of resources will hit exhaustion after a variable amount of runtime execution, given that every system is susceptible to failures and Byzantine fault-tolerant systems can only withstand f failures. One of the main aspects of this methodology is that the system have to have a way of recovering its nodes or its parameters periodically, bringing them to a safe state or rejuvenating these resources, this practice is called

proactive recovery. Many studies [Her+95; Her+97] suggest that this periodic refresh of the system is more effective than waiting for failures to happen or to be detected and then reacting.

Rather than simply trying to prevent or react to damage, proactive resilience involves creating structures that make systems adaptable and can recover quickly from damage with the help of architectural hybridization. Distributed systems usually have many uncertain characteristics and assumptions, like working with unreliable networks, and communicating messages between distinct or individual pieces of equipment or between different applications. However, there are also predictable and trustworthy parts, one way of leveraging that is with the use of hybrid components interconnected or the use of a wormhole subsystem[Ver03].

The Proactive Resilience Model (PRM) utilizes a combination of proactive recovery practices and architectural hybridization to create exhaustion-safe systems that are resilient to failure. Given that, in practice, proactive rejuvenation can not be coordinated properly in fully asynchronous systems[SNV05], the wormhole subsystem could provide the synchrony and security properties required to enable the system to recover safely. Therefore, throughout the contributions of this thesis, we will be leveraging the concepts of proactive rejuvenation and hybridization in order to make our systems and protocols more resilient.

2.1.3 Lazarus

Although the Proactive Resilience Model focuses on proactive strategies for improving system resilience through architectural hybridization and adaptive mechanisms, Lazarus[GBN19] presents an innovative approach to employ and manage diversity in Byzantine Fault-Tolerant systems automatically. Despite potential malicious attacks or failures, BFT systems rely on replicas to reach a consensus. However, these systems are vulnerable to attacks, and the system should guarantee that these faults are independent. Lazarus aims to enhance the resilience of BFT systems by introducing automatic diversity management techniques. It follows a hybrid fault model where each replica has a Local Trusted Unit(LTU) responsible for receiving messages from the system controller that is continuously being fed information from trusted sources on whether the software components used by replicas are at certain risk. Based on these assessments, Lazarus dynamically adjusts the replica configuration to maximize their diversity. It strategically selects replicas with different characteristics, such as hardware, software, or network environments, to minimize the impact of correlated failures or attacks. The selection process is guided by a diversity metric quantifying the dissimilarity between replicas. Through the automatic management of diversity, Lazarus enhances the fault tolerance and security of BFT systems, reducing the vulnerability to coordinated attacks and increasing their resilience. Experimental evaluations demonstrate the

effectiveness of Lazarus in improving the system's ability to tolerate malicious behavior and maintain high availability. By automating the diversity management process, Lazarus simplifies the deployment and maintenance of BFT systems, allowing them to adapt to dynamic environments and effectively withstand diverse attacks or failures.

2.2 Adaptive BFT-SMR protocols

Throughout this section, we will discuss various related works that also provide support for the works of this thesis, highlighting their contributions to the advancement of Byzantine fault tolerance. Traditional BFT-SMR protocols statically define a value for the size of the system and how many faulty replicas the system will be able to withstand during design time, which limits their performance during deployment time when compared to less resilient systems. Although dual mode and abortable protocols can switch modes at runtime, they usually maintain a constant fault threshold f, which constrains their performance optimization capabilities. Architectural hybridization can help distributed systems to minimize the number of messages required for consensus while maintaining fault tolerance guarantees, thereby improving efficiency and facilitating scalability. These adaptable features are iteratively important improvements over classical protocols like PBFT and Paxos that will lead to more modern and robust systems closer to achieve dynamic adaptive fault tolerance.

2.2.1 Abstract and Aliph

When designing BFT protocols and applying them to different applications, it is normal to feel the need to improve the protocol our tailor it to the target application, and this specialization can have its cost. Protocols that passed the scrutiny of time and were extensively tested, like PBFT and Zyzzyva have massive codebases, and calculating the possible outcomes of a specific change has on the whole protocol is not practical. Abstract [Gue+10] was designed to solve these issues because it enables the system to have different protocol solutions for different deployment challenges.

Abstract stands for Abortable Byzantine Fault-Tolerant State Machine Replication and it leverages on system's reconfiguration to provide new adaption possibilities. The authors further create AZyzzva and Aliph which have an Abstract instance and multiple protocols that can be switched during runtime execution of the distributed system. These protocols can have different objectives like optimizing for safety or optimizing for throughput, even with their own agreement techniques widening even more the scope of capabilities.

The switch between protocols or instances could be done safely, but it was still a static design-time solution. In the case of Aliph, for example, it was initially able to switch between the protocols Quorum and Chain. However, if we wanted to add another protocol to the mix we would not be able to because it was something statically defined, Aliph could change from Quorum to Chain automatically but it was not able to add another protocol to the list dynamically. This is also a common problem for dual-mode protocols.

2.2.2 MinBFT

The use of hybrid components can be seen as strong assumption when compared with the possibility of having abortable protocols, but it can also bring problems with it, like having limiting processing power and restricted storage. Addressing these problems, the authors create Unique Sequential Identifier Generator(USIG), their own Trusted Computing Base(TCB) for MinBFT [Ver+11], one using very minimal code base and small number of operations. USIG aligned with the reduction of request stages required by MinBFT and MinZyzza impacts directly the protocol's latency expected.

Architectural hybridization used by MinBFT enables it to only require $2f + 1$ replicas to withstand f replicas. The unique identifier provided by the trusted component remove the system's requirement of having f replicas in the intersection of quorums to be able to detect equivocation happening. The trusted component also allows MinBFT to have one stage less than PBFT on its normal-case execution because it can use USIG operations to uniquely identify messages.

MinBFT showed it is possible for a distributed system to have Byzantine fault tolerance without using three times the number of nodes to cope with f faults with the add of a minimal hybrid element. It was also showed that the communication steps could be reduced from 5 to 4 in normal-case execution and from 4 to 3 on more speculative executions of their protocols. Networks with chaotic amount of load and shortcomings like the Internet today benefits a lot from protocols like those, that spend less traffic with messages and requires less resources to provide services in a distributed fashion.

2.2.3 CheapBFT and ReBFT

Dual-mode protocols usually operate, with one of the protocols being the safest configuration of the systems, and the other part is the efficient version where the protocol achieves higher throughput values. CheapBFT [Kap+12a] is a dual-mode protocol where the system can either operate with CheapTiny, the most resource-efficient protocol, or to safely switch, using their own protocol, to MinBFT, a heavier but safer protocol.

The transition from one execution mode to another execution mode is done by a panic message issued either by clients or replicas. This change can be motivated by protocol's slow progress, in the case where clients do not receive $f+1$ matching replies, or by safety where replicas become suspicious about faults happening in the system. Either way, an agreement is needed to seal the abort history of which messages were executed up to that point, enabling the system to continue its execution from a checkpoint to where the next payload protocol starts.

CheapBFT applies architectural hybridization called CASH in a similar manner of MinBFT's USIG, while both CASH and USIG are used to ensure that messages are uniquely identified during runtime execution of consensus algorithms. CASH uses an FPGA or a hardware-based mechanism to generate unique, increasing counters, while USIG uses a software-based mechanism to generate unique, sequentially increasing identifiers, and they both employ a specific type of signature. Though both mechanisms prevent equivocation from happening, CheapBFT benefits from using a lighter version of the protocol when there are no faults on the system, and as soon as entities suspect that faults are happening, the protocol agrees on requests executed and moves to a heavier and safer payload protocol.

ReBFT [DCK15] or the updated version of CheapBFT comes also to enhance the efficiency of Byzantine Fault Tolerant (BFT) systems by optimizing resource utilization. BFT systems ensure consensus among replicas in the presence of faulty or malicious nodes, but they often suffer from high communication and computation overhead. Several techniques proposed by other works can be used to save resources with ReBFT like batching requests and using hashes instead of PRE-PREPARE and COMMIT requests. Additionally, it also allows a speculative execution technique where replicas can concurrently execute requests that do not conflict with each other, reducing the average consensus latency considerably. Another key aspect is the global commit history mechanism, which dynamically enables passive replicas to become active, passive replicas are nodes that were once not participating in consensus but were present the whole time at the system's network and infrastructure. Experimental evaluations demonstrate significant improvements in resource efficiency without compromising the fault tolerance and security of BFT systems. By minimizing communication costs, optimizing computation, and adapting resource usage, ReBFT provides a practical framework for achieving resource-efficient Byzantine fault tolerance, making BFT systems more scalable and practical for real-world applications.

2.2.4 BFTSmart

Despite the astonishing amount of classical distributed system's algorithms published before BFTSMaRt [BSA14b], they were the first to make publicly available a full fledge open source library that would enable the creation of many other BFT

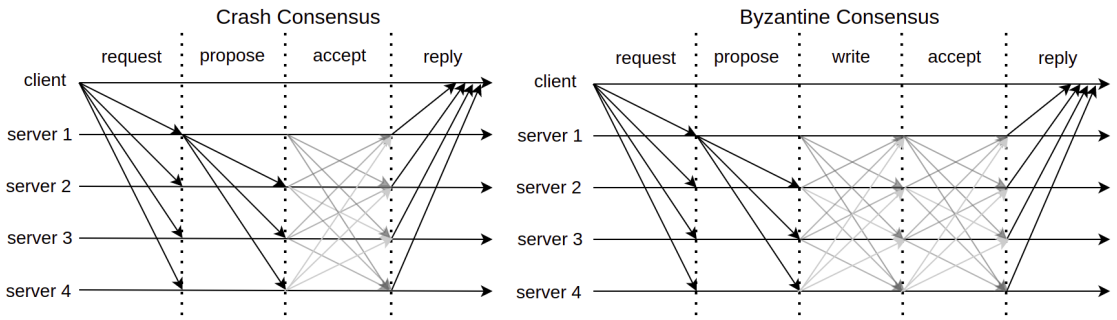

Figure 2.1: Comparison between different consensus algorithms, Crash fault-tolerant consensus does not require the write phase, different from Byzantine fault-tolerant consensus and both supported by default on BFTSMaRt library.

protocols. They choose to use a high level object-oriented language to increase the simplicity of the code, facilitate the re-use, to provide a library with a possibly smaller codebase. The modularity applied speeds up the creation of new protocols that improve specific aspects of the field, it gets easier to prototype, test new ideas without dealing with core mechanics of the system like the consensus algorithm if your new technique does not innovate on that. Replicas can be instantiated in different cores allowing this library to use industrial servers with strong processing power at full capacity and given that usually they have access to a well-connected network it also makes it easier to test and evaluate new developments. The exchange of messages or requests between entities and the ordering of these requests was also designed to be modular in the same way the fault model was programmed to be flexible, whether your system works with messages in batches or streamline of requests, whether it was thought to withstand crashing or Byzantine replicas. BFTSMaRt should be able to accommodate new protocols particularities without deeper changes.

All other BFT protocols explained before are using statically defined configurations for the size of the system and how many faults it will be able to withstand. BFTSMaRt [BSA14b] has its own module responsible for managing dynamic group membership actions and its own protocol to be able to add and remove replicas at runtime. The library does also uses a multi-threading approach where each replica can be using several threads depending on the amount of work services and operations are requiring from it, this way there is no throughput interferences if the replica is receiving too many requests because processing requests do not stall replicas' progress.

BFT-SMaRt library offers modularity and adaptability, making it a powerful tool for implementing both Crash Fault Tolerant (CFT) and Byzantine Fault Tolerant (BFT) consensus. It provides the flexibility to configure the consensus

behavior at the design stage, allowing developers to seamlessly switch between CFT and BFT modes by simply changing the configuration settings. This modularity enables system designers to choose the appropriate level of fault tolerance based on their application requirements. In CFT mode, BFT-SMaRt provides crash fault tolerance, where nodes can crash but do not exhibit Byzantine behavior, on this mode requests need also one phase less 2.1. In BFT mode, it guarantees Byzantine fault tolerance, allowing for the presence of malicious or faulty nodes. BFT-SMaRt's out-of-the-box capabilities in providing both CFT and BFT consensus, combined with its modular design, make it a versatile solution that can be tailored to various fault tolerance requirements. The simplicity applied to the development of BFTSMaRt made it to remain relevant even after half a decade of scrutiny from research and time. As result there is almost as many fork protocols as it can gets, with many of these adapting this library to be optimized for specific problem or application already existent, like wide-area application, throughput optimized protocols and latency-based optimized systems.

2.2.5 Adapt

Adapt[BGS15a] brings the adaptivity of systems to a higher level with its dynamically change of protocols on-the-fly, can be seen as a much improved version of Aliph, but it has much more to it. The creative combination of Byzantine fault-tolerant design with artificial intelligence brings a different perspective to the solution on how to tackle system's adaptability problem. This time differently from Aliph because Adapt has a list of available protocols that can be updated during runtime execution of the payload protocol, given the system manager provides numerical descriptions of the algorithm's features. It also adds an extra layer to answer the decision task of whether to abort the running protocol and go to the next one and to which protocol of the available ones it should change to.

Adapt may have dynamic switching mechanism, however the protocol depends strictly on the weights distributed in the characteristics of Throughput, Latency and Capacity. They have to do some training prior to run the real SMR system also to train the Machine Learning algorithm to better define values or weights for each one of the features. This partially restricts the dynamism of the system because if the administrator wants to add a brand-new protocol to the mix, he better give it accurate weights otherwise he could easily transform the dynamic change in a single point of failure of the system. Therefore, it is not clear how the addition of new protocols would be made, but it still expands possibilities to make BFT protocols more adaptive.

There are always two BFT protocols running with the same set of replicas one PBFT instance accounting for the quality control algorithm QCS and the other works with clients and requests like classical distributed systems are used

to. Adaptation here comes at the expense of throughput and latency to enable the system to take a distributed decision on which protocol would be the best fit to the system right at this specific moment. Adapt as well as Aliph, slightly achieve Adaptive Fault Tolerance one of the goals of this thesis, almost by accident because this adaptive improvement comes from the fact that different protocols implies on different system characteristics, like the fault threshold non-hybrid BFT systems is usually $3f+1$, speculative protocols can have $2f+1$ and so on, this work basically achieves adaptive fault tolerance by consequence.

2.3 Speculative Agreement protocols

We will describe here briefly how important Speculative agreement protocols are, their importance on pushing Byzantine fault tolerance to the masses given that they spend in normal-case scenarios less resources than PBFT for example. One of the critical challenges in distributed systems is reaching consensus in decision tasks among the distributed, interconnected nodes. To address these scalability issues, researchers have proposed speculative agreement protocols, which aim to reduce the communication overhead and increase the throughput of BFT-based consensus algorithms. These protocols leverage the speculative execution of transactions or requests to achieve faster agreement on the state of the system while maintaining the same level of security as BFT-based protocols.

2.3.1 Zyzzyva

In classical Byzantine Fault Tolerant (BFT) protocols, consensus must be reached in order to move from one request to the next. The system or clients wait for a majority of votes to make progress, this process can be time-consuming and resource-intensive. Zyzzyva [Abr+17a] aims to reduce this overhead by allowing nodes to execute transactions speculatively or without first receiving confirmation from almost the whole system before moving to the next transaction. This protocol uses a combination of two-phase voting and speculative execution, in the first phase, nodes broadcast their transactions to the network and collect votes from a subset of other nodes. These votes are used as a preliminary confirmation for the transaction, enabling the node to execute it speculatively. In the second phase, the node sends the transaction to all other nodes in the network and collects their votes. Only if the votes confirm the preliminary votes, the transaction will be then committed to the blockchain.

2.3.2 Thunderella

Combining speculative execution from Zyzzyva with the dual mode protocols architecture and we have Thunderella[PS18] where authors introduced a hybrid blockchain protocol that combines optimistic confirmation with a fallback mechanism for situations where optimistic confirmation is not feasible. Thunderella leverages a simple committee signature scheme and an asynchronous consensus protocol to achieve fast confirmation when favorable conditions are met, such as having a majority of honest network participants and an honest leader proposing transactions. In these cases, transactions can be quickly confirmed in just two rounds of communication. However, in less common scenarios, Thunderella falls back to an underlying traditional blockchain mechanism, such as proof of work or proof of stake. It can use proof-of-work as a slower fallback path and still optimistically confirm transactions much faster using the asynchronous consensus protocol when an accelerator node and three-fourths of an elected committee are honest. This hybrid approach allows Thunderella to balance between efficiency and security, leveraging speculative execution by optimistically confirming transactions in the fast path under favorable conditions, while still having a reliable and secure fallback mechanism in the slow path. By employing both optimistic and fallback strategies, Thunderella provides a flexible and efficient solution for blockchain consensus, addressing the need for instant confirmation while maintaining the robustness and resilience required in various scenarios.

2.4 Throughput-Based Optimization Protocols

Distributed systems will always face the challenge of maintaining high throughput while ensuring availability and safety. Byzantine fault tolerance (BFT) consensus algorithms are commonly used to address this challenge, but they can suffer from low throughput due to the communication overhead involved on the many stages required in the consensus process. To overcome this limitation, works have being proposed using throughput-based optimization procedures. Identifying the primary as a bottleneck, these protocols aim to increase the throughput of BFT-based consensus algorithms, usually by doing a view change where the next primary should have better processing and network capabilities than the actual primary. Prime, Aardvark, and RBFT are examples of protocols that make use of those performance measurements to achieve better throughput and in this section, we will explore these protocols in more detail and discuss their advantages and limitations.

2.4.1 Prime

A Byzantine primary can stall system's progress and jeopardize system's performance purposely. Prime[Ami+10] comes to address the performance degradation when under attack problem that happens on many known BFT protocols by periodically measuring primary's rate of ordering requests and measuring the latency between primary and pairs, in order to reconfigure the system to a more optimal state. These measurements allow the protocol to compute the maximum delay separating the sending of two ordering messages performed by a correct primary. The main objective of the protocol becomes to reduce this maximum delay as much as possible, improving as a consequence system's throughput, but this is only possible with correct round-trip-time measurements between pairs, even when Byzantine nodes are involved.

In classical distributed systems, protocols usually require the primary to be computationally very powerful and very well-connected to the rest of the nodes, because in many of those its primary's responsibility to prepare messages to be executed. The task of preparing messages involves two actions, the first one is to give a unique number to every request and keep track of these numbers to enable future primaries to continue from the numbers used, the second one is to send this message uniquely identified to every other replica on the network. When these two actions are well executed for a sequence of requests is when the system can reach its full throughput capacity. However, if the primary is faulty or lagging because of its own infrastructure, the system will be subsequently running at speeds far from optimal.

Prime applies periodic measurements to primary and the other nodes on the network, seeking to find the best suitable primary for this protocol execution moment. A primary with good processing capabilities should be able to order requests quickly and if its latency on messages exchanged with other replicas is good enough it will remain being the primary, otherwise after a period of time the system will change it automatically. The view change procedure done by PBFT was one of the heaviest parts of the protocol and done only when really required, here this procedure is way quicker, and it is actually preferred to spend resources changing primary than to hold protocol execution with a slow or badly connected node. It is then extremely important to Prime that replicas could measure precisely the round-trip-time of messages exchanged, if monitoring is inaccurate, part the robustness promised is lost, and it was seen that one faulty client and one faulty primary are able to degrade 22% of Prime's performance.

2.4.2 Aardvark

The ordering and forwarding requests to the respective replicas are responsibilities of a primary and they are not easy tasks to be performed fast at the system scale during runtime, Aardvark [Cle+09] addresses these problems by pushing the system to achieve higher and higher throughput numbers periodically. The performance requirements are updated every couple of seconds, and the leader at this moment in time has to be able to keep up, if it is not able to, the system will then reconfigure and change the leader. Eligible primaries will have their speed of ordering requests measured together with latency measurements between pairs, like in Prime, in order to enable the system to reconfigure to better configurations. It was also observed in this work that a malicious primary could still degrade the system's progress, but less when compared to Prime.

Aardvark leads the system to make a view change eventually by periodically checking the processing speed and quality of the network connection of the primary. The throughput expected from the protocol increases every period of time, decided at designing time, until the primary fail to fulfill the requirements, when this happens a view change is issued. Even with the system not processing any request during view change, the authors showed that Aardvark is still more efficient and more resilient to faults compared to Zyzzyva and PBFT.

There are protections built in place to protect system's progress from overload from clients or even from servers trying to spam the network or the system with fake requests. Aardvark also makes use of resource scheduling, separating the workload in different queues protecting the system against Byzantine clients trying to flood the system and also from replicas that may send to many messages given an arbitrary number of requests. It is also applied many checks to the request like the black list check that avoids blacklisted nodes from keep attacking the system, mac, sequence and retransmission check that protects the system against nodes re-sending the same messages or requests just to stall system's progress or to cause resource overload and many others interesting assertions are applied by Aardvark.

2.4.3 RBFT

Prime and Aardvark pushed the limits of the system's throughput to a greater level when compared with PBFT, they also showed that they are also more resilient but systems with dynamic load seem to be more prone to performance degradation caused by faulty entities. During attacks over those situations Prime decreases its performance in 78% and Aardvark 87%. Redundant Byzantine Fault Tolerance(RBFT)[AMQ13] comes as an alternative protocol to improve the former approaches, being able to hold system's performance under attack in scenarios where the request size varies or where it is at least bigger than 0 KB, which is

usually where huge performance numbers come from. RBFT follows an interesting and different infrastructure where each replica or server runs its own BFT protocol and there is a master instance of this protocol running on top of everything.

Changing the primary seems to improve the system if the next primary can be faster than the last one, problems start to happen if the measurements done were not accurate, leading the system to be guided by a faulty primary. One way to dodge the problem is to keep modifying primaries, like Spinning does but eventually, the system will elect a Byzantine replica as primary and the protocol should have protections in place to counter what ever possible damage a faulty primary can cause. Also, a faulty server or client should not have too much space to lie smartly and to not get caught by diminishing system's performance.

2.5 Weighted Voting Consensus

Weighted voting is a time-honored approach to making decisions that involve allotting each member of a group a certain number of votes, which are based on their expertise, wealth, level of authority, or knowledge [CP13]. This technique is frequently employed in various committees and organizations where certain individuals have access to more resources than others. In distributed systems, this can be represented by replicas that have more computational power, a superior location, or better network connectivity [Gif79]. The primary objective of weighted voting is to ensure that the most knowledgeable and competent members wield more influence in the decision-making process, resulting in more effective and efficient decision-making[Ber+20]. However, one potential drawback of this method is that it can result in uneven representation or disparities in voting power among group members. Therefore, it is necessary to carefully consider the criteria used to assign weight and ensure that they are equitable and aligned with the group's goals and objectives.

