Trusted Execution Environments in Byzantine-Tolerant Networks

Sebastien van Tiggele[1] , **Jérémie Decouchant**[1]
[1]TU Delft

Abstract

Achieving consensus in a network is one of the most important performance bottlenecks in distributed computing. This paper takes a look at the existing protocols for achieving Byzantine Reliable Broadcast on asynchronous partially connected networks and how these protocols change to leverage the fact that some nodes have access to Trusted Execution Environments. Modeling some nodes to be completely trusted improves the throughput and reduces latency but the impact changes heavily depending on the placement of these nodes. The second, more realistic approach is having all processes use a local trusted subsystem implemented in a TEE. We show that this reduces the upper bound of faulty nodes from $f < N/3$ to $f < N/2$ and reducing the amount of messages sent by up to 64% (N=30, f=5).

1 Introduction

Distributed systems are used globally to solve all kinds of tasks in lots of different fields. These days, these systems are widely used and have to be resilient and fault-tolerant. This paper looks into the problem of achieving consensus in a network in the presence of byzantine (faulty) nodes, and how the usage of trusted execution environments (TEEs) can improve known solutions.

According to the Byzantine tolerant fault model, some autonomous computing entities may exhibit arbitrary behavior making it very difficult to agree on a correct message.

Two abstractions have been defined in this context. The reliable communication (RC) abstraction helps us in some requirements regarding the reliability of the links between nodes. When a correct process broadcasts a message, every correct node in the network has to deliver that message. the second requirement is that when a correct node delivers a message, this has to be sent from a correct process. The next abstraction is called Byzantine Reliable Broadcast (BRB) and considers the additional case where the broadcaster may be arbitrarily faulty.

Dolev's Reliable Communication algorithm [7] considers a $2f + 1$ connected synchronous network, where f is the maximum number of byzantine nodes. This has been improved by Bonomi et al. [3]. Bracha's double echo authenticated broadcast [4] assumes a fully connected network with N nodes, where at most $N/3 + 1$ nodes are faulty and authenticated network links. Dolev guarantees RC, while Bracha adheres to BRB.

Combining the two abovementioned protocols e.g. providing BRB in a not-fully connected network has been studied recently by Wang and Wattenhofer [16]. This combination is improved by Bonomi et al [3] to account for the huge complexity by combining the Dolev and Bracha protocols.

A Trusted Execution Environment (TEE) is a secure area in the main processor, that can guarantees that the code ran on it will retain its integrity. The code ran on a TEE guarantees authenticity, integrity, confidentiality and remote attestation [12].

The impact of trusted nodes or trusted components in the network has been studied in various models and with differing assumptions. In the instance of a fault model that can tolerate up to f byzantine neigbors [8][11], Tseng et al. has studied the performance increase of placing trusted nodes in such networks and has shown a significant decrease in latency. [13].

In the model where processes are authenticated and use replicated state machines (protocols like PBFT) [5], simple trusted components have been implemented in these systems to decrease the upper bound of faulty services in the system from $f < N/2$ to $f < N/3$ [6] [9] [14] [15]. In this paper, the access to TEEs is studied to improve the performance of the Dolev and Bracha algorithms, which assume authenticated links. Firstly, an approach similar to Tseng et al. is used, where some nodes are assumed to be completely trustworthy en how this affects the protocols of Dolev and Bracha. After that, the improvements MinBFT proposed in the replicated state machine model are studied and translated to improvements on Bracha in the authenticated links model.

The report is structured as follows. Sec. 2 discusses the related work. Sec. 3 explains the system model. Section 4 describes trusted nodes in Dolev and Bracha. Section 5 discusses trusted components in the authenticated links model, and how the improvements from MinBFT over PBFT translate to improvements of Bracha's protocol in this model. Section 6 explains the experimental setup and the results. Section 7 is about the reproducibility and ethical impact of the research. Section 8 concludes the paper and discusses future work.

2 Related work

Gabriel Bracha described the first BRB protocol [4]. This protocol assumes a fully connected and reliable network. This means all processes in the network can connect with every other process, and messages can't get lost. This paper proposes an algorithm that can tolerate up to $f < N/3$ faulty nodes, where N is the number of processes in the network. Each process in this protocol has three phases $send$, $echo$ and $ready$, and a process will progress through the phase once it gets enough messages from other nodes in the network.

In the more common and general case of a not fully connected network, Danny Dolev proposed an algorithm that can tolerate up to f byzantine nodes if the network is sufficiently $(2f + 1)$-connected. Because of the pigeonhole principle, this means that there must be at least $2f + 1$ disjoint paths from a sender to each receiver. With a maximum of f byzantine nodes, This means that at least $f + 1$ disjoint paths do not containt a byzantine process, and a process can deliver a message once it has received it over $f + 1$ disjoint paths. Since this protocol has messages being sent through the whole network, and then has every process solving a disjoint paths problem, the worst-case complexity of this algorithm is very high. Bonomi et al [3] has improved the Dolev algorithm to reduce the number of redundant messages being sent in a practical sense.

