BLS Cosigning via a Gossip Protocol

Lukas Gelbmann

School of Computer and Communication Sciences

Decentralized and Distributed Systems lab

Semester Project

June 2019

Responsible
Prof. Bryan Ford
EPFL / DEDIS

Supervisor
Cristina Basescu
EPFL / DEDIS

Supervisor
Gaylor Bosson
EPFL / DEDIS

Abstract

Decentralized cosigning protocols, which collect digital signatures of a message from many peers, play a central role in several applications. In this semester project, a new gossip-based cosigning protocol using Boneh-Lynn-Shacham (BLS) signatures is designed, built and tested in simulations. The main goals are for the new protocol to be highly fault tolerant, relatively fast and not to overwhelm any node. The new gossip protocol is compared with BLS CoSi, an existing cosigning protocol, in experiments, with the new protocol performing better in the most important metrics, but not in terms of efficiency. The new protocol's performance is then analyzed, showing possible avenues for future improvement.

Contents

1 Introduction

There are many uses for a decentralized protocol which collects signatures of a given message from a large group of independent peers. Such a *cosigning protocol* can be used, for example, to ensure that a statement from a network authority (such as a certificate authority) has been seen and validated by a diverse group of witnesses [1]. This can protect clients from harm when the authority's secret keys are compromised and malicious actors attempt to abuse them. The same setup can also enforce software transparency in the context of closed-source software with regular updates [2]. Here, software makers could use a cosigning protocol to protect their users from updates that introduce secret backdoors—even if, for example, the inclusion of such a backdoor is ordered by a law enforcement agency [3].

In the context of blockchains and cryptocurrencies, a cosigning protocol is a central part of the *ByzCoin* protocol [4] developed at the DEDIS lab at EPFL. In ByzCoin, a proposed block needs to be signed by a threshold number of nodes to be accepted as part of the blockchain. One of ByzCoin's main improvements over other cryptocurrencies such as Bitcoin is its low transaction confirmation latency—in the range of seconds, compared to up to an hour for Bitcoin. To achieve a confirmation latency as low as possible, the cosigning protocol underlying ByzCoin should be fast, since a transaction is only confirmed when a block that contains it is appended to the blockchain. The cosigning protocol must tolerate a certain number of offline or malicious nodes. For the protocol to scale well, no single node should have to process an overwhelming amount of messages, and the size of messages should be kept low.

The aim of this semester project is to design and evaluate a gossip protocol for building collective signatures. An existing cosigning protocol built at the DEDIS lab, *BLS CoSi* [5], is used as a starting point. BLS CoSi uses the Boneh-Lynn-Shacham signature scheme [6], which supports *multi-signatures* [7]. Multi-signatures are short signatures that can be used to verify that a large number of parties signed a common message. To reduce the amount of data transferred and stored, this project's new cosigning protocol uses a multi-signature scheme based on BLS signatures, shown recently by Boneh, Drijvers and Neven [8]. The existing BLS CoSi implementation does not always handle faults in a small number of nodes well, and so there is room for improvement. A gossip protocol is well-suited for the task of improving fault tolerance. This report presents the new cosigning protocol and then analyzes it based on an experimental evaluation.

The main goal of the new protocol is to improve fault tolerance compared to the existing BLS CoSi protocol while still being fast and not overwhelming any participating node. A secondary aim is efficiency: the number of messages exchanged and the amount of data transferred between nodes should remain low. The new protocol is designed mainly for the use case of

ByzCoin, although other applications are certainly possible.

Section 2 of this report gives a little background on gossip protocols and the cryptographic tools used in the new cosigning protocol, and outlines the how BLS CoSi works. Section 3 presents the new gossip protocol's design and implementation. In Section 4, the methodology and setup used in the experiments is shown. Their results are given in Section 5. Section 6 discusses the findings and limitations, and gives an outlook on future work, after which the report ends with a short conclusion in Section 7.

2 Background

The main conceptual tools used in this project are gossip protocols and the BLS cryptographic scheme along with its extensions. The existing BLS CoSi protocol, which we aim to replace, uses BLS cryptography, but not gossip protocols.