2.6 Latency-Based Optimization Protocols

Another way of increasing the throughput is usually to reduce the time it takes the system to reach a consensus in a distributed system, and that is the objective behind protocols using latency-based optimizations. These protocols work by identifying the root cause of latency, which could be attributed to factors such as network latency, processing delays, or message transmission times. By addressing the underlying cause of latency or empowering nodes better connected, these protocols aim to reduce the time it takes for a consensus to be reached while maintaining safety and availability. WHEAT [SB15a] and AWARE[Ber+19], for

instance, reduce the time to reach agreement by using a weighted voting scheme where nodes are split into two groups, and they do not need the majority of nodes to vote requests but a voting power threshold to be reached. Kauri[NMR21], on the other hand, uses a pipeline technique to scale the dissemination of requests and achieves by consequence also better latency values. Overall, these latency-based optimization protocols are promising solutions to the challenge of achieving faster consensus in distributed systems.

2.6.1 WHEAT

Traditional distributed algorithms for geo-replicated state machine replication infrastructures rely on theoretical models like Mencius [Bar08] and EPaxos [MAK13], do not fully capture the complexities of real-world systems. In WHEAT [SB15a] authors evaluated their distributed system in a close scenario to what is seen as real-world deployment example. This was the first protocol to use a weighted vote assignment scheme that impacts directly the fault threshold and the quantity of spare replicas required to improve consensus's latency and protocol throughput. To have good representations of the network connection between servers and enable the protocol to work efficiently with its workload, in WHEAT the latency measurements are done by system administrators at system's start. During their evaluation from the outcomes observed in both experimental setups the crash fault-tolerant scenario and the Byzantine fault-tolerant one, authors concluded that co-locating clients with the leader does not necessarily result in a latency improvement for replicated state machines. Instead, the placement of the leader in the host with superior connectivity to the remaining replicas may lead to more consistent enhancements. Therefore, the advantage of reaching the leader quickly is not as crucial as locating the leader in the server with faster connection links to the other replicas. The most interesting counter-intuitive innovation WHEAT shows is the capability of reducing protocol latency by adding more spare replicas to the system.

2.6.2 AWARE

AWARE[Ber+19] can be considered an updated version of WHEAT, much more capable and much more dynamic. This protocol was also created with a focus in geo-replicated applications, where the latency discrepancies resulting from servers connected at WAN in different locations are much greater than applications interconnected to local area networks. It uses weighted quorum like in WHEAT, but this time voting power is being updated on the fly or during the runtime of the protocol, it is also being updated autonomously without the need of system administrators involved. These changes happen proactively every period defined

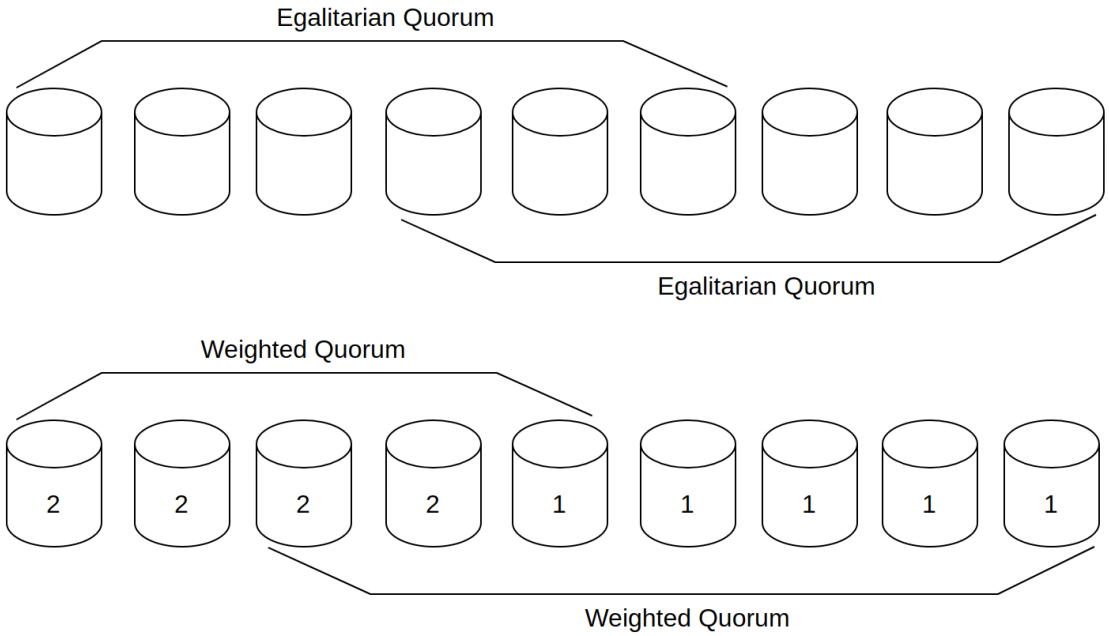

Figure 2.2: Classical consensus algorithms use egalitarian quorum where $\lceil \frac{n+f+1}{2} \rceil$ replicas have to vote to reach consensus, in AWARE's weighted quorum agreement can be reached even with $2f + 1$, usually requiring way less replicas.

during the designing time, and the system stops executing requests to execute the MonitoringWindow procedure where weights might be re-balanced. The criteria being used to provide more voting power to one arbitrary replica is the network connection with other nodes, better-connected nodes receive more voting power, and worst-connected nodes will receive less voting power. This difference in weights makes the quorum formation much more variable, which is seen as a characteristic that leads the system to more efficiency.

The MonitoringWindow is the procedure that happens every period of time defined during system's designing time, during this phase all nodes measure the round-trip-time of messages between them and others in a pair-wise manner. AWARE's standard period for MonitoringWindow is one after one execution per second, by the results of their experiments it seems to be enough to make the system optimal without spending too many resources or wasting valued payload protocol time. During the execution of this phase, the protocol stops executing payload requests and messages exchanged result on latency measurements that will further end up in a latency matrix. Before deciding on the next weight distribution that will favor the system and make it runs faster, nodes have to agree on the latency matrix, they start by sharing their local values and then they agree on a common matrix. Given that at this part of the procedure enough nodes have the

same measurement matrix, they can calculate the most optimal next configuration by themselves. The optimal configuration or close to optimal weight distribution is calculated by every replica using Simulated Annealing, where for each configuration is calculated how much it would be its consensus latency and gathering the distribution of all these values the annealing algorithm would enable replicas to find a good approximation of the global optimum. To close this procedure, they should agree on the next configuration, issuing with that a reconfiguration request, that will move every corresponding node to a new view.

In AWARE the voting weight distribution is split between two groups the one having minimal voting power called V_{min} and the one having maximal voting power called V_{max}. Nodes that are reportedly worst-connected with the rest of the network receive V_{min} voting power, nodes are shown good network performance during the MonitoringWindow will receive maximal voting power. V_{min} is equal to 1 and $V_{max} = 1 + \frac{\Delta}{f}$, where Δ is the number of spare replicas connected also to the system. Usually in classical distributed systems the more replicas a system has the more latency it will take to reach an agreement, but as observed in WHEAT the use of spare replicas together with weighted consensus improves the latency of payload protocol and AWARE as an updated version of WHEAT does also have the same advantage.

In conclusion, AWARE is an enhanced and dynamic version of WHEAT, designed specifically for geo-replicated applications that experience significant latency discrepancies due to wide area network connections. It utilizes weighted quorums, with voting power updated autonomously during runtime based on the network connections of each node. The protocol includes a MonitoringWindow procedure to measure latency and calculate an optimal weight distribution using Simulated Annealing. By splitting voting power into minimal and maximal based on network performance, AWARE achieves improved efficiency and reduced consensus latency compared to other classical distributed systems, making it a valuable solution for geo-replicated applications.

2.6.3 Kauri

While AWARE focuses on adaptively adjusting voting weights according to node's network connectivity, Kauri relies on latency between nodes and processing time to disseminate and aggregate messages in a pipeline architecture to achieve scalable Byzantine fault-tolerant consensus in distributed systems. Kauri employs a combination of dissemination and aggregation trees to efficiently propagate and gather information across the network. Dissemination trees are responsible for broadcasting messages from the leader to all nodes, while aggregation trees are used to collect and combine individual responses from nodes. These trees are dynamically constructed and adapt to changes in the network, ensuring efficient message dis-

semination and aggregation. To ensure the security and reliability of the consensus process, Kauri utilizes cryptographic techniques such as digital signatures and verifiable secret sharing. These mechanisms prevent malicious behavior and ensure the integrity of messages exchanged among nodes. Experimental evaluations confirm Kauri's scalability, outperforming existing BFT consensus protocols in terms of throughput and latency while maintaining strong consistency guarantees.

Dissemination and aggregation trees serve as decisive components of Kauri for efficient message propagation and information gathering. Dissemination trees are dynamically constructed hierarchical structures that facilitate the broadcasting of messages from the leader to all nodes in the network. Differently from a star topology in Kauri each node in the dissemination tree receives messages from its parent and forwards them to its children, ensuring reliable and efficient communication across the network, without broadcasting. Conversely, aggregation trees collect and combine individual responses from nodes, enabling the formation of a consensus on the collected information. These trees adapt to changes in the network topology and facilitate parallel processing, allowing for overlapping operations and reducing the overall consensus time. By leveraging dissemination and aggregation trees, Kauri achieves scalable and fault-tolerant consensus, ensuring consistent knowledge propagation and efficient information gathering in large-scale distributed systems. The construction and maintenance of these trees involve careful algorithms and protocols to optimize efficiency, fault tolerance, and load balancing, enhancing the protocol's overall performance and reliability.

2.7 Virtual Coordinate Systems

Virtual Coordinate Systems (VCS) such as Vivaldi [Dab+04] and Newton [Sei+13] are innovative approaches that aim to provide efficient and scalable solutions for network coordinate estimation in large-scale distributed systems. VCS algorithms seek to assign synthetic coordinates to network nodes based on network latency measurements, allowing nodes to estimate their relative positions in a coordinate space with good accuracy and a low error rate. Vivaldi, for instance, employs a spring-mass model where nodes iteratively adjust their virtual coordinates to minimize the discrepancy between estimated and actual latencies. On the other hand, Newton leverages Newton's laws of motion to compute node coordinates, simulating a physical force-based model where nodes attract or repel each other based on latency measurements. These VCS algorithms offer scalable solutions by enabling a reference set of nodes to estimate distances and paths in the absence of global knowledge or complete knowledge of where exactly is each and every node. They have been applied in various distributed systems, including peer-to-peer networks and content delivery networks, to improve routing efficiency, resource allocation,

and fault tolerance. While each VCS algorithms have its unique characteristics, they all contribute to the advancement of network coordinate estimation techniques and they will further, in this thesis, play an essential role in enhancing the performance and scalability of large-scale distributed systems.

2.7.1 Vivaldi

Vivaldi is a virtual coordinate system or an algorithm capable of issuing synthetic coordinates to nodes in the network, based on the latency matrix where each value is the pairwise measurement between every two nodes present in the system. It adopts a distributed approach to address the limitations of centralized systems, employing a spring-mass model where nodes are assigned virtual coordinates. These coordinates represent the node's position in a high-dimensional coordinate space. Through periodic exchange of coordinates between a reference set neighboring nodes, Vivaldi uses latency measurements to iteratively adjust the virtual coordinates, minimizing the discrepancy between estimated and actual latencies. This decentralized and iterative process allows nodes to gradually converge to a configuration that accurately reflects the network topology. Vivaldi's decentralized nature enables scalability and adaptability, making it resilient to network dynamics such as node failures or additions. The accuracy and efficiency of Vivaldi have been validated through extensive evaluations, making it a valuable tool for improving routing efficiency, load balancing, and resource allocation in various domains, including content delivery networks, peer-to-peer systems, and overlay networks.

The algorithm utilizes a spring-mass model shown in Figure 2.3 where a force vector is used to estimate network distances between nodes. Each node maintains a set of coordinates that represent its position in a virtual coordinate space. The force vector consists of attractive and repulsive forces that guide the movement of nodes towards an optimal configuration. By iteratively adjusting the coordinates based on these forces, the algorithm aims to achieve an accurate representation of the network topology. Additionally, the algorithm incorporates an adaptive timestep mechanism to dynamically adjust the rate at which nodes update their coordinates. The timestep determines the speed at which nodes converge to their optimal positions. Nodes more confident of their coordinates or with more stable latencies have smaller timesteps, allowing them to converge more slowly and reduce unnecessary coordinate oscillations. On the other hand, nodes with volatile latencies have larger timesteps, enabling them to adapt more quickly to changing network conditions.

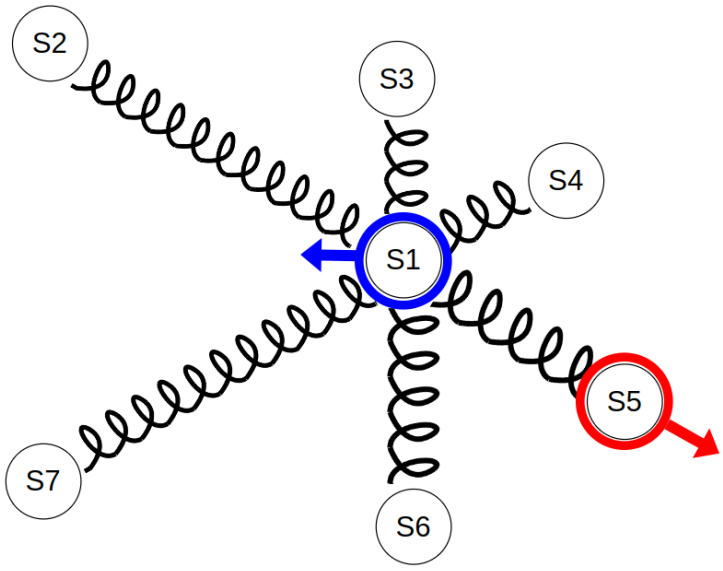

Figure 2.3: The mass-spring model is inspired by physics, where each node in the network is treated as a mass connected to other nodes by virtual springs. The model assumes that nodes strive for an equilibrium state where the forces exerted by the springs and their distances are balanced.

2.7.2 Newton

Newton extends on Vivaldi presenting a novel approach to secure virtual coordinate systems by leveraging physical laws. VCS algorithms, which assign virtual coordinates to network nodes based on latency measurements, are susceptible to malicious attacks that aim to manipulate node positions and disrupt network operations. In response, Newton introduces a physical force-based model where nodes simulate the attraction and repulsion forces based on their estimated latencies. By enforcing Newton's laws of motion, nodes iteratively adjust their virtual coordinates to minimize the discrepancy between estimated and actual latencies. This approach enhances the security of VCS by utilizing physical laws as a fundamental constraint, making it challenging for malicious nodes to alter their positions without affecting the overall consistency of the coordinate system. Newton also introduces mechanisms to handle potential inconsistencies caused by adversaries, allowing nodes to detect and mitigate malicious behavior. Experimental evaluations demonstrate the effectiveness of Newton in providing secure virtual coordinates, resisting attacks, and maintaining the accuracy of the coordinate system. By combining physical laws with virtual coordinates, Newton offers a promising solution for enhancing the security and reliability of distributed systems, particularly in scenarios where the integrity of node positions is crucial, such as overlay

networks, peer-to-peer systems, and content delivery networks.

Chapter 3

Threat adaptive Byzantine fault tolerant state-machine replication

Critical infrastructures have to withstand advanced and persistent threats, which can be addressed using Byzantine fault tolerant state-machine replication (BFT-SMR). In practice, unattended cyberdefense systems rely on threat level detectors that synchronously inform them of changing threat levels. However, to have a BFT-SMR protocol operate unattended, the state-of-the-art is still to configure them to withstand the highest possible number of faulty replicas f they might encounter, which limits their performance, or to make the strong assumption that a trusted external reconfiguration service is available, which introduces a single point of failure. In this chapter, we present ThreatAdaptive the first BFT-SMR protocol that is automatically strengthened or optimized by its replicas in reaction to threat level changes. We first determine under which conditions replicas can safely reconfigure a BFT-SMR system, i.e., adapt the number of replicas n and the fault threshold f, so as to outpace an adversary. Since replicas typically communicate with each other using an asynchronous network they cannot rely on consensus to decide how the system should be reconfigured. ThreatAdaptive avoids this pitfall by proactively preparing the reconfiguration that may be triggered by an increasing threat when it optimizes its performance. Our evaluation shows that ThreatAdaptive can meet the latency and throughput of BFT baselines configured statically for a particular level of threat, and adapt 30% faster than previous methods, which make stronger assumptions to provide safety.

3.1 Introduction

Cyber infrastructures, which are used in domains such as finance, public administration (e-government), social networks, or e-health, as well as cyber-physical

systems (CPS), such as the power grid or autonomous vehicles, increasingly face cyber attacks of various threat levels [Coo14]. Some of these attacks might successfully compromise a subset of the machines used in an infrastructure, which imposes periodical verification of a system's integrity and protection measures. However, the nature of cyber-physical systems, and the increasing complexity of both cyber-physical and cyber-only systems prevent manual attack surveillance and mitigation [Acc20]. Instead, systems have to operate through times of ongoing and possibly persistent incidents autonomously and unattended.

Fluctuations in the threat a system faces naturally arise from variations of environmental effects (e.g., radiation levels vary while planes taxi on ground and during flight [Jon05]), or when it comes to attacks, from the number and skill of adversarial actors having put their attention to a system and from the sophistication of the tools they use. In practice, cyber physical systems rely on threat detectors [11] that indicate the level of threat they are facing and that allows them to automatically adapt their performance and resilience.

As discussed briefly on Chapter 2, Byzantine fault-tolerant state machine replication (BFT-SMR) [CL+99], combined with rejuvenation [SNV06] and diversification methods [Hos+18; GBN19] can be combined and used to replicate servers and tolerate powerful adversaries. BFT-SMR protocols are typically configured with $n \geq 3f+1$ replicas to tolerate the largest number f of faulty replicas [CL+99] it might encounter, and require $2k$ additional replicas (i.e., $n \geq 3f+2k+1$) if up to k replicas are simultaneously rejuvenated every T_R seconds [SNV06]. However, the latency and throughput of BFT-SMR systems deteriorate when f increases. Our work is the first to allow BFT-SMR protocols to safely adapt to evolving threats by leveraging threat detectors.

Traditional protocols [CL+99] define a value for f once and for all at deployment time, and therefore have a lower performance than less resilient systems. Adaptive protocols [Ber+19] reclaim some performance, but they also maintain f constant. To some extent, dual mode and abortable protocols [Kap+12a; DCK15; Aub+15; BGS15b] can optimize their performance by executing a protocol switch at runtime, but they also keep the fault threshold f constant. On the other hand, group membership protocols adjust the system and quorum sizes using consensus [Cha+96; Rei96b; DGM02]. However, group membership protocols cannot guarantee that the system will enter a sufficiently resilient configuration before it gets compromised, in particular when network synchrony is lost. Finally, relying on an external reconfiguration service introduces a single point of failure.

Fortunately, as we show in this chapter, it is possible to circumvent the limitations of these approaches by leveraging threat detectors, which provide a lightweight trusted functionality. Threat detectors issue warnings well in advance of imminent increases of adversarial strength [MC14; Deh18; Boe18; 16]. However, threat detec-

tors also report about threat level decreases, when there is room for optimization.

On this chapter, we describe how to adapt distributed systems to fluctuating adversarial threats. We start by carefully analyzing the timing properties a threat detector should have to allow the system to react in time to adversarial strength increases. Building on those insights we describe a reconfiguration protocol, ThreatAdaptive, which allows the replicas of a system to: (i) use consensus to agree on and switch to a less resilient, but better performing configuration that is still resilient enough when possible; and (ii) return to a configuration that is strong enough, which has been agreed upon during the optimization phase, when the threat detector informs them about an imminent increase of adversarial strength. ThreatAdaptive allows a BFT-SMR protocol to save the resources it does not require to ensure safety at a given moment in time while increasing its efficiency by operating with fewer active replicas whenever possible.

Overall, this chapter makes the following contributions: (i) We establish when and how it is safely possible to reconfigure BFT protocols when the adversarial strength evolves over time. (ii) We present ThreatAdaptive, a BFT reconfiguration protocol that allows replicas to optimize its use of system resources and increase its resilience threshold(with respect to the perceived adversarial strength). (iii) We implement our Threat-Adaptive system design and evaluate its performance.

This chapter is organized as follows. §3.2 presents our system model and objectives. §3.3 discusses threat detectors and presents the required conditions for the safe reconfiguration of a BFT-SMR protocol that relies on rejuvenation. §3.4 describes our threat-adaptive reconfiguration protocol for BFT-SMR protocols. §3.5 presents our performance evaluation. §3.7 concludes this chapter.

3.2 System Model and Problem Statement

3.2.1 System Model

We assume a replicated service for which replicas coordinate agreement using a payload BFT-SMR protocol. We use PBFT [CL+99] for illustration purposes, but our dynamic membership protocol is generic and can be applied to other BFT-SMR protocols, or even to reliable broadcast protocols [Gue+20]. Replicas are interconnected by a partially synchronous network. We assume the availability of strong cryptographic primitives and abstract from the steps necessary to establish trust in replicas and their key material, initially and after they have been rejuvenated.

The configuration of a BFT-SMR system is captured by a tuple $(N_i, f_i, q_i, k_i, T_{R,i})$, where N_i is the set of active replicas that execute the payload protocol, f_i is the fault threshold, q_i is the quorum size, and, to support rejuvenation, k_i and $T_{R,i}$

respectively denote the number of replicas that can be rejuvenated simultaneously and the time a proactive rejuvenation of k_i replicas takes. After a duration of $\left\lceil \frac{n_i}{k_i} \right\rceil T_{R,i}$, the system has proactively rejuvenated all n_i replicas once, and at most k_i simultaneously. In general, the values of k_i and $T_{R,i}$ can be freely chosen as long as one takes into account that replicas cannot be rejuvenated arbitrarily quickly.

For simplicity, we focus here exclusively on homogeneous payload protocols (i.e., there are no trusted components that replicas can use for the payload protocol, but they rely on a threat detector for reconfigurations), where $q_i \leq n_i - f_i$ and $2q_i - n_i \geq f_i + 1$ must hold for safe and live configurations, which implies that $n_i \geq 3f_i + 2k_i + 1$ and $q_i \geq 2f_i + k_i + 1$. We assume an initial set of replicas N_{max} and define a World Configuration $C_{max} = (N_{max}, f_{max}, q_{max}, k_{max}, T_{R,max})$ as the initial system configuration. We require the world configuration to be safe, live and capable of masking faults and rejuvenating replicas as this will be the configuration the system returns to in the most severe circumstances. During times when the system experiences synchrony, group membership protocols can be used to adjust this world configuration. Finally, we assume that the system is equipped with a threat detector (TD), which we discuss in §3.3.