A different approach to the fault assumptions is presented by Koo, where a broadcast algorithm is described under the *t*-bounded fault model [8]. Tseng et al. explained this algorithm and named it CPA (Certified Propagation Algorithm) [11]. Tseng et al. extend the CPA algorithm with the usage of trusted nodes, and how they can be most efficiently placed in the graph [13]. This work shows that placing trusted nodes in the system does improve the latency in both sparse and dense networks.

All works described up till now assume a model where processes know that messages will be correctly sent over authenticated links. The Byzantine Fault Tolerance problem is also proposed in the Replicated State Machine model. Castro and Liskov. [5] popularized this with the PBFT algorithm, which made these algorithms practical to use. They make use of the Bracha protocol to tolerate up to $3f + 1$ faulty replicas, but this model differs in that it assumes authenticated processes that make use of encryption. This field of research is important to this paper because the usage of trusted components deployed in TEEs has been explored a lot. Chun et al. propose the use of Attested Append-Only Memory (A2M), which adds a trusted log to improve the PBFT algorithm to tolerate up to $2f + 1$ faulty processes [6]. Further improving this abstraction, Levin et al. proposed TrInc to change the log into a trusted incrementer to reduce overhead [9]. Veronese et al. present the use of a trusted service named USIG which not only guarantees the same $2f + 1$ faulty processes bound but also simplifies the trusted service [14]. Veronese et al. has done a short article explaining the reduction in the number of byzantine nodes using a trusted service and how this is achieved [15].

Madsen et al. looked at transforming byzantine faults into crash-stop faults by placing the entire system into a Trusted Execution Environment, in a 1-round $n = f + 1$ transformation using state machines [10].

3 System model and problem description

The network consists of a set $P = \{p1, p2, ..., p3\}$ of N processes that are interconnected and are uniquely defined by an ID. Up to $f < \lfloor (N/3) \rfloor$ of the N processes are byzantine, which means they can behave arbitrarily. The only information the processes have is N, the IDs in the system, and the fault threshold f. The communication occurs over a undirected graph $G = (V, E)$, where each process p_i is represented by a node in V. Every node also knows its own ID and the ID of its neighbors.

Nodes only communicate with each other directly over these edges. Otherwise, they use multi-hop communications where nodes in between two nodes have to relay the message. These communications can either be synchronous or asynchronous. We assume that up to f processes can behave arbitrarily faulty, e.g. byzantine.

The Byzantine Reliable Broadcast principle guarantees a couple of things during the communication between the nodes.

1. Validity: If a correct process p broadcasts a message m, then some correct process eventually delivers m.
2. No Duplication: No correct process delivers m more than once
3. Integrity: If a correct process delivers message m with correct sender p, then m was broadcast by p.
4. Agreement: If some correct process delivers m, every correct process has to deliver m eventually.

In sections 4 and 5, some contributions will be presented that make use of this System Model. In section 4,

3.1 Background on the Bracha protocol

Bracha's authenticated double echo broadcast describes a BRB protocol for asynchronous networks. This algorithm assumes the network is fully connected, e.g. every process can communicate with every other process in the network. It also assumes authenticated links, and it can tolerate up to f byzantine faults, where $f < N/3$ and N is the number of nodes in the network.

Every process in the network goes through three phases. When a process wants to broadcast the message, it does so by sending the message to every other node in the network. this first message is a SEND message. Upon receiving a SEND message, a process then sends an ECHO message to all nodes in the network. When a process has reached a quorum of $\left\lceil \frac{N+f+1}{2} \right\rceil$ received ECHO messages, the process proceeds to the last phase and sends out a READY message to all other processes. A READY message will also be sent once a process receives f+1 READY messages, indicating that at least one non-faulty process has moved to the READY phase. Once a process receives $2f + 1$ READY messages, it knows the payload is correct and delivers the message. In the end, all the correct processes will have delivered the message.

Algorithm 1: Bracha's protocol at process p_i

```
Result: Write here the result
On event initialization:
    $sentEcho = sentReady = delivered = false$;
    $echos = readys = \emptyset$;
On event broadcastMessage, $m$:
    forall $p \in thesetofprocesses$ do
        send(p, (SEND, m));
    end
On event receiveSENDMessage, p, $m$ and not $sentEcho$:
    $sentEcho = true$;
    forall $q \in setofprocesses$ do
        send(q, ECHO, m)
    end
On event receiveECHOMessage, p, $m$:
    echos.insert($p$);
On event $echos.size > quorumandnotsentReady$:
    $sentReady = True$;
    forall $q \in setofprocesses$ do
        send(q, READY, m)
    end
On event receiveREADYMessage, p, $m$:
    readys.insert($p$);
On event $echos.size > quorumandnotsentReady$:
    $sentReady = True$;
    forall $q \in setofprocesses$ do
        send(q, READY, m)
    end
On event $echos.size > quorumandnotsentReady$ 2:
    $delivered = True$;
    Bracha.deliver();
```

