Reliable Communication in Known Networks under the Hybrid Authentication Model

From Theoretical Guarantees to Real-World Deployments

Andrei Titu

Delft University of Technology

TU Delft

Reliable Communication in Known Networks under the Hybrid Authentication Model

From Theoretical Guarantees to Real-World Deployments

by

Andrei Titu

Instructor:	J. Decouchant
Project Duration:	October, 2024 - June, 2025
Faculty:	EEMCS, Delft

Cover:	Canadarm 2 Robotic Arm Grapples SpaceX Dragon by NASA under CC BY-NC 2.0 (Modified)
Style:	TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Preface

Reliable communication algorithms have existed for a while that assumed either a global authentication model backed by public key infrastructure or peer-to-peer authentication using shared session keys between pairs of neighboring nodes. Real-life networks, however, do not settle for only one or the other. Instead, they are dynamic, heterogeneous, and often composed of a mix of authentication capabilities across different nodes.

Recent work has introduced hybrid models that combine both authenticated links and authenticated processes to better reflect these real-world conditions. These models aim to preserve the strong correctness guarantees of Byzantine-resilient protocols while reducing their communication overhead and improving practical deployability.

This thesis builds on these ideas by introducing DualRC, a reliable broadcast protocol that explicitly supports hybrid authentication environments. It also presents a novel routed version of the protocol designed to shift complexity away from intermediate nodes and toward the sender and receiver, thereby improving scalability and efficiency. Alongside the theoretical contribution, this work includes the first implementation of both variants of DualRC and evaluates their performance across a range of network conditions, trust assumptions, and deployment scenarios.

The goal of this thesis is to demonstrate that reliable communication in partially trusted networks is not only possible but can be efficient, practical, and scalable—provided that protocol design embraces the diversity and complexity of modern distributed systems.

Andrei Titu
Delft, June 2025

Summary

This study evaluated DualRC under various trust configurations and network designs, highlighting how performance scales with the availability of authenticated and trusted nodes. It was found that while TEEs contribute modest gains, authentication—even when not globally deployed—leads to considerable reductions in latency and overhead. Routing optimizations further enhance performance, especially in dense or dynamic topologies. By framing Dolev's algorithm as a special case of DualRC, this work provided a clear comparative baseline and demonstrated how flexible trust assumptions can improve efficiency without compromising fault tolerance. These insights not only validate DualRC as a practical primitive for Byzantine-resilient communication, but also motivate further research into adaptable, trust-aware protocols for real-world deployment.

Contents

Nomenclature

If a nomenclature is required, a simple template can be found below for convenience. Feel free to use, adapt or completely remove.

Abbreviations

Abbreviation	Definition
DolevR	Routed version of Dolev's reliable communication protocol
DolevU	Unrouted version of Dolev's reliable communication protocol
RC	Reliable communication
TEE	Trusted execution environment

Symbols

Symbol	Definition	Unit
f	amount of faulty or Byzantine nodes in a network	integer
N	total amount of nodes in a network	integer
c	connectivity of a network - i.e. the minimum number of disjoint links that exist between any 2 nodes in a network	integer
A	number of authenticated nodes in the network	integer
T	number of trusted nodes in the network	integer

1 Introduction

Achieving reliable communication in adversarial environments remains a cornerstone challenge in distributed systems.

Traditional Byzantine Fault Tolerant (BFT) protocols, such as Dolev's algorithm, have long provided strong reliability guarantees by assuming a worst-case adversarial model and leveraging message redundancy through flooding. However, these protocols often incur prohibitive communication overhead, making them impractical for many real-world deployments. Recent advances in Trusted Execution Environments (TEEs), such as Intel SGX and ARM TrustZone, have opened new avenues for optimizing these protocols by enabling stronger authentication guarantees with lower trust assumptions.

Prior work presented DualRC, the first hybrid protocol that integrates Dolev's classical algorithm with a signature-based communication mechanism grounded in asymmetric cryptography, SigFlood. DualRC introduces a novel division between authenticated nodes, capable of cryptographic verification, and trusted nodes, those executing within TEEs and assumed to be uncompromised. This dual model enables a tunable trade-off between communication cost and trust assumptions, adapting to the heterogeneity of modern distributed systems.

We provide the first implementation of DualRC and evaluate its performance under varying system compositions, including differing ratios of authenticated and trusted nodes. Furthermore, we introduce a routed variant of DualRC that replaces the flooding behavior of classical Dolev-based protocols with routed forwarding. This variant significantly reduces message complexity while maintaining fault tolerance in systems where authenticated routes can be reliably established.

Our experimental results demonstrate the practical benefits and limitations of each mode of operation, offering insights into how TEEs and signature verification can be leveraged jointly for scalable, secure communication. This work lays the groundwork for future reliable communication protocols that combine cryptographic trust with hardware guarantees at the system level.

1.1. Research Questions

This paper tackles the following research questions:

1. **How does DualRC behave in practice?** Several assumptions need to be made about the way nodes authenticate each other, such as the availability of a public-key infrastructure, which implies additional overhead. Is the added benefit worth making these assumptions? How much better is DualRC compared to the basic Dolev's algorithm in practice?
2. **Do trusted execution environments bring a significant contribution to DualRC?** Theoretical claims about how authenticated or trusted nodes could impact performance have been made in the original paper. However, the benefit of using trusted execution environments remains an open-ended question.

3. **How would a routed version of DualRC perform compared to the non-routed version?** An even stronger assumption needs to be made with the routed version: that each node is constantly aware of the current network topology. Are the benefits of this protocol worth the effort of knowing the network topology?

1.2. Plan of the thesis

This thesis is organized to progressively build from the study of existing reliable communication protocols to the proposal and evaluation of a novel design.

The research paper begins with an overview of existing work in the field of Byzantine fault-tolerant consensus protocols 4, with particular emphasis on the mixed authentication model approach 3. By examining related literature and comparing key protocols, it aims to establish the context and identify gaps in performance or design. Following this, the thesis presents a detailed analysis of the Dual-RC protocol 5, including performance benchmarks and findings gathered during the implementation phase. This practical evaluation highlights the strengths and limitations of Dual-RC and motivates the design of a new consensus protocol. The final part of the thesis introduces this novel protocol, explains its design rationale, and provides an empirical evaluation of its performance compared to existing methods 5. All results are presented and discussed in their own section 6. This progression - from related work to analysis, to innovation and evaluation — serves to build a comprehensive understanding of the problem and contributes a new solution to the field.

2

The Reliable Communication Problem

Reliable communication refers to a property of a communication system in which messages sent from one party (the sender) are guaranteed to eventually be delivered, uncorrupted to the intended recipient (the receiver), despite any potential faults such as network delays, message loss, duplication, or reordering. In a byzantine scenario, this ensures that messages sent by correct (non-faulty) processes are delivered correctly, unambiguously, and consistently, even in the presence of up to f Byzantine faulty processes.

Specifically, the reliable communication primitive typically satisfies the following properties:

- **Validity:** If a correct sender broadcasts a message, then all correct processes eventually deliver that message.
- **Agreement:** If a correct process delivers a message, then all correct processes eventually deliver the same message.
- **Integrity:** Every correct process delivers the message at most once, and only if it was broadcast by the claimed sender.

These properties ensure that malicious actors cannot forge, alter, or prevent the agreement on messages being exchanged, even if some subset of the system behaves arbitrarily.

However, in this paper, it is always assumed that the sender is a correct node. Therefore the reliable communication with correct sender model is generally assumed. Therefore, this primitive does not prevent a faulty process from *equivocating*-that is, sending different conflicting messages to different neighbors, when it takes the role of the broadcaster.

To handle this, the *Byzantine Reliable Broadcast* primitive is used. It builds upon reliable communication and ensures that **all correct processes deliver the same set of messages**, even if the sender is Byzantine. This stronger primitive requires two assumptions:

- Reliable communication is available.
- At most one-third of the processes in the system are Byzantine faulty.

A more complex problem is the *Byzantine Consensus*. In this problem, each correct process starts with its own initial value m_i, and the goal is for all correct processes to agree on a single value m, where m is one of the m_i. Solving Byzantine Consensus requires the previously described primitives to exist.

2.1. Byzantine faults

A **Byzantine fault** is a type of failure in distributed systems where components (e.g., nodes or servers) behave in an *arbitrary or malicious* manner. Such components may send conflicting or incorrect information to different parts of the system. These faults are particularly challenging because their behavior is unpredictable and not easily detectable.

- **Arbitrary behavior:** Faulty nodes may act maliciously or erratically, including sending conflicting messages.
- **Difficult to detect:** It is not straightforward to determine which node is faulty.
- **Worst-case failure mode:** Byzantine faults encompass all other types of faults (e.g., crash, omission) as special cases.
- **Requires robust agreement protocols:** Solving the problem requires consensus algorithms that tolerate Byzantine behavior.

The concept comes from the **Byzantine Generals Problem** [25], a theoretical problem in distributed computing. In this problem, a group of generals must agree on a battle plan, but some generals may be traitors trying to mislead others. The goal is to achieve consensus despite the presence of such traitors.

Examples of Byzantine actors in real life scenarios:

- A node in a blockchain network broadcasting fake transactions to some peers and real ones to others.
- A malfunctioning server sending corrupted responses.
- A compromised insider node intentionally disrupting communication.

To tolerate Byzantine faults, systems employ **Byzantine Fault Tolerant** protocols. These protocols ensure correct primitives as long as certain requirements around the presence of Byzantine actors are met.

2.2. Reliable Communication as a Foundational Primitive

As seen before, at the heart of most distributed systems lies the fundamental assumption of *reliable communication*.

Reliable communication does not require strong timing guarantees or agreement between processes, it simply ensures that messages are not lost or tampered with in transit. This minimal yet crucial guarantee forms the base upon which more complex distributed protocols are built.

2.2.1. Building on Reliable Communication

From reliable communication, more advanced primitives can be constructed:

- **Message Ordering Guarantees:** With reliable delivery, systems can enforce stronger properties such as FIFO (first-in, first-out) or causal ordering. These are essential when the order of events must be preserved across distributed processes.
- **Failure Detection:** While reliable communication ensures eventual delivery, processes can use timeouts and heuristics to infer failures. These *failure detectors* are foundational for fault-tolerant algorithms.
- **Atomic Broadcast and Total Order Broadcast:** Building on reliable delivery, it is possible to ensure that all correct processes deliver the same messages in the same order. These primitives are vital for achieving agreement in distributed systems.
- **Distributed Consensus:** Protocols such as Paxos, Raft, and PBFT rely heavily on reliable communication. They require that messages sent by one node (e.g., a proposer or leader) reach other nodes (e.g., acceptors or followers) accurately. Without reliable delivery, the correctness guarantees of these protocols cannot be maintained.

The construction of distributed systems can be viewed as a hierarchy of abstractions:

1. **Reliable Communication** — ensures messages from a correct sender are eventually delivered to their destination.
2. **Ordering and Failure Detection** — adds context and responsiveness to communication.
3. **Reliable Broadcast [bracha1987asynchronous]** — enables nodes to agree on the delivery of a message
4. **Total Order Broadcast [25]** — enables agreement on message order.

5. **Distributed Consensus [25]** — achieves agreement on values across processes.
6. **Replicated State Machines, Blockchain, Coordination Services** — leverage consensus to implement complex distributed applications.

Each level in this hierarchy assumes the guarantees of the level below. In particular, *distributed consensus cannot be achieved without reliable communication*, even in asynchronous systems, because messages must eventually be delivered for any form of agreement or progress to occur.

2.3. Performance Evaluation Metrics

When comparing algorithms, the exact metrics after which the performance is measured are:

- Delivery latency: the averaged amount of time that has passed from the moment that a sender initiates the broadcast to the moment that each receiver node has delivered. In other words, the time elapsed until a node delivers, on average.
- Network consumption: Total amount of bytes traveling through the network. This metric is directly influenced by and can be broken down into the total amount of messages sent through the network and the average size of each message. In the experiments, network consumption has been measured in terms of total message bytes processed by every node, divided by the total number of nodes in the network. This includes the sender.

3

System model

In this section, we outline the system model assumed throughout the paper. Our work considers a static, asynchronous, and locally bounded Byzantine fault-tolerant environment.

3.1. System Model: Processes

In the model at hand, a *process* is defined as an independent computational entity operating within a distributed system [28, 4]. We consider a static system composed of a fixed set of n processes. Each process is uniquely identified by an integer identifier. We denote the process with identifier i as p_i, and let the set of all processes be $P := \{p_1, p_2, \ldots, p_n\}$.

Processes are stateful and communicate by exchanging messages over a network. Each correct process follows a common distributed protocol $\mathcal{P}$, which is assumed to be stored in a tamper-proof, read-only memory. This ensures that the logic governing the behavior of correct processes cannot be altered, even in the presence of an adversary.

We further distinguish between different levels of trust associated with processes:

Authenticated Processes: A process is said to be authenticated if its identity is verifiable by others through cryptographic means, such as digital signatures or message authentication codes (MACs). This ensures that a message claimed to be sent by p_i indeed originated from p_i and has not been tampered with. Authenticated processes are essential in settings where the communication links are insecure and adversarial manipulation of messages is possible.

Trusted Processes: A trusted process is assumed to always behave according to the protocol $\mathcal{P}$ and is not subject to adversarial control. Trust assumptions are usually minimized in Byzantine fault-tolerant systems, but some protocols rely on a small number of trusted processes (or trusted components such as secure hardware) to simplify protocol design or enhance security guarantees.

All processes that are not Byzantine faulty — that is, those that continue to follow protocol $\mathcal{P}$ correctly — are referred to as *correct*.

3.1.1. Process Knowledge Assumptions

We make several assumptions regarding the knowledge held by processes in the system.

First, we assume that every process knows the upper bound f on the number of Byzantine faulty processes in the system, depending on the considered failure model — global, local, or mobile. However, processes do not know the identity of the faulty peers and cannot determine which processes are compromised.

Processes are assumed to have no global knowledge of the communication topology, unless otherwise stated. That is, they are unaware of the graph $G = (V, E)$ representing the network including the set of nodes and the links between them.

We distinguish between two levels of local topology awareness:

- **Know Neighborhood (KN):** Processes are aware of the identities of their direct neighbors — the set of processes they share communication links with.
- **Unknown Neighborhood (UN):** Processes do not know their neighbors explicitly. Instead, communication is supported via a `send_link` primitive that broadcasts a message to all immediate neighbors without requiring knowledge of their identities.

3.1.2. Authentication Model

We consider a model in which some processes are *authenticated*. An authenticated process possesses a private cryptographic key and has access to a Public Key Infrastructure (PKI). The PKI enables authenticated processes to obtain and verify the public keys of other authenticated peers. This infrastructure allows for secure message origin authentication and integrity verification between authenticated processes.

Authentication provides the ability to cryptographically sign and verify messages, thus enabling the recipient to verify both the sender's identity and the message's integrity, assuming the sender is authenticated and honest.

Link Assumptions

The communication network is composed of point-to-point links between processes. We consider *perfect reliable and authenticated links*, as if the two ends of the link have access to a shared authenticated session - cryptographically speaking. However, authenticated links can also be implemented directly at hardware level [39], without any need for cryptography. These links satisfy the following properties that help prevent impersonation attacks [15]:

- **Reliable delivery:** If a correct process p_i sends a message m to a correct process p_j, then p_j eventually receives m.
- **No duplication:** Messages are delivered at most once; no correct process receives the same message more than once.
- **Authenticity:** If a correct process p_j receives a message m from sender p_i, then m was indeed sent by p_i to p_j.

This model allows us to distinguish between authenticated and unauthenticated processes and to analyze how authentication influences fault tolerance and the design of secure distributed protocols.

3.2. Communication Model

A **static network** assumption implies that the topology of the network is fixed throughout the execution of the protocol. Nodes have stable identities and the communication links between them do not change over time. This assumption simplifies protocol design and analysis, as it eliminates the complexity introduced by dynamically joining or leaving nodes and varying communication paths. In this paper a static network will always be assumed.