3.2.2 Threat Adaptive Reconfigurations

We consider an adversary whose strength evolves over time. We consider two ways in which this evolution can happen: i) the adversary can corrupt less or more replicas overall (the adversarial fault threshold f_{adv} evolves); and ii) the adversary requires less or more time to corrupt f_{adv} replicas. We capture those two aspects in the notion of adversarial strength, which is a function $T_A(t, f)$ of time t and fault threshold f. We do not instantiate the strength function for our analysis (cf. §3.3).

The system can be configured to run the payload protocol either more efficiently or with more resilience. A replica r transitions through configurations and into a passive but responsive operation mode [Kap+12a; DCK15], e.g., a deep sleep mode with wake-on-LAN, if it is not involved in the current configuration (i.e., if $r \notin N_i$). We assume that passive replicas are correct upon wake up and substantiate this assumption by frequently rejuvenating passive replicas. Such a rejuvenation is only limited by the rejuvenation time T_R and not by k_i. Passive replicas can be rejuvenated simultaneously.

Classically, system reconfigurations are decided by a system administrator. We opt for an automatic and internal reconfiguration orchestrated by the replicas themselves. Assuming an external reconfiguration service only relocates the problem we tackle, since this service has to be safe at all times. In order not to introduce a single point of failure, this service would also need to be replicated

and either configured to the maximum fault threshold or have the capability to reconfigure itself.

Our goal is to define a resilient system able to automatically react to evolving threats safely and rapidly enough to outpace the adversary in its attempt to compromise the system. Replicas reconfigure the system to counter imminent threats, or to optimize its performance when it is safely possible, based on a threat detector's signal. We call *reactions* and *optimizations* the adaptations in consequence of an increasing or decreasing adversarial strength, respectively.

3.3 Threat Detectors and Requirements for Adapting to Changing Adversarial Strength

This section first discusses threat detectors. We then present our threat model, which encapsulates the notion of a changing adversary, and state more precisely the threat detector requirements (i.e., how well in advance it should warn of a treat change) using the proactive recovery threat model.

3.3.1 Threat Detectors

Threat detectors are fundamentally different from intrusion detectors. Whereas the latter has to identify whether parts of the system have been compromised, threat detectors merely have to assess the risk of severe faults happening.

Threat levels have been defined by Singer as a product of the estimated capabilities of malicious actors and their intent [Sin58]. Defense against cybersecurity threats requires identifying such actors, their points of entry, attack vectors and known vulnerabilities of the system to protect. The perception of adaptive threats requires continuous monitoring of changes of malicious actor capabilities (i.e., whether new, exploitable vulnerabilities have been exposed or old ones patched), as well as changes of a malicious actor's intent (system in focus of enemy military or intelligence actors, higher/lower black market financial incentives for breaching the system).

The answer to this threat perception challenge lies in implementing Cyber Threat Intelligence, which allows for continuous and responsive cybersecurity information collection, dissemination and processing, and, as a result, enables educated decisions on how to prepare the system to face (perceived) threats [Boe18]. Operational frameworks (STIX) and standards of information exchange (TAXII) have been designed to automatically evaluate threat levels [Qam+17]. Open threat feeds [Gou; 11] are already available and the feasibility of such systems is confirmed by their existence in the frameworks of notable institutions like the Bank of England [16].

3.3.2 Centralized Threat Detectors

Centralized threat detectors are a possible solution for enhancing the resilience of distributed systems and two existent solutions that could be integrated are threat-intelligence providers like FireEye and ThreatConnect. These platforms enable organizations to proactively detect, analyze, and respond to cyber threats, ultimately strengthening the security posture of distributed systems. Being used as tools in threat intelligence-based frameworks, they promote collaboration, knowledge sharing, and preparedness, allowing organizations to improve their incident response capabilities and develop effective countermeasures. The European framework for threat intelligence-based ethical red-teaming framework TIBER-EU provides a comprehensive standardized framework for risk assessment, centralized continuous monitoring, and event response coordination in the financial sector.

FireEye can test your's systems defenses and help improve its security posture by conducting TIBER-EU tests that mimic the tactics, techniques, and procedures of real-life attackers, revealing the strengths and weaknesses of system's infrastructure and enabling it to reach a higher level of cyber maturity. FireEye and its consultants' team use cyber threat intelligence to develop and execute their testing plans. This process ensures that the testing efforts, findings, and observations are aligned with the tested entity's real-world threat profile. FireEye monitors hundreds of threat groups, including over 40 advanced persistent threat groups, ten financial threat groups, and hundreds of uncategorized groups. Comprehensive profiles of these threat groups are built and maintained and include target industries, attack motivation, tools, and procedures.

Differently from FireEye, ThreatConnect is not an application but more like a platform that offers centralized threat intelligence services. It aggregates data from various sources, including open-source intelligence, commercial feeds, and internal sources, to provide comprehensive and contextualized threat intelligence. ThreatConnect's capabilities give organizations a holistic view of the threat landscape, identify potential risks, and prioritize mitigation efforts across distributed systems. ThreatConnect also facilitates proactive threat hunting and incident response by correlating threat intelligence with security events and indicators of compromise. By centralizing threat intelligence management, organizations can streamline their incident response efforts, making them more coordinated and efficient. This centralized approach helps to identify and respond to threats across distributed systems, ensuring a unified and cohesive defense strategy.

During the development of this work, we took as an assumption the existence of a threat detector connected to replicas sending the actual threat level. One possible idea, out of the scope of this thesis, would be to integrate FireEye or ThreatConnect, acting as a trusted centralized controller resembling what was done in Lazarus [GBN19], enabling a robust threat intelligent approach to secu-

rity and resilience. Even more, advances could be made if we connect the diversity management provided by the trusted controller in Lazarus. The centralized threat detection and incident response capabilities of FireEye and ThreatConnect would ensure a comprehensive defense against potential attacks and vulnerabilities. The combination of diverse replicas, continuous monitoring, and threat-intelligence applications would help the distributed system designers to identify and create mitigation mechanisms capable of responding to threats proactively, enhancing the overall resilience and security of the BFT systems.

3.3.3 Distributed Threat Detectors

FireEye and ThreatConnect are tools that can act as a centralized threat-intelligence applications, but Byzantine fault-tolerant systems should be able, in the forceable future, to distribute the threat detection skills over the replicas on the network. We could consider two fault models one homogeneous with the threat detector being as trustworthy as any other replica or a hybrid fault model where the threat detector would be a hardware component attached to each and every replica. Without the presence of trusted component means that threat detectors would have their own network and their own BFT protocol running, like Quality Control System (QCS) network works in Adapt to update their machine learning weights. With threat detectors being a trusted component, we could think of them sharing any threat suspicions with others whenever any different status is detected, in a better be safe than sorry way. Each model has its own implications and benefits.

In the homogeneous fault model, threat detectors function as independent entities with dedicated networks and BFT protocols. They utilize their local knowledge and detection capabilities to identify potential threats within their replicas. Drawing inspiration from the QCS network in the Adapt system, threat detectors can dynamically update the number of faulty replicas or which threat level the system is handling at this moment. These actions could bring the system to safer or more efficient configurations.

The hybrid fault model involves integrating trusted threat detectors as hardware components within each replica. These threat detectors can share their threat suspicions with other replicas in real-time whenever they detect a different status or abnormal behavior. This collaboration ensures that any potential threats or anomalies are promptly communicated across the network, allowing all replicas to remain informed and take necessary precautionary measures.

By distributing threat detection skills over the replicas, BFT systems can achieve several benefits. First, the system becomes more resilient to attacks, as the compromise of a single replica or threat detector does not compromise the entire system's detection capabilities. Second, the distributed nature of threat detection ensures scalability, allowing the system to handle large-scale deployments with

ease. Third, it promotes adaptability and agility, as threat detectors can update their detection algorithms independently, keeping pace with evolving threats.

For the sake of this chapter, the threat detector (TD) used on our experiments generates and delivers indications of changes in the perceived adversarial strength T_A to replicas. TD endpoints at each replica are connected through a synchronous network, that is separated from the regular partially synchronous network that replicas use for the payload protocol. This separated network is a wormhole [Ver06], which on our system is primarily used for sending TD signals, but is also is used for waking up passive replicas and could also be used for sending state update packages that keep passive replicas up to date with the last executed commands from payload protocol.

3.3.4 Adversarial strength

We now determine how much in advance replicas should be informed of an increasing adversarial strength so that they can reconfigure the system to maintain it safe. Sousa et al. [SNV06] assume that during any time interval of duration T_A the adversary can compromise at most f replicas. A system with n replicas is then said to be *exhaustion safe* if all faulty replicas are repaired faster than T_A. In the absence of perfect failure detectors, this can be achieved by rejuvenating all n replicas proactively faster than T_A. For example, if k replicas can be rejuvenated simultaneously well within T_R, the system is exhaustion safe if and only if $\lceil \frac{N}{k} \rceil \ T_R \leq T_A$. In this scheme, the number of replicas should be at least $n \geq 3f + 2k + 1$ to ensure safe and live quorums.

We extend Sousa et al.'s model and their exhaustion safety notion by characterizing the combined adversarial strength T_A as a function of time t. However, while their adversary model faces systems with a constant fault threshold f, we strive for systems that adapt f in response to changing adversarial strength. Therefore, to understand how strong an adversary of strength $T_A(t)$ is against different configurations of the systems, $T_A(t)$ must itself be a function mapping each configuration's fault threshold f to the length of the time interval during which no more than f replicas can be compromised. That is, $T_A(t, f)$ resembles Sousa et al.'s time interval for an adversary with strength $T_A(t)$ at time t when it faces a system that is capable of tolerating up to f simultaneous faults.

Definition 1 (Adversarial strength)**.** Let $T_A : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ be a function that maps every point in time $t \in \mathbb{R}$ and every $f \in \mathbb{N}$ to a duration $T_A(t, f)$ such that at time t the adversary cannot corrupt more than f replicas before a duration $T_A(t, f)$ has elapsed. We shall assume in this work that T_A remains constant for extended periods of time and that the duration of such a period $P_i = [t^i, t^{i+1})$ is larger than $T_A(t, f)$ for all involved fault thresholds f (e.g., when the system transitions from

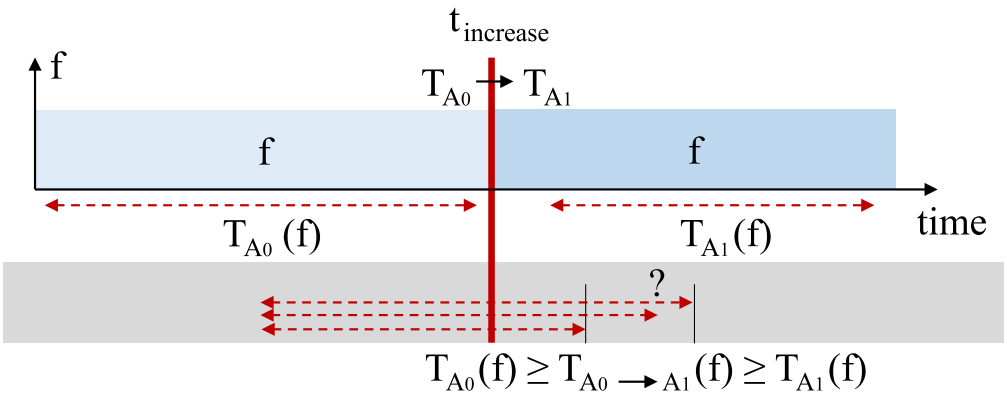

Figure 3.1: Time to compromise before, during and after the increase of adversarial strength at $t_{increase}$. During red-dashed intervals, the adversary cannot compromise more than f replicas. The length of intervals including $t_{increase}$ is not known precisely and hence requires careful consideration.

f_1 to f_2). We call $T_A(t, f)$ the *strength of the adversary* during this period against a system with fault threshold f.

In the following, for simplicity, we shall write $T_{A_i}(f)$ instead of $T_A(t, f)$ where $t \in P_i$ and $T_{A_{i+1}}(f)$ for $T_A(t, f)$ while $t \in P_{i+1}$, to denote these noticeable changes in the adversarial strength during subsequent periods of time P_i, P_{i+1}. We say that the adversary becomes ***stronger*** relative to a given f if it evolves from being characterized by T_{A_i} and is now characterized by $T_{A_{i+1}}$ where $T_{A_{i+1}}(f) < T_{A_i}(f)$. We define similarly a ***weaker*** adversary. We require T_{A_i} to be monotonic (i.e., $T_{A_i}(f_0) \leq T_{A_i}(f_1)$ for all $f_0 \leq f_1$), but make no further assumptions on T_{A_i} (such as linearity).

We define the time an adversary takes to corrupt f replicas:

Definition 2 (Time-to-compromise intervals)**.** A time-to-compromise interval $[t_l, t_r)$ is any interval, between t_l and t_r such that the adversary cannot corrupt more than f replicas. If T_{A_i} remains constant between t_l and t_r it implies that $t_r - t_l \leq T_{A_i}(f)$.

The duration of time-to-compromise intervals that include a change in the adversary's strength requires careful consideration. We denote by $T_{A_i \to A_{i+1}}(f)$ the adversarial strength over such an interval and require only that $T_{A_i}(f) \geq T_{A_i \to A_{i+1}}(f) \geq T_{A_{i+1}}(f)$. Fig. 3.1 illustrates this point.

In the following, we discuss how a system can react to an increasing adversarial strength, namely: i) by increasing the rejuvenation rate; or ii) by activating additional replicas. For each situation, we describe how the system should be reconfigured, and obtain a temporal bound before which the reconfiguration should

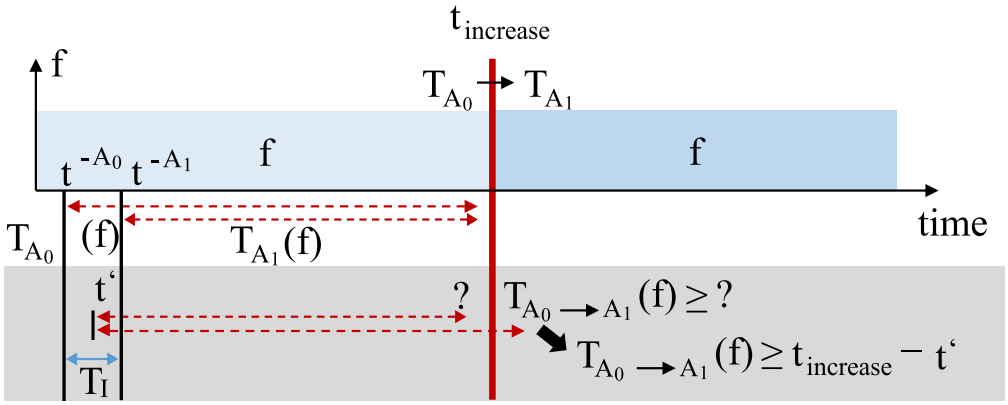

Figure 3.2: Increase of adversarial strength.

be effective. We finally combine both results to identify all possible reconfigurations of the system.

3.3.5 Accelerating rejuvenation

Let us assume an adversary that becomes stronger at time $t_{increase}$, characterized through the functions T_{A_0} and T_{A_1}. In particular, we have $T_{A_0}(f) \geq T_{A_1}(f)$. We first study how and when the rejuvenation parameters can evolve so that no more than f replicas are simultaneously faulty. This situation is illustrated in Fig 3.2.

Exhaustion safe systems allow up to f faults to happen in any window of size $T_{A_0}(f)$ that starts before $t^{-A_0} = t_{increase} - T_{A_0}(f)$, and in any window of size $T_{A_1}(f)$ that starts after $t_{increase}$, provided the system is as well exhaustion safe with the adjusted rejuvenation rate and the strong adversary. We now investigate what happens over the time intervals where the adversary becomes stronger, i.e., those that contain $t_{increase}$. Those intervals start at a time t such that $t^{-A_0} < t < t_{increase}$, and have a duration comprised between $T_{A_1}(f)$ and $T_{A_0}(f)$. We can therefore over-approximate the adversary's strength and state our first theorem:

Theorem 1 (Reacting by accelerating rejuvenation)**.** *From an exhaustion safe configuration* (N, f, q, k_0, T_{R_0})*, if the adversary's strength evolves from* T_{A_0} *to* T_{A_1} *at time* $t_{increase}$*, the system remains safe if it is reconfigured to an exhaustion safe configuration* (N, f, q, k_1, T_{R_1})*, where* $\left\lceil \frac{|N|}{k_1} \right\rceil T_{R_1} \leq T_{A_1}(f)$ *before* $t_{increase} - T_{A_1}(f)$*.*

Proof. We know that before $t_{increase}$, the additional adversary strength has no effect. Therefore, there exists an interval included in $T_I = [t^{-A_0}, t^{-A_1} = t_{increase} - T_{A_1}(f)]$ such that the time-to-compromise windows that start in this interval (e.g.,

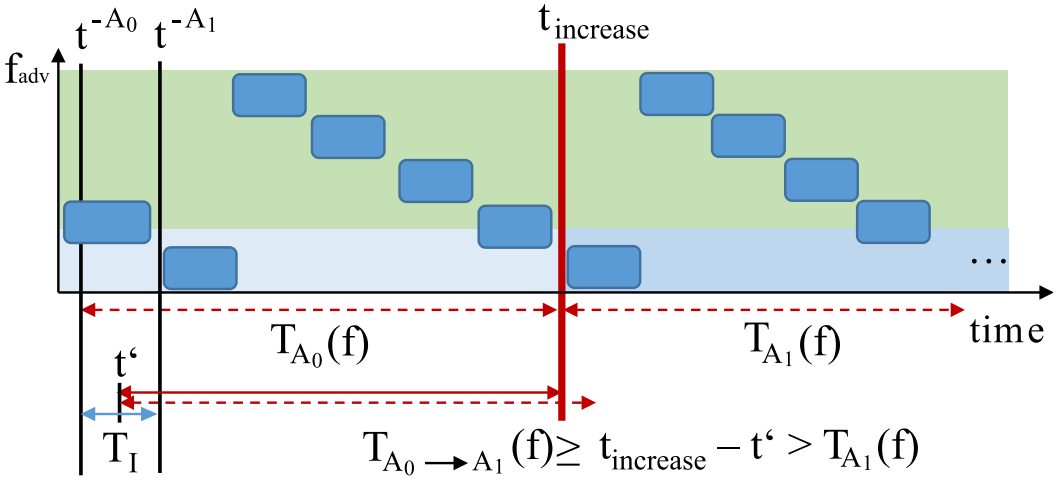

Figure 3.3: More frequent rejuvenation to counter a stronger adversary. A blue rectangle shows the rejuvenation of a replica.

at $t' \in [t^{-A_0}, t^{-A_1}]$) cannot compromise the system in less than $t_{increase} - t'$ because this window is entirely included in the interval where the adversary is still weak.

The existence of this interval allows us to derive the point in time when the system has to adjust its proactive rejuvenation rate to counter the increase of adversarial strength. Having over-approximated the adversary's strength with $T_{A_1}(f)$ for any window that starts after t^{-A_1}, any attack launched during any of these windows is countered by rejuvenating all replicas in N faster than $T_{A_1}(f)$ (i.e., after adjustment, $\left\lceil \frac{|N|}{k_1} \right\rceil T_{R_1} \leq T_{A_1}(f)$ holds, which guarantees exhaustion safety for all sliding windows after t^{-A_1}).

The system does not have to switch to this rate before t^{-A_1} because all time-to-compromise windows that start in the interval T_I contain interval $[t^{-A_1}, t_{increase}]$ during which all n replicas are rejuvenated. The time-to-compromise windows that start earlier than t^{-A_0} find all replicas repaired before $t_{increase}$ since we assume the weak system to be exhaustion safe while the adversary is weak, and their possible overlap with interval $[t^{-A_1}, t_{increase}]$ only speeds up rejuvenation, Figure 3.3 illustrates this point. It is important to note that despite the change of rejuvenation parameters of the configuration (from k_0 to k_1 and from T_{R_0} to T_{R_1}), the order in which replicas are rejuvenated must be preserved. □

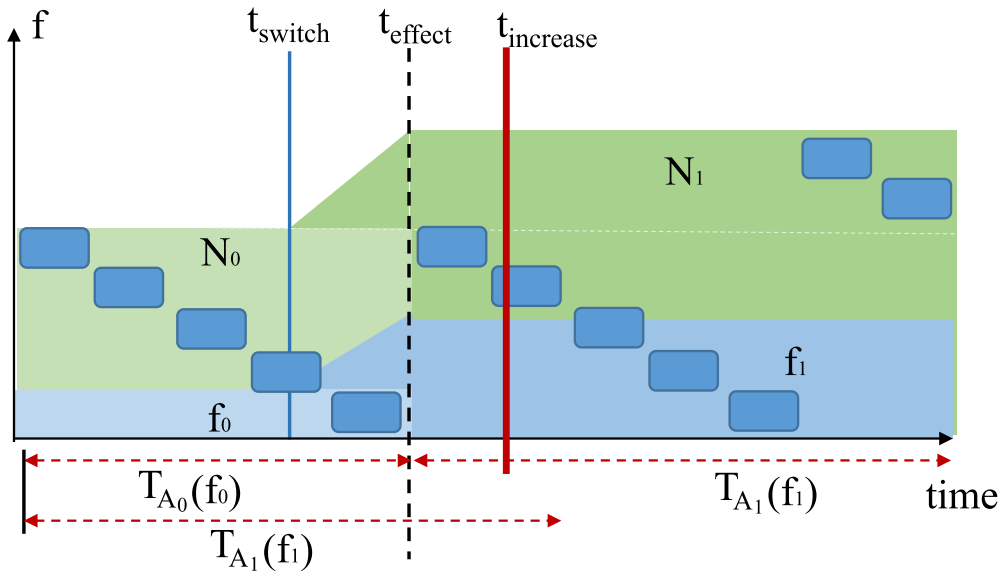

Figure 3.4: Adding replicas to counter a stronger adversary.

3.3.6 Adding replicas

Replicas cannot be rejuvenated arbitrarily fast. For example, Garcia et al. found that rejuvenating a replica running Ubuntu 16.04 requires 40s [GBN19]. Therefore, it might happen, that a system is already rejuvenating replicas as fast as possible. In this case, it will no longer be possible to react only by accelerating the rejuvenation rate. Fortunately, the system can also react to an increasing adversarial strength by increasing its fault threshold, i.e., by adding replicas. 3.4 illustrates this strategy and Theorem 2 captures its effectiveness.