Algorithm 2: Dolev's protocol at process p_i

```
Result: Write here the result
On event initialization:
    delivered = false;
    paths = empty;
On event broadcastMessage, $m$:
    forall $p_x \in neighbours(p_i)$ do
        send($p_x$, $(m, [])$);
        delivered = true;
        deliver;
    end
On event receiveMessage, $(m, path)$, $p_j$:
    paths.insert($path + p_j$);
    forall $p_x \in neighbours(p_i)$ do
        send($p_x$, $(m, path + p_i)$);
    end
On event $p_i$ is connected to the broadcaster through f+1 disjoint paths and delivered = false:
    Dolev.Deliver();
    $delivered = True$;
```

3.2 Dolev's algorithm

Dolev's protocol is not used for BRB, but for a weaker abstraction called RC (reliable communication). This also assumes authenticated links, but it assumes the broadcaster is not faulty. This is why Dolev's algorithm is also called "communication with Honest dealer". The network also doesn't have to be fully connected, but at least $(2f+1)$-connected. In this algorithm, the broadcaster sends the message to all neighbors, which they also forward to all neighbors except the one who sent the message. This makes the message get flooded over the network, and every message also contains the path of nodes it traversed through. A process will use these paths to compute the maximum number of disjoint paths. Once a process has received at least f+1 disjoint paths with a content, it will deliver it. This can be verified through Mengers theorem, which says that if a network is k-connected there exist at least k disjoint paths through all nodes. This means that at least $2f + 1$ disjoint paths exist between all nodes and that since at most f nodes are byzantine a process will at least correctly get a message through $f + 1$ disjoint paths.

Bonomi et al. improved on Dolev's theorem in a practical sense, introducing 5 modifications to reduce redundancy of the number of messages sent. These modifications are as follows:

- **Mod 1.** *If a process p receives a content directly from the source s, then it is directly delivered by p*
- Mod 2. *If a process p has delivered a content, then it can discard all the related pathsets and relay the content only with an empty pathset to all of its neighbors*
- Mod 3. *A process p relays pathsets related to a content only to the neighbors that have not delivered yet*
- Mod 4. *If a process p receives a content with an empty pathset from a neighbor q, then p can discard from relaying and analyzing any further pathset related to the content that contains the label q*
- Mod 5. *A process p stops relaying further pathsets related to a content after it has been delivered and the empty pathset has been forwarded*

3.3 BRB in 2f+1 connected networks

Byzantine Reliable Agreement in $2f+1$-connected networks can be implemented by layering Bracha's algorithm with a second algorithm that takes care of the not-fully connectedness of a network. In this paper, we will focus on the combination of Bracha and Dolev, since this takes the global fault model into account. This combination was recently described by Wang and Wattenhofer [16] by designing a randomized agreement protocol on sufficiently connected networks. These protocols can be combined by replacing the broadcast-to-all operations from Bracha with a Dolev-broadcast, and a link-deliver from Bracha with a Dolev-deliver. Bonomi et al. [2] Improved this algorithm with 12 modifications each either addressing the latency or the number of bits sent over the network. The modifications are either improvements on the Bracha layer or they are cross-layer improvements and they include combining messages of various types, optimiz-

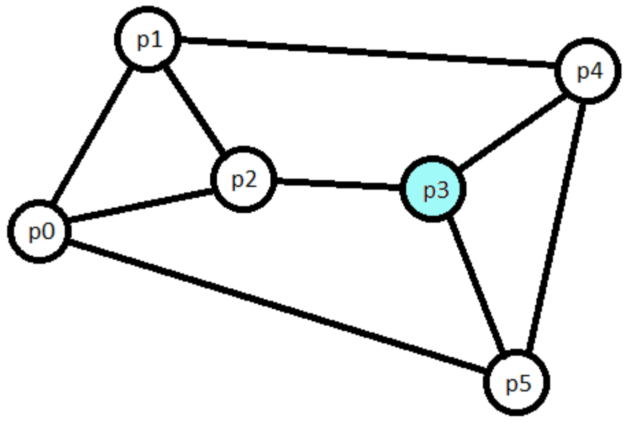


Figure 1: A Network with a connectivity of 3 and one trusted node

ing messages, and ignoring messages when they are redundant.

4 Trusted nodes in the system

We are going to study the use of trusted execution environments in the authenticated links model in two ways.

1. Nodes that are deployed completely in a TEE. This allows the protocols to be modified to take into account that some nodes are known to be trusted, and thus cannot behave in a byzantine way. This forces these nodes to tell the truth, which the other nodes can use to reach Byzantine reliable agreement faster.
2. Deploy a component in a TEE that all nodes have access to. This means nodes can still behave arbitrarily faulty like in the defined system model, but this component will force Byzantine nodes to follow a stricter protocol. In section 5 some possibilities of how this can be realized are explored.

With the first notion, we explore the capabilities of processes that are deployed in a TEE, and thus cannot lie to other nodes. These nodes will make use of signatures to let other nodes know that they are trusted, so every process does in practice know which nodes are to be trusted in a network.

4.1 Trusted nodes in Dolev

We recall the algorithm that is used in Dolev's paper, where nodes propagate a message with a path to all neighbors. Normally, a process only accepts once it has received $f + 1$ disjoint paths, but with the introduction of honest nodes, this can change. Figure 1 shows a network with one trusted node.