Additionally, we adopt an **asynchronous communication model**, making no assumptions about the timing of message delivery, network delays, or the responsiveness of nodes. This means that messages sent between nodes may be delayed arbitrarily, delivered out of order, or even experience temporary losses before eventual delivery. Importantly, the system cannot distinguish between a slow node and a faulty one based on timing alone.

This is in contrast to the **synchronous model**, in which the system assumes known, fixed upper bounds on message delivery times and processing delays. In synchronous systems, nodes can rely on timeouts and scheduled actions to detect faults or coordinate with one another. Such timing guarantees simplify the design of consensus protocols and failure detectors.

However, real-world distributed systems often operate in unpredictable environments, where such timing assumptions are unrealistic. Networks may experience congestion, variable latency, or partial outages, and node responsiveness may fluctuate due to load or failures. The asynchronous model captures this uncertainty and represents a more robust and realistic foundation for protocol design.

By embracing the asynchronous model, this work targets protocols that remain correct and secure regardless of the timing or order of messages. This model reflects the worst-case conditions of real networks and ensures that the correctness of the system does not depend on any form of timing synchronization or reliable scheduling.

3.3. Fault Model

In distributed systems, the fault model defines the types of failures and adversarial behaviors that a protocol must tolerate. It is a crucial part of the system design, particularly in Byzantine fault-tolerant systems, where nodes may behave arbitrarily or maliciously.

In this work, we assume a powerful adversary that has complete knowledge of the system and can influence its behavior by corrupting processes. When a process is corrupted, it becomes *Byzantine faulty*, meaning it can act in any arbitrary or malicious way — such as failing to send messages, sending contradictory messages to different peers, or providing invalid information. This type of fault model is considered the most general, as it subsumes simpler failures like crashes, message omissions, and passive behaviors like eavesdropping [28].

There are three different models that describe how faults can be distributed across the system:

1. Globally Bounded Faults: At most f processes in the entire system can be Byzantine faulty at any given time. This set of faulty nodes is fixed and does not change over time.

2. Locally Bounded Faults: Instead of limiting the total number of faulty nodes globally, this model restricts the number of faulty neighbors each process can have. Every process may connect to at most f Byzantine nodes.

3. Mobile Faults [3]: Here, the number of faulty processes at any point in time is still limited to f, but the actual set of faulty nodes can change over time. This models adversaries who control mobile agents capable of moving between nodes. When an agent resides on a process, it renders it Byzantine faulty. Once the agent moves away — which it can do over links connecting processes, respecting a minimum time interval λ between movements — the previously faulty process returns to correct behavior. We refer to such processes as *cured*. Depending on the model, cured processes may or may not retain awareness of having been previously faulty.

All processes that are not currently faulty are considered *correct* and are assumed to follow the specified distributed protocol faithfully.

In this work, the **global fault model** is assumed. Also, a single, omnipresent adversary controls all faulty nodes in the system. This adversary is capable of coordinating the behavior of corrupted nodes in a fully synchronized manner, potentially observing all network traffic and exploiting timing or delivery patterns to subvert the protocol. This model represents a powerful adversary and is often used to prove the robustness of protocols under worst-case conditions.

The **static adversary** assumption further constrains the threat model by requiring that the set of nodes under adversarial control be fixed before the protocol begins. The adversary must choose which nodes to corrupt ahead of time and cannot adaptively change its targets based on protocol execution or observed messages. This is in contrast to a dynamic adversary, which may compromise nodes during the execution of the protocol.

4

Related work

The problem of achieving reliable communication in the presence of Byzantine faults has been studied extensively in distributed systems literature. Early foundational work established the theoretical limits of fault tolerance and proposed baseline broadcast primitives. Since then, numerous protocols have been developed to address varying assumptions about synchrony, authentication, and network reliability. In this section, we briefly survey key lines of research that inform and motivate further research presented in this paper.

4.1. Applications in Modern Systems

Reliable communication primitives form the cornerstone of many Byzantine fault-tolerant systems deployed today. In particular, Dolev's Reliable Communication and Bracha's Reliable Broadcast primitives provide formal guarantees about message delivery and agreement, even when some nodes in the system behave arbitrarily or maliciously. Their authenticated variants, which incorporate digital signatures or message authentication codes (MACs), offer more efficient implementations in realistic system models and are extensively used in practice.

Blockchain and Decentralized Consensus

Authenticated broadcast primitives are fundamental in the design of consensus protocols used in blockchain systems. For example, permissioned blockchain platforms like Hyperledger Fabric use digital signatures to ensure message authenticity and to achieve agreement among participants. Bracha's broadcast protocol, or variants thereof, often underpin the dissemination of transactions and blocks in such systems, ensuring that all honest nodes reach the same state despite adversarial behavior by a subset of participants. The use of authenticated messages significantly reduces the communication complexity, which is critical for scalability and performance.

Replicated State Machines

Reliable broadcast is also a key component in replicated state machine (RSM) replication protocols. Systems such as PBFT (Practical Byzantine Fault Tolerance) and BFT-SMaRt use authenticated reliable broadcast mechanisms to ensure that all replicas process the same sequence of requests. This is vital for consistency in distributed services such as databases, financial systems, and critical control infrastructure. The correctness guarantees of these primitives directly translate into service reliability and data integrity.

Secure Messaging Protocols

Group messaging systems like Signal and Matrix rely on cryptographic forms of reliable broadcast to ensure that messages sent in a group reach all recipients reliably and in order, even in the presence of malicious actors or network partitions. Authenticated variants of reliable communication are used to verify message origins and prevent tampering, which is essential for end-to-end security and trust in modern communication applications.

Cloud-Native and Microservice Architectures

In microservice-based architectures and cloud deployments, reliable message dissemination is necessary for configuration management, service discovery, and failover handling. Authenticated reliable communication primitives ensure that critical updates (e.g., configuration changes, container orchestration events) are propagated correctly across the system. This reduces the likelihood of service inconsistencies or cascading failures in large-scale deployments, especially when operating over potentially unreliable network substrates.

Software Update and Patch Distribution

Secure and reliable software update mechanisms, such as those used in automotive systems or IoT devices, benefit from authenticated reliable communication. These systems must guarantee that only valid updates from trusted sources are installed, and that all intended recipients receive the same update version. Using primitives such as Dolev's with authentication ensures tamper-resistant propagation, even in partially connected or hostile network environments.

Overall, the practical deployment of these primitives demonstrates their essential role in maintaining correctness, security, and availability in distributed systems that face both faults and adversarial behavior.

4.2. Reliable Communication in the authenticated link model

Dolev's reliable communication algorithms rely on a set of relaxed assumptions about the network connectivity, the nature of the requests and the byzantine participants present in the system:

- The processors are arranged in a k-connected bidirectional network with N.
- Every processor knows the members of the network.
- Processors communicate only by sending messages along links.
- Every processor can identify the neighbor from which it receives each message. But it can only identify its neighbors and no one else meaning the authenticated links authentication model is used.
- A message includes the route through which it was delivered.
- A reliable processor relays a message only after it appends the name of the processor from which it received the message to the message's route.
- A reliable processor relays messages without neither altering them nor eavesdropping on their values.
- There exists an upper bound on the delay of relaying a message by a reliable processor. Even though this is a rather minimal constraint it is nevertheless an assumption about the arrival time of a message, which means a synchronous communication model is used.
- There exists an upper bound, f, on the number of faulty processors in the system.

Additonally, Dolev's reliable communication protocol heavily references Menger's theorem on connected graphs: "if the connectivity of a graph is k, then there exist at least k disjoint paths between every pair of nodes of that graph."

Under the above above conditions, both the routed and the unknown network variants of Dolev's reliable communication protocol offers the following guarantee:

Reliable communication is achieved in a given network of processors if and only if:

- The amount f of byzantine nodes is smaller than a third of the amount N of total nodes in the network.
- The amount f of byzantine nodes is smaller than half of the amount k denoting the network's connectivity.

4.2.1. Dolev's protocol on routed networks: DolevR [14, 13]

In the routed newtwork variant of the protocol a few very handy pre-condition is present: every receiver knows the entire network topology and therefore knows beforehand the routes through which the transmitter will send the messages.

In this setup, a transmitter sends multiple copies of the same message along separate, disjoint paths to each receiver. The purpose of using multiple paths is to limit the impact of any single faulty processor along a given route. If one route is compromised due to faults, other independent routes can still carry the correct message to the receiver.

Upon receiving messages, the receiver verifies that the actual paths taken by the messages align with the planned routes to filter out obvious cases where the faulty nodes have tempered with the route history. If a message's route does not match the intended path (suggesting potential tampering or faults in the processors along that route), the message is discarded. The remaining messages, which have traversed the correct routes, are termed "good messages". At this point good messages aren't all neccessarily true, because the faulty nodes could have still not modified the route information sent along the paths, but only modified the sent value in the messages which is a more subtle attack.

After filtering, the receiver analyzes the remaining good messages to find the correct value. If the receiver finds at least f + 1 good messages that carry the same value, it accepts this value as the transmitted message. If no value has at least f + 1 good messages, the receiver defaults to a special "empty" value, indicating uncertainty about the transmitted value. This typically occurs if the transmitter itself is faulty and has sent conflicting messages. Dolev's algorithm is proven to work in a network where the connectivity is at least 2f + 1, meaning there are enough disjoint paths to bypass faults. If a reliable transmitter sends 2f + 1 copies of its message along disjoint routes, and the number of faulty processors is at most f then the receiver is guaranteed to receive at least f + 1 correct messages. Since there can be at most f conflicting messages from faulty routes, the correct message value will be the majority among the good messages, allowing the receiver to reliably determine it.

The algorithm also prevents "false alarms," meaning if the transmitter sends no message, the receiver will not mistakenly infer a valid message from noise or faulty transmission; it will simply receive the special "empty" value.

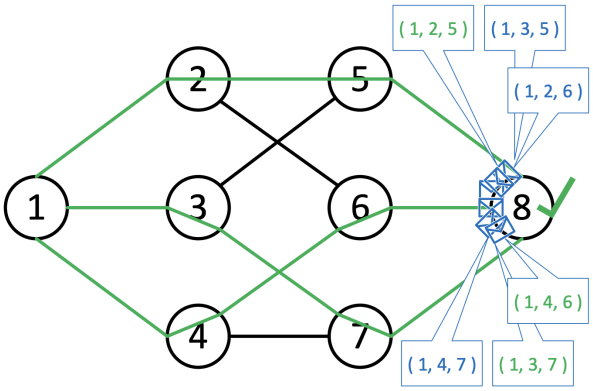


Figure 4.1: Message transmission in DolevR

4.2.2. Dolev's protocol on unknown topology: DolevU [14]

In this version, neither the transmitter (sender) nor the receiver knows the specific paths or routes that messages will take through the network. Unlike in a routed network where paths are predetermined and known, in an unknown network, messages are broadcasted through the network in a flood-like manner, with nodes forwarding messages to their neighbors, to guarantee that the message will eventually reach the intended recipient. An additional and significant issue with this setting is the potential for faulty processors to generate an unbounded number of messages. This could result in an overload of messages at reliable nodes, making it harder to filter out the correct value.

The algorithm has each node implement a 'Send' function to propagate messages through the network.

When a transmitter sends a message, it does so by broadcasting to its immediate neighbors, who in turn forward it to their neighbors (leaving out the direction from which they received the message to avoid an endless loop), propagating the message throughout the network. Each message contains the sender's value 'a' and a list of nodes (the "route") that it has passed through. This list is used later to filter out unreliable messages. In the 'Send' algorithm, each node checks if the route embedded in the message contains valid system members, with each processor appearing only once. If the route meets these criteria, the message is accepted; otherwise, it is ignored. If the message's route seems valid according to the previous check, the receiving node appends its own identifier to the route list and forwards the message to all its neighbors not already in the route. This process ensures that each reliable processor receives multiple, independent copies of the message through different routes, which is crucial for the next stage of filtering and validation. After a receiver collects multiple copies of the message, it applies the 'Purifying ' algorithm to identify and isolate the correct value. This algorithm's purpose is to filter out any "fictive" (false) values that may have originated from faulty processors. Two messages are considered independent if they have been relayed by entirely disjoint sets of processors. A set of messages is deemed independent if every pair within the set is independent. The receiver looks for a subset of messages with at least f + 1 independent copies. If all messages in this subset carry the same value, that value is accepted as the correct "purified" value. If no such subset exists, the algorithm outputs a default "empty" value, indicating uncertainty. The Purifying Algorithm is designed to yield a non-empty value only if the transmitter actually sent one. This is critical to prevent faulty processors from producing misleading results. If no legitimate message is sent by the transmitter, any messages received are either invalid (incorrect routes) or insufficiently independent, leading the receiver to settle on an empty value.

If a reliable transmitter uses the 'Send' algorithm to broadcast a value, every reliable receiver in the network will be able to obtain this value using the ''Purifying algorithm. This is because a reliable transmitter broadcasts the same message to all its neighbors. Since there are at least f + 1 independent paths to each reliable receiver, the receivers will eventually gather enough independent, correct messages carrying the transmitter's value. The 'Purifying' method then filters out any conflicting values from faulty processors, leaving the correct value.

To address potential flooding from faulty processors, the algorithm imposes a finite time window in which messages are collected and processed. This limits the receiver's exposure to endless faulty messages.

If the transmitter is byzantine, the protocol can never reach consensus.

Optimizations

To enhance the efficiency of content dissemination in gossip-based protocols, Bonomi et al. [**bonomi2019multi**] proposed a series of optimizations aimed at reducing overhead and improving scalability. These optimizations are summarized below:

- **MOD_1_BONOMI** – *Deliver if received from source*:
  When a node receives content directly from the source, it immediately delivers it without further validation. This reduces latency and eliminates redundant checks.
- **MOD_2_BONOMI** – *Upon delivery, send empty path*:
  After a node successfully delivers the content, it forwards a message with an empty path to inform neighbors of its delivery, helping them adjust their forwarding strategy accordingly.
- **MOD_3_BONOMI** – *Relay only to neighbors that haven't delivered*:
  To minimize message redundancy, a node relays the content only to neighbors that have not yet delivered it, conserving bandwidth and processing resources.
- **MOD_4_BONOMI** – *Ignore paths containing a node that sent an empty path*:
  If a node receives an empty path from node q, it will ignore any future paths that contain q. This prevents wasting resources on paths that involve nodes which have already delivered the content.
- **MOD_5_BONOMI** – *Stop processing content after delivery*:
  Once a node has delivered the content, it ceases any further forwarding or analysis of that content, further reducing computation and communication overhead.