Theorem 2 (Reacting by adding replicas)**.** *From an exhaustion safe configuration $C_0 = (N_0, f_0, q_0, k, T_R)$, if the adversarial strength evolves from T_{A_0} to T_{A_1} at time $t_{increase}$, the system remains exhaustion safe if it is reconfigured before $t_{increase}$ to a configuration $C_1 = (N_1, f_1, q_1, k, T_R)$, such that C_1 is exhaustion safe relative to the stronger adversary (i.e., $\left\lceil \frac{|N_1|}{k} \right\rceil T_R$ holds), provided that $T_{A_0}(f_0) \leq T_{A_1}(f_1)$, and replicas continue to be rejuvenated in the same order.*

Proof. The system starts with $n_0 = |N_0|$ replicas and fault threshold f_0, and is reconfigured to involve $n_1 = |N_1|$ replicas with fault threshold f_1. Let t_{switch} be the point in time where the system starts reconfiguring itself, and t_{effect} the time when the new configuration is operational and capable of tolerating f_1 faults. During a time-to-compromise window that starts before $t_{effect} - T_{A_0}(f_0)$ the replicas in N_0

are rejuvenated by the rejuvenation scheme of the weak configuration. Similarly, time-to-compromise windows that start after t_{effect} are exhaustion safe by our assumption that the strong configuration fulfills this property.

We now consider the windows that overlap with t_{effect}. Let us assume that f_1 was chosen so that $f_0 < f_1$ and $T_{A_0}(f_0) \leq T_{A_1}(f_1)$, and that the rejuvenation order of replicas is maintained through the reconfiguration. In this case, any window that starts before t_{effect} and goes beyond $t_{increase}$ does not lead to the compromise of more than f_1 replicas.

The windows that start in $[t_{effect} - T_{A_0}(f_0), t_{effect}]$ overlap with the new configuration. However, their length is larger than $T_{A_1}(f_1)$, because the adversary's strength over them is sometimes lower than T_{A_1}. However, since replicas that get activated at t_{effect} are correct, maintaining the same rejuvenation order ensures that the replicas that were active in the previous configuration, and which might be faulty, are rejuvenated first. In fact one could regard such a correct instantiation as a free simultaneous and instantaneous rejuvenation at t_{effect}. As a consequence, no more than f_1 replicas get compromised within any window of size $T_{A_1}(f_1)$ that starts after $t_{effect} - T_A(f_0)$, since all n_1 replicas in N_1 are rejuvenated in such a window. □

3.3.7 Accelerating rejuvenation and adding replicas

We have seen that the system can adjust the rejuvenation rate to $T_{A_1}(f_1)$ before time $t_{increase} - T_{A_1}(f_1)$ so that the $|N_0|$ replicas are rejuvenated over all time-to-compromise windows located before or containing $t_{increase}$ (Theorem 1). We have also seen that replicas can be added so that they are active before $t_{increase}$, so that the $|N_1|$ replicas are rejuvenated over all time-to-compromise windows that start after $t_{increase}$ (Theorem 2). These results can be combined as follows.

Theorem 3 (Reacting by accelerating rejuvenation and adding replicas)**.** *From an exhaustion safe configuration $(N_0, f_0, q_0, k_0, T_{R_0})$, if the adversarial strength evolves from T_{A_0} to T_{A_1} at time $t_{increase}$, the system remains safe if it is reconfigured to an exhaustion safe configuration $(N_1, f_1, q_1, k_1, T_{R_1})$, where*

1. *f_1 is such that $f_0 \leq f_1$ and $T_{A_0}(f_0) \leq T_{A_1}(f_1)$*
2. *the rejuvenation frequency is changed before $t_{increase} - T_{A_1}(f_1)$ so that $\left\lceil \frac{|N_1|}{k_1} \right\rceil T_{R_1} \leq T_{A_1}(f_1)$*
3. *the passive replicas become active before $t_{increase}$*
4. *replicas that appear in both configurations are rejuvenated in the same order after t_{effect}*

Proof. by Theorem 1 and 2. □

3.3.8 Consistent and Synchronized Information of Replicas

In addition to advance notice, consistency of the information replicas receive and possible synchronization requirements are further concerns to be considered in the threat detector network. Clearly, if all replicas receive the same information about imminent increases/decreases of adversarial strength at the same time, no confusion can arise when some lagging replicas are still reacting while others already optimize their configuration. However, such a time synchronized response introduces significant complexity and overhead in the threat detector wormhole.

In Section 3.4.4, we therefore show how our protocol avoids this complexity, by first returning to the strongest required configuration since the replica last reacted to increasing threats before allowing this replica to advance with optimizations since then. Threat detectors inform replicas about both the strongest required configuration and whether it is safe to enter the targeted configuration during optimization.

3.4 A Threat-Adaptive Reconfiguration Protocol

Assuming the availability of a threat detector that reports threat increases sufficiently in advance, as discussed in the previous section, we now present our reconfiguration protocol that replicas use to reconfigure the system.

3.4.1 Intuition

Receiving an adversarial strength decrease signal from the threat detector allows replicas to optimize performance and switch into a more efficient configuration that involves fewer active replicas and possibly an adjusted rejuvenation scheme. If threat detection provides replicas with sufficient confidence that T_A remains stable for a significant duration, they can optimize their performance by executing the payload in a less resilient but more efficient configuration. In the event of threat-level increasing, the system switches back to the former safer configuration, executing the payload in a more resilient but less performant configuration. Replicas involved in this configuration remain active and return to the payload protocol after adopting the rejuvenation scheme of this configuration. Replicas not involved in this configuration silence themselves by not participating on payload protocol, but remain attentive to activation requests. In the meantime, they rejuvenate themselves without coordinating with other replicas to remain available (remember, in configuration C_i active replicas are limited to rejuvenate at most k_i replicas at a time; passive replicas are not constrained in this way).

After the threat-detector wormhole informs the replicas about an imminent increase of adversarial strength, correct replicas must react within a bounded amount

of time (see Section 3.3.5), which, as we have seen, rules out using consensus. Key to circumventing this impossibility is the observation that the decision about how to react to a given threat can be already taken well before the threat manifests itself. In this chapter, we will suggest preparing for the reaction to increasing adversarial strength well ahead when reaching consensus about how to optimize. Clearly, a first decision must be taken at this point in time, but it is as well possible to revisit this decision several times during the execution of the target configuration (e.g., to compensate for permanent failure in passive replicas).

For simplicity, in this chapter we shall only discuss reaction by returning to the very same configurations from where the system came from. That is, if the system, starting in the world configuration C_{max} progressed through a sequence of configurations $C_1, C_2, \ldots C_i$, where C_i is the current configuration. It will react by returning along this chain (i.e., from C_i to C_{i-1} to $C_{i-2}, \ldots$) until it has reached a configuration that is capable of withstanding the currently reported adversarial strength. Since any such chain starts with C_{max}, which is by definition the most resilient configuration, it is always possible to find such a configuration, provided the system can withstand an adversary of this strength in the first place.

3.4.2 Reaching Consensus for Optimizations

The goal of our reconfiguration protocol is to safely transition the system from its current configuration C_s to a proposed new configuration C_t, which is resilient enough to withstand the current adversary of decreased strength. At the same time, the protocol prepares for a later increase of adversarial strength by creating the possibility to return to C_s and to previous configurations in the chain without having to reach consensus. In general, the replica set N_t of configuration C_t does not need to be a subset of the current configuration's replica set N_s. In particular, quorums formed in N_t do not have to be safe quorums in N_s and vice versa.

Our reconfiguration protocol has the following properties.

P.1 Replicas can prove to lagging replicas (and clients) that C_t is the next configuration.

P.2 If C_t becomes active, enough replicas in this configuration must know it so that they can report any progress made in C_t in case the system requires to return to C_s. Otherwise replicas in C_s or in a previous configuration would wait forever for the progress of C_t and the system would not be live.

P.3 If C_t becomes active, enough replicas in C_s need to know about this fact so that no quorum can be formed in C_s to agree on a second configuration C'_t. Otherwise, both C_t and C'_t could become active simultaneously and the system would not be safe (i.e., clients could receive inconsistent replies if both C_t and C'_t return to the payload protocol).

```
1  reconfiguration (v,seq):
2    // leader s_l ∈ N_s
3    compute C_t
4    propose ⟨PrePrepareCfg, l, v, seq, C_s, C_t⟩_σ(s_l)
5    // backups s_i ∈ N_s
6    relay PrePrepareCfg as ⟨PrepareCfg, i, v, seq, C_s, C_t⟩_σ(s_i)
7    // all replicas s_i, s_l ∈ N_s
8    wait for q_s matching Pre/PrepareCfg messages
9    if TD.is_valid(C_t)
10     send ⟨CommitCfg, i, v, seq, ℂ_s→t⟩_σ(s_i)
11          to all replicas in N_s ∪ N_t
12   // new mode replicas s_j ∈ N_t
13   wait for q_s matching CommitCfg messages
14   if ℂ_s→t is valid
15     send ⟨ConfirmCfg, j, v, seq, ℂ_s→t⟩_σ(s_j)
16          to all replicas in N_s
17   // witnesses s_i ∈ N_s
18   wait for n_t matching ConfirmCfg messages
19   send ⟨AckCfg, i, v, seq, ℂ_s→t⟩_σ(s_i) to all replicas in N_t
20   if s_i ∉ N_t wait passively for C_t to return
21   // new mode replicas s_j ∈ N_t
22   wait for q_s matching AckCfg messages
23   resume payload in C_t with view v + 1
```

Figure 3.5: Reconfiguration Protocol.

We provide P.1 thanks to configuration certificates $\mathbb{C}_{s\to t}$, which loosely resemble view-change certificates of PBFT. To enforce P.2 all $n_t = |N_t|$ replicas of the target configuration C_t need to participate in a configuration certificate such that at least $n_t - f_t$ correct replicas are able to report any progress made in C_t. The payload protocol liveness is guaranteed despite waiting for a response from all n_t replicas in C_t, as long as subsequent optimization attempts are interleaved with minimal progress in the payload protocol. Note that an optimization may never succeed if target configurations continue to include non-responsive replicas, but the above guarantees progress nonetheless. Property P.3 requires a quorum of replicas in the source configuration C_s to witness the activation of C_t. We shall return to this aspect in 3.4.5.

Fig. 3.5 shows the pseudocode of our optimization reconfiguration protocol. Let s_l be the current leader of the payload protocol (i.e., the x^{th} replica in the set N_s where $x = v \bmod |N_s|$). We assume that the leader s_l progressed in view v to request sequence number $seq - 1$ and proposes with seq the reconfiguration of

the system as $\langle \mathit{PrePrepareCfg}, l, v, \mathit{seq}, C_s, C_t \rangle_{\sigma(s_l)}$ (Ln 4). Here $\langle \ldots \rangle_{\sigma(s_i)}$ denotes a message signed by replica s_i. C_s, C_t are the source and target configurations. Proposing this configuration in view v and with sequence number *seq* ensures that backups (i.e., non-leader replicas) will try to optimize the system at the same relative point in time. Following PBFT, backups relay the leader proposal as $\langle \mathit{PrepareCfg}, i, v, \mathit{seq}, C_s, C_t \rangle_{\sigma(s_i)}$ (Ln 6) before both leader and backups in C_s wait for q_s matching *PrePrepageCfg* and *PrepareCfg* messages from different replicas (Ln 8). Forming configuration change certificate $\mathbb{C}_{s \to t}$ out of these q_s messages, the replicas in C_s validate with their threat detector TD, whether C_t is a valid configuration given the current adversarial strength (Ln 9) and if so, they send this certificate in a *CommitCfg* message to both the replicas of the source configuration and all target configuration replicas (Ln 10), which wake up passive replicas in N_t.

Replicas of both configurations then proceed with a handshake to transition the system to C_t. Replicas of the target configuration C_t confirm the receipt of a valid configuration change certificate (Ln 14) and the replicas in the source configuration C_s acknowledge the receipt of such confirmations from all replicas in C_t (Ln 18-19). We call ***witnesses*** the source configuration replicas that have seen these confirmations. Whereas target configuration replicas resume execution of the payload protocol in view $v + 1$ (Ln 23), replicas that are not in the target configuration become passive but they can still react to messages (Ln 20).

For better readability, we omit timeout handling in Fig. 3.5. All participating replicas (i.e., all $s_i \in N_s$ and after activation all $s_j \in N_t$) set a timer when engaging in the configuration change. Timeouts cause replicas to abort their attempt to change to configuration C_t, which typically leads replicas in C_s to retry the configuration change. However, there are two exceptions: (i) having sent *AckCfg* witnesses wait for replicas from C_t to return, even if their timeout fires (Ln 31). This ensures that progress in the target configuration C_t is not lost; and (ii) replicas in the target configuration C_t return if the timeout fires during the configuration change (see 3.6 and Ln 24–29). They will stop doing so after having received q_s matching *AckCfg* messages, which marks a successful reconfiguration.

3.4.3 Reacting to Increasing Adversarial Strength

Reaction closely resembles the switch protocol of ReBFT [DCK15] extended to traverse the chain of visited configurations to the more resilient one(see Section 3.3).

Once in the targeted safe configuration, replicas resume executing the payload protocol, possibly after catching up with the progress their peers made relative to the history, which has to be done using the synchronous wormhole. The construction of local histories lh_j and the application of the global history gh depend on the payload protocol (Ln 27, 35). For example, for PBFT, the latest checkpoint

```
24 on TD strength increase or reconfig timeout:
25   // replica s_j ∈ N_t in view v at seq
26   stop payload protocol
27   create local history lh_j
28   send ⟨History, j, v, seq + 1, lh_j⟩_σ(s_j) to all replicas in N_s
29   stop processing view v messages
30   // replica s_i ∈ N_s
31   wait for q_t History messages
32        from different replicas (=Fig.4 Line 22)
33   combine lh_j into global history gh
34   if C_s is strongest visited
35     apply gh
36     resume in C_s with view v + 1
37   else continue returning to the next config.
38          in the chain with lh_i = gh
```

Figure 3.6: Protocol to react to increasing adversarial strength, by returning from C_t to the configuration C_s that activated C_t.

and the progress made since then have to be reported. Prepared messages (i.e., those receiving q_t matching *PrePrepare* or *Prepare* messages from replicas in C_t) are executed. Client requests not in this state are proposed again by the leader of the current configuration. The combined global history merely reports all received local histories and confirms with the signature of s_i the transition if C_s is not the strongest visited.

Notice that although the return may proceed through several configurations in the chain, only the final configuration will resume the payload protocol. Moreover, the replicas initiating this process will continue being rejuvenated in the pattern of their configuration while it is active. All other replicas and those that become passive will undergo frequent reconfigurations without first having to coordinate with others. Therefore, between the moment where the reconfiguration starts t_{switch} and the moment t_{effect} where it is effective even if multiple configurations are passed, Theorems 2 and 3 hold because none of these transitionally passed configurations become active in the sense of entering the payload protocol.

To prevent a client from ever accepting replies from a faulty quorum of replicas, in particular after a threat increase, replicas drop the private key they were using to interact with the client when they receive a threat increase notification. Replicas then generate a new set of keys to interact with the clients and broadcast the public keys to other replicas. To identify the currently active configuration, clients contact the world configuration and successively walks down the optimizations.

Note that we could also rely on a forward-secure digital signature scheme [KT20] to replace the keys of replicas.

3.4.4 Lagging Replicas and Successive Reconfigurations

One important detail concerns passive or lagging replicas that have not been able to enter the current configuration before the system optimizes to decreasing threats. Without further precautions, faulty replicas could activate a configuration with the lagging replicas while agreeing on a different operation to activate with the replicas that aim to optimize the system. To avoid this issue, replicas do not engage in optimizations before they have returned to the stronger configuration that can withstand the increasing adversarial strength. (see Section 3.3.8). From this configuration, lagging replicas follow the configurations the system was reconfigured into until they reach the currently active configuration or become passive in the progress. Only in this active configuration will they resume participating in potential optimizations.

The amount of requests executed in other system configurations can increase severely, the time being used to install payload checkpoint updates into passive replicas when reconfiguring toward resilience. An alternative could be to have a deadline for optimized configurations, and whenever it expires, the system would automatically reconfigure back, enabling passive replicas to catch up without jeopardizing liveness. Another alternative would be to keep sending checkpoints every x amount of requests from the leader to passive replicas, avoiding, in this way, the cumulative difference between executed requests on active and passive replicas.

3.4.5 Witnesses

To conclude the discussion of our protocol, let us give further details on the witness role (Ln 17–20) and show how it ensures property P.3, i.e., safety and liveness of the system.

Liveness. Replicas in the target configuration C_t react to increasing adversarial strength once they receive q_s matching *CommitCfg* messages (Ln 13). However, they only start processing the payload protocol after receiving the same number of q_s matching *AckCfg* messages (Ln 22). This step ensures that replicas from C_t will already return control to C_s, even if they are not yet ready to advance to entering the payload protocol. Conversely, witnesses in C_s will only acknowledge the configuration transition if they are sure that enough replicas in C_t will report the progress C_t might make (Ln 13ff). This is the case after all n_t replicas confirmed, because then $n_t - f_t \geq q_t$ correct replicas are guaranteed to communicate back their progress. In combination, this ensures liveness, even if the target configuration C_t is only partially activated.

Safety. We have to ensure that no two configurations ever execute the payload protocol (i.e., configurations may only be simultaneously active as part of the above transition protocols). Witnesses ensure this by refusing to participate in re-executing the reconfiguration protocol in 3.5 (e.g., for agreeing on a different configuration in case the optimization failed) unless the activated target configuration returned. For a partially activated target configuration (i.e., one receiving only fewer than q_s acknowledgments from witnesses), this is the case after the reconfiguration timeout expires. But in this case, already n_t replicas of C_t confirmed, which guarantees that those witnesses that have acknowledged C_t receive a correct history (although with no progress made since the payload protocol did not restart). This refusal to participate in a re-election before C_t returns ensures safety.

Replay Attacks. Replay attacks would not lead to the activation of two configurations. Key to preventing such replay attacks is the fact that to obtain all messages required for the activation (i.e., q_s *AckCfg* messages), a quorum in C_t must confirm this configuration. From this moment onward, replicas no longer react to messages with the confirmed view v after replicas in C_t returned (see Line 32 in 3.6). Moreover, correct witnesses do not produce *AckCfg* messages before they receive confirmation from C_t. Consequently, if the required messages are available, C_t can be activated only until the point in time when C_t returns, but such a return is necessary for the witnesses to resume in the protocols of Fig. 3.5 and 3.6 and hence to agree on a different configuration to activate.

3.5 Performance Evaluation

Setup. We implemented our reconfiguration protocol, which we name Threat-Adaptive, on top of BFT-SMaRt [BSA14a]. For our experiments we use 4 Ubuntu 18.04+ desktops each equipped with an Intel i7 6th generation processor and 8 GB of memory. The desktops are interconnected with a Linksys WRT54G router. For each experiment, replicas are evenly distributed on the machines. We ran BFT-SMaRt's microbenchmarks with 160 clients that send 100 Bytes requests every $150ms$ to keep replicas continuously busy. Messages are delivered in batches of 1024 and experiments start with fresh views. Our goal is to demonstrate that despite the threat adaptation mechanisms we described, the performance of the payload protocol is maintained close to the respective native configurations.

Baselines. We compare the performance of Threat-Adaptive to the following five baselines.

1. *BFT-SMaRt-4* and *BFT-SMaRt-10* are unmodified BFT-SMaRT [BSA14a] deployments with n=4 (i.e., f=1) and n=10 (i.e., f=3) without rejuve-

Protocol	Safety depends on
BFT-SMaRt-4	#`faults` ≤ 1
BFT-SMaRt-10	#`faults` ≤ 3
StoppableBFT	sysadmin being faster than attackers to reconfigure (not guaranteed)
Group Membership	reaching consensus during attack (not guaranteed)
ReBFT	#`faults` $\leq f$ (mode if #`faults`$\neq 0$)
Speculative	#`faults` $\leq f$ (gracious execution if #`faults`$=0$)
ThreatAdaptive (this work)	Safe if TD signal arrives early enough so that Thm. 3's conditions are verified.

Table 3.1: Safety conditions of the protocols considered.

nation (i.e., k=0). *BFT-SMaRt-6* and *BFT-SMaRt-12* are the equivalent configurations (i.e., f=1 or 3) with rejuvenation and k=1.

2. *StoppableBFT* stops the system execution before restarting it in a new configuration. This protocol emulates the unrealistic scenario where a sysadmin would immediately reconfigure the system before each threat increase.

3. *GroupMembership* is a consensus-based reconfiguration protocol, similar to Rampart et al. [Rei96a].

4. *ReBFT* [DCK15] is the optimistic mode of BFT-SMaRt where the system runs with $n-f$ replicas (here 7), but would need to return to a configuration with $n = 10$ replicas to handle faults.

5. *Speculative* implements an optimization presented where clients wait for $3f+1$ replies [Abr+17b]. Replicas rollback and re-execute requests if these replies are not received within a bounded time.

Metrics. We measure the throughput and latency of the payload protocol, in particular when reconfigurations happen, and the delay that is required to react to increasing threat levels. We measure latency as the average time required to process all client requests received within a one second interval at the replica side. To improve the readibility of the graphs, we plot the average of 11 measurements per sliding-window and indicate the standard deviation σ in the figure captions.

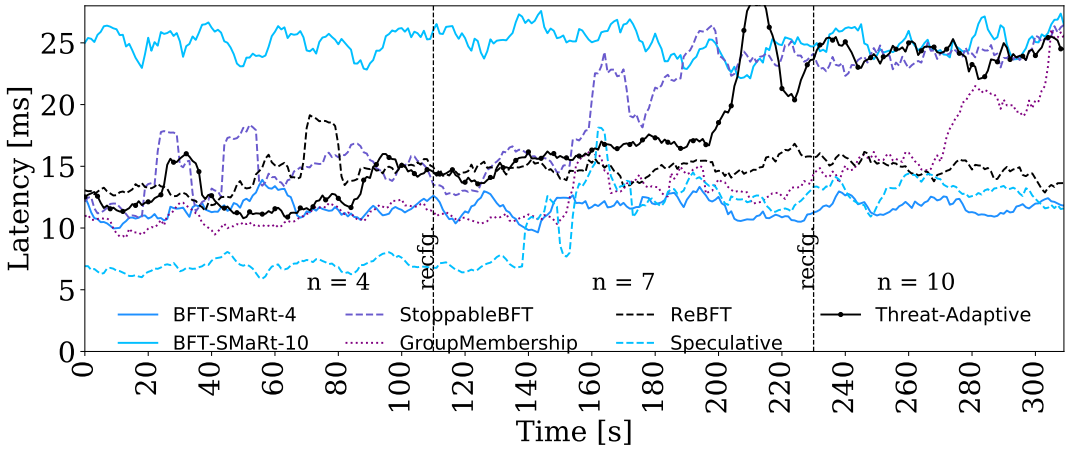

Figure 3.7: Latency with increasing adversarial strength. σ=3 during transitions, and 2 otherwise (no rejuvenation).