2.1 Gossip protocols

Gossip protocols can be used for a range of tasks where information needs to be shared between different nodes. Among other things, they are used for information dissemination and computing an aggregate between many nodes [9]. They provide many useful properties at a reasonable efficiency. One important such property is logarithmic *mixing time*: if the number of nodes is n, a correctly designed gossip protocol propagates a new piece of information to all peers in $O(log(n))$ time, assuming nodes are not overwhelmed by the rate of information transfer [9]. At the same time, the number of messages that a single node sends and receives is constant in the number of peers. Another advantage is gossip protocols' simplicity: all participating nodes can often run the same code.

A typical basic gossip protocol repeatedly runs three steps in a loop: *peer selection*, *data exchange* and *data processing* [10]. In the first step, peer selection, a node chooses one or more peers to exchange information with, typically at random. Choosing a new set of random peers each time is a simple and effective way of achieving fault tolerance.

In the second step, data exchange, data is sent to or received from the selected peers. In a *push model*, information is sent to the peer and no reply is expected. In a *pull model*, a request is sent, to which the peer responds with known information. Is is also possible to combine the two models in one protocol.

In the third (optional) step, the new information is processed or passed to a further-up application layer. How processing is done depends on the domain and application of the protocol.

A gossip protocol that is concerned with information dissemination can be analyzed by looking at the different states in which a node can find itself

in regards to an update [11]. An update is here a piece of information that is being spread. There are usually up to three states called *susceptible (S)*, meaning the node does not know about the update, *infected (I)*, meaning the node is actively spreading the update, and *removed (R)*, meaning the node knows about the update, but is not actively spreading it (anymore). In the SI model, a node starts in state S and switches to state I when it learns about an update, after which it can no longer change its state with regards to this update. In the SIR model, a change to state R is possible after the node takes on state I.

Gossip protocols are typically long-running protocols, but in this project, we employ a very short-running protocol. A protocol instance is started when a node makes a request for a message to be signed. When the requesting node has received enough signatures, the protocol has fulfilled its task. In our setup, every node knows and can exchange messages with every other node. The set of nodes does not change during the course of a protocol instance. These properties as well as our specific requirements affect our protocol design.

Our protocol sends push messages in a very short interval to allow signatures to be spread very quickly. We use the SI model because we value the speed that is gained from continuing to spread signatures pro-actively throughout the protocol's lifetime. We use only push messages during the main part of the protocol. Pull messages are not necessary to guarantee complete dissemination in the SI model. Although they have some advantages over push messages, they also have the disadvantage that its takes a full roundtrip to spread a piece of information after sending a pull message, compared to half a roundtrip for a push message.

2.2 Multi-signatures

A *digital signature* is a tool of public-key cryptography, computed from a message and a private key, that be verified using the corresponding public key and the message. *Cosigning* or collective signing, as used in this report, means that multiple nodes digitally sign the same message or statement. A *multi-signature* is a compact value that proves that multiple nodes have signed the same message [8]. It is computed by aggregating multiple signatures from different nodes. The advantage of a multi-signature over a simple list of signatures from different nodes is that a multi-signature is smaller. This is especially important in protocols where many cosignatures are exchanged between nodes, as is the case in ByzCoin [4].

The multi-signature scheme used in the new protocol is based on work by Boneh, Drijvers and Neven [8] and implemented under the name *BDN* in the cryptographic library *Kyber* [12]. BDN multi-signatures are based on the Boneh-Lynn-Shacham (*BLS*) signature scheme [6] and use commutative groups of prime order both for signatures and for public keys. In the version

of the BDN scheme that is applied in the new cosigning protocol, the public keys of all nodes have to be known in advance, even if not all of them end up signing the message. The public keys are used collectively to create a unique integer coefficient for each signatory: α_1 for the first node, α_2 for the second, and so on. In a set of three nodes signing the same message, where S_i is the signature of node i, the multi-signature S is then computed as $S = \alpha_1 * S_1 + \alpha_2 * S_2 + \alpha_3 * S_3$ (using additive group notation). A multi-signature from just part of the nodes, such as $\alpha_1 * S_1 + \alpha_3 * S_3$, is equally possible. Thanks to commutativity and associativity, partial multi-signatures from two or more disjoint sets of peers can be added to form a new valid multi-signature. A more in-depth explanation of the BDN scheme is found in the paper presenting the scheme [8] or an article by Snigirev [13].