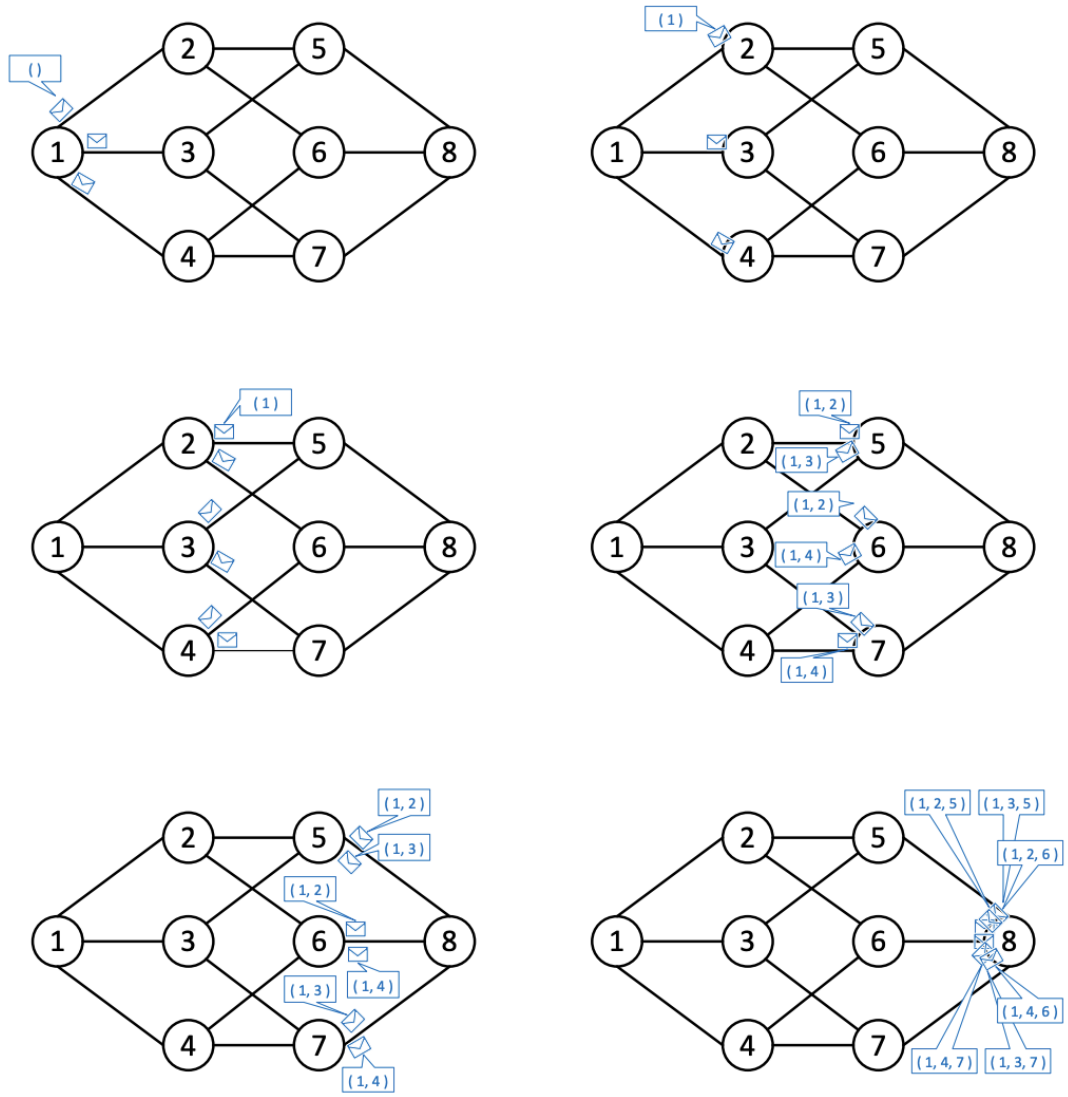


Figure 4.2: Message transmission in DolevU

4.2.3. Explorer: from DolevU to DolevR

Routing has considerable advantages when it comes to bandwidth and latency. However, to employ such a feature network participants need to be aware of the network topology. Due to the ever growing and changing nature of networks nowadays, statically defining routing tables for each node ahead of participation is not viable. Therefore, processes need to find a way to reliably and dynamically determine a view of the real topology in addition to the direct links and neighbors.

The **Explorer** algorithm, proposed by Nesterenko and Tixeuil [**Nesterenko2002**], addresses the *Byzantine fault-tolerant topology reconstruction* problem in distributed systems, with asynchronous communication, global fault model and static adversary and network. Its objective is to allow each correct process to reconstruct the communication topology of a distributed network, even in the presence of Byzantine (arbitrarily faulty) nodes, given a reliable communication primitive is already at hand.

Nesterenko and Tixeuil established two impossibility results:

No algorithm can determine whether a two-faulty edge exists. An edge $\{p_i, p_j\}$ known only to faulty nodes p_i and p_j cannot be reliably verified, as both endpoints may lie or remain silent.

No algorithm can reconstruct a topology that includes only real nodes and edges while also including all correct and one-faulty edges. In unknown networks, each link is known only to its endpoints. A faulty node can

falsely declare or deny the existence of a link with a correct node, making verification impossible.

Each process p_i executes the following two procedures:

1. **Broadcast Neighborhood Information**:

$$\text{Broadcast } \Gamma(i)$$

   where $\Gamma(i)$ is the set of neighbors of p_i.

2. **Collect and Store Topology Information**:

   Upon receiving reliable messages from other processes, each process p_i builds a dictionary:

$$\texttt{cTop}_i := \bigcup \langle j, \Gamma(j) \rangle$$

   where each pair $\langle j, \Gamma(j) \rangle$ represents the neighborhood of process p_j as received by p_i.

Each process p_i uses the collected data $\texttt{cTop}_i$ to compute a local view G_i of the global topology G. Since Byzantine processes can equivocate (send inconsistent information to different receivers), it is possible that:

$$G_i \neq G_j \quad \text{for some correct processes } p_i \text{ and } p_j$$

This algorithm comes, whatever, with certain limitations:

- **Equivocation**: Byzantine processes can send conflicting neighborhood information due to unicast links.
- **Divergent Reconstructions**: Correct processes may compute inconsistent topologies.
- **Partial Visibility**: Some edges or nodes may be missing or falsely reported.
- **No Liveness Guarantee**: A later result [**Bonomi2019**] shows that the reliable communication primitive used in Explorer does not ensure liveness in all scenarios.

4.2.4. Explorer2

Explorer2 [30] is a revised version of the Explorer protocol designed for reconstructing the network topology in the presence of Byzantine faults. It is specifically built to handle the *unknown neighborhood* scenario, where processes do not initially know which other processes they are connected to.

At the beginning of execution, each process sends a simple `HELLO` message to discover its neighbors. Upon receiving such a message, a process adds the sender to its list of neighbors. This list may grow over time as more neighbors are discovered through message exchanges.

Whenever a process detects new neighbors and its neighborhood list grows, it reliably broadcasts this updated list using a Byzantine Fault Tolerant (BFT) primitive. Other processes that receive these updates replace any older version of the neighborhood with the new, expanded version. Since the neighborhood list is monotonically growing, this ensures consistency over time.

Explorer2 includes an implicit failure detector: if a process receives two inconsistent neighborhood lists from another process (that do not follow a strict growth pattern), the sender is considered faulty and excluded from the reconstruction.

Each process uses the neighborhood information it has collected to reconstruct a local view of the network:

- A node is added to the topology if it has been directly declared or has been mentioned by a sufficient number of other nodes.
- A connection between two nodes is added if at least one of them declares the link.
- The reconstruction can be made more conservative by requiring mutual acknowledgment of links to reduce the risk of including spurious edges.

Explorer2 guarantees several key properties:

- All correct nodes and correct edges are eventually included in the reconstructed topology.

- Spurious nodes and edges, which do not exist in reality, are excluded unless declared by faulty processes.
- Faulty processes may be excluded or have only partial links included, especially in the unknown neighborhood setting.
- With local broadcast links, all correct processes eventually compute the same network topology.

The protocol's communication cost depends on the neighborhood knowledge:

- Under the known neighborhood assumption, each process broadcasts its neighborhood only once, leading to $O(n)$ reliable broadcasts.
- Under the unknown neighborhood assumption, processes discover neighbors incrementally and may broadcast multiple times, leading to $O(n^2)$ reliable broadcasts in the worst case.

Explorer2 is a Byzantine-resilient topology discovery algorithm that ensures eventual consistency and correctness across all non-faulty processes. It is especially notable for its ability to work under minimal assumptions (e.g., unknown neighborhoods) and for leveraging a reliable broadcast primitive to disseminate and validate neighbor information robustly.

4.3. Reliable Communication in the authenticated process model

As public key cryptography evolved and reliable libraries became available to a wider audience, solutions have been implemented to improve reliable communication bottlenecks [2, 16, 5].

4.3.1. SigFlood

SIGFLOOD [8] is a broadcast protocol designed for networks of authenticated nodes, where each node possesses a unique private signing key and a corresponding public key known to all other participants. The protocol achieves reliable communication by leveraging digital signatures to ensure the authenticity and integrity of transmitted messages. When a sender node wishes to broadcast a message, it first signs the message using its private key. This signed message is then sent to all other nodes in the network. Upon receiving a message, each recipient verifies the signature using the sender's public key. If the signature is valid and the message has not already been delivered, the node accepts and delivers the message. Because each message must carry a verifiable signature, SIGFLOOD guarantees that nodes only deliver messages genuinely produced by the claimed sender, thereby preventing forgery and tampering. This protocol assumes a cryptographic model in which signature forgery is computationally infeasible, ensuring that even in the presence of Byzantine nodes, no node can impersonate another. SIGFLOOD avoids the complexity of quorum-based or path-based Byzantine fault tolerance protocols by relying solely on cryptographic identity, making it lightweight and suitable for scalable deployments. However, it does not prevent a malicious sender from broadcasting multiple conflicting messages, and thus assumes the sender itself is non-Byzantine or must be handled through additional mechanisms in higher layers of the system.
However, a key drawback of SIGFLOOD is its reliance on a public key infrastructure (PKI) to distribute and manage the public keys of all nodes. This requirement introduces operational overhead, as PKI systems can be complex to deploy, manage, and keep secure—especially in dynamic or large-scale systems where keys must be rotated or trust relationships updated over time.

4.3.2. From Authenticated Links to Authenticated Processes: CryptoRC

We assume the presence of an adversary whose capabilities are limited: they cannot forge digital signatures, meaning that no malicious process can generate fake messages that appear to be legitimately signed.

CryptoRC [19] is a prototype protocol that ensures reliable communication with authentication. It works in two phases. First, each participant generates a pair of cryptographic keys: one public and one private. These keys are used for digital signatures. Every process then shares its public key with the others using an existing reliable communication method, such as the DolevU protocol. When another process receives this key, it associates the key with the identity of the sender.

Once this key exchange is completed, the system switches to using a SIGFLOOD, which takes advantage of the shared cryptographic keys to provide efficient and authenticated reliable communication.

CryptoRC guarantees the safety of reliable communication. Assuming no two processes can have the same key pair and that the digital signature system cannot be broken by the adversary, it follows from previous remarks that the communication remains secure and authentic. CryptoRC guarantees the liveness of reliable communication if the network connectivity is strictly greater than the number of faults. Since the public keys are distributed using a reliable communication protocol that works correctly under the current system assumptions, all correct processes will eventually receive all public keys from other correct processes. Given the robustness of the network and the strength of the assumptions, the system ensures that messages will eventually be delivered.

CryptoRC begins with an initialization step where all processes share their public keys using a reliable communication method. This step requires each process to participate in a communication instance, resulting in a total number of such instances proportional to the number of processes. After this setup, the actual authenticated communication becomes significantly more efficient.

4.4. Trusted hardware components

Trusted Execution Environments (TEEs) are isolated regions of a processor that provide security guarantees for code and data loaded within them. TEEs ensure confidentiality and integrity, even in the presence of a potentially compromised operating system. Examples of TEEs include Intel SGX, ARM TrustZone, and AMD SEV.

These hardware-backed environments are increasingly being used in distributed systems for a range of security properties, including privacy-preserving computation and integrity assurance. TEEs can execute sensitive operations securely and attest to their correct execution, making them a valuable primitive for building secure distributed protocols.

4.4.1. Applications in Privacy-Preserving Systems

Several works have leveraged TEEs to provide strong privacy guarantees in distributed and federated learning settings. For instance, [34, 22, 29] explore the integration of TEEs for training machine learning models over sensitive data without compromising user privacy. TEEs are used to isolate model parameters, gradients, or entire training loops, ensuring data confidentiality even when run on untrusted infrastructure.

- In [34], the authors present DeepLearn, a TEE-based system for secure multi-party training that minimizes the trust placed on cloud infrastructure.
- FedSGX [22] incorporates Intel SGX into federated learning to protect model updates and aggregation, providing a practical balance between performance and security.
- In [29], TEEs are employed to ensure secure parameter exchange in privacy-preserving federated learning across untrusted devices.

4.4.2. Applications in Integrity-Critical Protocols

Beyond privacy, TEEs are also employed to guarantee integrity in distributed protocols, particularly in the context of Byzantine fault tolerance (BFT) and consensus algorithms. Notable examples include:

- **MinBFT** and **CheapBFT** utilize TEEs to reduce the number of replicas required for Byzantine fault tolerance by simplifying trust assumptions [38, 9].
- **Damysus** and **OneShot** use TEEs to streamline view changes and reduce complexity in BFT protocols [11, 12].
- Trusted components have also been proposed to facilitate **leaderless consensus** and **reliable broadcast**, where the TEE ensures that only valid messages are processed and relayed.

These approaches exploit the attestable and tamper-proof nature of TEEs to remove or simplify some of the cryptographic or quorum-based checks traditionally required for integrity.

4.4.3. Reliable Communication with Trusted Components

To the best of our knowledge, **DualRC** [8] is the first protocol that employs TEEs specifically for *reliable communication*, a foundational building block of distributed systems. Although previous protocols have

used TEEs for higher-level primitives such as consensus and broadcast, DualRC is unique in targeting reliable point-to-point communication.

In this work, we build on the insights from DualRC and extend it by proposing the first **routed reliable communication protocol for known networks** that leverages trusted components. Our protocol demonstrates *for the first time*, the empirical benefits of trusted components on both **communication complexity** and **latency**. This represents a significant step in the use of trusted hardware for core communication tasks in distributed systems.

4.4.4. Attacks on Trusted Components

Despite their strong security guarantees, TEEs are not impervious to attack. Known classes of attacks include:

- **Side-channel attacks:** TEEs like Intel SGX have been shown vulnerable to cache timing attacks, branch prediction attacks, and speculative execution-based attacks such as Foreshadow [36], SGXpectre [6], and Plundervolt [37]. Other attacks exploit branch shadowing [26] to infer enclave control flow.
- **Software vulnerabilities:** Bugs in the enclave software itself can undermine the security guarantees [10].
- **Rollback and replay attacks:** Without persistent and tamper-proof storage, TEEs can be tricked into reprocessing stale or manipulated data [21, 33].

These vulnerabilities highlight the importance of careful design and threat modeling when incorporating TEEs into distributed protocols. While TEEs offer powerful capabilities, they are not a silver bullet and must be complemented by robust software engineering and complementary security mechanisms.

TEEs represent a powerful tool for improving both privacy and integrity in distributed systems. From privacy-preserving machine learning to integrity-critical consensus protocols, TEEs have shown promise in enhancing security without excessively compromising performance. Our proposed routed reliable communication protocol marks a novel use case of TEEs and empirically demonstrates their positive impact on communication efficiency in known network settings.

4.5. Reliable communication in the local fault model

The existence of other fault models has led the field to protocol variations.

For example, the **Certified Propagation Algorithm (CPA)** [24, 32, 35] is a method for ensuring that messages can be reliably delivered between processes in a distributed system, even when some processes may behave maliciously (known as *Byzantine faults*). CPA is specifically designed for systems that with static and asynchronous network and local fault model.

The system assumes that each correct (non-faulty) process may be connected to at most f faulty (Byzantine) neighbors. This is known as the *locally bounded Byzantine fault model*.

The algorithm proceeds according to three simple rules:

1. **The origin (source) process delivers its message and sends it to all of its neighbors.**
2. **Any process that receives the message directly from the source delivers it and forwards it to all of its neighbors.**
3. **If a process receives the same message from more than f different neighbors (i.e., at least $f+1$), it delivers the message and then forwards it to all of its neighbors.**

These rules ensure that even if up to f neighbors are malicious and send false or conflicting information, correct processes can still determine which message is authentic and relay it correctly.

Whether CPA works correctly depends on certain properties of the network, which are defined in terms of how the nodes are connected. To understand this conditions, several concepts need to be defined:

An f-local set [27] refers to a set of potentially faulty processes such that **no correct process is directly connected to more than f of them**. In other words, even in the worst-case placement of faults, each

process will still have a majority of correct neighbors.

For a given process (let's call it P_i), imagine trying to organize all other processes into levels based on how the message spreads from P_i under the CPA rules. The value associated with P_i, called **the maximum spreading depth for reliable communication from that process**, is the deepest such level structure that can be formed starting from P_i, denoted by $j_i(G)$.