3.5.1 Reacting to Increasing Threat Levels

We first consider the situation where the threat level increases. Fig. 3.7 illustrates ThreatAdaptive's latency when the threat level increases from f=1 to 2 at 110s, and from f=2 to 3 at 230s. ThreatAdaptive's latency evolves from the one of BFT-SMaRt-4 before 110s to the one of BFT-SMaRt-10 at 230s, while obtaining an intermediary performance between 110s and 230s, and remains stable despite the reconfigurations. Protocols with a lower latency are those that do not guarantee safety or that are executing without faulty replicas. To clarify this point, we summarize in Table 3.1 the safety conditions for the baselines and ThreatAdaptive in that situation. The GroupMembership and the BFT-SMaRt-4 baselines may no longer be safe in the scenario we consider, and StoppableBFT would be safe only if the administrator reacts quickly enough. GroupMembership replaces replicas using consensus, which would always finish in time if consensus is provided as a functionality of a synchronous wormhole. ThreatAdaptive assumes a simpler synchronous wormhole whose task is to inform replicas sufficiently in advance of a threat level increase so that histories can be transmitted in time (through the synchronous wormhole) to the replicas involved in the next configuration. We now precise this condition.

3.5.2 Outpacing Adversaries

Table 3.2 shows the time required for ThreatAdaptive to return to a safe configuration after the adversarial strength increases. We evaluate two extremes supported by our witness scheme: (i) *inclusive* returns from C_t to an inclusive configuration

System	Recfg. increasing f ($n=3f+1$)			
	1 → 2	**2 → 3**	**3 → 4**	**4 → 5**
GroupMembership	3561	3656	3825	3948
ThreatAdaptive (inclusive)	2404	2445	2589	2781
ThreatAdaptive (overlapping)	2690	2856	3012	3379

Table 3.2: Reaction time ($t_{effect}-t_{switch}$) in ms (no rejuvenation).

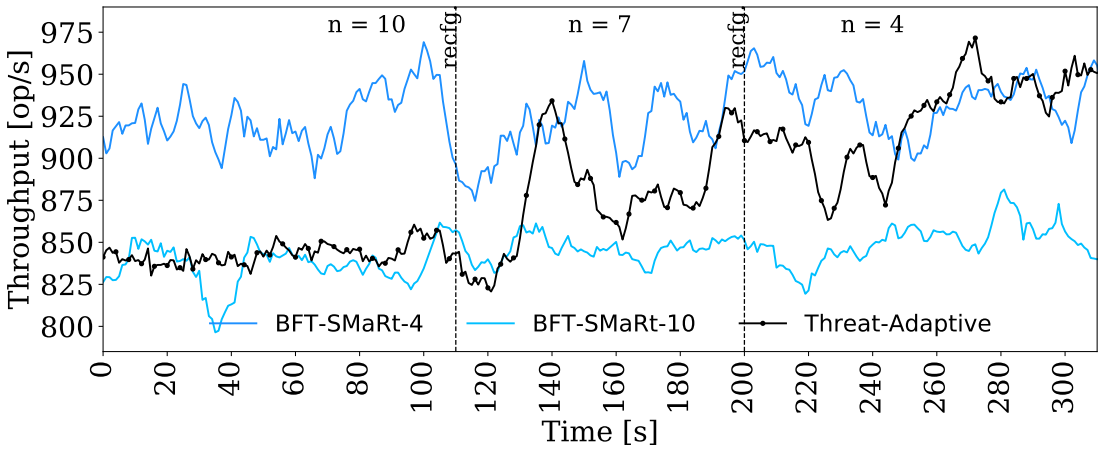

Figure 3.8: Throughput with optimizations at 110s and 200s (no rejuvenation, σ=65 at 110s and 200s, and 34 otherwise).

C_s, where $N_t \subseteq N_s$; and (ii) ***overlapping*** returns to a configuration that activates passive replicas that have never been active in previous configurations. The ***inclusive*** scheme outperforms GroupMembership by 30% since the reconfiguration decision is made in advance. As expected, replicas s_i that remain active in both configurations (i.e., $s_i \in N_t \cap N_s$) speed up the reconfiguration process. These results precise condition 3 of Theorem 3 to indicate the time required for replicas to transmit their histories and for passive replicas to become active. For example, the threat detector would need to inform replicas that the threat level evolves from f=1 to 2 only 3,561 ms in advance with the GroupMembership baseline, whereas with our approach (with the inclusive witness scheme) only 2,404 ms would be required.

3.5.3 Optimization Reconfiguration

We now consider the case where the threat level decreases from f=3 to 2 at 110s, and from f=2 to 1 at 230s. Fig. 3.8 shows that the throughput of ThreatAdaptive

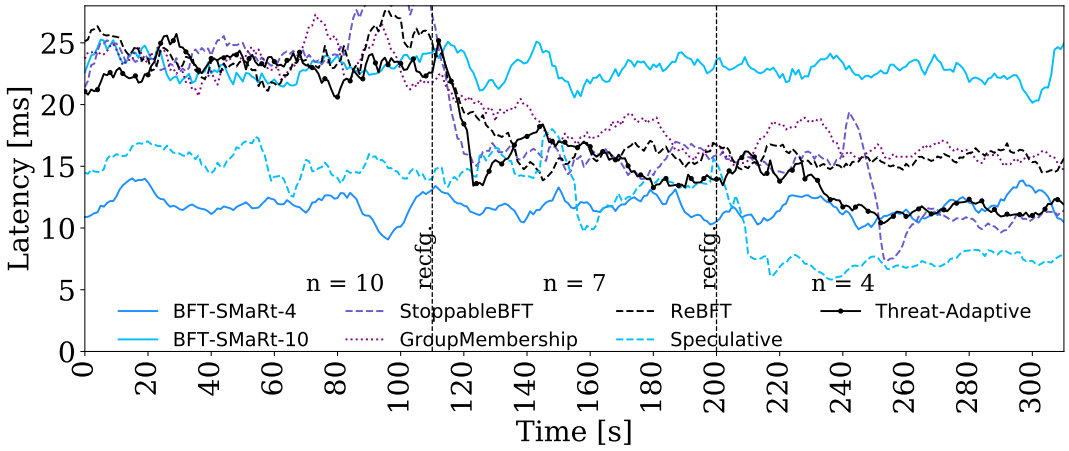

Figure 3.9: Latency of ThreatAdaptive and the 5 baseline protocols (no rejuvenation, $\sigma \leq 3$).

during two optimization reconfigurations is similar to the two BFT-SMaRt baselines (i.e., BFT-SMaRT-10 and BFT-SMaRT-4) when they are configured for a similar threat level. In addition, between 110s and 230s ThreatAdaptive is able to use 7 replicas, which provides a better latency and throughput than BFT-SMaRt-10 and is safe, contrary to BFT-SMaRt-4 in that interval.

For the same experiment, Fig. 3.9 illustrates the latency of the protocols. ThreatAdaptive comes close to the BFT-SMaRt baselines when the threat level allows it. In the second phase, we see slightly higher latencies than StoppableBFT, this is because outliers present in StoppableBFT reconfigurations were averaged out (i.e., masked in the sliding window average). StoppableBFT's restart costs are much higher than other protocols because the payload protocol has to be stopped, reconfigured and relaunched, which is much slower than executing view changes.

3.6 Challenges faced during the development with BFTSMaRt

Programming using the BFTSMaRt Library, particularly in the implementation of the BFT-SMaRt-v1.0-beta version, presents several challenges. One significant challenge encountered is the tendency of this version to switch leaders frequently, which hampers the system's progress during experiments. In order to overcome this issue, the solution found was to silence certain leader change methods within the library, like this one on Figure 3.10 responsible for forwarding requests to the

```
for (ListIterator<TOMMessage> li = pendingRequests.listIterator(); li.hasNext(); ) {
    TOMMessage request = li.next();
    if (!request.timeout) {

        logger.info("Forwarding requests {} to leader", request);

        request.signed = request.serializedMessageSignature != null;
        tomLayer.forwardRequestToLeader(request);
        request.timeout = true;
        li.remove();
    }
}
```

Figure 3.10: The code snippet handles timed-out requests in a system during leader change protocol execution in BFTSmart Library. It checks if there are pending requests, logs the timed-out requests, forwards non-timed-out requests to the leader, marks them as timed out, and removes them from the pending requests list. This process ensures effective management of pending requests and maintains system progress even if a leader change is happening.

leader. Selectively loosening deadlines involving the detection of faulty leaders was also helpful, the system could continue to function and make advancements in the experimental setup. These changes were extremely important on helping the development and debugging of our solutions during implementation. Another notable challenge involves the requirement of a Trusted Third Party (TTP) class called TTPReceiverThread to facilitate the addition and removal of replicas. This reliance on a specific component adds complexity to the programming process, as it necessitates the proper implementation and management of the TTPReceiver-Thread class. There was also something like a proper time to add and remove replicas noticed, for example if the system was in the middle of a reconfiguration or a view-change the procedure of adding or removing would probably fail. The inclusion of such a class indicates the importance of a trusted entity in the operation and maintenance of the BFTSmart Library, further emphasizing the intricacies involved in programming with this particular library.

There were also challenges concerning the process of transitioning replicas from a passive mode to an active state and vice versa. The described procedure of activating and deactivating replicas, as required by the system, led to errors related to replicas catching up. These errors likely involved difficulties in synchronizing the state of the replicas and ensuring that they were up-to-date with the latest changes. Managing the activation and deactivation of replicas while ensuring proper synchronization is a complex task that requires careful attention to detail. This problem highlights the intricacies of implementing a full-fledged distributed system framework like BFTSMaRt and the need for meticulous handling of replica

state transitions to avoid errors and maintain the system's integrity.

There were instances where the reconfiguration procedure took an excessive amount of time, adversely affecting the experiment. This delay resulted in all replicas initiating a continuous catching-up process in a loop, leading to disruptions and potentially invalidating the experiment's results. Dealing with the extended reconfiguration time and mitigating the repetitive catching-up process requires careful consideration and optimization of the system's configuration mechanisms. This issue underscores the importance of efficiently managing reconfiguration in distributed systems protocols to ensure smooth execution and reliable system behavior.

The modularity found on BFTSMaRt is one of its kind, and without all the documentation and experienced distributed systems authors behind it, it would be way harder to create new BFT protocols extending from this library. These challenges highlight the complexities involved in programming real-world deployment distributed systems software like BFTSmart Library and the need for careful consideration and adjustments to ensure smooth operation and progress within a distributed system environment.

3.7 Final Remarks

In this Chapter, we established the conditions that allow a BFT-SMR protocol, which potentially rejuvenates its replicas, to safely reconfigure itself to tolerate an increasingly powerful adversary based on a threat detector that communicates synchronously with replicas. We designed ThreatAdaptive, a novel BFT-SMR protocol that proactively agrees on more resilient fall-back reconfigurations before optimizing its configuration. Our results allow a protocol to dynamically and safely optimize its performance by reducing the number of replicas that actively participate in the protocol's execution. ThreatAdaptive is the first protocol that guarantees safe reconfigurations directly executed by the replicas assuming that a threat increase signal is received sufficiently in advance. Our experiments showed that our threat adaptive protocol achieves a throughput and latency comparable to the non-adaptive baselines in stable phases, and that reconfigurations are 30% faster than using previous methods, which make stronger assumptions to provide safety.

Chapter 4

Robust and Automatic Reconfiguration for BFT State-Machine Replication

As shown in Chapter 3, not all nodes in an adaptive distributed system have the same role at all times. In this chapter, we will take a look at other factors that can impact replicas differently, such as workload shifts and changes in network properties, and how they can be used to optimize the system. These changes present an opportunity to minimize the system's latency by reconfiguring it whenever necessary. Ideally, these reconfigurations should be automatically determined and executed by the nodes themselves, eliminating the need for human intervention. To achieve this, pairs of nodes can measure and report their communication latency, generating a node-to-node latency matrix. This matrix can then be used as input for an optimization algorithm, ensuring that the system's latency remains optimized at all times. Workloads may shift and network properties change for various reasons. These changes allow a system's latency to be minimized by reconfiguring it, whenever network properties evolve. Ideally, these reconfigurations should be automatically decided and executed by the nodes themselves, so that the system's latency remains optimal at all times without requiring human intervention. To do so, pairs of nodes would measure and report their communication latency to produce a node-to-node latency matrix that would then be given as input to an optimization algorithm. Unfortunately, in practical settings one cannot assume that all nodes are correct. In particular, Byzantine nodes might report incorrect latencies, or even collude to lead the system towards a configuration with poor performance. In this chapter, we first confirm this intuition by evaluating the negative effects of coordinated attacks on a state-of-the-art automatic reconfiguration method. We start by projecting nodes latencies into a 3D space using a virtual coordinate system (VCS). In this space, distances between nodes

resemble node-to-node communication latencies, and latency changes resemble the movements of nodes in this space. From this projection, we derive two methods that allow correct nodes to eliminate inconsistent latencies. This is possible because movements in the VCS need to remain coherent from the point of view of correct nodes, which is more difficult to manipulate in a multidimensional space. We introduce three new latency sanitization methods that aggregate the information coming from a tampered latency matrix and output a much more accurate version of it. Our evaluation of three real-world networking datasets reveals that the resultant latency matrix sanitized by our methods is up to 86% more accurate than previous solutions, providing much more precision to reconfigurations of latency-based optimization protocols.

4.1 Introduction

Latency and throughput of network links are two predominant factors contributing to a distributed system's overall performance. Optimizing them remains, until today, a subject of active research, in particular to compensate change. Indeed, distributed systems change in consequence of their nodes assuming different roles, which in turn affects how significantly their connectivity contributes to the overall system performance. Workloads may shift, for example to follow regional day-night cycles, and network properties may fluctuate in response to internal and external events.

A prominent example of nodes changing roles is leadership change in Byzantine Fault-Tolerant State Machine Replication (BFT-SMR) protocols when rotating [Ver+09] or when electing a new leader in case the current replica in this role is suspected faulty. Leaders typically send and receive significantly more messages than other replicas and should therefore ideally be well-connected to other replicas and active clients.

Considering link properties when performing such reconfigurations allows a distributed systems to keep operating near its optimal latency, in particular if optimizations are performed automatically by the nodes of the system and without human intervention. To perform such optimizations in an automated and unattended manner, pairs of nodes would measure their communication latency and share these values to form a system-wide node-to-node latency matrix. This matrix is subsequently passed to optimization algorithms to identify optimal configurations. The literature reports several such optimizations, including leader election [ED18], consensus group formation [LV16] and various network level optimizations, such as latency-aware routing [Ari18] and multicast [Le+13].

Unfortunately, distributed systems and infrastructures are increasingly exposed to malicious and coordinated attacks. This is while growing complexity compli-

cates manual supervision and attack resolution [Acc20], which asks for automated and unattended mitigation [Gor+11]. Cyberattacks may affect all nodes of a distributed system, all components of such nodes and all functionality executed by them, including the systems' latency optimization strategy. The goals of such latency attacks are plentiful and range from denying service to gaining competitive advantages, possibly in preparation of subsequent attacks.

In this chapter, we focus on a specific form of latency attacks, called ***latency collusion attacks***. The goal of adversaries mounting such attacks is to place compromised nodes into a favorable role to more severely affect the system once these nodes start acting in a Byzantine manner. Compromised nodes might report carefully-crafted false node-to-node latency information and thereby trick the system's latency optimization algorithm into granting them a favorable role without raising suspicion. Then in that role, they might jeopardize the performance of the system, by delaying responses, by jamming nearby nodes or by ceasing cooperation when it is most critical for the system [Sin+08]. Compromised nodes might also lead the system in a configuration where its latency is increased, in which case they do not even have to stop participating to degrade the system's performance.

In this chapter, we set out to mitigate the effects of latency collusion attacks by making it more complicated for compromised nodes to report false node-to-node latency information. The contributions of this chapter can be summarized as follows:

- We start in Section 4.2 by showing that state-of-the-art automatic reconfiguration methods for a BFT-SMR system are indeed vulnerable to attacks that aim at manipulating the timing information on which these methods base their optimization.
- Section 4.3 gives an overview of the System model and the mechanism of a MonitoringWindow assumed by our protocol in this chapter, and also what are the assumptions that enable our system to work.
- In Section 4.4, we leverage findings from Vivaldi [Dab+04] and explain how the 3d projection of a virtual coordinate system (VCS) can help mitigating latency collusion attacks in the latency matrix. On the VCS, distances between nodes resemble node-to-node communication latencies and movements of nodes resemble latency changes. Since these movements must be coherent from the point of view of correct nodes, it is much harder for an adversary to manipulate this multidimensional space and hence forge node-to-node latencies. We introduce two new approaches leveraging the VCS to enable latency-based optimization systems to make more robust decisions.
- We evaluate our methods in Section 4.5 on three real-world networking datasets.

Section 4.6 draws conclusions and gives directions for future work.

4.2 Impact of Latency Collusion Attacks

Latency collusion attacks aim at placing compromised nodes in R_{max} to then, in a subsequent step, disrupt or slow down the distributed system and the service it provides. Let us return to AWARE to understand the impact of such an attack. Reporting pair-wise low latencies among faulty replicas and high latencies when communicating with healthy replicas, adversaries may trick AWARE's optimization algorithm into selecting faulty replicas for the best-connected group(R_{max}) and hence into assigning them the higher voting weight V_{max}. The worst case occurs if half of the replicas receiving V_{max} are faulty. This is because if these replicas cease to cooperate, higher costs will occur since consensus must be reached among many replicas outside R_{max} and among the remaining healthy replicas in R_{max}. Figure 4.1 illustrates this effect[1]. In the figure, we plot throughput (i.e., the number of operations per second) for a system of 13 nodes (we observe similar effects also for systems of a different size). Results show that under attack when AWARE needs to operate with replicas outside R_{max} the time to throughput drops from 60 operations per second to 20. Our goal is to avoid such attacks and the performance degradation they entail.

The methods described in this chapter enhance the resilience of systems that apply latency-based optimizations against coordinated attacks by implementing effective latency matrix sanitization algorithms. Former works, like AWARE and Kauri, described systems capable of improving the system's latency and throughput by applying latency optimization techniques. What is not covered by them is if their replicas fail to measure round-trip time as a consequence of collusion attacks. In the presence of such attacks, the once great optimization becomes a hurdle. The impact of this obstacle on the system's progress is significant because there are times when it can cause a complete pause in the system's progress. Therefore our approach offers distributed systems a more robust optimization by providing the system with more accurate knowledge of its point-to-point real-world communication latencies.

4.3 System Model and Monitoring Window

We consider a distributed system comprised of $N = 3f + 1 + \Delta$ nodes that tolerate up to f Byzantine nodes, and that has Δ additional nodes, all n perform pairwise

[1]Measurements were performed on a Dell Precision 7920 Rack running openjdk 17.0.5 on Ubuntu Linux 18.04.6 LTS. We deploy clients and replicas on the different cores of this system.

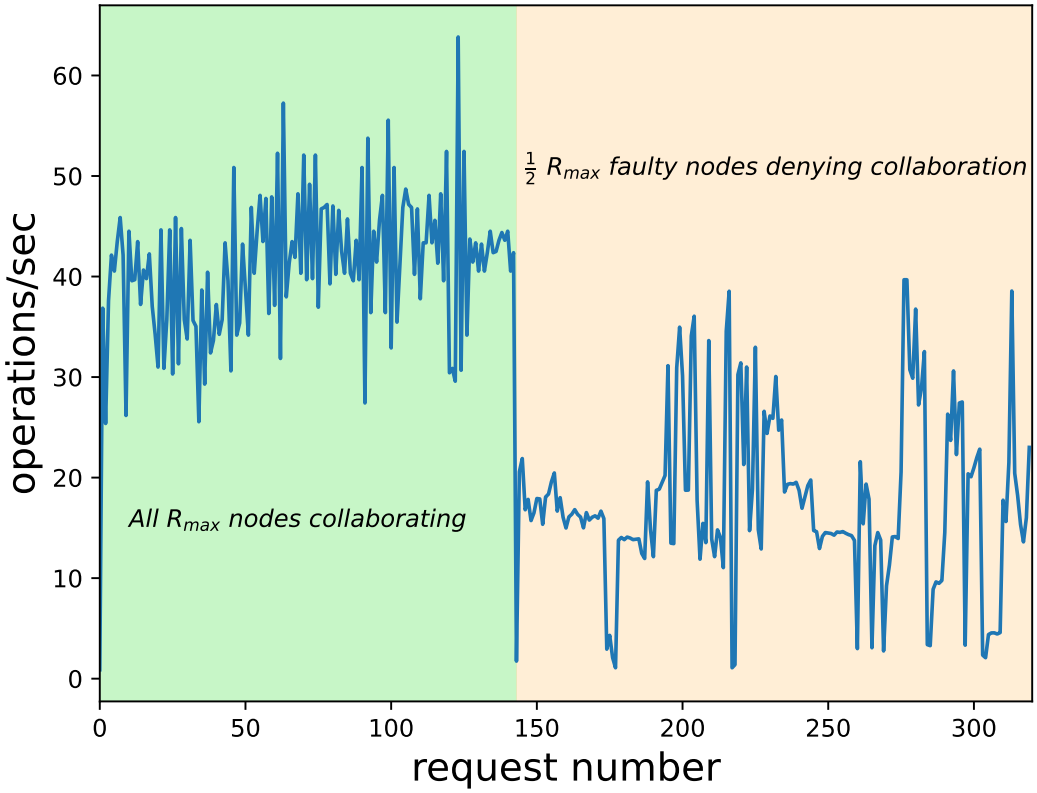

Figure 4.1: Effect of a latency collusion attack on AWARE. On the left, the performance of AWARE is shown as operations per second assuming all nodes with weight V_{max} collaborate. During that time, adversaries have influenced AWARE's latency optimizing algorithm to also grant the f Byzantine nodes this maximum weight. The moment these f nodes cease to collaborate, at request 145, the performance drops as consensus requires including also nodes with V_{min} and more in total.