If a node receives a message with a content, and the path only consists of nodes that are trusted (including the broadcaster, since the Dolev algorithm considers an honest-dealer abstraction), it can deliver. We can represent this modification as a additional piece of code at line 11 with $if($path only contains trusted nodes $)then\{Dolev.deliver()\}$

This follows through from the first modification of the Dolev improvements of Bonomi et al. [3] where the nodes connected directly to the honest dealer immediately deliver. Since a trusted node will adhere to the protocol, every correct process will accept the fact that this message must be correct, and can immediately deliver the message.

Due to the second modification introduced in the above-mentioned paper, namely a process p relaying an empty path after delivering a content, this is a good way of confirming that a content has been delivered by a protocol.

Since a trusted node will always correctly relay a message, we can conclude that a process p receiving a message containing a content with an empty path from trusted neighbor q can immediately deliver. This is true since the trusted node only relays a message with an empty set after delivering that message, which means neighbor q has successfully authenticated the content.

The more nodes are modified to be trusted, the more efficient the Dolev algorithm naturally becomes. Since nodes will deliver earlier depending on the amount and placement of trusted nodes, we can expect the number of messages throughout the propagation process to be lower than without trusted nodes. Since the trusted nodes only provide confirmation of contents from already delivered processes, and the topology of the network is unknown (e.g. nodes cannot just send all messages specifically to the trusted nodes) the effect of the trusted nodes might be heavily dependent on how far away they are placed from the honest broadcaster and the connectivity of the network.

4.2 Trusted nodes in Bracha

Introducing trusted nodes in the Bracha algorithm is quite straightforward. To leverage the fact that the broadcaster itself might be faulty, the protocol should be modified so nodes only accept messages from a trusted node. Let the broadcaster be p_0, and one trusted node be p_1. Since all nodes in the network know which node is trusted, they will not accept any incoming message (from potentially byzantine processes). The trusted node itself will wait for one message from the broadcaster, send the message correctly to all nodes, deliver the message, and refrain from sending anything else. Every other node waits for this message, and if a potentially faulty broadcaster refuses to send a message to the trusted node no process will deliver. When a potentially faulty broadcaster does send a message to the trusted node, it will make sure all other correct processes deliver this message, and thus BRB-Agreement is not violated. Note that the broadcaster could also be trusted, trivializing the problem even more.

4.3 Trusted nodes in Bracha-Dolev

Since Dolev is just an abstraction layered under Bracha to assure the multi-hop process takes place, the modification to the Dolev algorithm can directly be implemented in the Bracha-Dolev and optimized Bracha-Dolev protocols.

5 Processes with a trusted component

In the real world, deploying even a few processes completely in a TEE is not a feasible solution. Running code on TEEs is more expensive than running it on another part of the system, and assuming some processes in a network are just to be "trusted" isn't always viable. Having nodes in the system that all have access to a trusted subsystem may allow nodes to provide remote attestation, e.g. trust a part of a message even if the host might be faulty.

5.1 Different models

There are a few options regarding a model where we only assume the links are authenticated. We first have to make a few assumptions on how the trusted component is defined. In the system model defined in section X, we can see that every process p from the set of all processes is connected to a few neighbors with authenticated links, all of which make a graph G. Now we do not assume a few nodes are trusted, and up to f nodes can still be byzantine. We also assume the processes themselves cannot sign the messages using encryption. The addition here is that every process is connected to a form of a trusted computing base (TCB), deployed in a Trusted Execution Environment (TEE) (?). Even if a process that hosts this component is faulty, it is guaranteed that everything in the TCB still runs tamperproof.

A1 *The host is connected to the network and the trusted computing base cannot use encryption in communication with the host.*

A2 *The host is connected to the network and the trusted computing base is able to sign messages.*

B1 *The TCB is the one with the connection to the network, but it still cannot sign messages. The TCB will communicate everything to the host*

B2 *The TCB is connected to the network, and it can authenticate messages.*

The different models will be discussed briefly in the following subsections.

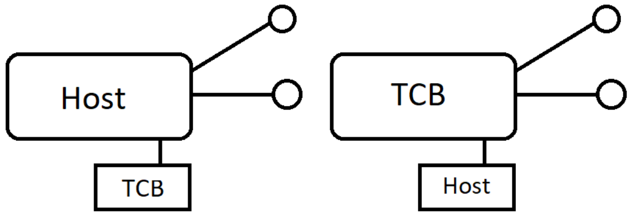
Model A1

This model assumes the host is connected to the network, with a TCB that cannot sign messages. Having access to a trusted subsystem is helpful in that it can control messages sent by other nodes, but it quickly becomes clear how this is not helpful if the TCB cannot sign messages. Assume process p receives a message from a neighbor q. If this message contains some approval of the TCB of q, the authenticity of this message cannot be verified since some byzantine process might have forged this. We can conclude that this model is not more useful than just leaving out the TCB entirely.