The **worst-case spreading depth** among all processes in the network is denoted by $j(G)$. That is, we look at the maximum reliable communication depth for every process and pick the smallest one.

Finally, we define a value that tells us **how many faulty neighbors a process can tolerate** and still manage to reliably send a message to everyone, no matter how the faulty processes are placed (as long as the f-local condition is satisfied). It's defined as the maximum value of f such that CPA still works correctly even in the presence of any f-local set of faulty processes. This is denoted $h_i(G)$.

- **One-to-All Communication from a Given Process**: [31, 23] This is possible if:
  - The maximum number of faulty neighbors that the process can tolerate (i.e., its fault tolerance depth) is greater than f, or
  - The process can spread its message across more than $2f$ reliable layers in the network.
- **Any-to-Any Communication in the Entire Network**: This is possible if:
  - Every process in the network can tolerate more than f faulty neighbors, or
  - The worst-case message spreading depth (across all processes) is more than $2f$.

Computing whether each process meets the first condition (based on fault tolerance depth) is a very hard (NP-hard) problem. However, checking the second condition (based on message spreading levels) can be done efficiently using known algorithms.

- **Message Complexity**: Each process forwards a message only once to each of its neighbors. So, the total number of messages is proportional to the number of connections (edges) in the network.
- **Delivery Complexity**: Each process only needs to wait for up to $(f + 1)$ identical messages. So the time it takes a process to decide is proportional to f.
- **Communication Latency in Synchronous Systems**: The time it takes for a message to reach all processes depends on the network structure and how the faulty processes are placed. Using theoretical tools like **Minimum k-Level Ordering**, upper and lower bounds on delivery time can be estimated.

4.6. Reliable communication in the hybrid authentication model

Recent work has introduced protocols that adapt to both assumptions about authentication methods to more accurately represent the heterogenity of real-world networks.

4.6.1. DualRC

In distributed systems, achieving reliable communication in the presence of Byzantine failures has traditionally relied on uniform assumptions about authentication. Many classic protocols presume either (i) the presence of *authenticated links* between all communicating pairs, or (ii) the universal availability of *authenticated processes*—that is, every process can sign and verify messages using a public key infrastructure (PKI). However, these assumptions are often unrealistic in modern heterogeneous networks such as IoT deployments, edge computing environments, or cross-organizational systems, where nodes may have widely varying trust guarantees and cryptographic capabilities.

DualRC (Dual-mode Reliable Communication) addresses this challenge by proposing a protocol that operates under a *hybrid authentication model*, one that accommodates both authenticated links and authenticated processes within the same system. This flexibility allows DualRC to adapt to heterogeneous trust and security configurations without requiring uniform security properties across all nodes or links.

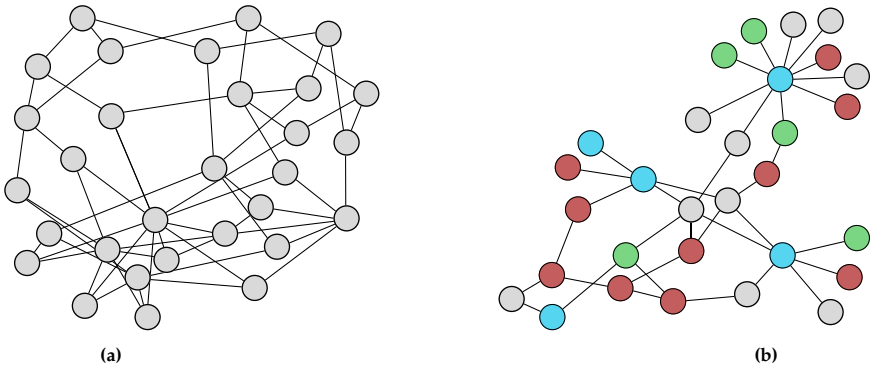


Figure 4.3: (a) In a network of 30 authenticated nodes with connectivity (3), both SigFlood and DualRC ensure reliable communication when $f = 1$. (b) In a complex network with 10 authenticated (red), 10 non-authenticated (gray), 5 authenticated trusted (green), and 5 non-authenticated trusted (blue) nodes, only DualRC achieves reliable communication.

Motivation and Design Principles

The central premise of DualRC is that combining authenticated links and authenticated processes can yield stronger fault tolerance and more efficient communication than relying on a single mode of authentication. In practice, this means that messages can propagate through the network using the most secure and efficient available route—either through authenticated point-to-point links (e.g., via MACs over TLS) or through verifiable signatures when such links are not available.

To support this, DualRC modifies the underlying reliable broadcast algorithm to:

- Leverage digital signatures for end-to-end authentication when messages pass through untrusted or unauthenticated links;
- Utilize hop-by-hop trust on paths that consist of authenticated links to reduce cryptographic overhead;
- Dynamically adapt path selection based on the authentication capabilities of intermediary nodes;
- Integrate petitioning mechanisms, whereby non-authenticated nodes can validate a message by collecting a sufficient number of independent signatures from authenticated or trusted nodes.

Protocol Behavior and Correctness

DualRC operates by sending messages through multiple *vertex-disjoint paths*, leveraging network redundancy to tolerate Byzantine behavior. For each destination node, the sender computes the maximum flow in the network to determine the number of disjoint paths, ensuring that the message can reach its target even in the presence of up to f Byzantine faults.

Authenticated nodes can sign messages, while non-authenticated nodes forward messages with *petitions*—accumulated sets of signatures from trusted sources that attest to the message's correctness. A receiving node delivers the message once it either:

- Receives a valid signed message from a trusted source;
- Collects sufficient (i.e., $f + 1$) distinct valid signatures in the petition;
- Receives the message through at least $f + 1$ disjoint paths.

Correctness verification in DualRC involves checking that no forged or inconsistent message can be accepted by an honest node, even when up to f nodes in the system behave arbitrarily. The hybrid model requires reasoning over both the cryptographic guarantees provided by authenticated processes and the structural guarantees provided by authenticated paths.

Role of Trusted Execution Environments

DualRC further enhances reliability and efficiency by incorporating *Trusted Execution Environments* (TEEs), such as Intel SGX. These are hardware-isolated environments that can perform sensitive operations—like signature generation or validation—securely and attestably. When deployed in key nodes, TEEs:

Protocol	Auth. links	Auth. nodes	Trust. nodes	Trust. comp	Routed	Correctness
DolevU [14]	✓	✗	✗	✗	✗	$2f+1$ vertex-connectivity
DolevR [14]	✓	✗	✗	✗	✓	$2f+1$ vertex-connectivity
SigFlood [18]	✗	✓	✗	✗	✗	$f+1$ vertex-connectivity
CryptoRC [19]	✓	✗	✓	✗	✗	$f+1$ vertex-connectivity
DualRC [7]	✓	✓	✓	✓	✗	$2f+1$ vertex-connectivity
Routed-DualRC (this work)	✓	✓	✓	✓	✓	$2f+1$ vertex-connectivity

Table 4.1: Reliable communication protocols, their system models, and correctness conditions.

- Reduce the cost of trust by minimizing the number of processes that must be fully authenticated;
- Act as cryptographic anchors in portions of the network with limited authentication capabilities;
- Increase the effective number of "trusted" paths without requiring global PKI enrollment.

Benefits and Drawbacks

One of the key practical benefits of DualRC is its *graceful degradation*: the protocol remains correct and partially efficient even when only a subset of the network is authenticated. This makes it highly suitable for incremental deployment in existing systems. Additionally, by not requiring full PKI enrollment or universal link authentication, DualRC reduces administrative and computational overhead, making it practical for deployment in edge computing, cloud federations, or cross-domain secure collaboration.

However, the algorithm present several limitations due to:

- **Network consumption and latency** Given that each node broadcasts all messages it receives, the total amount of messages in the network grows following a factorial function. This floods the network and also causes nodes to mostly receive and process redundant messages.
- **Time complexity per node** Finding N-disjoint paths in a pool of received paths turns out to be an NP-hard problem so the time complexity in each node is exponential.

DualRC represents a significant advancement in Byzantine-resilient communication by moving away from rigid trust assumptions. Through its hybrid design, support for trusted components, and efficient routing optimizations, it aligns with the needs of modern, heterogeneous distributed systems. It provides a scalable, flexible foundation for secure communication in real-world settings where trust is partial, uneven, or evolving.

5

Routed DualRC

This chapter introduces Routed DualRC, a novel protocol designed to achieve reliable broadcast in distributed systems composed of both authenticated and unauthenticated nodes. Routed DualRC builds upon the foundational concepts of the original DualRC algorithm, enhancing its performance by incorporating explicit routing and signature-based message validation. The protocol is inspired by two main lines of work: (1) the Dolev protocol, which guarantees reliable communication via multiple disjoint paths, and (2) SigFlood, which exploits cryptographic guarantees provided by public key infrastructures (PKIs) to accelerate agreement among authenticated participants.

Unlike its unrouted counterpart, Routed DualRC pushes more responsibility to the sender and receiver nodes, relieving intermediary forwarders from the need to actively validate or reinterpret message content. This shift allows for a simpler forwarding model—nodes merely forward messages along predefined routes—which in turn facilitates more scalable and bandwidth-efficient message propagation. Additionally, Routed DualRC introduces a mechanism for path truncation, where intermediate nodes that have already delivered a message can strip redundant routing information, further reducing overhead.

A full implementation of Routed DualRC has been developed in C++ as part of this research effort [1]. The implementation is now open source, providing the community with a working prototype that adheres closely to the algorithm's theoretical design. This codebase has been used extensively to benchmark the protocol in various network configurations and trust models. It serves both as a practical tool for evaluating performance trade-offs and as a reference for future extensions or deployments in secure networked systems.

In the sections that follow, we present the algorithmic design of Routed DualRC in detail, explain its core components, and discuss optimizations that enhance both delivery latency and message complexity. We also explore the protocol's behavior under varying trust assumptions and demonstrate its effectiveness through empirical evaluation.

5.1. Motivation for implementing the Unrouted version of Dual-RC

Before proposing and validating the routed variant of Dual-RC, it was essential to implement and thoroughly evaluate the unrouted version. This step served as the foundational layer for both theoretical and empirical understanding of the algorithm. Dual-RC, as originally introduced, was described only in pseudocode form and had never been implemented or tested in a real system. Consequently, the unrouted implementation was a critical prerequisite to verify correctness, identify corner cases, and understand the practical behavior of the algorithm in both fault-free and Byzantine scenarios.

The unrouted implementation also serves as a *performance and correctness baseline*, helping answer questions like: How does Dual-RC perform in the absence of optimizations? What is the effect of varying trust assumptions? And most importantly, where are the algorithm's bottlenecks in terms of communication, latency, and redundancy? These insights were indispensable before introducing the

additional complexity and trade-offs inherent to a routed version.

Furthermore, during the implementation process, several *discrepancies and omissions in the original pseudocode* were discovered, which could have led to incorrect behavior in specific network configurations. These findings were not only corrected in the working code but also documented for future research. A public C++ implementation of Dual-RC, including both the unrouted and routed versions, has been made available for reproducibility and further analysis [1]. This implementation serves as a reference and testbed for ongoing and future work in reliable communication over partially trusted networks.

Therefore, this chapter contributes in the following ways:

- **Implementing and Improving Dual-RC:**
  The original Dual-RC paper outlined the protocol in pseudocode but did not provide a reference implementation. As part of this research, a robust implementation of the unrouted version was developed from scratch in C++. In the process, a number of logical inconsistencies and inefficiencies were identified. For instance, the original pseudocode failed to address certain conditions under which a node should avoid forwarding a message—for example, if a neighbor had already delivered it, or if the path was empty. Furthermore, a critical optimization opportunity was overlooked: if a node collects more than f valid (non-faulty) signatures all attesting to the same message, it can deliver that message without further verification. This insight is particularly powerful in environments with frequent broadcast events and is now integrated into the updated implementation.
- **Highlighting Edge Cases and Security Vulnerabilities:**
  The unrouted implementation also revealed subtle security vulnerabilities not discussed in the original design. One such issue was the potential for *replay attacks* in the presence of repeated messages from the same sender. Because the message payload alone was being signed, malicious nodes could replay older messages with valid signatures to trick others into delivering outdated information. To mitigate this, the updated implementation now includes a *broadcast nonce* or *per-sender counter* as part of the signed payload. This ensures that each broadcast instance is uniquely identified and immune to replay.
- **Providing a Foundation for the Routed Version:**
  The unrouted implementation also served as the control group for evaluating the performance gains introduced by the routed variant, which shifts much of the responsibility from the intermediate nodes to the sender and receiver. Without this baseline, it would be difficult to quantify whether the routed version meaningfully improves efficiency or simply redistributes complexity.
- **Open Source Contribution:**
  The implementation, benchmarks, and experiments described in this chapter are all publicly available in the released C++ repository [1]. This codebase includes detailed logging, configuration support for varying trust assumptions, and a modular design that facilitates experimentation with additional optimizations, such as path caching, signature batching, and TEE-based acceleration.

In summary, the unrouted version of Dual-RC was more than a stepping stone—it was a vital tool for understanding the protocol's strengths, limitations, and real-world behavior. It enabled a clearer vision for designing the routed version, validated several theoretical assumptions, and highlighted areas where performance could be improved through optimizations or integration with trusted hardware.

The next section builds directly upon this work, showing how the use of trusted execution environments (TEEs) can affect performance in practical deployments.

5.2. Implementing and improving DualRC

In this paper, the first implementation of the DualRC algorithm [7] is provided. During the implementation process, several issues and improvements in the original description of the algorithm were identified and addressed to ensure correctness and robustness.

Several small improvements which have been collaboratively addressed in the original pseudo-code and which will not be amply discussed include: omitting to relay an empty path, also after the signed handler delivered a message; and omitting to check whether neighbours have already delivered before

relaying the message to them in some scenarios. But most significantly, a delivery condition which could drastically improve delivery speed has been overlooked. Namely, when a number of distinct non-trusted signatures that is higher than the number of faulty nodes jointly attest the same value, then that value must be the real one. DualRC was already checking for the total amount of disjoint signed paths, but was not checking for different signatures. Even though outside of the scope of the original paper, of important notice when dealing with multiple broadcasts in the same network coming from the same sender is the possibility of a replay attack. Signing the message payload alone leaves room for a replay exploit where the attacker remembers signatures of previous values and sends them in the network to cause nodes to deliver this previous message because the signature will verify. However, this could be easily avoided by also adding a nonce to the signature, or a broadcast counter per node. That is, a value that is never the same for two distinct broadcasts. The presented implementation adheres closely to the intended methodology while incorporating these necessary corrections, making it a reliable reference for future applications and evaluations of the algorithm. The repository with the C++ implementation has been made public.

This paper contributes an analysis of the performance characteristics of DualRC under varying trust assumptions. In the absence of authenticated or trusted nodes, except for the broadcaster, DualRC reduces to the classic Dolev reliable communication protocol. This special case serves as a baseline for comparing the performance improvements introduced by enhanced trust models. The evaluation shows that the presence of authenticated nodes leads to significant efficiency improvements, reducing latency by up to 40% and network overhead by approximately 30%. These findings indicate that although public key infrastructures (PKIs) can be complex and resource-intensive to deploy, even partial adoption of authentication mechanisms across a distributed system can lead to substantial performance benefits.

5.3. Determining the impact of TEEs

DualRC leverages trusted components to optimize the speed of reliable message delivery across a distributed network. While the use of trusted execution environments (TEEs) is not strictly necessary for DualRC to achieve its fundamental goals, it is hypothesized that TEEs can, in certain scenarios, accelerate message propagation and reduce overhead. This hypothesis stems from the ability of TEEs to provide secure and efficient execution of critical operations, minimizing computational bottlenecks and reducing the need for redundant message verification. However, the actual impact of TEEs on DualRC's performance remains an open question.

To validate or challenge this assumption, I conducted a series of experiments designed to systematically compare the performance of DualRC with and without trusted execution. These experiments involved deploying DualRC in controlled network environments with different configurations, measuring key performance indicators such as the total number of messages sent, the total amount of data transmitted over the network, and the time required for full message delivery across all participating nodes. By running these benchmarks under varying conditions, including high-latency and lossy networks, I aimed to assess whether TEEs consistently provide a measurable advantage.