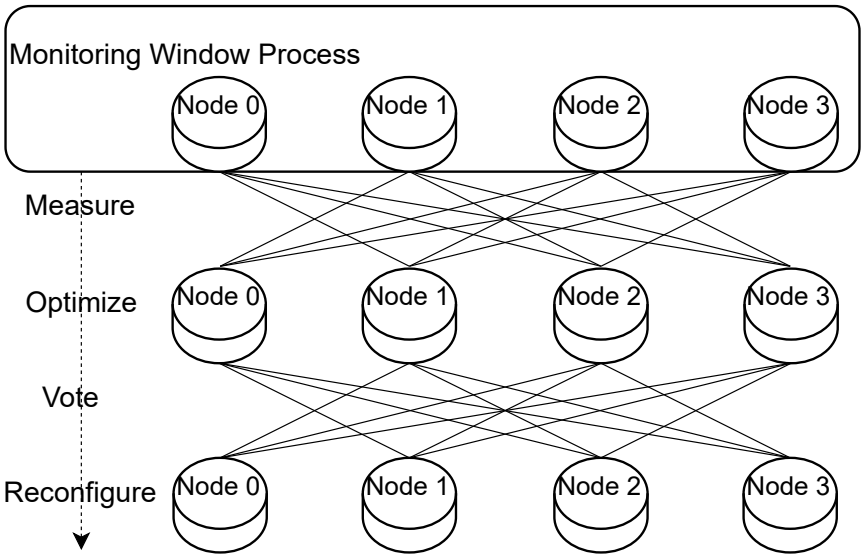

Figure 4.2: Illustration of our fault-tolerant reconfiguration pipeline that nodes independently and automatically execute.

node-to-node latency measurements to optimize the overall system's latency and improve its service. More precisely, we assume that this service can be provided with optimal latency by a subset R_{max} of well-connected replicas. Optionally, the system may still be able to provide the service using other subsets of nodes, albeit at reduced performance or while generating other undesirable costs.

We assume a partially synchronous system, so that consensus can be reached. We do not constrain the topology of the network, but assume that any two nodes can communicate with each other. Differently from [Obe+16] on our system, the network links properties may improve or worsen over time. While nodes in the distributed system have to measure these pairwise latencies, we do not want to constrain adversaries and assume them to have full knowledge over the true latencies between nodes at all times. We do so assume for this chapter that the adversary cannot manipulate the network and modify the latency of links. We shall return to this assumption as part of our future work.

AWARE [Ber+20] is an example of such a system. It implements BFT-SMR using two kind of weights: V_{max} and V_{min}. Well-connected nodes (i.e., those with low node-to-node latencies) receive a higher weight V_{max} and participate in AWARE's weighted consensus algorithm. Weights are distributed such that agreement among nodes in R_{max} suffice to reach consensus, but AWARE is also able to reach consensus using a larger number of nodes, by including nodes outside R_{max}. In the latter

case, performance degrades as more communication is required to reach agreement among this larger set of nodes. Other examples of latency optimizing systems include blockchain-algorithms such as hyperledger [Fos+20] that elect replicas to form the consensus group and that would have to re-elect a different group if replicas cease to cooperate or distributed systems that silence not so well-connected nodes [Kap+12b], but are prepared to reactivate them in case active nodes fail.

We exemplify how our methods work by applying them on AWARE's settings. AWARE periodically performs automatic reconfigurations during a phase called MonitoringWindow. The first latency measurements are already done on system setup like in WHEAT [SB15b]. To reach an agreement in a weight voting scheme, replicas have to sum more voting power than a threshold, that for AWARE consists of $2(f + \Delta) + 1$. Nodes must agree on a common latency matrix that will later help define the best weight configuration. The weight configuration of a system of $(3f+1+\Delta)$ replicas is a composition of two subsets, the first called R_{max} composed by the $2f$ best-connected replicas, according to the latency matrix, and R_{min} composed by the other $(f + 1 + \Delta)$ replicas. The system may experience latency inconsistencies caused by network artifacts and/or by the action of Byzantine replicas, tampering with their latency matrix in an attempt to portray them as best-connected replicas on the network. The system's designer defines the period in which replicas update their latency matrix, ranging from $500ms$ to one second. Whenever this period of update expires, replicas start this monitoring phase. This window can be subdivided in the following four steps, illustrated by Figure 4.2.

Step 1: Measure. In this stage, replicas broadcast their latency measurements to each other and store these values on matrix M. These measurements are done by sending one *MONITORING* message from each and every replica that should respond as quickly as possible. When those replicas receive their answer for its monitoring request, timestamped correctly, they will store the round-trip time it took for this specific replica receiver. Correct or Byzantine replicas cannot send responses faster than the network time required, because there is a challenge-response mechanism on every measurement message, forcing replicas first to receive the message with the challenge and then respond to the replica sender that requested a response in the first place. These latency measurements are broadcasted and stored in a matrix locally on every replica.

Step 2: Optimize. With the latency matrix ready the next step is to sanitize their matrix, execute our 3d fault-tolerant vector approximation algorithms and start the local optimization process executed by every replica. Sanitization, in AWARE, is done by applying the following operation $\widehat{M}[i, j] = \max(M[i, j], M[j, i])$ to every term belonging to the matrix [Ber+20]. Here is where our latency algorithms make the difference, because this sanitization method is not resilient against collusion of latency measurements between Byzantine nodes. Therefore,

every replica executes our latency correction techniques locally in order to obtain a much more accurate latency matrix. Replicas finalize this step by executing simulated annealing to find the best weight configuration possible, the one with the best consensus latency, using current corrected values as input.

Step 3: Vote. Now that every replica found locally a configuration with the fastest consensus latency, they need to agree about that. Byzantine replicas could try to tweak their selection of the best configuration, but it does not change a thing for the system because replicas will run a consensus on which is the best configuration. The agreement also uses a weighted voting scheme, and with that, as soon as the voting power threshold is met, replicas will start reconfiguring the system.

Step 4: Reconfigure. On the last step, replicas will have the checkpoint of a well-succeeded monitoring window with the agreed best consensus latency, and if the system performs correctly, they will issue a reconfiguration request. A subsequent consensus after the voting step can be required to verify whether the consensus latency promised was matched. In case performance does not match the latency expected by a threshold defined by the system designer, the system will then leave this MonitoringWindow process with the same configuration it started, not to jeopardize safety for performance. The system designer could also configure greater consequences to change the simulated algorithm by adding more inputs, like for example which replicas are finishing the processing of requests first. Adding more measurements could incline the simulated annealing algorithm to statistically decrease the chance of choosing arbitrary replicas to have more voting power on the next MonitoringWindow.

4.4 Latency matrix sanitization

A latency matrix is composed of columns representing how one node is seen by everyone else in the system and rows that depict how every node in the system is seen from one node's perspective. The projection of the node-to-node latency matrix into a virtual coordinate space follows the projection suggested by Dabek et al. [Dab+04] for the Vivaldi system. Vivaldi's algorithm is executed once during system setup and the original virtual 3d coordinates are shared. Our algorithm assesses each column and virtually emulates the viewpoint of every network node towards the corresponding node represented by that column.

The nodes receive at deployment time an initial latency matrix. From this matrix, nodes obtain the initial position of the nodes in a virtual coordinate space. This initial matrix is assumed to be correct. Later on, before each reconfiguration, each node maintains a latency matrix that is collectively formed at the beginning of the Monitoring Window phase. The network latency between every two nodes

is measured by the involved parties and reported to all other nodes. During the Measure step, Byzantine nodes can tamper with their own latency measurements being reported. After the nodes agree, through consensus, on a latency matrix, each node executes a latency sanitization algorithm on the latency matrix, making it more accurate.

Our sanitization methods rely on virtual coordinate systems. Indeed, when one only considers the latencies to detect discrepancies, nodes can collude or lie, and it would not be possible to detect these deviations. This occurs because only the replicas that are reporting their latencies are able to do so, and it is not possible, from the point of view of correct nodes, to detect whether one or all of the two reporting nodes is faulty. However, in a virtual coordinate system, it is possible for nodes to analyze whether or not the latency changes that are reported by others seem coherent from their point of view.

After executing Vivaldi's algorithm with the latency matrix as input, each replica will have its VCS representation, where every replica has a 3d coordinate, in a way that the distance between every pair of nodes is represented by a round-trip-time measurement. If the latency matrix is correct, all replicas will have the same 3d coordinates for every replica on the system. However, if the values in the latency matrix differ for any reason, there will be different 3d positions for the same node, and nodes will have to agree on an approximation of these 3d positions. We created two variants of a novel 3d approximation agreement algorithm based on geometry from the information given by VCS, and we also created a baseline inspired by the state-of-the-art robust aggregation algorithm MT-KRUM [Bla+17] as a latency matrix sanitization technique.

4.4.1 Spheres intersection

Comparing points in a 3d space to determine the correct positions of nodes despite Byzantines is not an easy task. The 3d coordinates of a replica should make sense with the values represented in the latency matrix, so that it does not appear to be in two different places at the same time. We have to find the point of intersection between sufficiently enough correct representations. For example on Figure 4.3 to determine replica R2's position we would have to find the intersection between the perspective of every other replica besides R2. For our algorithms **Geometric** and **RndSamp** we choose to represent these perspectives as Spheres, where every Sphere has as its center the 3d coordinate of a replica, and the radius is represented by the RTT between those two replicas. During the setup of the system, the Spheres are generated by GeneratingSpheres1 function using the original latency matrix as input.

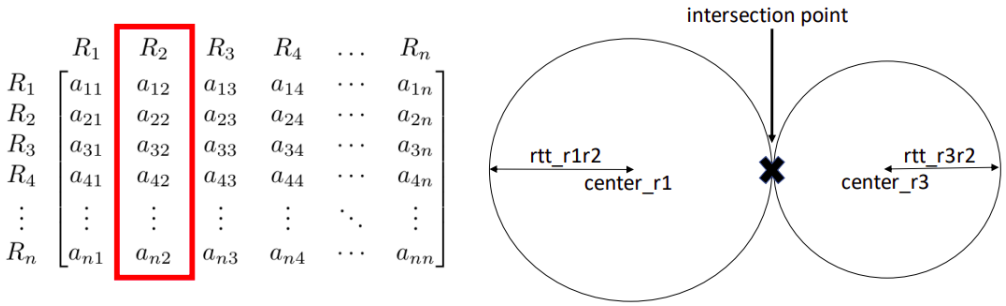

Figure 4.3: Illustration of the sphere intersection method that identifies compatible reported latencies using 3d coordinates, given by Vivaldi's 3d model. The column for node R2 contains the RTT reported between R2 and all other nodes.

Algorithm 1: *GeneratingSpheres* uses Vivaldi's algorithm to provide 3d coordinates for every node given a latency matrix as input. Our methods use a Sphere abstraction to represent every node, where the Sphere's center is the 3d position of a node given by Vivaldi, and the radius is the RTT between the two nodes in which the latency change is being analyzed.

Input : Virtual 3d coordinates, Latency Matrix, n, f,nodei
Output: List of Spheres

```
spheres_list = []
for nodej in node_ids do
    rtt = latencies[nodej][nodei]
    center = coords[nodej]
    radius = rtt
    sphere = Sphere(center,radius)
    spheres_list.append(sphere)
end
return spheres_list
```

4.4.1.1 Geometric and Random Sampling

Our first latency correction algorithm computes the intersection of spheres to verify whether the latencies reported between a given target node and all the other remaining ones seem to indicate that it is faulty. This verification is executed for each node in the system. The spheres we consider are defined as follows. One sphere is created for each of the $n-1$ nodes that verify the latencies of a target node. This sphere is centered at the 3d coordinate provided by Vivaldi's algorithm, and its radius is equal to the RTT between this node and the one being verified. Since latency indicates the distance in the VCS between two nodes, from the point of view of each of the $n-1$ nodes, the target node position should be located in an intersection between these Spheres. Therefore, if in $n-f-1$ spheres there is no intersection point, then we consider that the target node's reported position is inaccurate.

The Geometric solution illustrated by Algorithm 2 iteratively computes the intersections of spheres, which are either empty, contain a single point or form a circle. In case an intersection is equal to a circle, we continue computing its intersection with the remaining spheres. For example, if spheres from replicas r1 and r3 intersect in a circle, then this circle will be tested for intersection against other spheres. The result of a circle-sphere intersection is either empty or a point. Whenever the algorithm finds one point that belongs in the intersection of $n-f-1$ spheres it finishes the evaluation of the latencies of a target node and moves to the next one. In terms of complexity, for each column of the latency matrix our algorithm computes in the worst case $\binom{n-1}{2}$ sphere-sphere intersections times $\binom{(n-1)/2}{2}$ circle-sphere intersections. However, in the average case, solutions are found using less than $10 * \binom{n-1}{2}$ iterations. Figure 4.3 illustrates the evaluation of the latency matrix by our abstraction where column R2 shows the connectivity from every node with node 2.

Our geometric solution finds the approximation of R2's position by finding the intersection between all the spheres, where each sphere is centered on the node's Vivaldi's 3d position and with a radius equal to the RTT to this node. Exemplified by Figure 4.3 and pseudocode 2 the algorithm starts by looking at spheres S_{12} and S_{32}, i.e., R2's 3d coordinate from the perspective of R1 and R2's 3d coordinate from the perspective of R3, and finding the intersection of these spheres to determine the position of this node. Repeating this procedure for $n-f$ nodes results in a close approximation of R2's position, which is then used to recompute the latencies between nodes in the latency matrix as their distance in the VCS. Once sanitized, the latency matrix can then be used in the latency-based optimization. The geometric solution might fail to find a large enough intersection of spheres if the VCS is too imprecise. In this case, we progressively increase the radius of the spheres until a satisfying intersection can be found.

Algorithm 2: *Geometric* finds the most plausible 3d position of a target node in a virtual space by finding a point that belongs in enough spheres centered around other nodes and whose radius are equal to the RTT with the target node.

Input : n, f, List of Spheres, Latency Matrix
Output: 3d Coordinate

```
# Drop f extreme coordinates
spheres = dropfDistantSpheres(n,f,spheres)
# Find all intersections between spheres
combinations = (spheres 2)
circles = []
for combination in combinations do
    if combination.intersect() then
        circles.append(combination)
    end
end
# Find intersections between circles and spheres
points = []
for circle in circles do
    for sphere in spheres do
        intersection = sphere.intersect(circle) if intersection then
            points.append(intersection)
        end
    end
end
# Find intersections between intersection points and spheres
points = []
for p in points do
    for sphere in spheres do
        intersection = sphere.intersect(point) if intersection then
            intersection_count++
        end
        if intersection_count > f then
            return p
        end
    end
end
```

Algorithm 3: *RndSamp* finds the most accurate 3d position of one node in a virtual space by looking into points on the surface of spheres and checking which point belongs to the largest number of intersections.

Input : n, f, List of Spheres, Latency Matrix
Output: 3d Coordinate

```
1  # Drop f extreme coordinates
2  dropfDistantSpheres(n, f, spheres)
3  # Randomly generate points on Spheres surface
4  points = []
5  for s in spheres do
6      for 1000 times do
7          α = random(0, 360)
8          β = random(0, 360)
9          dx = s.radius * cos α * cos β
10         dy = s.radius * sin α
11         dz = s.radius * sin α * cos β
12         p = Point(s.x + dx, s.y + dy, s.z + dz)
13         points.append(p)
14     end
15 end
16 # Find intersections between points and Spheres
17 for p in points do
18     intesection_count = 0
19     for sphere in spheres do
20         if sphere.intersect(p) then
21             intesection_count++
22         end
23         if intesection_count > f then
24             return p
25         end
26     end
27 end
```

The RndSamp solution finds an approximation of R2's position without exactly computing the intersection of spheres. Instead, it randomly generates points on the surface of spheres to find the first it intersects the most with. Exemplified by Figure 4.3 and pseudocode 3 the algorithm starts by looking at spheres S_{12} and S_{32} and it then generates points on the surface of these spheres and tests if these particular points belong in both spheres and in other spheres. Repeating this process for $n - f$ nodes results in an accurate approximation of where R2 is located in the 3d space. This approach might seem inefficient but it takes on average the same time as our geometric solution and yields better results against specific coordinated attacks. By applying both of these methods, each replica is able to achieve a latency matrix that portrays a more accurate representation of the point-to-point connections between replicas in the system when compared to former methods.

4.4.2 Latency matrix correction: Byzantine gradient averaging

Replicas are represented by 3d coordinates in the virtual coordinated system, to approximately identify other replica positions they have to be able to average out Byzantine connection measurement reported values. MT-KRUM [Bla+17] is a robust fault-tolerant aggregation technique, that cleverly combines gradients computed by nodes in order to mitigate the impact of Byzantine nodes. In our context, when studying the latencies of a target node, a gradient is a vector that is located on the line that connects them, and whose direction and length are computed based on the latency change between them. All nodes reconcile their gradients through Byzantine gradient averaging and determine the most plausible location of the target node in the 3d space. The Byzantine gradient average method works as follows after computing all gradients, the average of the gradients is computed, and all gradients are ordered according to their distance to this average. The top k smallest scoring gradients are kept for aggregation and their mean is the final gradient that indicates the movement of the target node in the 3d space. This procedure enables MT-KRUM to exclude vectors that are too far away from the average coordinate reducing the damage capabilities of Byzantine measurements. Once the location of the target node is determined its latencies with all other nodes are recomputed and inserted in the latency matrix. Once sanitized, the latency matrix can then be given as input to the optimization process.

4.5 Performance Evaluation

Setup. We implemented the centralized version of the Vivaldi algorithm, which is executed by each node to calculate the VCS on the system setup. We are using three different datasets: i) the first one, CloudPing[2], contains latency measurements coming from Amazon EC2 instances around the globe; ii) the second one is the well known King dataset [GSG02], which has already been used in the literature to demonstrate the limits of the Vivaldi virtual coordinate systems; and iii) the Wondernetwork dataset[3], which comes from a large global networking solution provider, with up to date data from many different sites. These three datasets represent a good variety of latency values and are a good reflection of the real-world topologies that nodes have on publicly available and global services.

Solutions and baselines. We implemented our 3d approximate agreement algorithms, Geometric and RndSamp, which are respectively described in pseudocodes 2 and 3. We compare these solutions to MTKRUM [4] adapted to our solution needs, because it is also a d-dimensional approximate agreement technique and to AWARE [Ber+20], because it is a well known BFT protocol with open codebase that apply latency-aware optimizations to their replicated reconfiguration mechanism. In our experiments, we are simulating systems that contain from $n = 7$ nodes, where 2 nodes might already collude, to $n = 31$ nodes where up to 10 nodes could be reducing the latency between themselves in a coordinated attack. In general, one experiment is subdivided in rounds where collisions follow powers of 2, in the first round there faulty replicas will divide their latency by 2, on the next round they will divide their latency by 4 and so on until 128. Our goal is to demonstrate that our 3d vector averaging techniques can help the system to progress by reducing the impact that coordinated Byzantine nodes could have on latency-aware optimized reconfigurations.

Coordinated Attacks. Coordinated attacks can occur in many different scenarios. We consider three representative scenarios in our experiments: i) Byzantine nodes can directly collude to modify their reported communication latencies (simple instantaneous attack); ii) with their processing power Byzantine nodes could try to change the latency matrix with more arbitrary changes to push the system to pick a configuration where they have more voting power (sophisticated attack).

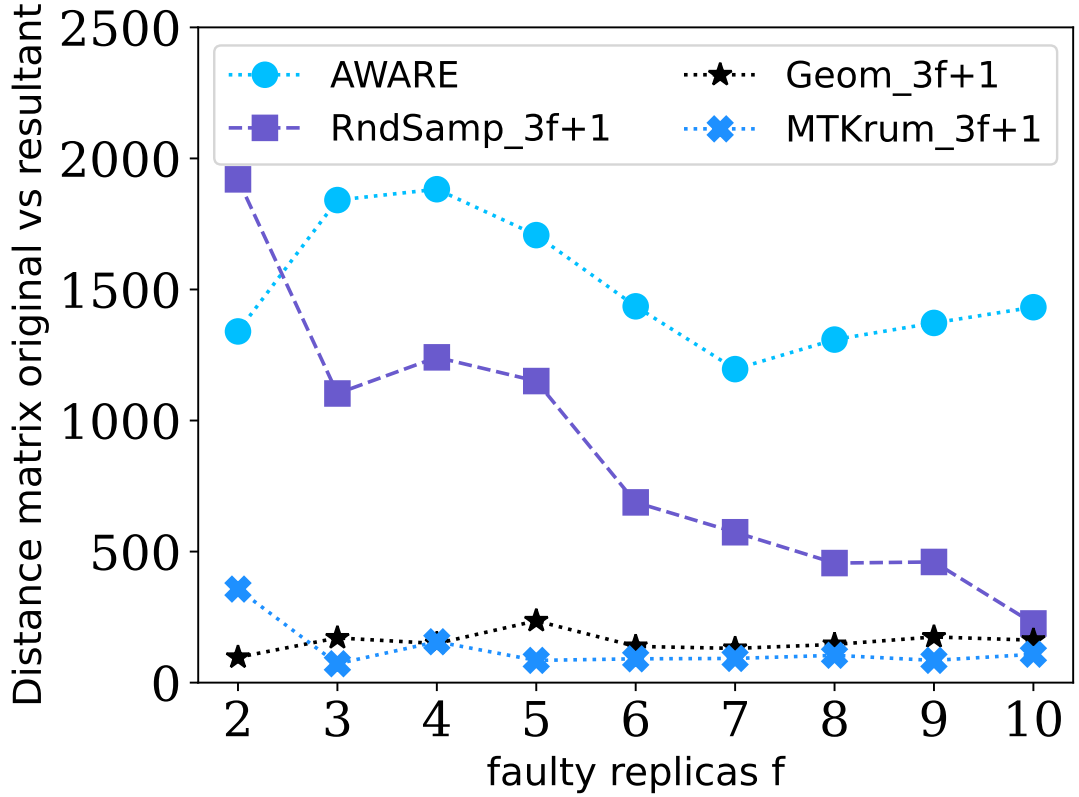

Figure 4.4: The Mean Squared Error(MSE) distance between the sanitized latency matrix and the one reported by the nodes depending on system sizes varying from $f = 0$ to 10 ($n = 3f + 1 + \Delta$) with the CloudPing dataset after our Simple Instantaneous attack.