To verify a multi-signature, the verifier must know the public keys of all the nodes that are part of the group, including those who did not end up signing. This is not a problem in the context of the new cosigning protocol, as all nodes know the public keys of all peers. The verifier must also know which nodes contributed to a multi-signature. This is solved by storing a bitmap together with each multi-signature that indicates the contributing nodes.

2.3 The pre-existing BLS CoSi protocol

In BLS CoSi [5], a protocol instance is initiated when a cosignature is requested on some node by a client from a higher application layer. Authenticated communication with other nodes is handled by the *Cothority* framework [14] which BLS CoSi is a part of, as is the distribution of public keys and membership of nodes. It can therefore be assumed that all nodes know all public keys from the start and that the set of peers is known and constant for the duration of the protocol.

Generally, there is a certain threshold of nodes that must sign a message it before it is considered a valid cosignature. In this report, we assume *Byzantine consensus* is needed, which means we assume that up to f out of $n = 3f + 1$ nodes may be faulty (this includes offline or malicious nodes) and we require $2f + 1$ nodes, i.e. just above two thirds of nodes, to sign a message [4].

BLS CoSi works by arranging all participating nodes in a tree of depth 3. The root of this tree is the node where the protocol instance is initiated, which is simultaneously the node where the cosignature needs to be returned to an application layer in the end. An initial signature request containing the message to be signed is sent by the root to all its child nodes in the tree. These nodes then send a signature request to their children, which are leaf nodes. The leaf nodes reply by sending their signature to their parent nodes, who then aggregate the received signatures. They add their own signature and return the aggregated signature to the root, who then combines the

aggregates and its own signature to form the final cosignature. Of course, each node can also choose not to sign the message. In that case, it sends a refusal message instead of a signature.

If a child of the root is faulty and does not return a proper aggregate, the root potentially misses out on a lot of signatures. To mitigate this problem, the root rearranges the tree and tries again if, for example, it has not heard back from a child after a few seconds. This solution is not ideal, however, as it is possible that multiple attempts are needed, which can cost a lot of time.

3 The new protocol's design and implementation

Like the old BLS CoSi implementation, the new cosigning protocol is written in Go and fits into the Cothority project [14]. It makes use of different tools developed by the DEDIS lab, including Kyber [12].

The interface for calling the new protocol is the same as for BLS CoSi. As before, the protocol starts with a single node, the *root*, needing a cosignature. Any node can start a protocol instance and will then be the root for this instance. The root has a special role in the protocol, as the root needs to return a cosignature to an application layer in the end.

3.1 Rumor messages

During the main part of the protocol, only one type of message is exchanged between nodes, called a *rumor message*. It contains the statement to be signed as well as all the signatures the sender has seen so far. Of course, at the moment the protocol starts, only the root knows about the protocol. The other nodes start participating after they receive a rumor message for the first time.

The behavior of each node during this initial phase of the protocol is simple. After receiving a rumor message for the first time, the node adds its own signature to its collection of known signatures (assuming that the node chooses to sign the statement). A node also adds all signatures to its collection when they are received for the first time. Each node periodically sends a rumor message, containing the full collection of known signatures, to a set of r different randomly selected peers (r is smaller than the number of nodes n). This sending of rumor messages happens at a regular interval t, typically a fraction of a second in a context like ByzCoin. The parameters r and t have the same fixed value for all nodes.