One of the main factors influencing delivery speed in DualRC is the efficiency of cryptographic operations and message authentication. TEEs, such as Intel SGX or ARM TrustZone, can offload cryptographic computations to dedicated hardware, potentially reducing the latency associated with signature verification and encryption. Furthermore, TEEs can facilitate more efficient delivery mechanisms by providing attestation guarantees, which may reduce the number of redundant message exchanges required for agreement. In theory, this should lead to lower message complexity and a reduction in the overall data transmission volume. However, TEEs also introduce additional overhead due to enclave management, context switching, and memory constraints, which could offset the anticipated benefits.

The benchmarking process involved deploying DualRC across a simulated network topology with the amount of nodes ranging between 20 and 50, testing both TEE-enabled and non-TEE implementations. Each experiment was repeated multiple times to ensure statistically reliable results, and performance metrics were collected for comparative analysis. The results revealed that, under low latency conditions, the TEE enabled version of DualRC achieved a 10% reduction in the delivery time of the message compared to the baseline implementation. However, in high-latency environments, the benefits of TEEs were less pronounced, with delivery time reductions averaging only 2%. Similarly, while TEEs

helped reduce the number of messages exchanged in some cases, the overall data transmission volume was sometimes higher due to additional signed path data added to each message required by the cryptographic operations and integrity checks performed within the trusted execution environment.

These findings suggest that while TEEs can enhance the efficiency of DualRC in certain network conditions, their effectiveness depends on a trade-off between computational acceleration and the overhead introduced by secure enclave management. Future optimizations could explore hybrid approaches, selectively leveraging TEEs for specific tasks such as initial authentication while relying on conventional execution for routine message passing.

Ultimately, this paper contributes empirical data to the ongoing discussion about the practical benefits of TEEs in distributed systems. By systematically evaluating their impact on DualRC, these experiments provide valuable insights into when and where TEEs are most beneficial, helping inform future design choices in secure and efficient network communication protocols.

A thorough presentation of the results and methodology can be found in the experiments section of the paper.

5.4. Introducing Routed DualRC

The prior work section has discussed the routed version of Dolev, whose main idea is that the sender needs to find $2f + 1$ disjoint paths in the known topology between itself and the target, and relay the message through all of these paths. Also, SigFlood was brought up, which is based on the cryptographic guarantees of signatures created by keys that are tied to nodes' identities. Here, the source node only has to compute $f + 1$ disjoint paths to the target and relay the signed message through each of them, to ensure that the target eventually delivers.

Like its unrouted version before it, Routed DualRC attempts to tackle the reliable communication problem in mixed networks which are composed of both authenticated (that are enbaled by public key infrastructure and are able to verify signatures as well generate them) and unauthenticated nodes. The algorithm does this by combining concepts from SigFlood and the routed version of Dolev's algorithm for reliable communication. This combination is presented next. Three node types can be distinguished in the Routed DualRC presentation: sender, forwarder and receiver. The main difference between the routed version compared to the unrouted alternative is the fact that most of the work is shifted towards the sender and receiver from the intermediary nodes. Most of the time, the sender node will pull all the ropes and decide where and how messages should travel, while intermediary nodes will only obey the sender's directions. The receiver nodes are to verify the delivery conditions. For this, each message should carry a linked list of nodes that represents the route on which a message should travel. Each intermediary node only needs to find itself in the linked list, verify that the node prior to itself is indeed the link sender, and then forward the message untouched to the node that follows in the list.

5.4.1. Starting from the basics

From now on, an authenticated node will be denoted by A ease of writing and reading, and an unauthenticated one by U. The algorithm could be analysed via four different cases:

- Source is A, and target is A. Intuitively, SigFlood can be applied here to send a signed message f+1 times.
- Source is U, and target is U. Basic Dolev approach could be used and $2 * f + 1$ paths are computed for the message to be routed through.
- Source is A, and target is U. Naive approach dictates that we should also compute $2 * f + 1$ paths. Because receiver cannot verify signed messages.
- Source is U, and target is A. Naive approach again dictates that we should also compute $2 * f + 1$ paths, because sender does not know how to sign messages.

With the basic approach, three out of four times this version is not really doing better than the original routed Dolev algorithm. However, several optimizations could be made to improve the performance.

5.4.2. Optimizations

Some optimizations can be performed even in the absence of trusted node.

In terms of delivery speed, the improvement discovered in the regular DualRC can be applied. The the mixed case U to A can be improved by leveraging a "petition" of signatures from authenticated nodes that delivered and are found on the paths between the U source and A target. When the target accumulates $f + 1$ distinct signatures all claiming the same message value, it knows that $f + 1$ amount of nodes claimed to have delivered the same message. Assuming only f faulty nodes exist, then it must be that this is the real value.

In terms of the total amount of messages being sent but with slight implications in the delivery speed, the sender can pre-compute the paths for all receivers by maximizing the number of shared paths between distinct receivers. In this way, the "2 flies with 1 hit" principle can be applied. Also,

And lastly but not least, in terms of message size, an intermediary node that delivered, can truncate the linked-list representing the route existing in the message before forwarding it, by removing all nodes up to itself.

If trusted nodes exist in the network, their contribution is towards delivery speed, but also making delivery possible in network topologies that made it impossible before, further relaxing the graph connectivity requirement.

5.4.3. Finding disjoint paths between a source and a target

This detail is the fundament of the entire algorithm. There are several ways of finding disjoint paths in a graph whose topology is known. This paper will focus on the *max-flow* approach for solving this problem [20].

The maximum flow approach is a well-established method for finding disjoint paths in a graph between a given source and target. The main idea is to model the problem as a network flow instance, where each edge has unit capacity, and the objective is to maximize the flow from the source to the target. The number of paths found by this approach corresponds to the maximum number of edge-disjoint paths.

Consider a simple undirected graph $G = (V, E)$ with $V = \{s, a, b, c, t\}$ and edges:

$$E = \{(s, a), (s, b), (a, c), (b, c), (c, t)\}$$

This graph has multiple routes from s to t, but they are not all node-disjoint. To find the maximum number of node-disjoint a max-flow algorithm (e.g., Ford-Fulkerson) is applied on this directed graph from source s to target t (which are not split). The resulting flow value corresponds to the maximum number of node-disjoint paths. In this case, the result is 2 node-disjoint paths:

$$s \to a \to c \to t \quad \text{and} \quad s \to b \to c \to t$$

These paths overlap at c, so only one can be used if we require node-disjoint paths. After transformation, the graph ensures only one of these paths can use node c, so the max number of node-disjoint paths is 1.

A significant detail is that a transformation of the graph is necessary, since the current model requires *node-disjoint* paths instead of edge-disjoint ones. Each node v is split into two nodes: v_{in} and v_{out}, connected by a directed edge with unit capacity. The incoming edges of v are connected to v_{in}, while outgoing edges are connected from v_{out}. This transformation ensures that each node can be used at most once in a path, enforcing node disjointness. More exactly, the following transformation are exerted on the graph prior to max flow computation:

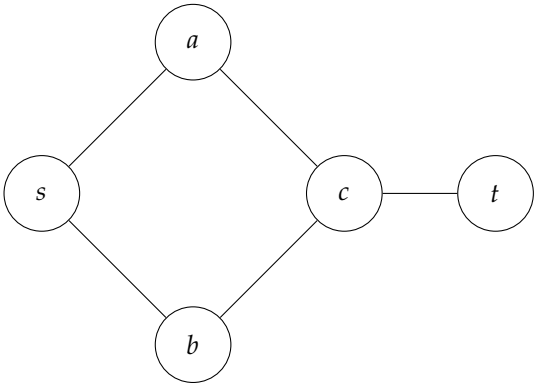
- Each node $v \in V \setminus \{s, t\}$ is split into two nodes v_{in} and v_{out} connected by a directed edge $(v_{in} \to v_{out})$ of capacity 1.
- All incoming edges to v are redirected to v_{in}, and all outgoing edges are redirected from v_{out}.

The intuition of this algorithm is clear when the graph is envisioned as a train rail network, where trains are of infinite lengths. The goal of the train network is to avoid collisions between trains while also trying to get travelers to their destinations. There is only one train allowed on each rail, and if two rails

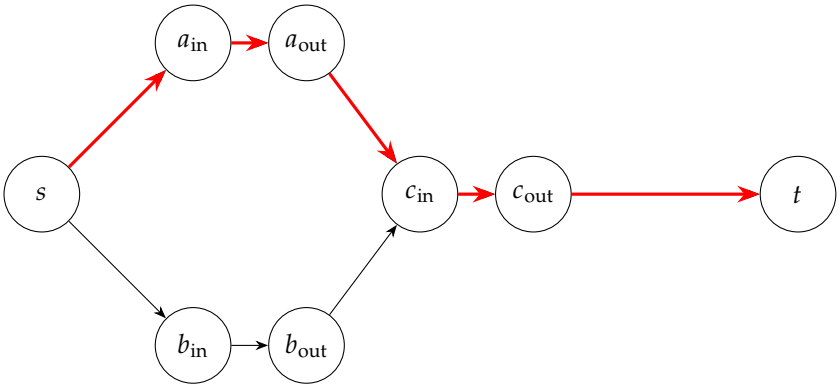
intersect (i.e., two paths share a node), only one train can pass, while the other must take an alternative path or stop to avoid a crash.

A maximum flow computation on the transformed graph then yields the maximum number of node-disjoint paths.

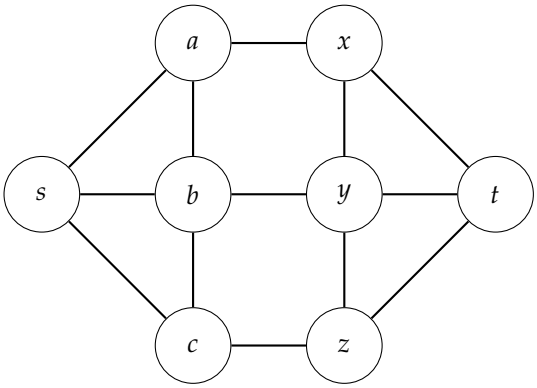
For example, the above graph would look like this:



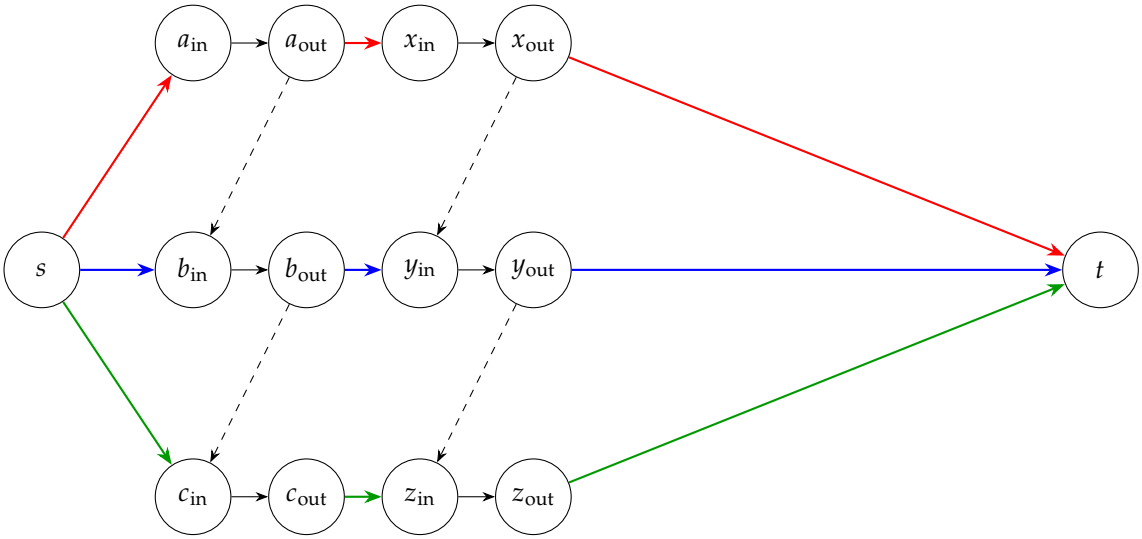
While the transformed graph on which the max flow is computed looks like this:



A more complex example is shown in the graph below:



After the transformation, the max flow is computed yielding a maximum configuration of three disjoint paths.



The constructed graph yields exactly **3 node-disjoint paths** from the source node s to the target node t, and this is the **maximum possible** number of such paths. Below, we provide a detailed explanation of why this is the case.

Definition Recap. A set of *node-disjoint paths* between two nodes in a graph means that:

- No two paths share any intermediate nodes (i.e., nodes other than s or t).
- These paths may share edges or neighbors, but not the internal nodes they traverse.

Identified Paths. From the graph presented, the following three node-disjoint paths from s to t can be identified:

1. $s \to a \to x \to t$
2. $s \to b \to y \to t$
3. $s \to c \to z \to t$

Each of these paths uses a unique sequence of intermediate nodes (a, b, c, x, y, z) which are not shared between any two paths, satisfying the condition for node-disjointness.

Node Capacity Limitation. The number of node-disjoint paths between s and t is upper-bounded by the minimum *vertex cut* between them. In this graph:

- Node s connects to only 3 distinct nodes: a, b, c.
- Each of these connects uniquely to x, y, z, respectively, which then connect to t.
- To satisfy node-disjointness, no intermediate node (from a to z) can be reused across paths.

As a result, the maximum number of disjoint paths is inherently limited to 3 by the structure of the graph.

Impact of Extra Edges. Although we introduced extra edges such as $a \leftrightarrow b$, $b \leftrightarrow c$, and $x \leftrightarrow y \leftrightarrow z$ to illustrate potential additional connections:

- These edges do not enable more disjoint paths.
- Using such edges would cause the reuse of intermediate nodes, violating the disjointness condition.

Therefore, while useful in fault tolerance or alternate routing scenarios, these additional edges do not increase the number of *node-disjoint* paths.

Confirmation via Max-Flow. After applying the standard *node-splitting transformation* (each node v becomes v_{in} and v_{out}, connected by a capacity-1 edge), we run a max-flow algorithm (e.g., Ford-Fulkerson [20]) from s to t:

- Each node's internal capacity limits the number of paths using it.
- The max-flow returned is 3, confirming the existence of exactly 3 node-disjoint paths.

Conclusion. The maximum number of node-disjoint paths in this graph is 3 due to:

- The structural limitation imposed by the intermediate nodes,
- The disjointness requirement on node usage,
- And the confirmation provided by the max-flow/min-cut theorem.

This example illustrates how the topology of a graph and its node connectivity directly influence the resilience and routing capacity in protocols like DualRC.

This max-flow can be computed via algorithms such as Ford-Fulkerson [20] or Edmonds-Karp [17].

In this paper, the basic Ford-Fulkerson method was adopted due to its simplicity of implementation. From a complexity standpoint, there was no compelling reason to adopt a more optimized algorithm, as the true bottleneck was not the flow computation, but the high connectivity requirement (i.e., large number of edges) of the graphs involved. Hence, upper bounding by E was not a limiting factor in our case.

The time complexity of Edmonds-Karp is given by:

$$O(|V||E|^2)$$

A more informative expression is:

$$O(|E| \cdot \min(f_{\max}, |V||E|))$$

where:

- $|V|$ is the number of vertices,
- $|E|$ is the number of edges,
- $f_{\max}$ is the value of the maximum flow in the network.

While the time complexity for Ford-Fulkerson is simply

$$O(|E| \cdot f_{\max})$$

The exact paths are then selected based on the computed flow, by choosing paths whose edges are exercised with unit flow. If more paths are present than the required number, paths are selected according to a heuristic such as choosing the shortest paths, or simply at random.

5.4.4. Implementation

Algorithm 1 initializes a reliable broadcast from process p_i using disjoint paths to all other nodes. The main logic begins at line 12, where the broadcast value v is delivered locally. The algorithm then iterates over all potential target nodes in line 14.