Model A2

Let's now assume the TCB can sign messages. This brings the system closer to the authenticated processes model, but there is a key difference. Since the host itself still cannot sign any message, the authenticity of each host is still not clear. In this model, messages that are sent through the network have to be signed by the TCB before they get sent to the neighbors of the host.

This might be very helpful since the TCB can make sure messages that are sent behave more appropriately than a potentially Byzantine host. The TCB cannot create the messages and only serves as an agent between the authenticated link and the host, but it can control the messages sent by the host. We can make some assumptions about this model. The TCB can make sure that only one type of message is sent by the host, which will prevent a potentially malicious host to send different messages to different neighbors. This does not prevent a host to change the message before sending it, since the TCB only serves as a forwarding tool.



(a) The host is connected to the network (b) The TCB is connected to the network

Figure 2: The difference between models A1&A2 and B1&B2

Model B1

Here we assume the TCB is connected to the network, but neither the host nor the TCB can sign the messages. For the same reasons as in model A1, not allowing the TCB to sign messages makes this model not helpful.

Model B2

This model works like B1, where the TCB is connected to the network and can also sign messages. This model is very powerful since the Trusted Computing Base has access to the network. This means that the TCB can choose when and how to broadcast and forward messages, which makes the BRB problem trivial.

Having considered these four models, model A2 will be used in the authenticated links model moving forward in this paper. It provides a model that is quite similar to how TEEs and TCBs have been used in related work, and that can be implemented realistically. Compared to B2, it does not trivialize the problem but offers a way for nodes to provide remote attestation, e.g. that they do not directly trust processes but can trust the TCB's to set limits on what a faulty process can do.

5.2 Trusted component usage in Replicated State Machine Protocols

Related work has tackled the Byzantine Fault Tolerance problem with the use of replicated state machines [5]. This model works with a *client* who sends a request to a set of *servers* who have to reach agreement. These processes are assumed to be authenticated, which differs from the models described earlier in this section. However, the communication protocols used in these systems are directly drawn from Bracha's double echo protocol [4]. One of the more recent papers describing using trusted components is the paper by Veronese et al. [14]. Here, MinBFT is proposed as an improvement of the PBFT algorithm. To illustrate how the PBFT algorithm works, a short illustration is provided in figure 2.

1. The client sends a request to the primary.

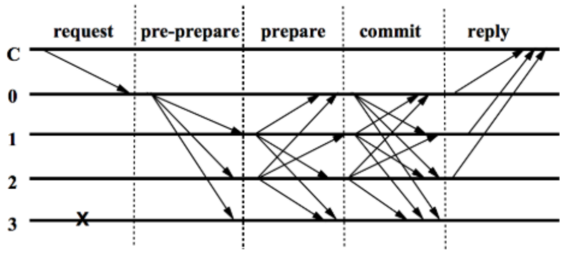


Figure 3: Bracha in the PBFT algorithm

2. The primary service then proceeds to send a PREPREPARE message to all services, with a sequence number and signature.
3. Every service multicasts a PREPARE statement after validating the PREPREPARE
4. Once a service receives enough PREPARE statements, it multicasts a COMMIT message
5. After receiving $2f + 1$ commit messages the service executes the request.

As you can see, PBFT uses Bracha's algorithm to achieve their way of reliable broadcast in the replicated state machine protocol. The pre-prepare, prepare, and commit phases are direct parallels to the send, echo, and ready phases. To recall why Bracha's algorithm works, we refer to the proofs in their paper [4], but in short, it works like this:

- f nodes could be faulty, and with $(N - f)$ replies a node knows it should proceed
- Since the $(N - f)$ replies could contain up to f conflicting replies, a node expects at most $(N - f) - f = n - 2f$ correct replies
- in Bracha (and in PBFT), this must be the majority as nodes do not know whether a faulty broadcaster sent conflicting messages, which is why $N - 2f > f$ and therefore N must be bigger or equal to $3f + 1$

The trusted component in the MinBFT algorithm provides a way for nodes to check the broadcasted message since it has a corresponding counter value that only the trusted component can verify. With a form of remote attestation, all processes can trust that every message corresponding to some counter value and therefore need less confirmation from different processes. More specifically, in the regular Bracha protocol, the amount of nodes that have to agree on a message $N - 2f$ has to be greater than f since the processes have to agree on the same message. Since the trusted component in the MinBFT algorithm prevents equivocation and every node receives the same request corresponding to the same counter, $N - 2f$ has to be greater than 1, thus leveraging a network of size $N \geq 2f + 1$.

Trusted Component in MinBFT

The MinBFT protocol uses USIG, *Unique Sequential Identifier Generator*, to assign a unique identifier to each message and then signs it. It uses a function to create an *unique identifier* for a message m, and it has another function to verify the message using this unique identifier. "These three properties imply that the USIG 1) will never assign the same identifier to two different messages (uniqueness), 2) will never assign an identifier that is lower than a previous one (monotonicity), and 3) will never assign an identifier that is not the successor of the previous one (sequentiality)." [14]

5.3 Trusted Counter in Bracha

Bracha's algorithm will be modified to leverage the fact that all processes including the broadcaster have a trusted computing base (TCB) deployed in a TEE, that will behave in the same way as in the MinBFT algorithm. A short description of the algorithm is as follows.