3.2 Shutdown mode

As soon as the root node has received a threshold number of signatures through this process of gossip, it sends an aggregated cosignature to the

application that requested it. At this point, the protocol has fulfilled its task, and all that is left to be done is to wind down the protocol. This needs to be done with some care; after all, the other nodes do not know when the protocol is done. To signal that the protocol is finished, the root sends a *shutdown message* to s different randomly selected peers and enters *shutdown mode*, where s is another parameter with the same fixed value for all nodes. In shutdown mode, incoming rumors are answered with a shutdown message and incoming shutdown messages are ignored. The stored collection of signatures is no longer needed in shutdown mode. When a node receives a valid shutdown message, it sends the message on to s different randomly selected peers and enters shutdown mode. Invalid messages are simply ignored.

After the root has built a cosignature, all nodes should enter shutdown mode relatively quickly. However, nodes generally do not know the state of other nodes, and so they remain in shutdown mode, listening for rumor messages and responding with a shutdown message, for a while longer. This is to ensure that all nodes enter shutdown mode eventually and can cease sending out rumor messages. After a fixed amount of time, long enough to not interfere with the course of the protocol, the node stops running the protocol.

To prevent an attack where a malicious node causes other nodes to enter shutdown mode prematurely, shutdown messages have to be signed by the root. (The root, as the node where the signature request originates, can be assumed to be correct.) In order to prevent a replay attack where a malicious node resends a shutdown message from a previous protocol instance signed by the same root node, the signed shutdown message needs to be unique to a protocol instance. For this reason, we chose to use the aggregated cosignature in the shutdown message: the shutdown message contains the original statement to be signed, a valid cosignature of this statement, and a signature of the cosignature by the root.

3.3 Signature aggregation

The simplest way of handling signature aggregation is to let the root to it only when it has received the threshold number of signatures. Indeed, this was one of the variants we implemented. However, less data needs to be transferred if signatures are aggregated sooner in the protocol. Aggregating signatures sooner is tricky because signatures can only be aggregated if they come from disjoint sets of signatories. For example, a multi-signature from the set of signatories $\{A, B\}$ can be aggregated with the signature of node C. However, if we try to aggregate two multi-signatures from signatories $\{A, B\}$ and $\{B, C\}$, we run into a problem because the sets of nodes are not disjoint. Without additional information, we cannot validly aggregate these multi-signatures.

To avoid this problem while still allowing some level of early aggregation, signatures can be aggregated along a conceptual binary tree. Each leaf in this tree stands for a signature from a single signatory. All other nodes in the binary tree stand for the aggregation of their children. For example, with four potential signatories A, B, C and D, there are four leaves, one for each of them. One level up, we might have one parent for the aggregation $\{A, B\}$ and one for $\{C, D\}$. At the top of the binary tree is the aggregation of these two: $\{A, B, C, D\}$. The second protocol variant we implemented aggregates signatures if and only if the aggregation exists in this binary tree. This way, there are never any two multi-signatures from disjoint sets of signatories.

3.4 Further details

In gossip protocol terminology, the rumor messages act as push messages most of the time. However, when they reach a node in shutdown mode, they act more like a pull message, pulling the shutdown message.

The protocol is designed to spread information quickly, with efficiency only being a secondary concern. This is why entering shutdown mode is only possible after it has been proven that the root has received a cosignature.

The parameters r, s and t are sent along with rumor messages in our current implementation to help us test the effects of these parameters in experiments. In a real-world implementation, they would be fixed in advance.

4 Methodology for experimentally evaluating the protocol

To evaluate the new protocol, it was tested in simulations and compared to BLS CoSi under varying conditions. These conditions include the number of nodes, the number of failing nodes, and an artificial message delay, which is used to simulate real-world network delays. The delay for any given message was randomly picked from a uniform distribution between a fixed minimum and maximum delay. A failing node, in our simulations, drops all incoming messages and sends no messages, effectively causing the node to be offline. If a node is not failing, it is *active* and functions correctly. The root was active in all simulations. All experiments were run on single machine with eight 3.10 GHz Intel Core i7 processors under Ubuntu 18.10.