A check in line 15 ensures that the process does not send to itself. For each target, the available max-flow is computed (line 16), and based on the authentication status of the sender and receiver (lines 17–18), the number of disjoint paths k is determined.

If the flow is insufficient (line 24), a warning is printed and the target is skipped. Otherwise, disjoint paths are retrieved in line 27. Each path is used to send a message (lines 29–33), and if authentication is supported, a signature is added to the message (lines 31–32)

Algorithm 2 defines how a process handles a received routed message. Upon receiving a message (line 16), the node locates itself in the message's route (line 17) and verifies the route's integrity (line 19). If the current node is not the destination, it prepares to forward the message (line 31). Authenticated processes that have already delivered the message sign it (lines 23–28). Trusted signatures are added to the message; otherwise, they're appended to the petition. If the node is the destination (line 33), it checks whether the message has already been delivered (line 34). If not, it evaluates whether the sender or signature is sufficiently trusted to justify delivery (lines 37–42). If no such trust is available, it counts valid petition signatures (lines 44–50) and delivers if a threshold is met. If authentication fails or path info is not cached (line 54), it computes the max-flow and drops the message if the flow is insufficient (lines 55–57). Otherwise, it stores and verifies paths, ensuring at least $f + 1$ are collected before delivering the message (lines 62–64).

5.4.5. Performance Analysis

This paper also evaluates a routed variant of DualRC, in which nodes are assumed to have consistent and accurate knowledge of the current network topology. This stronger assumption enables messages to be forwarded along precomputed efficient paths, rather than relying on the broader flooding mechanisms of the unrouted protocol.

Algorithm 1: InitBroadcast Let $n = |V|$ be the number of processes in the network, and let f be the maximum number of Byzantine nodes. We analyze both time and space complexities.

- **Time Complexity:**
  - The algorithm iterates over all n nodes: $O(n)$.
  - For each node, it calls `maxFlow`, which takes at most $O(n^3)$ using the Edmonds-Karp algorithm, or better with more efficient algorithms.
  - If sufficient flow exists, it calls `getDisjointPaths` to extract up to $2f + 1$ vertex-disjoint paths, which can be done in $O(k \cdot n^2)$, where $k = 2f + 1$.
  - It constructs and sends k messages, where each message travels along a path of length $\leq n$.

  Overall Time Complexity:

  $$O(n \cdot (\texttt{maxFlow} + \texttt{getDisjointPaths} + k)) = O(n \cdot (n^3 + fn^2 + f)) = O(n^4 + fn^3)$$

  assuming $k = O(f)$ and using naive flow/path algorithms.
- **Space Complexity:**
  - Storing all k disjoint paths to each of the n processes requires $O(n \cdot k \cdot n) = O(fn^2)$.
  - Each message includes a route (path), signature, and petition data of size $O(n)$.

  Overall Space Complexity:

  $$O(fn^2)$$

Algorithm 2: HandleRoutedMessage Let l be the length of the route in a received message (at most n), and let p be the number of petition signatures (at most n).

- **Time Complexity:**
  - Verifying membership in the route and validating message structure: $O(n)$.
  - Signature checks: at most $O(n)$ for validating each of the petition signatures.
  - Caching and computing expected paths (only once per sender–receiver pair):
    - max-flow: $O(n^3)$,
    - disjoint path extraction: $O(f \cdot n^2)$.
  - Route matching and path collection: $O(f \cdot n)$.

Worst-Case Time Complexity per message:

$$O(n^3 + fn^2)$$

- **Space Complexity:**
  - Per message, space to store petition, signatures, and the route: $O(n)$.
  - Cached expected paths per source–destination pair: $O(f \cdot n)$.
  - Overall storage across n nodes can be $O(fn^2)$ in the worst case.

Overall Space Complexity per process:

$$O(fn^2)$$

Both algorithms are polynomial in the number of nodes n and linear in the number of Byzantine faults f, with the bottlenecks arising from the use of max-flow computations and the processing of disjoint paths. In practice, precomputing flow-based path information or caching results may mitigate runtime overhead.

In practice, compared to the unrouted counterpart, the routed version of DualRC demonstrates significant improvements in terms of latency. By directing messages through well-defined routes, the protocol avoids unnecessary delays caused by redundant transmissions and indirect paths. As a result, end-to-end message delivery is considerably faster, especially in larger or denser network topologies.

In terms of network consumption, the benefits of routing are more nuanced. While the routed protocol reduces the total number of messages transmitted, the improvement is less pronounced compared to the gains in latency. The reduction in network usage becomes more apparent as network connectivity increases, that is, when nodes have more neighbors and routing choices. In sparse networks, the routed and unrouted versions tend to exhibit similar message overhead, but in well-connected graphs, the routed DualRC can prune many unnecessary paths and avoid message duplication more effectively.

Overall, routing significantly enhances DualRC's responsiveness, making it particularly suitable for environments where timely communication is essential. The improvement in network consumption, while secondary, still supports the use of routing in more complex and dynamic deployment scenarios.

Algorithm 1 InitBroadcast: Reliable broadcast initialization from process p_i using disjoint paths.

Parameters:
  f: max. number of Byzantine processes in the system.
Uses:
  • $G = (V, E)$: network topology.
  • `isAuth`(p): indicates if process p is authenticated.
  • `maxFlow`(s, t): computes max-flow from source s to target t.
  • `getDisjointPaths`(s, t, k): extracts k vertex-disjoint paths from s to t.
  • `sign`(v, bId): signs payload v and broadcast ID with the sender's private key.
  • `send`(m, p): sends message m to process p.
  • `myBroadcastId`: local broadcast identifier.
  • v: value to broadcast.

upon event ⟨InitBroadcast⟩ **do**
  Deliver v locally at p_i
  forall `target` $\in V$ **do**
    if `target` $= p_i$ **then continue**
    $flow \leftarrow$ `maxFlow`$(p_i,$ `target`$)$
    if `isAuth(target)` **then**
      if `isAuth`(p_i) **then**
        $k \leftarrow \min(flow, f + 1)$
      else
        $k \leftarrow \min(flow, 2f + 1)$
    else
      $k \leftarrow 2f + 1$
    if $flow < k$ **then**
      Print warning: delivery to `target` not possible
      continue
    $\mathcal{P} \leftarrow$ `getDisjointPaths`$(p_i,$ `target`$, k)$
    forall `path` $\in \mathcal{P}$ **do**
      Create message $m =$ [`path`, `petition` $= \emptyset, \sigma = \emptyset, v,$ `myBroadcastId`]
      if `isAuth`(p_i) **and** `isAuth`(`target`) **then**
        $\sigma \leftarrow$ `sign`$(v,$ `myBroadcastId`$)$
        Add σ to m
      Send m to first hop in `path`
  Increment `myBroadcastId`

Algorithm 2 HandleRoutedMessage: Processing a routed message at process p_i

Parameters:
f: max. number of Byzantine nodes in the system.
Uses:
- p_i: this process's ID.
- bId: this broadcast's ID.
- `sender`: the direct sender of the current message (from which hop this message was received).
- `broadcaster`: the ID of the node initiating the broadcast.
- `route`: list of process IDs in message route.
- `hasTC`(p), `isAuth`(p), `isTrusted`(p): trust/authentication indicators.
- `isDelivered`(m): whether the current node delivered the message already.
- `v`: the actual value sent by the broadcaster as a message.
- σ: the signature on the value v and bId.
- `sigValid`(σ): whether the signature is valid.
- `petition`: list of signatures from authenticated nodes claiming the message is correct.
- `expectedPaths`: set of possible disjoint paths from broadcaster to the current node.

upon event ⟨HandleRoutedMessage, `Receive` | $m = [route, petition, \sigma, v, bId]$⟩ **do**
Locate p_i in message route `route`
if $p_i \in$ `route` **then**
  if `prevHop` ≠ sender **or** broadcaster ≠ first in route **then**
    Print error and **return**
  if p_i is not the last node in the route **then**
    Create new message from received data: m′
    if `isAuth`(p_i) **and** p_i has already delivered m **then**
      $\sigma = sign(v, bId)$
      if `isTrusted`(p_i) **or** `hasTC`(p_i) **then**
        Add trusted signature to message
      else
        Add signature to `petition`
      end if
    end if
    Forward message to next hop
  else
    if p_i is last node in route (destination) **then**
      if isDelivered(m) **then**
        return
      end if
      if sender = broadcaster **or** isTrusted(sender) **then**
        Deliver message and **return**
      end if
      if `isAuth`(p_i) **then**
        if `isAuth(signer)` **and** (`isTrusted(signer)` **or** `hasTC(signer)` **or** broadcaster = signer) and sigValid(σ) **then**
          Deliver message and **return**
        else
          for each $\sigma \in$ `petition` **do**
            if `sigValid`(σ) **then**
              Store σ
            end if
          end for
          if stored petition signatures > f **then**
            Deliver message and **return**
          end if
        end if
      end if
      if `expectedPaths` not cached **then**
        Compute max-flow from broadcaster to p_i
        if max-flow ≤ $2f$ **then**
          Drop message and **return**
        end if
        Cache extracted flow paths
      end if
      if route ∈ `expectedPaths`(m) **then**
        Collect route
        if more than f routes collected **then**
          Deliver message and **return**
        end if
      end if
    end if
  end if

6 Evaluation

This section describes the setup and overall methodology of the experiments, what data has been collected, as well as the results of each individual measurement.

6.1. Experimental Setup

All experiments were conducted on a cluster of Docker containers emulating a distributed environment. Each container represents a node in the network and is assigned a unique identifier. The containers communicate over a virtual network configured using Docker's bridge mode to simulate inter-process communication in real-world deployments.

Each container runs an instance of our system under test, compiled from the same source code and instrumented to record relevant metrics (e.g., delivery latency, number of messages, and path disjointness). Logs are written to container-local files in /app/log/, which are periodically copied to the host for aggregation and post-analysis. To reduce I/O contention and avoid cross-container interference, each instance logs to a uniquely named file.

The host system runs Ubuntu 22.04 on a MacOS machine with 10-core CPU with 8 performance cores and 2 efficiency cores, 32-core GPU, 400GB/s memory bandwidth and 64 GB of RAM. All experiments were conducted with no other significant workloads on the host.

6.1.1. Methodology

Each experimental run begins with the initialization of all containers and the distribution of configuration parameters (e.g., number of nodes N, fault tolerance level f, amount of authenticated nodes, amount of trusted components and TEEs, and the protocol choice). The experiment proceeds in several rounds, usually 2 to 5. In each round, a designated node initiates a broadcast, and all other nodes participate according to the protocol under evaluation. The network topology is generated at random in each experimental run, as well as the configuration of which nodes are authenticated, trusted, equipped with a trusted component, or faulty.

To evaluate system behavior under fault, we injected Byzantine behavior into a configurable number of nodes. These faulty nodes deviate from the protocol by either omitting messages, modifying payloads, or violating path constraints. The system is expected to maintain correctness as long as the number of faulty nodes does not exceed f. In practice, fault injection was achieved via shutting down docker containers at random or delaying network sends in some nodes for very large amounts of time or indeterminately.

The experiment configuration included setting the amount of authenticated nodes to around 50% of the total nodes, while keeping the trusted nodes and components at around 5-10% of the total nodes. In addition, the network connectivity and the number of fault nodes were varied. Several experiments have been conducted for each configuration, and the results averaged out. However, noticeable outliers were also recorded.

In these experiments, network connectivity ranged from 4 to 15, while the amount of faulty nodes ranged from 1 to 7. The total amount of nodes in the network ranged from 20 to 100. These ranges were enough to yield key observations as presented in the following.

6.1.2. Metrics Collected

We focused on the following key metrics:

1. **Delivery Latency:**
   Delivery latency is defined as the time elapsed between the initiation of a broadcast and the point at which a non-broadcasting node successfully delivers the message. Due to synchronization artifacts, the broadcasting node occasionally exhibited negative latency values; these were excluded from all latency-related calculations. For the remaining nodes, delivery latency was analyzed using both the mean and maximum values across the network to capture typical and worst-case performance, respectively.
2. **Delivery Threshold:**
   The minimum number of disjoint paths required for a node to deliver a broadcast message. This threshold directly reflects the fault tolerance of the system, ensuring that even in the presence of Byzantine failures, delivery decisions are based on independently verified, non-overlapping communication paths. A higher threshold improves resilience but may increase latency and message overhead, making it a key trade-off parameter in the protocol's design.

   From a theoretical standpoint, the delivery threshold reflects the system's resilience to Byzantine faults. A higher threshold improves fault tolerance and agreement guarantees, but it also imposes more stringent requirements on connectivity and message propagation. In contrast, a relaxed threshold maintains correctness under slightly weaker assumptions while unlocking several key advantages. If certain nodes are authenticated or trusted, then the need for a large number of confirmations decreases, as the trustworthiness of received information is inherently higher. This makes it possible to safely reduce the delivery threshold without exposing the system to incorrect deliveries.
3. **Message Overhead:**
   The total number of messages exchanged across the network during protocol execution. This metric is evaluated either by aggregating the total number of messages sent or by computing the average number of messages received per node, depending on the analysis context.
4. **Network Consumption:**
   Network consumption refers to the total volume of message bytes transmitted across the network during protocol execution. This metric captures both the number of messages exchanged and the size of each message, offering a comprehensive view of the protocol's communication cost.

All metrics were logged in real-time and exported to the host for centralized analysis using Python scripts. Inconsistent or missing logs were flagged to detect potential data loss or container failures.

6.1.3. Reproducibility

To ensure repeatability, the entire setup was automated using Python and Docker. Logs were timestamped and stored with a unique experiment ID. All configuration files, scripts, and raw data have been archived for future reference.

6.2. Results

This section presents the collected values for the metrics described above and interprets them.

6.2.1. Impact of authenticated nodes in DualRC

One of the key design parameters in the DualRC consensus protocol is the number of nodes that participate as authenticated replicas. We analyze the protocol's performance under varying levels of authenticated participation, ranging from 0 to 80 authenticated nodes (in increments of 10), out of a total of 80 nodes, under fixed connectivity of 10 and four Byzantine faulty nodes. When the amount of authenticated nodes is zero, we are evaluating the basic DolevU implementation while when all nodes

are authenticated, we are measuring SIGFLOOD's performance.

As mentioned previously, for each parameterised round, several runs were aggregated to smooth out the difference between when the broadcaster itself is authenticated or not. Naturally, when the percentage of authenticated nodes increases, most of the time the broadcaster will be authenticated.

Figure 6.1 presents the observed latency values for each configuration. To quantify the trend, we compute the average latency at each authentication level, summarized in Table 6.1. A stark improvement in latency is observed as the number of authenticated nodes increases. This trend is best understood in the context of protocol behavior: authenticated nodes bypass the overhead of routing via untrusted intermediaries, reducing cryptographic verification and coordination costs.

Table 6.1: Average Latency vs. Number of Authenticated Nodes (total system contains 80 nodes)

Authenticated Nodes	Average Latency (micro seconds)
0	36,614,699
10	32,573,946
20	28,400,886
30	25,354,328
40	17,633,656
50	14,765,268
60	10,503,773
70	5,660,497
80	**878,026**

The data shows that with zero authenticated nodes, the average latency exceeds 36 million microseconds. With full authentication (i.e., all 80 nodes authenticated), latency drops dramatically to under 1 million microseconds—a reduction of more than **97.5%**.

The largest marginal gains occur between 30 and 40 authenticated nodes (a drop of ~30%), and between 60 and 80 nodes, where latency drops by nearly half every 10 nodes added. These inflection points suggest a nonlinear benefit, where reaching a critical mass of authenticated participants drastically reduces coordination complexity and network delays.

This analysis underscores the importance of maximizing the number of authenticated nodes in DualRC deployments. Not only does it improve trust, but it also provides exponential benefits in latency reduction, pushing the system closer to real-time performance under Byzantine settings.