4.5.1 Simple instantaneous attack

On this first simple attack, the objective of the Byzantine nodes is to reduce the latencies they report between themselves to portray them as being better connected than other replicas. The results of this attack are indicated in Figure 4.4 where we measured the mean squared error(MSE) distance between the original latency matrix and the latency matrix that is obtained by the attack following our sanitization methods. Our solutions Geometric and RndSamp maintain a low error comparable to the one obtained by MT-KRUM, the 3d Byzantine gradient aggregation method. All our solutions improve over AWARE's sanitization method.

Because we are analyzing this attack as its impact in a 3d virtual coordinated space, the outcome of this simple attack would be that Byzantine nodes would get nearer and nearer from each other in the VCS. Once again, let us argue that it would not be possible to detect this simple attack if we were trying to correct Byzantine latencies only based on the reported latency matrix, because it would not be possible to distinguish between an attack happening and the network latencies normally evolving. After executing our solutions the system is capable of accurately reaching the decision about the nodes final positions.

The results from Figure 4.4 are that the system using our latency sanitization methods will have a much more accurate latency matrix. The struggle suffered using AWARE's baseline against this coordinated attack is present from $f = 2$ to $f = 10$ showing that their sanitization method is not resilient enough. The results obtained by RndSamp are good if compared with AWARE but not as good as Geometric and MT-Krum that achieve latency matrices up to 95% more accurate. Also Geometric seems to do even better when operating in a network where the RTT between nodes varies more, something further illustrated in Table 4.4. On this table, we are indicating in % how much better our solutions perform compared to AWARE's sanitization method during a simple instantaneous attack with all three datasets and all solutions.

4.5.2 Optimized instantaneous attack

In this second attack type, the objective of the Byzantine nodes remains the same, but they can report reduced or increased latencies with any other node of the system. They determine how to do so by running an optimization algorithm to lead the system in the worst configuration they can. Byzantine nodes rely on a

[2]https://www.cloudping.co/grid
[3]https://wondernetwork.com/pings
[4]https://github.com/LPD-EPFL/AggregaThor/blob/master/aggregators/krum.py

Table 4.1: Latency matrix distance reduction (in %) during an Simple Instantaneous Attack and using our sanitization methods (Geometric, RndSamp, MT-Krum) compared to the state-of-art (AWARE) over all datasets used.

Sys. Parameters	3f+1+Δ		
Datasets	CloudPing	King	Wonder
Geometric	91.97%	**77.55%**	**95.51%**
RndSamp	86.29%	36.4%	77.67%
MT-KRUM	**95.97%**	77.16%	**95.07%**

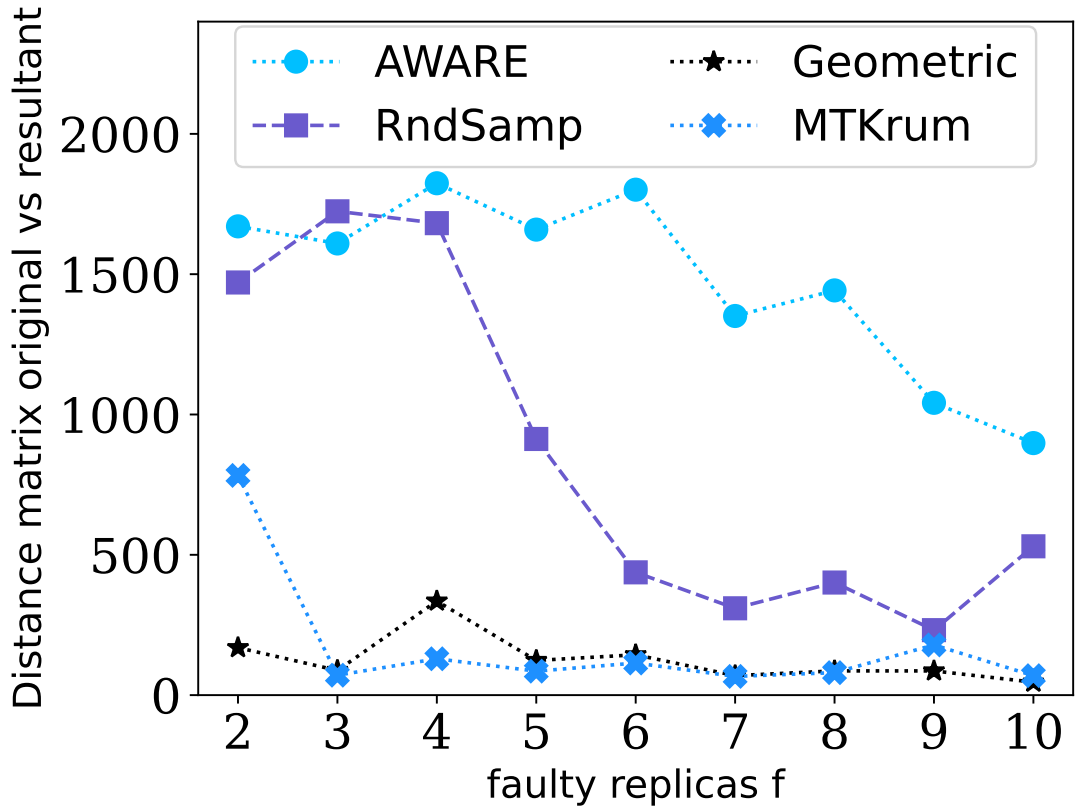

Figure 4.5: The Mean Squared Error(MSE) distance between the sanitized latency matrix and the one reported by the nodes depending on system sizes varying from $f = 0$ to 10 ($n = 3f+1+\Delta$) with the Wondernetwork dataset after our Optimized Instantaneous attack.

Table 4.2: Latency matrix distance reduction (in %) during an Optimized Instantaneous Attack and using our sanitization methods (Geometric, RndSamp, MT-Krum) compared to the state-of-art (AWARE) over all datasets used.

Sys. Parameters	3f+1+Δ		
Datasets	CloudPing	King	Wonder
Geometric	93.67%	**76.55%**	**95.99%**
RndSamp	81.43%	31.36%	77.67%
MT-KRUM	**96.14%**	76.12%	**95.59%**

simulated annealing (SA) algorithm to execute this attack. SA usually is used for finding or approximating the global optimum of any given function, and it was used in AWARE to enable replicas to choose the best possible configuration. In this attack, Byzantine nodes are using SA to find the global optimal change they can apply to portray them as being better connected. In Figure 4.5, our solutions Geometric and RndSamp showed severe improvement over AWARE and comparable progress to MT-KRUM implementation.

The results from Figure 4.5 are that systems using our latency sanitization methods are resilient even to more intelligent attacks. AWARE's implementations seem to have a tough time against coordinated attacks of any kind, with smaller system's configuration like $f = 2$ to more resilient configurations like $f = 10$. Again the results obtained by RndSamp are good if compared with AWARE but not as powerful as MT-Krum and Geometric that achieve latency matrices with up to 95% more accuracy on their node-to-node connection values. Geometric and MT-Krum keep operating with good resilience against attacks on network topologies where the RTT has more variance, as illustrated in Table 4.4. Once again, we summarized the results of this attack with all three datasets and all solutions into Table 4.2 for better readability. This Table also indicates in % how much better our solutions perform compared to AWARE's sanitization method during an optimized attack.

4.5.3 Impact on configuration latency

During an Optimized Instantaneous attack, the impact caused on the configuration selected as the best one is meaningful since Byzantine nodes can still increase their voting power. Experiments showed that this attack, when optimized for this purpose, can change the combination of nodes chosen as the best connected replicas in the system, and usually attributes a larger weight to Byzantine nodes. However, because our protocol follows AWARE directives on this regard, which means that

there are $2f$ nodes with a larger weight, the system will never be dominated in voting power by faulty nodes.

In AWARE coordinated attackers could stall the system's progress indefinitely by tweaking the latency matrix and consequently holding half of the voting power. Our solutions fix this problem by dramatically reducing the damage they could do on the latency matrix, so that Byzantine nodes would rarely represent 50% of the replicas with more voting power set.

We envision that one could address this worst case scenario using a subsequent consensus and a reputation mechanism. The nodes would have to verify whether the performance promised by the reconfiguration was actually achieved and, later on, the system could then decide whether to stay in this configuration or to restart the reconfiguration procedure. A complementary scheme would penalize through a reputation scheme the nodes that were previously selected as being well connected and guarantee that future reconfiguration would decrease the probability to select the same nodes. This method might lead the system to choose slower configurations rather than the fastest possible but it would make the latency-aware optimization subsystem more resilient over the long run.

4.5.4 Deviations over multiple reconfigurations

Former works allow collusion not only to happen but also to persist on the latency matrix over reconfigurations, and because of that, they are doomed to experience more significant damages than our methods. In our experiments, every reconfiguration happens as if it was the first because Byzantine nodes cannot hold onto leverage acquired from one reconfiguration to another. For the MT-KRUM solution, in a system with classical distributed systems configuration, every reconfiguration procedure is unique, or there are no traces of Byzantine tweaking on the latency matrix. For Geometric and RndSamp there are usually slightly tampered values on the latency matrix with $3f + 1 + \Delta$ or more resourceful systems, but even after multiple reconfigurations the interference on the system's progress is negligible. Therefore our solutions do not suffer from deviations in the latency matrix over multiple reconfigurations.

4.5.5 Sanitization running time

Other works applied fast sanitization methods that were as simple as matrix additions, but in consequence these protocols are prone to coordinated attacks. By applying our techniques, the time dedicated for the sanitization of the latency matrix is increased. In our experiments, we found that AWARE's sanitization method can take on average $500ms$. Our solutions are heavier and can take at most 3 times more time to sanitize the latency matrix, which however remains

practical. The latency matrix sanitization process takes visibly more time, but it does not disrupt the system. Using our solutions the system can keep progressing even in the most adversarial scenarios that a fault-tolerant distributed system may face.

4.6 Final Remarks

In this Chapter, we introduced two novel methods to mitigate the effect of latency collusion attacks. Adversaries mount attacks of this kind on distributed systems with a built-in latency optimization mechanism to place compromised nodes in favorable roles from where they have greater effects in subsequent attacks. Our methods complicate the reporting of false node-to-node latency information through colluding Byzantine nodes, by requiring reported latency information to be consistent in a multidimentional virtual coordinate space. The first requires latency-representing spheres to move consistently whereas our second method leverages gradient aggregation algorithms for detecting and correcting false latency information. Our performance evaluation over three real-world networking datasets showed that our approach protects latency after a reconfiguration against Byzantine nodes up to 95% better than previously published state-of-the-art solutions.

As future work, we plan to extend our approach to other measures commonly used when reconfiguring distributed systems, such as throughput and the energy supply of nodes. Moreover, we plan to return to the quite restrictive assumption that adversaries cannot affect network latencies. Given a traffic shaping mechanism, pairs of nodes should be able to measure connection latencies, while anticipating the maximal allowed adversarial interference by such mechanisms.

Chapter 5

Threat-Resilient BFT-SMR Protocol: Robustness through Adaptive Measures

BFT protocols are known for using too much resources and that is one of the reasons why they are not mainstream yet, with the developments shown on Chapter 3 this work introduced a protocol capable of optimizing resources at runtime, instead of having the protocol configurations hard-wired and pre-fixed during architecture designing time. We also showed that the same protocol capable of optimizing its configuration automatically would also be able to return the system back to safer configurations, if needed, without requiring consensus to take immediate action. Instead of only quantitatively changing configurations where only the size of the system is changed, we could use the techniques developed in Chapter 4 to create more optimal and minimal system configurations where replicas are chosen qualitatively by how fast they are interconnected with other nodes and using our methods, we can create more efficient quorums extracting even more performance from the payload protocol. Another adaptive measure found during the development of our solutions is the strong correlation between replicas that finish requests first and the fastest replicas on the network. Integrating all these solutions enables the system to make more robust optimizations, self-regulating itself in an unattended manner, allowing an efficient resource use of the system's infrastructure even on safer configurations, and also with the addition of reputation the system can get progressively better and more efficient over time.

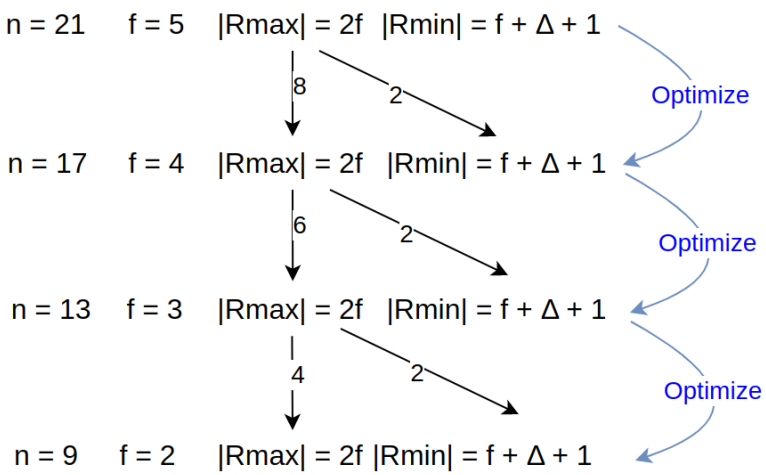

Figure 5.1: Using our methods systems can reconfigure themselves to move iteratively from bigger and slower configurations like $n = 21$ to faster and optimal configurations like $n = 9$. When going down in configurations most of target R_{max} will remain the same as source R_{max}, with the exception of the two slowest replicas from R_{max} that will be having their voting power downgraded to V_{min}.

5.1 Proactive Reconfiguration for Performance Optimization

We assume that for every period of x amount of time(decided by the system designer) a MonitoringWindow will be executed, like in AWARE, and this is the moment where our system should strive for optimization. Connecting our Threat Detector also to this system would provide a better grasp of when is the right moment for optimizing the system. When the reconfiguration goal is optimization the system respects the monitoring window period. It will then further, re-calculate the latency matrix and apply our sanitization methods from Chapter 4 to take a more robust decision on the configuration change.

When optimizing systems infrastructure, the replicas that make up the $R_{max}(2f)$ set in the source configuration will also make up the R_{max} in the target configuration as shown in Figure 5.1. The two slowest replicas in R_{max} of the source configuration will become R_{min} on the target configuration. The $f + \Delta + 1$ fastest replicas from R_{min} on the source configuration will be going to the target configuration, also on R_{min}. Taking these replica allocation decisions will not only shrink the system on the number of replicas, but also on the choice of which replicas remain active. The nodes not included in the target configuration will become passive, like in ThreatAdaptive, and they will be further re-activated when the

system needs them.

A valid assumption we can make is that $\Delta = f$ always. The reason behind this is that in AWARE, it is necessary for V_{max} to be greater than V_{min} in order to enable a more varied quorum formation. Knowing that $V_{max} = 1 + \frac{\Delta}{f}$, the number of spare replicas is at least f.

Right after optimizing system's configuration the consensus latency promissed can be easily tested by checking which replicas are finishing the first consensus after reconfiguration and how long did it took to reach agreement. If it takes less than expected, it is even better for the system and there is no reason to reconfigure back or again, but if new configuration does not deliver the new efficient consensus latency proposed than the system could reconfigure it back to a safer configuration. Checking the consensus latency on the agreements after a reconfiguration was not cover by AWARE and it is actually a good way of guaranteeing that the system is not doing fake optimizations or optimizations that would stall more system's progress than help.

5.2 Determining Conditions for Safe Reconfiguration

Differently from optimization, the system does not need to wait for a Monitoring-Window to happen to move the infrastructure to a safe state and possibly save the system from increasing adversarial strength scenario. Reconfiguration with safety goal is triggered by the Threat Detector signal, if the system is or the region where the system is becomes attacked, TD will signalize that threat level is high, meaning that the system should protect itself. If the threat level does not change, the system will keep going up on reconfigurations and whenever reaches the safest configuration then the system will re-calculate the latency matrix and apply our sanitization methods, only after that the system will be well-prepared to take the decision on the next configuration change.

When reconfiguration towards safety happens, the replicas that make up the $R_{max}(2f)$ set in the source configuration, that is now the most efficient one, will also make up the R_{max} in the target configuration, but now the two fastest replicas from R_{min} will be also being part of R_{max} as shown in Figure 5.3. All the replicas from R_{min} on the source configuration will be going to the target configuration also on R_{min} and there will be more Δ passive replicas being re-activated. Taking these replica allocation decisions will increase the system's redundancy capabilities and its robustness against increasing adversarial strength. There will still be enough passive nodes that can be re-activated on each step of reconfiguration, like in ThreatAdaptive, and they will be further re-activated when more resilience to

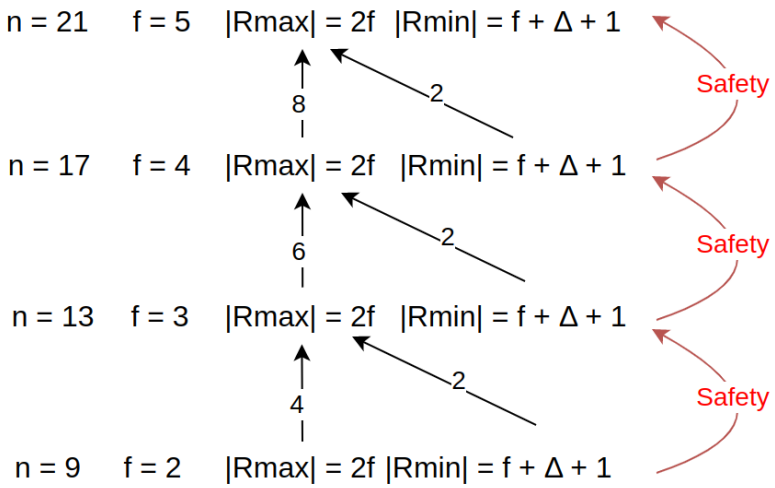

Figure 5.2: Using our methods systems can reconfigure themselves to move from smaller and insecure configurations like $n = 9$ to safer and reliable configurations like $n = 21$. When going up in configurations, most of the target R_{max} will remain the same as source R_{max}, except for the two fastest replicas from R_{min} that will be having their voting weights upgraded to V_{max}.

faults is required.

The fact of having $\Delta = f$ can be seen as increasing expenditure of resources, but it can bring improvements to the infrastructure. As observed in WHEAT, in fault-tolerant weighted voting schemes to have more replicas makes consensus to be reached faster. In the case of reconfiguring the system in chase of safer configurations, the system actually speeds up, when compared to egalitarian quorum configuration of the same size, and increase resilience at the same time.

After reconfiguring the system's configuration towards safety, the system will consult the threat detector again to see whether it should keep reconfiguring to safer configurations or it can start executing the payload protocol again. If the threat level is still high, the system will keep reconfiguring and awakening passive replicas, but if the threat level went from high to medium depending on the system's designer choice the system can stay on the same configuration or keep reconfiguring until it hits the safest configuration. MonitoringWindows are stopped until the system returns to normal payload execution, the latency matrix is also not updated during this procedure because it is still not the right time to strive for efficiency but to strengthen the system against increased adversarial strength.

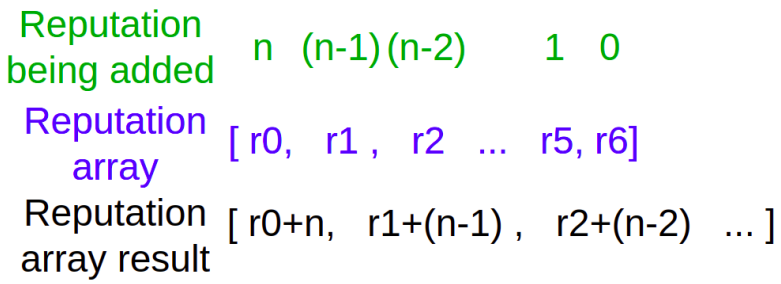

Figure 5.3: Nodes accumulate reputation points by finishing consensus messages, who finishes the first earns more points. Here in this example, the first replica $r0$ will have n summed to the actual reputation value because it was the first replica to finish, $r1$ the second replica finishes in the second position and earns $n-1$ points, and so on. Every MonitoringWindow, each node, will have its own reputation array that reflects the quality of the interconnection between replicas from another perspective.

5.3 Reputation improves consensus latency over time

If every node takes note of which replicas are finishing consensus first and add an arbitrary value to the reputation of nodes proportionally to their performance, the system achieves a reputation array that reflects the latency matrix information from a different perspective. Having those two inputs the system could easily detect discrepancies on the latency measurements done, because how come nodes X and Y can have really good rtt values if when checking the reputation array replicas see them as badly connected. Once these co-relations are detected the system can wisely reconfigure to a safer configuration given that faulty replicas possibly in Rmax have been tampering with the latency measurements on the former MonitoringWindow.

We could also add this reputation array as input of simulated annealing to improve safety and efficiency of the result found by the jumps of this optimization algorithm. The system would not be only looking for the most efficient voting weight distribution between nodes but also the most trustworthy efficient configuration, given that configurations in which nodes with highest reputations are in Rmax would be preferred.

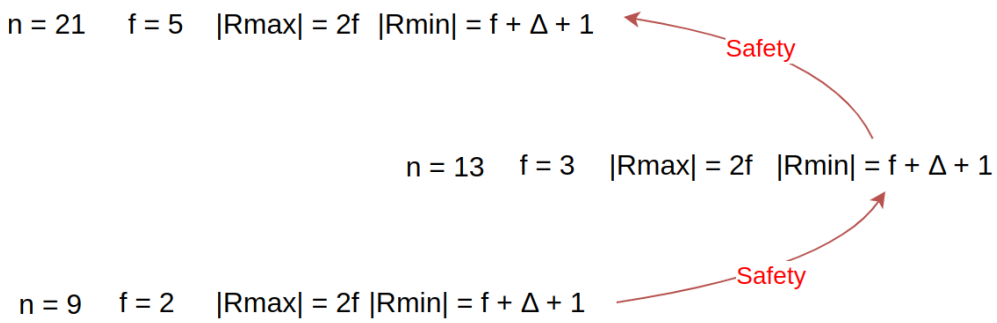

Figure 5.4: Nodes are able to evaluate the efficiency of a potentially more resilient target configuration, when reconfiguring towards safety, using an updated latency matrix. If the current target configuration does not meet the efficiency criteria, replicas could collectively agree that there are superior configuration choices available, and if the threat level is not excessively high, replicas could be allowed to reconfigure to the new target configuration.