Three metrics were used to evaluate the protocols. The most important metric is the time it takes until the root returns the cosignature, called the *protocol duration*. The other two metrics we measured are the number of messages sent per active node and the amount of data sent per active node. Both rumor messages and shutdown messages are included in these metrics.

Altogether, ten sets of simulations were run. Each set of simulations varied one or two parameters to isolate their effect and analyze their influence.

The other parameters were set to default values. The parameters come in two groups: environment parameters that are not within the nodes' control, and protocol parameters that affect how the nodes behave.

4.1 Default parameter values in simulations

The default values for environment parameters are as follows.

- Number of nodes: $n = 25$
- Number of failing nodes: 0
- Minimum message delay: 0.095 seconds
- Maximum message delay: 0.105 seconds

It follows that the mean message delay is 0.1 seconds by default. The small range of possible delays (0.01 seconds between the minimum and maximum delay) is specified to prevent unnatural effects where, for example, messages from different nodes always arrive at the same time.

The threshold of signatories for a valid cosignature is always set to $n - \lfloor \frac{n-1}{3} \rfloor$ because this is the value that is used in ByzCoin. The default values for the other protocol parameters are as follows.

- Number of recipients for each rumor message: $r = 3$
- Interval between sending out rumor messages: $t = 0.07$ seconds
- Number of initial shutdown-message recipients: $s = 2$
- Early aggregation along a binary tree: on

These default values for protocol parameters were chosen as a compromise between protocol speed and efficiency.

5 Results

Whenever a parameter is not mentioned in the description of an experiment, it takes on the default value given in Subsection 4.1.

5.1 Comparison of BLS CoSi and the new gossip protocol

In an initial experiment, the new gossip protocol was compared to BLS CoSi with varying numbers of nodes and failing nodes. We found that BLS CoSi sometimes fails when the number of active nodes was close to the threshold of signatories (but still high enough that the threshold can be reached). This happens when no valid signature is produced after four tries. In the

most extreme case in our experiments, with $n = 36$ and 11 failing nodes, BLS CoSi failed 52% of the time.

Even when it did not fail, BLS CoSi often took many times longer to produce a cosignature than the gossip protocol, depending on the number of retries it needed. The duration of the gossip protocol was fairly stable, always measuring in below 1.3 seconds in this set of experiments. With very few failing nodes, however, BLS CoSi was consistently faster than the gossip protocol. The timings are shown for $n = 36$ in Figure 1.

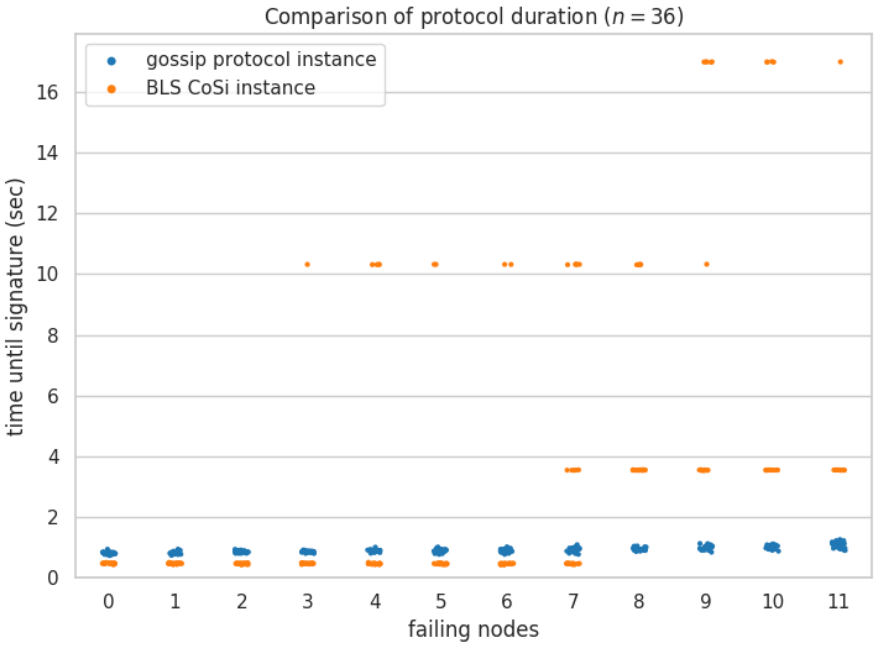


Figure 1: Timings until a cosignature was returned using BLS CoSi versus the new gossip protocol. Each dot stands for a single protocol instance.