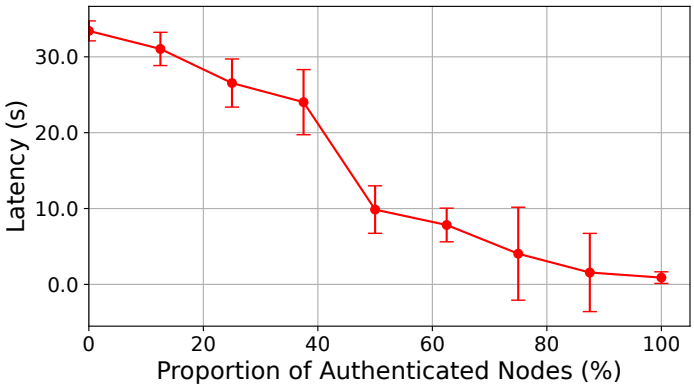


Figure 6.1: Average latency as percent of authenticated nodes in the network increases. c=10, f=4, N=100

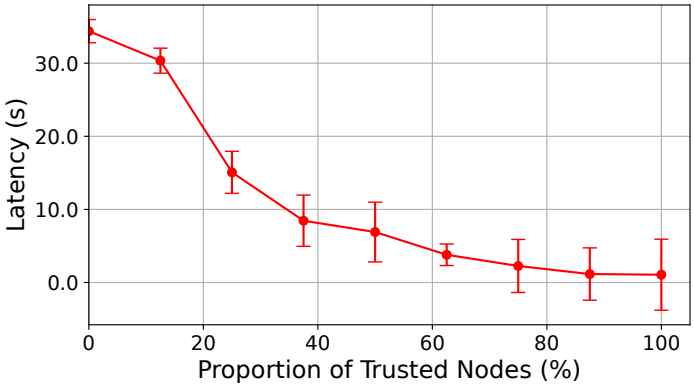


Figure 6.2: Average delivery latency against percentage of trusted nodes out of the total amount of nodes: N=100, c=10, f=4, A=0

6.2.2. Impact of trusted nodes in DualRC

Given latency data L_{T_p} for various trust percentages $T_p \in \{0, 10, \ldots, 80\}$, we define the mean latency:

$$\mu_{T_p} = \frac{1}{n} \sum_{i=1}^{n} L_{T_p}^{(i)}$$

Figure 6.2 plots the mean latency. We observe that increasing the trusted percentage leads to a significant decrease in average latency.

There is a clear exponential decrease in latency as trusted nodes increase. Notably:

- Moving from 0% to 20% trust yields nearly a 50% latency drop.
- From 0% to 70%, latency drops by over **95%**.
- The curve suggests a diminishing return after 70% trust, but still significant.

This suggests that trust-based routing mechanisms substantially reduce communication latency, particularly once a threshold (~30–50%) of trusted nodes is reached.

6.2.3. Impact of TEEs in DualRC

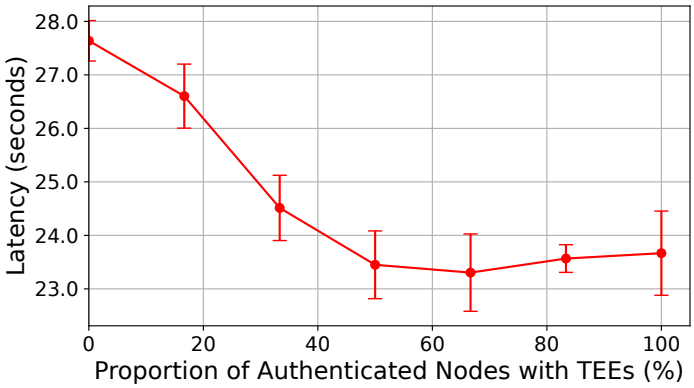


Figure 6.3: Average delivery latency against percentage of TEE equipped nodes out of the total amount of authenticated nodes: N=100, c=10, f=4, A=60

Figure 4.2 depicts how increasing the number of Trusted Execution Environments (TEEs) affects the average message delivery latency in a system operating under the Dual-RC protocol.

The measurements were done on a node with 80 nodes, out of which 60 are always authenticated. In the amount of authenticated nodes, some nodes are chosen at random to contain TEEs (this amount

varies from 10 to 60). The same topology was used to compare the impact of the increasing number of TEEs during one round. Several rounds have been conducted, so results from several topologies were aggregated in the above graph. A particular detail is that the sender was never authenticated; otherwise no significant observations could have been made since authenticated nodes would have delivered based on the sender's signature alone.

The horizontal axis (x-axis) represents the number of TEEs deployed in the network, ranging from 0 to 60. The vertical axis (y-axis) shows the average latency in seconds, which spans from approximately 23 to 28 seconds. A red line with square markers traces the change in average latency as the number of TEEs increases.

Initially, when there are zero TEEs, the average latency is at its peak, around 27.5 seconds. As TEEs are added, the latency drops sharply, reaching a minimum value near 23.2 seconds when there are 20 TEEs. This significant decrease indicates that the presence of trusted nodes greatly enhances the efficiency of message delivery in the Dual-RC system. The minimum point between 20 and 30 TEEs suggests that a sufficient number of TEEs has been reached to optimize trust-based operations.

Beyond 30 TEEs, the latency begins to increase slightly, reaching about 24.8 seconds at 50 TEEs. At 60 TEEs, there is a small decrease, but the latency remains above the minimum observed at 20–30 TEEs. This trend implies diminishing returns when more TEEs are added, and in some cases, performance may even degrade slightly.

Dual-RC leverages TEEs to expedite secure message delivery through two main mechanisms: validation via trusted paths and petition-based delivery. TEEs, being inherently trusted, allow nodes to bypass the petition process when certain conditions are met. This accelerates consensus and reduces the overall latency.

The steep drop in latency as TEEs are introduced highlights their critical role as accelerators of trust. However, when too many TEEs are introduced, network overhead and coordination costs may begin to outweigh the benefits. Increased signature verifications, routing inefficiencies through trusted nodes, and redundant validations can introduce latency rather than reduce it.

The graph reveals a nonlinear relationship between the number of TEEs and delivery latency. A moderate number of strategically placed TEEs significantly enhances performance by reducing the burden of message validation. However, excessive TEEs may degrade performance due to added complexity. Therefore, the deployment of TEEs in a Dual-RC network must be done judiciously to strike a balance between trust and efficiency.

6.2.4. Computing routing tables

In this system, the broadcaster is responsible for computing disjoint paths to every other node in the network. To achieve this, the entire network is represented as a directed graph, where each link has an associated capacity (typically set to one to enforce disjointness). The broadcaster performs the following steps:

1. **Graph Construction**: The broadcaster maintains the full network topology as a capacity graph in memory.
2. **Max-Flow Computation**: For each target node, the broadcaster runs the Ford-Fulkerson algorithm to compute the maximum flow from itself to the target.
3. **Path Extraction**: From the computed residual graph, disjoint paths (corresponding to individual flows) are extracted.
4. **Routing Table Population**: The identified disjoint paths are stored in the routing table, indexed by destination node.

The experimental setup is similar to the other measurements:

- Number of nodes: $20 - 100$
- Average node degree: 10
- Total edges: $E = 1000$

- Graph type: undirected, represented via adjacency list
- Algorithm: Ford-Fulkerson (with DFS traversal)

In terms of memory usage, the following realistic upper-bounded estimates have been made by analyzing the used data structure:

- **Graph Representation** (edges):
  - Each edge (bidirectional): ≈ 32 bytes
  - Total for 1000 edges: 32 KB
- **Routing Metadata**:
  - Per-path storage (average 10 hops): 100 bytes
  - Total for 99 destinations: 9.9 KB
- **Algorithm Overhead (e.g., queues, visited flags)**: ≈ 30 KB

Component	Estimated Memory
Graph representation	32 KB
Routing metadata	10 KB
Computation overhead	30 KB
Total RAM usage	≈ 70–80 **KB**

Table 6.2: Estimated memory usage for routing table computation

The measured latencies are in line with the polynomial complexity for computing max flow for each node in the graph:

Amount of nodes	Total Latency (sequential)
20	43.472 ms
50	98.593 ms
100	185.582 ms

Table 6.3: Estimated latency for routing table computation

6.2.5. Routed DualRC vs Unrouted DualRC: Latency

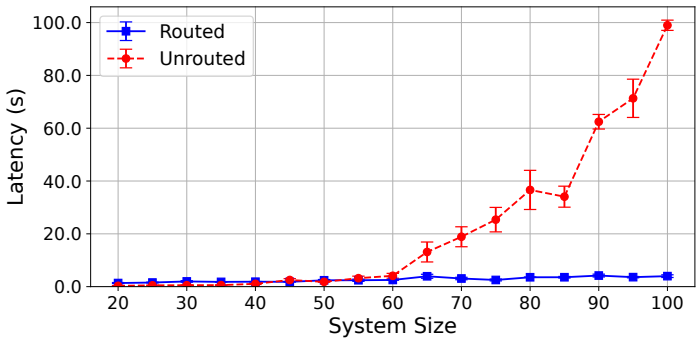


Figure 6.4: Average delivery latency comparison between routed and unrouted versions of DualRC

The routed protocol exhibits a sublinear to moderate growth in latency. The unrouted protocol displays exponential growth, especially beyond 55 nodes. Routed latencies have moderate variance. Unrouted latencies show high to extreme variance, suggesting instability.

20-30 Nodes

- **Routed:** Latency ranges from approximately 0.1M to 2.2M μs.
- **Unrouted:** Latency mostly below 0.6M μs.
- **Conclusion:** Unrouted mode is comparable or even faster at small scale because routing introduces overhead of computing net flow and the exponential nature of the amount of messages being broadcasted isn't yet noticeable.

35–50 Nodes

- **Routed:** Median latency increases modestly to ~2–3M μs.
- **Unrouted:** Latency rises but remains mostly under 5M μs, with rising variance.
- **Observation:** Unrouted remains competitive on average, but signs of instability appear.

55–60 Nodes

- **Routed:** Median latency stabilizes between 3M–4M μs.
- **Unrouted:** Latency spikes to 12M μs; outliers appear frequently.
- **Observation:** Unrouted begins to break down under scale as the runtime starts to be dominated by network latency.

65 Nodes

- **Routed:** Gradual increase, majority below 6M μs.
- **Unrouted:** Dramatic rise to 25M μs and above.
- **Observation:** Unrouted becomes unreliable.

70–100 Nodes

- **Routed:** Latencies remain bounded, typically 4M–6M μs.
- **Unrouted:** Latencies explode to 70M+ μs; extreme variance.
- **Observation:** Unrouted becomes infeasible; routed clearly superior.

Scalability Summary

Node Count	Routed Median (μs)	Unrouted Median (μs)	Comment
20	~1.3M	~0.3M	Unrouted faster (low overhead)
35	1.5M–2.5M	0.7M–1.2M	Unrouted still faster
50	2.5M–3.5M	3M–4M	Routed catching up
60	3.5M–5M	8M–12M	Unrouted worsens
70	4M–6M	20M–40M+	Routed significantly better
80	5M–7M	40M–70M+	Unrouted breaks down

Based on the previous observations, it can be concluded that routing introduces overhead at small scale but amortizes well at large scale. The unrouted version is only suitable for small systems due to poor scalability. Routed protocols provide robust and predictable latency across network sizes.

6.2.6. Routed vs Unrouted: Network consumption

As the number of nodes increases, average network consumption rises in both routed and unrouted networks. This trend is expected; larger networks naturally involve more communication between nodes, which leads to increased consumption. However, the rate and nature of this increase vary significantly depending on whether the network uses routing. In routed networks, the growth in consumption is comparatively gradual and controlled, while in unrouted networks, the consumption grows more steeply, indicating a much less efficient communication mechanism as the system scales.

A key observation is the consistent performance advantage of routed networks across all network sizes. Routed networks consume noticeably less bandwidth than their unrouted counterparts, and the gap

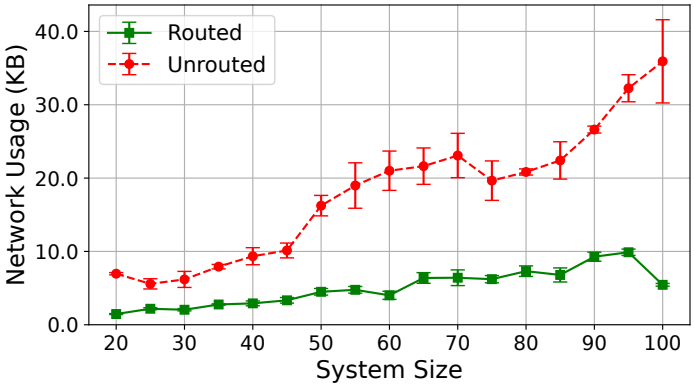


Figure 6.5: Memory consumption in Routed-DualRC versus Unrouted-DualRC

widens as the number of nodes increases. In smaller networks with only 10 or 15 nodes, the difference in consumption between the two approaches is present but relatively small. However, as the network size grows beyond 30 or 40 nodes, the inefficiencies of the unrouted network become more pronounced. The unrouted approach begins to exhibit dramatically higher levels of consumption, suggesting that its communication strategy becomes increasingly wasteful as more nodes are introduced.

The core reason for this disparity lies in the nature of communication strategies. Unrouted networks often rely on broadcasting or gossip-like mechanisms to propagate messages, which leads to message duplication, redundant transmissions, and general overuse of the network. Every message may reach multiple unintended recipients, especially in fault scenarios, in an effort to ensure reliability. In contrast, routed networks employ optimized paths for message delivery. These paths avoid unnecessary transmissions, reduce message duplication, and are often resilient to failures by dynamically bypassing faulty nodes. This allows routed networks to achieve communication objectives with far less overhead, even in the presence of faults.

The presence of two faulty nodes further exacerbates the inefficiencies in unrouted networks. Because unrouted networks have no explicit paths or routing intelligence, they often compensate for potential message losses by increasing redundancy—effectively spamming the network to ensure delivery. This not only increases the consumption under fault conditions but also contributes to network congestion. Routed networks, on the other hand, tend to be more fault-aware. They can isolate or avoid faulty nodes during route computation, maintaining efficient communication with minimal additional cost. This strategic handling of faults allows routed networks to maintain low average consumption even when the network experiences node failures.

Finally, the scalability implications are significant. Routed networks demonstrate much better scalability characteristics. As node count increases, they continue to maintain a reasonable growth in network consumption, making them suitable for large-scale deployments. Unrouted networks, however, scale poorly. The sharp rise in consumption with node count suggests that such networks would become unmanageable and inefficient at scale, especially when reliability in the face of faults is required.

In summary, the analysis of this plot clearly supports the conclusion that routed networks offer superior efficiency, fault tolerance, and scalability compared to unrouted ones. The benefits of routing grow increasingly important as networks grow larger and more complex, and their ability to contain the impact of faults without flooding the network with redundant communication makes them the preferable choice in most practical scenarios.

6.2.7. Impact of authenticated nodes in Routed-DualRC

In the routed variant of the DualRC protocol, each node maintains routing paths computed centrally by the broadcaster. These paths allow messages to be relayed efficiently through authenticated intermediaries, effectively reducing the communication overhead compared to the unrouted or unauthenticated variants.

A key observation in this setup is that the number of authenticated nodes significantly influences the

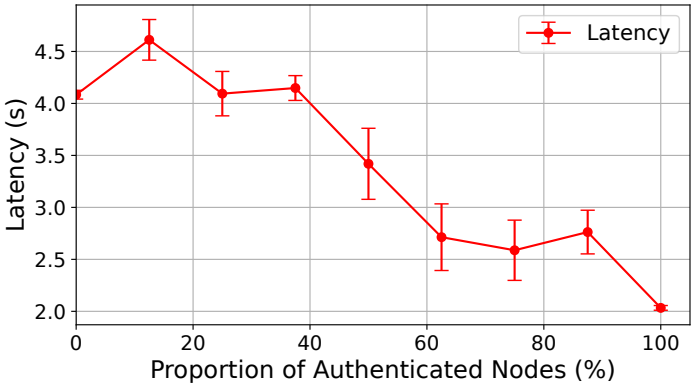


Figure 6.6: Average delivery latency against percentage of authenticated nodes in Routed-DualRC: N=100, c=10, f=4.

protocol's overall latency. With more nodes authenticated, the protocol can leverage a richer set of trusted routing paths, avoiding bottlenecks and enabling parallelism in message propagation.