5.4 Adaptive Measures: Strengthening BFT-SMR against Threats

Imagine the scenario where the Threat Detector was triggered and the system has to reconfigure itself to more resilient configuration, as explained already, using our methods this can be done at runtime without waiting for an agreement, because nodes initially agreed on which configuration to come back when the system was optimizing. The system before reconfiguring could calculate based on the latency matrix if this more resilient target configuration is efficient enough using a threshold, if it is then the system continues reconfiguring like before, but if its collectively verified that there are better choices as configurations and the threat level is not extremely high, replicas could then agree in a new configuration to go. The problem is that requiring a system to reach consensus in a scenario where adversarial strength can be increasing is dangerous and three different outcomes can happen. The first outcome is that if everything works correctly, agreement can be reached and the system switches to a different combination of nodes like in Figure 5.4 when reconfiguring from $n = 9$ to $n = 13$.

The second outcome is that what if the strength of the adversary is worse than expected and replicas can not reach agreement on which configuration to go. For that matter the system has to have a timeout on this change of thoughts about which configuration to go and if it runs out of time the system should follow the old configuration agreed. In the end is better to go to a not optimal configuration than miss the system adaptation completely, that's why this timeout is required

and can save the system in an almost deadlock situation.

The third outcome is the one that should be mostly avoided, and happens in the case that adversary is more powerful than expected such that it can lean the consensus to be reached towards a doomed configuration. An intelligent and coordinated attack unexpected by the Threat Detector could try to mess up the latency matrix, by applying collusion to every pair of faulty replicas and also increasing the latency against some correct target replicas, the fastest ones on the network usually, doing that attackers could try to make the system agree on the configuration they wanted instead of the right one. One of the core assumptions of BFT protocols is that the number of faulty replicas does not surpass f, but in the case of this scenario faulty replicas from different configurations could be combined to invalidate this assumption or even faulty passive replicas can also never re-activate and perhaps make the system lose its majority. Our system can avoid this outcome by rejuvenating passive replicas before joining or whenever they get re-activated rejuvenation should the first thing to be executed.

5.5 Conclusion

In conclusion, the proposed system offers a comprehensive solution for system reconfiguration, addressing both optimization and safety goals. During periodic MonitoringWindows, the system focuses on optimization by recalculating the latency matrix and applying sanitization methods from Chapter 4. This allows for robust decision-making regarding configuration changes, leading to improved system performance.

Reconfiguration with a safety goal does not require waiting for a monitoring window like defined by ThreatAdaptive in Chapter3. Instead, it is triggered by the Threat Detector, signaling an increase in the threat level. This proactive approach enables the system to move to a more resilient configuration promptly, potentially saving the system from increasing adversarial strength scenarios. Once the threat level is identified and remains unchanged, the system continues to reconfigure gradually moving towards the safest configuration or it can stay on the same configuration depending on its design. The system will only be deciding on the next configuration change when it's safe to do it, then it will recalculate the latency matrix and applies the sanitization methods, ensuring a well-prepared reconfiguration process.

The system also leverages reputation arrays to gain insight into the interconnection between nodes from a different perspective. By attributing reputation scores to nodes based on their performance, the system can identify discrepancies when compared to the latency measurements. Inconsistencies between good latency values and poor reputation scores indicate potential tampering or faulty

replicas.

Furthermore, the proposed system avoids the potential dangers of requiring consensus in an environment where adversarial strength may be increasing. Instead, it allows for collective verification and agreement on new configurations without relying on consensus. This dynamic reconfiguration process enables the system to adapt to changing circumstances efficiently and effectively. In summary, the proposed system combines optimization and safety-oriented reconfiguration techniques, leveraging MonitoringWindows, threat detection, reputation arrays, and latency-based optimizations. This comprehensive approach ensures improved system performance, robust decision-making, and efficient adaptation to changing threat levels, ultimately enhancing the overall resilience of the system.

Chapter 6

Conclusions and Future Work

This thesis presents resilient adaptation techniques for critical infrastructures in the face of advanced threats. It makes substantial contributions to the field of Byzantine fault-tolerant state-machine replication (BFT-SMR), specifically through the introduction of the Threat Adaptive protocol explained in Chapter 2 3 and advancements in latency optimization for distributed systems covered by Chapter 3 4. Threat Adaptive automatically adjusts its configuration based on threat level changes, allowing it to outpace adversaries, optimize performance, and adapt faster than previous methods while maintaining comparable latency and throughput to statically configured BFT baselines. We have presented two novel approaches that address critical challenges and improve the robustness and performance of distributed systems that apply latency-based optimizations. In Chapter 5, we combined the ideas and methods from both works into a single protocol and had a glimpse of the possible advantages and results of all these novel adaptation techniques combined could bring.

Firstly, we established the conditions for safe reconfiguration in a BFT-SMR protocol, enabling it to adaptively tolerate increasingly powerful adversaries based on the detected threat level. This led to the design of ThreatAdaptive, a groundbreaking BFT-SMR protocol that proactively agrees on resilient fall-back reconfigurations before optimizing its configuration. Our results demonstrate that ThreatAdaptive allows dynamic and secure performance optimization by reducing the number of actively participating servers. It is the first protocol that ensures reconfigurations towards safety to be executed by servers without the need for consensus to be reached, assuming detected threat level signals are received in advance. Experimental evaluations confirmed that our threat adaptive protocol achieves comparable throughput and latency to non-adaptive baselines while achieving reconfigurations 30% faster than previous methods with stronger safety assumptions. Possible extensions to ThreatAdaptive include extending or changing its fault model from homogeneous to hybrid with the inclusion of trusted compo-

nents to allow either smaller system configurations or to act as the distributed threat detector to be present on each and every replica, another possible future work could be to consider a weaker threat detector model.

Secondly, we introduced three novel methods to mitigate the impact of latency collusion attacks, which adversaries employ to exploit distributed systems with latency optimization mechanisms. Our methods make it difficult for faulty nodes to report false node-to-node latency information by requiring consistency in a multidimensional virtual coordinate space. The first latency sanitization algorithm, called Geometric, verifies the accuracy of latencies reported between a target node and other nodes in a system. It uses spheres to represent the distances between nodes, where the center of these spheres is defined by the VCS 3d position of each and every node, and the radius is represented by the round-trip time (RTT) between an arbitrary node and the others in the network. The new position of the target node is determined by identifying the point of intersection among a sufficient number of spheres. The second method called RndSamp is similar to the first with the main difference being the method utilized to calculate the intersection between n spheres. The third method is inspired by byzantine gradient averaging of MT-KRUM, that in our scenario is used in a virtual coordinated system with 3D coordinates in order to mitigate the impact of faulty nodes on the final latency matrix. Gradients, representing one node position calculated from the latency measurements between nodes, are computed and reconciled through Byzantine gradient averaging to determine the plausible location of a target node in 3D space. The sanitized latency matrix, obtained by recomputing latencies and aggregating them, is then used as input for the optimization process. Performance evaluations using real-world networking datasets demonstrated that our three approaches could provide up to 95% better protection against Byzantine nodes in terms of latency after reconfiguration, surpassing state-of-the-art solutions. Possible extensions to this work could involve using a punishing and rewards system to increase or decrease the impact of faulty servers when aggregating possible coordinates for an arbitrary node.

In future work, we plan to extend our approaches to incorporate other adaptation mechanisms commonly used in distributed systems, such as reputation and accountability. Reputation systems can be used during the payload protocol to indicate to the system which inputs should be taken more seriously, also there seems to be a strong correlation between nodes that broadcast the reply of a request first and the fastest nodes in the network. This can be further exploited to be used together with the measured latency matrix to make it even closer to the real latency matrix. Accountability could be attested to protocols that use latency-based optimizations, where for example, 1 or 2 agreements after reconfiguring the system, servers could regard consensus latency closely whether to check

if the optimization was worth it. If the latency promised as optimization was delivered, the system could generate a certificate signed by servers to verify that every step of the optimization process happened correctly. Overall, the findings presented in this thesis advance the understanding and practical implementation of threat-resilient BFT-SMR protocols and effective mitigation techniques against coordinated latency collusion attacks. The proposed solutions contribute to developing more secure, adaptive, and efficient distributed systems, paving the way for further research in this evolving field.

Bibliography

[11] *Proactive detection of network security incidents*. Tech. rep. ENISA, European Union Agency for Cybersecurity, Dec. 2011.

[16] *CBEST Intelligence-Led Testing: Understanding Cyber Threat Intelligence Operations*. Tech. rep. Bank of England, 2016.

[Abr+17a] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. "Revisiting fast practical byzantine fault tolerance". In: *arXiv preprint arXiv:1712.01367* (2017).

[Abr+17b] Ittai Abraham, Guy Gueta, Dahlia Malkhi, Lorenzo Alvisi, Rama Kotla, and Jean-Philippe Martin. "Revisiting fast practical byzantine fault tolerance". In: 2017.

[Acc20] Accenture". *State of Cybersecurity report 2020*. Tech. rep. 2020.

[Ami+10] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. "Prime: Byzantine replication under attack". In: *IEEE transactions on dependable and secure computing* 8.4 (2010), pp. 564–577.

[AMQ13] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. "Rbft: Redundant byzantine fault tolerance". In: *2013 IEEE 33rd International Conference on Distributed Computing Systems*. IEEE. 2013, pp. 297–306.

[Ari18] Andis Arins. "Blockchain based Inter-domain Latency Aware Routing Proposal in Software Defined Network". In: *2018 IEEE 6th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2018, pp. 1–2. DOI: 10.1109/AIEEE.2018.8592203.

[Aub+15] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. "The Next 700 BFT Protocols". In: *ACM Trans. Comput. Syst.* 32.4 (2015). DOI: 10.1145/2658994.

[Bar08] Catalonia-Spain Barcelona. "Mencius: building efficient replicated state machines for WANs". In: *OSDI*. USENIX. 2008.

[Ber+19] C. Berger, H. P. Reiser, J. Sousa, and A. Bessani. "Resilient Wide-Area Byzantine Consensus Using Adaptive Weighted Replication". In: *SRDS*. 2019, pp. 183–18309. DOI: 10.1109/SRDS47363.2019.00029.

[Ber+20] Christian Berger, Hans P Reiser, João Sousa, and Alysson Neves Bessani. "AWARE: Adaptive wide-area replication for fast and resilient Byzantine consensus". In: *IEEE TDSC* (2020).

[BGS15a] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. "Making BFT protocols really adaptive". In: *2015 IEEE International Parallel and Distributed Processing Symposium*. IEEE. 2015, pp. 904–913.

[BGS15b] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. "Making BFT protocols really adaptive". In: *PDPS*. 2015.

[BGS89] Daniel Barbara, Hector Garcia-Molina, and Annemarie Spauster. "Increasing availability under mutual exclusion constraints with dynamic vote reassignment". In: *ACM TOCS* 7.4 (1989), pp. 394–426.

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. "The latest gossip on BFT consensus". In: *arXiv preprint arXiv:1807.04938* (2018).

[Bla+17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. "Machine learning with adversaries: Byzantine tolerant gradient descent". In: *NeurIPS* 30 (2017).

[Boe18] Sergei Boeke. "National cyber crisis management: Different European approaches". In: *Governance* 31.3 (2018), pp. 449–464.

[BSA14a] Alysson Bessani, João Sousa, and Eduardo Alchieri. "State machine replication for the masses with BFT-SMART". In: *DSN*. 2014.

[BSA14b] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. "State machine replication for the masses with BFT-SMART". In: *2014 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE. 2014, pp. 355–362.

[CAA89] Shun Yan Cheung, Mustaque Ahamad, and Mostafa H Ammar. "Optimizing vote and quorum assignments for reading and writing replicated data". In: *IEEE TKDE* 1.3 (1989), pp. 387–397.

[Cha+96] Tushar Deepak Chandra, Vassos Hadzilacos, Sam Toueg, and Bernadette Charron-Bost. "On the Impossibility of Group Membership". In: *PODC*. 1996.

[CL+99] Miguel Castro, Barbara Liskov, et al. "Practical Byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999. 1999, pp. 173–186.

[CL99] Miguel Castro and Barbara Liskov. "Practical byzantine fault tolerance". In: *OSDI*. Vol. 99. 1999. USENIX. 1999, pp. 173–186.

[Cle+09] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, Mirco Marchetti, et al. "Making Byzantine fault tolerant systems tolerate Byzantine faults". In: *Proceedings of the 6th USENIX symposium on Networked systems design and implementation*. The USENIX Association. 2009.

[Coo14] Price Waterhouse Coopers. "The Global State of Information Security® Survey 2018". In: *Price Waterhouse Coopers* (2014).

[CP13] R Morris Coats and Gary M Pecquet. "The calculus of conquests: the decline and fall of the returns to Roman expansion". In: *The Independent Review* 17.4 (2013), pp. 517–540.

[Dab+04] Frank Dabek, Russ Cox, Frans Kaashoek, and Robert Morris. "Vivaldi: A decentralized network coordinate system". In: *CCR* 34.4 (2004), pp. 15–26.

[Dav89] Danco Davcev. "A dynamic voting scheme in distributed systems". In: *IEEE TSE* 15.1 (1989), pp. 93–97.

[DCK15] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. "Resource-efficient Byzantine fault tolerance". In: *IEEE transactions on computers* 65.9 (2015), pp. 2807–2819.

[Deh18] Ali Dehghantanha. *Cyber Threat Intelligence*. Springer, 2018.

[DGM02] An Das, Indranil Gupta, and Ashish Motivala. "SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol". In: *DSN*. Feb. 2002, pp. 303–312. DOI: 10.1109/DSN.2002.1028914.

[ED18] Michael Eischer and Tobias Distler. "Latency-aware leader selection for geo-replicated Byzantine fault-tolerant systems". In: *DSN-W*. IEEE/IFIP. 2018, pp. 140–145.

[FMC11] Nicolas Falliere, Liam O Murchu, and Eric Chien. "W32. stuxnet dossier". In: *White paper, symantec corp., security response* 5.6 (2011), p. 29.

[Fos+20] Luca Foschini, Andrea Gavagna, Giuseppe Martuscelli, and Rebecca Montanari. "Hyperledger Fabric Blockchain: Chaincode Performance Analysis". In: *ICC*. 2020, pp. 1–6. DOI: 10.1109/ICC40277.2020.9149080.

[GBN19] Miguel Garcia, Alysson Bessani, and Nuno Neves. "Lazarus: Automatic Management of Diversity in BFT Systems". In: *Middleware*. 2019.

[Gif79] David K Gifford. “Weighted voting for replicated data”. In: *ACM SOSP*. 1979, pp. 150–162.

[Gor+11] Katarzyna Gorzelak, Tomasz Grudziecki, Paweł Jacewicz, Przemysław Jaroszewski, Lukasz Juszczyk, Piotr Kijewski, and A Belasovs. “Proactive detection of network security incidents”. In: *ENISA Report* (2011), pp. 114–116.

[Gou] Bob Gourley. *Cyber Threat Intelligence Feeds*. Available at `https://thecyberthreat.com/cyber-threat-intelligence-feeds/` (2021/04/07).

[GSG02] Krishna P Gummadi, Stefan Saroiu, and Steven D Gribble. “King: Estimating latency between arbitrary internet end hosts”. In: *ACM SIGCOMM Workshop on Internet measurement*. 2002, pp. 5–18.

[Gue+10] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. “The next 700 BFT protocols”. In: *Proceedings of the 5th European conference on Computer systems*. 2010, pp. 363–376.

[Gue+20] Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. “Dynamic Byzantine reliable broadcast”. In: *OPODIS*. 2020.

[Her+95] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. “Proactive secret sharing or: How to cope with perpetual leakage”. In: *Advances in Cryptology—CRYPT0’95: 15th Annual International Cryptology Conference Santa Barbara, California, USA, August 27–31, 1995 Proceedings 15*. Springer. 1995, pp. 339–352.

[Her+97] Amir Herzberg, Markus Jakobsson, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. “Proactive public key and signature systems”. In: *Proceedings of the 4th ACM Conference on Computer and Communications Security*. 1997, pp. 100–110.

[Hos+18] Shohreh Hosseinzadeh, Sampsa Rauti, Samuel Laurén, Jari-Matti Mäkelä, Johannes Holvitie, Sami Hyrynsalmi, and Ville Leppänen. “Diversification and obfuscation techniques for software security: A systematic literature review”. In: *Information and Software Technology* 104 (2018), pp. 72–93.

[JM90] Sushil Jajodia and David Mutchler. “Dynamic voting algorithms for maintaining the consistency of a replicated database”. In: *ACM TODS* 15.2 (1990), pp. 230–280.

[Jon05] Captain J. Bryn L. Jones. “Space Weather Effects on Aircraft Operations”. In: *Effects of Space Weather on Technology Infrastructure*. Ed. by Ioannis A. Daglis. Dordrecht: Springer, 2005, pp. 215–234.

[Kap+12a] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. "CheapBFT: Resource-Efficient Byzantine Fault Tolerance". In: *EuroSys*. EuroSys '12. Bern, Switzerland: Association for Computing Machinery, 2012, pp. 295–308. DOI: 10.1145/2168836.2168866. URL: https://doi.org/10.1145/2168836.2168866.

[Kap+12b] Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. "CheapBFT: Resource-efficient Byzantine fault tolerance". In: *EuroSys*. ACM. 2012, pp. 295–308.

[KT20] Petr Kuznetsov and Andrei Tonkikh. "Asynchronous Reconfiguration with Byzantine Failures". In: *DISC*. 2020.

[Le+13] Duc Tai Le, Thong Le Duc, Vyacheslav V. Zalyubovskiy, and Hyunseung Choo. "Latency aware Broadcast Scheduling in Duty Cycled wireless sensor networks". In: *ICOIN*. 2013, pp. 48–53. DOI: 10.1109/ICOIN.2013.6496350.

[LV16] Shengyun Liu and Marko Vukolić. "Leader set selection for low-latency geo-replicated state machine". In: *IEEE TPDS* 28.7 (2016), pp. 1933–1946.

[MAK13] Iulian Moraru, David G Andersen, and Michael Kaminsky. "There is more consensus in egalitarian parliaments". In: *SOSP*. ACM. 2013, pp. 358–372.

[MC14] Robert Mitchell and Ing-Ray Chen. "A survey of intrusion detection techniques for cyber-physical systems". In: *ACM Computing Surveys (CSUR)* 46.4 (2014), pp. 1–29.

[MP17] Savita Mohurle and Manisha Patil. "A brief study of wannacry threat: Ransomware attack 2017". In: *International Journal of Advanced Research in Computer Science* 8.5 (2017), pp. 1938–1940.

[NMR21] Ray Neiheiser, Miguel Matos, and Luis Rodrigues. "Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation". In: *SOSP*. ACM. 2021, pp. 35–48.

[Obe+16] Daniel Obenshain, Thomas Tantillo, Amy Babay, John Schultz, Andrew Newell, Md Edadul Hoque, Yair Amir, and Cristina Nita-Rotaru. "Practical intrusion-tolerant networks". In: *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2016, pp. 45–56.

[PS18] Rafael Pass and Elaine Shi. "Thunderella: Blockchains with optimistic instant confirmation". In: *Advances in Cryptology–EUROCRYPT 2018: 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29-May 3, 2018 Proceedings, Part II 37*. Springer. 2018, pp. 3–33.

[Qam+17] Sara Qamar, Zahid Anwar, Mohammad Ashiqur Rahman, Ehab Al-Shaer, and Bei-Tseng Chu. "Data-driven analytics for cyber-threat intelligence and information sharing". In: *Computers & Security* 67 (2017), pp. 35–58. DOI: https://doi.org/10.1016/j.cose.2017.02.005.

[Rei96a] Michael Reiter. "Distributing Trust with the Rampart Toolkit". In: *Commun. ACM* 39.4 (1996), pp. 71–74.

[Rei96b] Michael K Reiter. "A secure group membership protocol". In: *IEEE Transactions on Software Engineering* 22.1 (1996), pp. 31–42.

[SB15a] Joao Sousa and Alysson Bessani. "Separating the WHEAT from the Chaff: An Empirical Design for Geo-Replicated State Machines". In: *SRDS*. 2015.

[SB15b] João Sousa and Alysson Bessani. "Separating the WHEAT from the chaff: An empirical design for geo-replicated state machines". In: *SRDS*. IEEE. 2015, pp. 146–155.

[Sei+13] Jeff Seibert, Sheila Becker, Cristina Nita-Rotaru, and Radu State. "Newton: securing virtual coordinates by enforcing physical laws". In: *IEEE/ACM Transactions on Networking* 22.3 (2013), pp. 798–811.

[Sil+21] Douglas Simoes Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Völp, and Paulo Esteves-Verissimo. "Threat adaptive byzantine fault tolerant state-machine replication". In: *SRDS*. IEEE. 2021, pp. 78–87.

[Sin+08] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. "BFT Protocols under Fire". In: *Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*. NSDI'08. San Francisco, California: USENIX Association, 2008, pp. 189–204.

[Sin58] J. David Singer. "Threat-perception and the armament-tension dilemma". eng. In: *The Journal of Conflict RSesolution* 2.1 (1958), pp. 90–105.

[SNV05] Paulo Sousa, Nuno Ferreira Neves, and Paulo Verissimo. "How resilient are distributed f fault/intrusion-tolerant systems?" In: *2005 International Conference on Dependable Systems and Networks (DSN'05)*. IEEE. 2005, pp. 98–107.

[SNV06] Paulo Sousa, Nuno Ferreira Neves, and Paulo Verıssimo. "Proactive resilience through architectural hybridization". In: *Proceedings of the 2006 ACM symposium on Applied computing*. 2006, pp. 686–690.

[SS16] Dean F Sittig and Hardeep Singh. "A socio-technical approach to preventing, mitigating, and recovering from ransomware attacks". In: *Applied clinical informatics* 7.02 (2016), pp. 624–632.

[Tan11] Colin Tankard. "Advanced persistent threats and how to monitor and deter them". In: *Network security* 2011.8 (2011), pp. 16–19.

[Ver+09] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. "Spin One's Wheels? Byzantine Fault Tolerance with a Spinning Primary". In: *2009 28th IEEE International Symposium on Reliable Distributed Systems*. 2009, pp. 135–144. DOI: 10.1109/SRDS.2009.36.

[Ver+11] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. "Efficient byzantine fault-tolerance". In: *IEEE Transactions on Computers* 62.1 (2011), pp. 16–30.

[Ver03] Paulo Veríssimo. "Uncertainty and predictability: Can they be reconciled?" In: *Future Directions in Distributed Computing*. Springer, 2003, pp. 108–113.

[Ver06] Paulo E. Verıssimo. "Travelling through Wormholes: A New Look at Distributed Systems Models". In: *SIGACT News* 37.1 (2006), pp. 66–81.

[Yin+19] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. "Hotstuff: Bft consensus with linearity and responsiveness". In: *PODC*. ACM. 2019, pp. 347–356.