1. The broadcaster sends a SEND message to all other processes, which apart from the usual contents also contains a unique identifier signed by the trusted component.
2. When a process receives a SEND message it sends an ECHO to all other processes. The trusted component checks whether the unique identifier matches the content of the message, and when it doesn't match it does not send the ECHO.
3. A node will execute the previous step if it receives an ECHO before a SEND since a faulty broadcaster might not send the SEND message to all processes.
4. once a process has received $f + 1$ ECHO messages that are verified by the trusted component, it delivers.

This algorithm makes use of a single echo authenticated broadcast because processes do not need to decide on the correct content of a message as the trusted component forces every broadcast to have a unique identifier, which prevents conflicting contents of a message. This is why a process p only needs $f+1$ messages to know that this message has been forwarded by at least one other correct process and is not some fabrication of the faulty nodes (which would lead to for example only p delivering the message). Since at least one correct process has sent this ECHO, process p knows all other correct processes received this message since a correct process will correctly relay all messages. It can therefore assume all other correct processes are eventually going to deliver this message and deliver this message itself.

Since this protocol only needs one broadcast and then has all processes forward the message it once, the complexity of the number of messages is $N + N^2$ instead of $N + 2N^2$. Needing fewer processes to tolerate f byzantine nodes also reduces the number of processes needed.

remark An additional optimization if the amount of byzantine nodes isn't tight, e.g. it is less than $\lfloor \frac{N}{2} - 1 \rfloor$, is to refrain from sending SEND messages to more processes than needed. Since only one correct process has to receive the SEND message for it to correctly broadcast its ECHO to all others, having the broadcaster send $f + 1$ SEND messages is enough for one correct process to receive it. However, let us note that our new algorithm lets processes send their own ECHO if it receives a correct ECHO before a SEND.

5.4 Trusted Counter in Dolev

The Dolev protocol floods a $2f + 1$-connected network with messages containing their path and then solves the disjoint path problem at every node to compute whether it has received the same message over $f + 1$ disjoint paths. In related work, the usage of public-key encryption has shown that this leads to a protocol tolerating a $f + 1$ connected network [1]. This works because the broadcaster is assumed to be a correct process which then encrypts the message. Every process can then just verify whether the content is unmodified or not, reducing the problem to just having to forward the message to all processes instead of solving a disjoint path problem. The network has to be at least $f + 1$ connected to avoid a faulty process to not forward the encrypted message to the rest of the network. Using a trusted Counter here will unfortunately not leverage such an improvement of the protocol, since it just provides a way for processes to do remote attestation. This is a limitation of the model we use, where we assume authenticated links instead of authenticated processes, and does not allow a Dolev protocol that can assume network connectivity of $f + 1$.

5.5 Trusted Counter in Bracha-Dolev

In the combination of the protocols, the changes to Bracha can be applied to the algorithm. One particular remark is that since the new algorithm leverages $N \geq 2f + 1$ and Dolev's algorithm requires the Newtork to be $2f + 1$-connected, it is not possible to test the upper bound of byzantine nodes without having a fully connected network.

6 Experimental Setup and Results

This section will explain the setup used to get the results and then explains them.

6.1 Trusted nodes in Dolev

The introduction of trusted nodes into the network will be simulated using the OMNeT++ Network simulator v.5.6.2 which runs on C++. The experiments are run on random generalized graphs where the number of nodes, connectivity, and amount of trusted nodes will differ. The graphs are simulated using the Networkx python library, where the trusted and byzantine nodes are selected randomly based on the amount of them present. Every time a random network is created the average performance is reported of 5 runs with each run containing randomly differing IDs of byzantine and trusted nodes. The amount of nodes K ranges from 10 to 50 nodes, the amount of trusted nodes is tested at levels of 10%, 30%, and 50% respectively. Also, the sparsity of the network is tested, first where f is roughly $N/5$ and k is $2f + 1$ and after that where f is 1 and k is 4. The payload is 128b.

Fig. 4 shows having trusted nodes in the network improves the throughput of messages a lot depending on the size of the network. In small networks, the reduction is from 9% up to 23% depending on the level of trust. In large networks, even having a small number of trusted nodes (10%) reduces the number of messages by 58%, and by 92% with a trust level of 50%. Let us note that during evaluation, a lot of runs had big outliers in terms of performance. The placement of these nodes has an enormous impact on how quickly the trusted nodes let others deliver, and in evaluation the dispersion of trusted nodes was random every run. Fig. 5 shows that the