To study this, we evaluated the end-to-end latency of the protocol for increasing counts of authenticated nodes, ranging from 0 (i.e., no authenticated relays) up to 80. The results, averaged across multiple trials, are plotted in Figure 6.6, and clearly demonstrate a strong inverse relationship between the number of authenticated nodes and the protocol latency.

As seen in the figure, initial configurations with zero authenticated nodes result in very high latency—on the order of 7–9 seconds in many runs. This is expected, as routing is effectively disabled, and all messages must be disseminated without intermediate authenticated relays.

However, as we incrementally increase the number of authenticated nodes, latency begins to drop sharply. For instance, at just 20 authenticated nodes, median latency drops by nearly 40%, and continues to decrease toward sub-second latencies as authentication approaches full coverage. At 80 authenticated nodes, the lowest latency recorded is under 1 second, highlighting the protocol's high efficiency under dense authentication.

These results underline the critical role of authentication in the routed DualRC model: by enabling intermediate routing through trusted nodes, the protocol achieves both lower latency and improved scalability.

To highlight the decrease in latency caused by authenticated nodes between the routed and un-routed strategies, we bring the lines to the same plot 6.7. It can be noticed how in the routed version, authentication makes a smaller impact compared to the unrouted version.

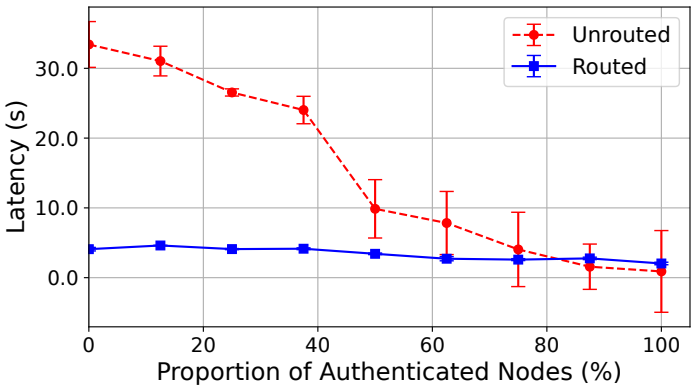


Figure 6.7: Average delivery latency against percentage of authenticated nodes in Routed-DualRC and DualRC: N=100, c=10, f=4.

7

Conclusion

7.1. Conclusion

This paper presented a comprehensive analysis of DualRC, a resilient broadcast protocol designed for Byzantine environments, with a focus on understanding how trust assumptions and network design impact performance. By treating Dolev's classic algorithm as a foundational baseline, the study demonstrated how incremental enhancements in trust—via authentication mechanisms and trusted execution environments—can lead to significant improvements in both latency and communication overhead. Notably, even partial deployment of authenticated nodes within the network produced up to 40% reductions in delivery latency and message volume, illustrating the practical viability of integrating lightweight public-key infrastructure into Byzantine fault-tolerant protocols. Additional performance gains were observed when trusted execution environments (TEEs) were used, though these gains were comparatively modest.

The evaluation also showed that the routed variant of DualRC substantially improves delivery latency, especially in well-connected topologies, while offering modest benefits in overall message cost. These results affirm the value of adapting protocol behavior to network conditions and available trust guarantees.

A central contribution of this work is the formalization and analysis of the routed version of DualRC, which improves on the state of the art for Byzantine-resilient reliable broadcast. Classical reliable broadcast protocols, such as Dolev's algorithm, suffer from exponential message complexity ($O(n^{f+1})$) due to the combinatorial explosion of message forwarding in worst-case scenarios. In contrast, the routed DualRC protocol leverages topology-aware path planning to reduce this burden. By routing messages along precomputed disjoint paths, it achieves significant reductions in communication overhead and time-to-delivery. Specifically, routed DualRC reduces the message complexity to $O(n \cdot f)$ in practical cases, while reducing the time complexity from $O(d \cdot n)$ to $O(d)$, where d is the network diameter. Furthermore, the protocol maintains space complexity of $O(n \cdot f)$ at each node—substantially lower than the exponential state-tracking required in traditional protocols. These improvements make the routed variant particularly attractive for large-scale or latency-sensitive deployments, providing a practical and scalable alternative to the status quo.

The evaluation showed that the routed variant of DualRC substantially improves delivery latency, especially in well-connected topologies, while offering modest benefits in overall message cost. These results affirm the value of adapting protocol behavior to network conditions and available trust guarantees.

- **RQ1: How does DualRC behave in practice?** The empirical evaluation confirms that DualRC offers substantial improvements over Dolev's original protocol in realistic network settings. Although the use of authentication infrastructure introduces cryptographic and computational overhead, the trade-off is favorable: even with partial authentication coverage, the protocol achieves up to 40% reductions in delivery latency and message volume. These results validate that the benefits

of DualRC outweigh its additional trust assumptions in many deployment scenarios.

- **RQ2: Do trusted execution environments bring a significant contribution to DualRC?** The evaluation shows that TEEs yield moderate improvements in performance metrics, primarily by allowing nodes to be recognized as trusted without relying on heavy-weight cryptographic signatures. However, the contribution of TEEs is smaller than that of authentication and appears to be most effective when deployed alongside other trust mechanisms. These findings nuance earlier theoretical claims and suggest a complementary, rather than foundational, role for TEEs in Byzantine-resilient protocols.
- **RQ3: How would a routed version of DualRC perform compared to the non-routed version?** The routed version of DualRC assumes that nodes have access to the current network topology—a nontrivial assumption. Nevertheless, it consistently provides the best latency results in simulations, especially in topologies with higher connectivity. Although the benefits in message complexity are more modest, the routing-based approach proves valuable in scenarios where fast message dissemination is critical and topology information is accessible or can be efficiently maintained.

Several promising directions for future research emerge from this study:

- **Decentralized Routing Mechanisms:** While the routed version of DualRC delivers strong performance benefits, its dependence on full topology knowledge limits its applicability. Future work could explore decentralized or probabilistic routing strategies that approximate optimal paths using only local or partial network views, thereby relaxing the assumptions required for routing-based broadcast.
- **Lightweight Cryptographic Optimizations:** The integration of more efficient cryptographic primitives, such as BLS or aggregate signatures, may reduce authentication overhead and improve scalability. Investigating trade-offs between verification speed, signature size, and fault tolerance will be important for optimizing DualRC in resource-constrained environments.
- **Resilience to Network Dynamics:** DualRC, in its current form, assumes a relatively stable network. An important avenue for future work is extending the protocol to cope with dynamic membership, link failures, or partitioning. Techniques from epidemic protocols and reactive path repair could be incorporated to maintain correctness and liveness in unstable conditions.
- **Security Under Stronger Adversary Models:** This work assumes a static Byzantine adversary. Future studies could evaluate DualRC under stronger models, such as mobile Byzantine faults or adaptive adversaries, to assess its robustness in more hostile scenarios. Modifying the protocol to mitigate equivocation or collusion in these models is an open challenge.
- **Deployment in Real-World Systems:** Finally, a practical direction is to implement DualRC in distributed systems with concrete use cases—e.g., permissioned blockchains, consensus-based databases, or critical infrastructure networks. Empirical studies on deployment costs, fault detection latency, and protocol interoperability would help bridge the gap between theory and practice.

In conclusion, this work underscores the importance of flexible Byzantine communication primitives that can exploit existing trust structures. By quantifying the impact of such structures on protocol efficiency, this study contributes both a practical perspective on deploying DualRC in modern systems and a foundation for further optimizations in authenticated and trusted broadcast protocols.

References

[1] J. Decouchant A. Titu. *Distributed algorithms*. https://github.com/jdecouchant/distributed_algorithms. 2025.

[2] Amos Beimel and Matthew Franklin. "Reliable communication over partially authenticated networks". In: *Theoretical computer science* 220.1 (1999), pp. 185–210.

[3] Silvia Bonomi, Antonella Del Pozzo, and Maria Potop-Butucaru. "Optimal Self-Stabilizing Synchronous Mobile Byzantine-Tolerant Atomic Register". In: *Theoretical Computer Science* 709 (2018), pp. 64–79.

[4] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. 2nd ed. Springer, 2011.

[5] Miguel Castro and Barbara Liskov. "Practical Byzantine Fault Tolerance". In: *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*. Vol. 99. 1999, pp. 173–186.

[6] Guoxing Chen et al. "SgxPectre: Stealing Intel secrets from SGX enclaves via speculative execution". In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 142–157.

[7] Rowdy Chotkan et al. "Reliable Communication in Hybrid Authentication and Trust Models". In: *28th International Conference on Principles of Distributed Systems*. 2025.

[8] Rowdy Chotkan et al. "Reliable Communication in Hybrid Authentication and Trust Models". In: *28th International Conference on Principles of Distributed Systems (OPODIS 2024)*. Ed. by Silvia Bonomi et al. Vol. 324. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025, 25:1–25:26. ISBN: 978-3-95977-360-7. DOI: 10.4230/LIPIcs.OPODIS.2024.25. URL: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.OPODIS.2024.25.

[9] A. et al. Clement. "CheapBFT: Resource-efficient Byzantine Fault Tolerance". In: *EuroSys*. 2009.

[10] Victor Costan and Srinivas Devadas. "Intel SGX Explained". In: *IACR Cryptol. ePrint Arch.* 2016 (2016), p. 86.

[11] Jérémie Decouchant et al. "DAMYSUS: streamlined BFT consensus leveraging trusted components". In: *Proceedings of the Seventeenth European Conference on Computer Systems*. EuroSys '22. Rennes, France: Association for Computing Machinery, 2022, pp. 1–16. ISBN: 9781450391627. DOI: 10.1145/3492321.3519568. URL: https://doi.org/10.1145/3492321.3519568.

[12] Jérémie Decouchant et al. "OneShot: View-Adapting Streamlined BFT Protocols with Trusted Execution Environments". In: *2024 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2024, pp. 1022–1033. DOI: 10.1109/IPDPS57955.2024.00095.

[13] Danny Dolev. "The Byzantine Generals Strike Again". In: *Journal of Algorithms* 3.1 (1982), pp. 14–30. DOI: 10.1016/0196-6774(82)90004-9.

[14] Danny Dolev. "Unanimity in an Unknown and Unreliable Environment". In: *22nd Annual Symposium on Foundations of Computer Science (FOCS)*. Nashville, Tennessee, USA: IEEE Computer Society, 1981, pp. 159–168. DOI: 10.1109/SFCS.1981.53.

[15] John R. Douceur. "The Sybil Attack". In: *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*. Ed. by Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron. Vol. 2429. Lecture Notes in Computer Science. Springer, 2002, pp. 251–260.

[16] Vadim Drabkin, Roy Friedman, and Marc Segal. "Efficient Byzantine Broadcast in Wireless Ad-hoc Networks". In: *Dependable Systems and Networks (DSN 2005), Proceedings of the International Conference on*. IEEE, 2005, pp. 160–169.

[17] Jack Edmonds and Richard M. Karp. "Theoretical improvements in algorithmic efficiency for network flow problems". In: *Journal of the ACM (JACM)* 19.2 (1972), pp. 248–264. DOI: 10.1145/321694.321699.

[18] Patrick T Eugster et al. "Epidemic information dissemination in distributed systems". In: *Computer* 37.5 (2004), pp. 60–67.

[19] Giovanni Farina. "Tractable Reliable Communication in Compromised Networks". English. Distributed, Parallel, and Cluster Computing [cs.DC], NNT: 2020SORUS310. PhD thesis. Sorbonne Université and Università degli Studi La Sapienza (Rome), 2020.

[20] L. R. Ford and D. R. Fulkerson. "Maximal flow through a network". In: *Canadian Journal of Mathematics* 8.3 (1956), pp. 399–404. DOI: 10.4153/CJM-1956-045-5.

[21] Yawei Gu et al. "Secure data deduplication with dynamic ownership management in cloud storage". In: *2017 IEEE Conference on Computer Communications (INFOCOM)*. Discusses rollback and replay protection in secure storage. IEEE. 2017, pp. 1–9.

[22] Zhiwei He, Yike Li, and Kui Ren. "FedSGX: Privacy-preserving Federated Learning with SGX". In: *Proceedings of the 2022 ACM Conference on Computer and Communications Security*. 2022.

[23] Akira Ichimura and Maiko Shigeno. "A new parameter for a broadcast algorithm with locally bounded byzantine faults". In: *Information Processing Letters* 110.12-13 (2010), pp. 514–517.

[24] Chiu-Yuen Koo. "Broadcast in Radio Networks Tolerating Byzantine Adversarial Behavior". In: *Proceedings of the Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC)*. Ed. by Soma Chaudhuri and Shay Kutten. St. John's, Newfoundland, Canada: ACM, 2004, pp. 275–282.

[25] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. "The Byzantine Generals Problem". In: *ACM Transactions on Programming Languages and Systems* 4.3 (1982), pp. 382–401.

[26] Sangho Lee et al. "Inferring fine-grained control flow inside SGX enclaves with branch shadowing". In: *arXiv preprint arXiv:1702.07521* (2017).

[27] Chris Litsas, Aris Pagourtzis, and Dimitris Sakavalas. "A Graph Parameter that Matches the Resilience of the Certified Propagation Algorithm". In: *Ad-hoc, Mobile, and Wireless Network - 12th International Conference, ADHOC-NOW 2013*. Ed. by Jacek Cichon, Maciej Gebala, and Marek Klonowski. Vol. 7960. Lecture Notes in Computer Science. Wrocław, Poland: Springer, 2013, pp. 269–280.

[28] Nancy A. Lynch. *Distributed Algorithms*. San Francisco, CA, USA: Morgan Kaufmann Publishers, 1996. ISBN: 978-1-55860-348-6.

[29] Xueyuan et al. Mo. "PPFL: Privacy-Preserving Federated Learning using Trusted Execution Environments". In: *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2021.

[30] Mikhail Nesterenko and Sébastien Tixeuil. "Discovering Network Topology in the Presence of Byzantine Faults". In: *IEEE Transactions on Parallel and Distributed Systems* 20.12 (2009), pp. 1777–1789.

[31] Aris Pagourtzis, Giorgos Panagiotakos, and Dimitris Sakavalas. "Reliable broadcast with respect to topology knowledge". In: *Distributed Computing* 30.2 (2017), pp. 87–102.

[32] Andrzej Pelc and David Peleg. "Broadcasting with Locally Bounded Byzantine Faults". In: *Information Processing Letters* 93.3 (2005), pp. 109–115.

[33] Daniel Rogers et al. "MYST: Secure communications using trusted hardware". In: *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 2017, pp. 555–569.

[34] Yash et al. Sharma. "DeepLearn: Privacy-preserving Deep Learning using Trusted Execution Environments". In: *arXiv preprint arXiv:2208.08361* (2022).

[35] Lewis Tseng et al. "Reliable broadcast in networks with trusted nodes". In: *GLOBECOM*. IEEE. 2019.

[36] Jo Van Bulck et al. "Foreshadow: Extracting the secrets of Intel SGX with transient out-of-order execution". In: *27th USENIX Security Symposium (USENIX Security 18)*. 2018, pp. 991–1008.

[37] Jo Van Bulck et al. “Plundervolt: Software-based Fault Injection Attacks against Intel SGX”. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2020, pp. 1466–1482.

[38] G. S. et al. Veronese. “MinBFT: Minimal Byzantine Fault Tolerance”. In: *DSN*. 2012.

[39] Kai Zeng, Kannan Govindan, and Prasant Mohapatra. “Non-cryptographic authentication and identification in wireless networks”. In: *IEEE Wireless Communications* 17.5 (2010), pp. 56–62.