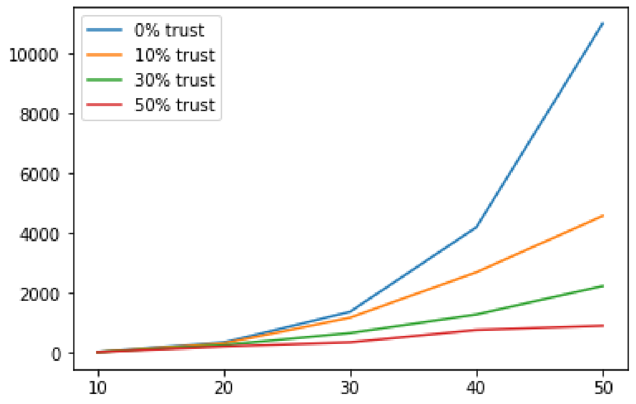


Figure 4: Dolev with trusted nodes, $f = \frac{N}{5}$ and $k = f * 2 + 1$

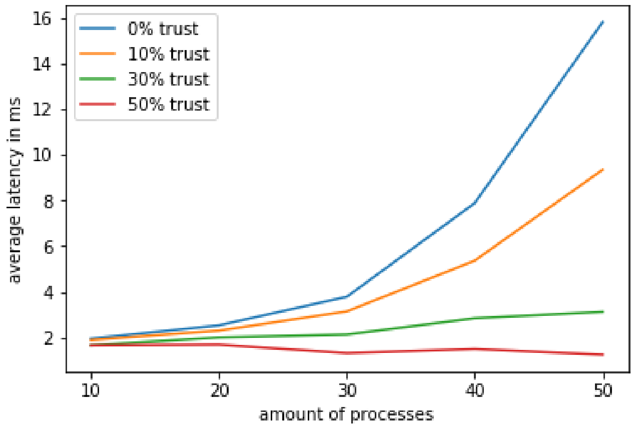


Figure 5: Dolev with trusted nodes, $f = \frac{N}{5}$ and $k = f * 2 + 1$

impact on the latency is similar, with reductions in big networks also ranging from 50% to 92% depending on the level of trust.

6.2 Trusted Components in Bracha and Bracha-Dolev

The performance improvement of the Bracha protocol is exactly as the theory states it is: the broadcaster sends N messages, after which every correct process sends N messages itself instead of 2N representing the double ECHO rounds. Since it is more interesting to see the results in the Bracha-Dolev algorithm, Performance has been tested with the OMNET++ simulator just like the performance evaluation of the trusted nodes. The algorithm used to implement the Trust-Bracha protocol is the Bracha-Dolev protocol with the improvements made to Dolev [3]. We leave the modifications of Bonomi et al. for future research, since the cross-layer implementations combining messages have to be modified themselves. Let us note that the results presented in fig. 6-8 show the number of messages and not the throughput in bits. This is done intentionally since we did not actually implement a trusted component that sends a signature and a unique counter with every message. As this performance evaluation only benchmarks the performance of the correct processes, byzantine nodes are not modeled to perform attacks in the Network Simulator and it is not needed to have redundant checks on the messages that are sent.

Fig. 6 shows that in regular networks, with N ranging from 10 up to 30, the message reduction ranges from 19% in N=10, to 39% in N=20, to finally 64% in networks of N=30. In sparse networks (fig.7) the impact is still high (up to 44% reduction in messages). This indicates that the improvements scale well with bigger networks, and that this effect is higher the more connected the network is. Fig. 8 confirms this, as the amount of messages in a N=30 network is tested with varying degrees of connection. In Networks with k=15 the reduction is 77%. The latency is not shown, as in almost every graph the latency was reduced to < 3 ms no matter the size, connectivity or amount of byzantine nodes.

6.3 Discussion

Both models are shown to have a very positive impact on the message and latency reduction. The placement of trusted nodes and byzantine nodes has been done randomly, which could be a realistic approach to modeling actual byzantine processes who can show up everywhere. This does mean that no worst case scenarios have been tested. Other graphs could haven been selected to model worst-case byzantine process placements, but we chose regular graphs to model realistic scenarios. Regarding the trusted components, there would probably be a cost to implementing and using it in the form of message size. Whether this has a big impact on the performance depends on the payload size and the complexity of the signature, and could have been tested.

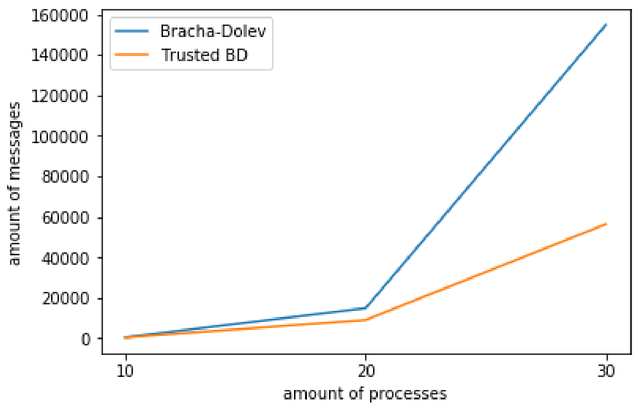


Figure 6: Message nr. impact, $f = \lfloor \frac{N}{5} \rfloor$ and $k = 2 * f + 1$

7 Responsible Research

This research question talks about reducing messages and latency in the presence of trusted execution environments, which in itself does not impose any ethical questions. It merely improves the efficiency of existing algorithms and protocols, and does not introduce real-world issues.

The code that has been produced during the research will be published along with this paper, and therefore anyone can

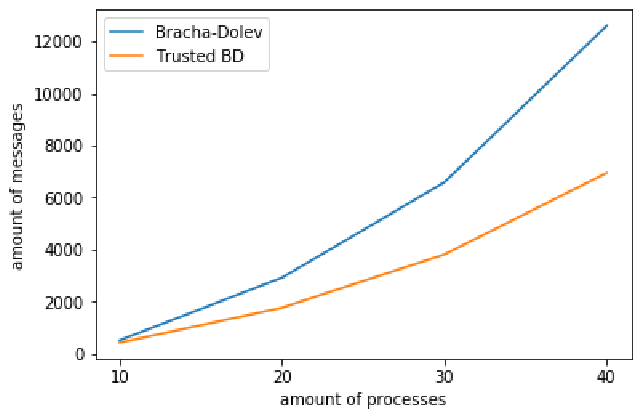


Figure 7: Message nr. impact, $f = 1$ and $k = 3$

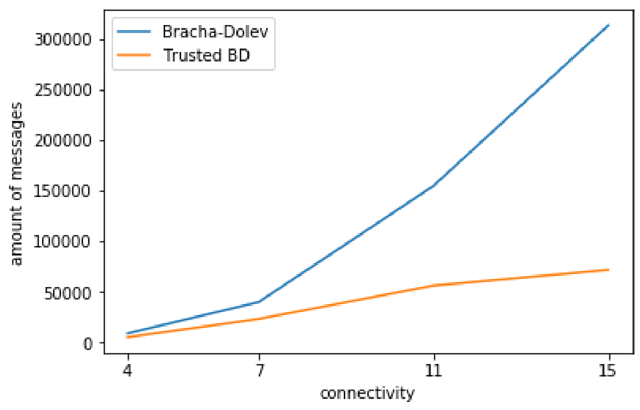


Figure 8: Message nr. impact, $N = 30$ and $f = \frac{k-1}{2}$

reproduce the results. It is written in C++ and works with the OMNET++ library, which itself is documented online. The evaluation can be reproduced almost completely, apart from the fact that we chose random IDs for the byzantine nodes every run.

8 Conclusions

The assumption of having trusted nodes in the network decreases the amount of messages and the latency (up to 92% in large networks), but the impact heavily depends on the placement of these trusted nodes in the system. In future works, the additional assumption of a known topology might increase the viability of these trusted nodes. Letting processes make use of a trusted component in the form of a counter like in MinBFT has a great impact on the theoretical bounds of Bracha. This has been tested in the Dolev-Bracha algorithm and the amount of messages sent is reduced from 42% to 70% in large networks (N=30).

Future research on trusted nodes should look at the most efficient placement of them in the network. The trusted component can be expanded to take into account optimized versions of Dolev to model a tight bound of Byzantine nodes in the Bracha-Dolev protocol. Furthermore, the use of trusted components can also be tested in CPA, which itself can be combined with Bracha to take care of not fully connected networks.

Combining the findings of this paper with having nodes know the network topology, let the processes use encryption or signatures might improve this field of research.

References

[1] Amos Beimel and Matthew Franklin. Reliable communication over partially authenticated networks. *Theoretical computer science*, 220(1):185–210, 1999.

[2] Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. Practical byzantine reliable broadcast on partially connected networks. *arXiv preprint arXiv:2104.03673*, 2021.

[3] Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Multi-hop byzantine reliable broadcast with honest dealer made practical. *Journal of the Brazilian Computer Society*, 25(1):1–23, 2019.

[4] Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, November 1987.

[5] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, OSDI '99, page 173–186, USA, 1999. USENIX Association.

[6] Byung Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: Making adversaries stick to their word. 2007.

[7] D. Dolev. Unanimity in an unknown and unreliable environment. In *22nd Annual Symposium on Foundations of Computer Science (sfcs 1981)*, pages 159–168, 1981.

[8] 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*, pages 275–282, 2004.

[9] Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. 2009.

[10] Mads Frederik Madsen, Mikkel Gaub, Malthe Ettrup Kirkbro, and Soren Debois. Transforming byzantine faults using a trusted execution environment. 2019.

[11] Andrzej Pelc and David Peleg. Broadcasting with locally bounded byzantine faults. *Information Processing Letters*, 93(3):109–115, 2005.

[12] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[13] Lewis Tseng, Yingjian Wu, Haochen Pan, Moayad Aloqaily, and Azzedine Boukerche. Reliable broadcast in networks with trusted nodes. 2019.

[14] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Verissimo. Efficient byzantine fault-tolerance. *IEEE Transactions on Computers*, 62(1):16–30, 2011.

[15] Giuliana Santos Veronese, Miguel Correia, and Lau Cheuk Lung. From crash to byzantine consensus with 2f+ 1 processes.

[16] Ye Wang and Roger Wattenhofer. Asynchronous byzantine agreement in incomplete networks. In *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, AFT '20, page 178–188, New York, NY, USA, 2020. Association for Computing Machinery.