In the amount of data sent per active node, the gossip protocol performs slightly worse than BLS CoSi for $n = 36$. Still, the amount of data sent by the gossip protocol is usually less than 170% of that sent by BLS CoSi, as Figure 2 shows. The message count is typically around 300% higher for the gossip protocol than for BLS CoSi. See Appendix B for more detailed results.

5.2 The new gossip protocol under varying conditions

A range of experiments were conducted to analyze the effects of environment parameters and protocol parameters. In these experiments, the protocol

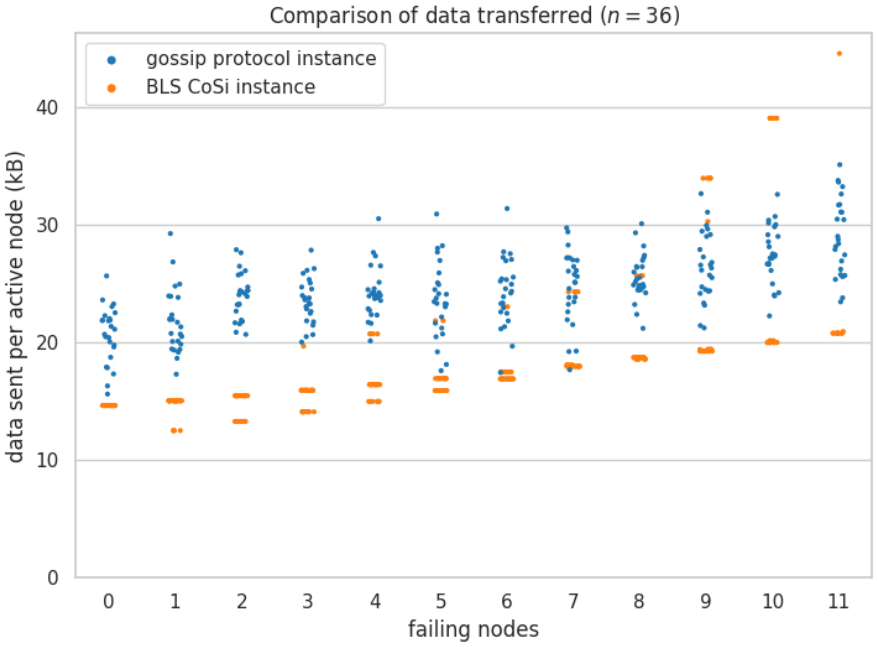


Figure 2: Amounts of data transferred per active node using BLS CoSi versus the new gossip protocol

was run multiple times under each set of parameter values, typically around 20 times. We report and plot the mean of the measurements for these runs.

The effect of the message delay on the protocol duration is shown in Figure 3. In this experiment, the protocol was run with different mean message delays ranging from 0.0 to 0.4 seconds. The minimum and maximum delay are always 0.01 seconds apart (except for the special case where the minimum and maximum delay are 0.0 seconds). The plot shows that the mean protocol duration is highly dependent on the mean message delay and that there is an approximately linear relationship between message delay and protocol duration.

The experiment was run for different numbers of failing nodes. Figure 3 shows that the mean protocol duration is consistently slightly higher for higher numbers of failing nodes. A similar roughly linear relationship was also observed for both the number of messages and the amount of data sent per active node. This is expected, because the number of messages sent per unit of time per node is constant while the node is sending rumor messages. Therefore, the longer a protocol takes to complete, the more messages each node is expected to send. A plot showing the number of messages sent under these varying conditions is given in Figure 4.

The number of nodes also affects the protocol duration, as shown in

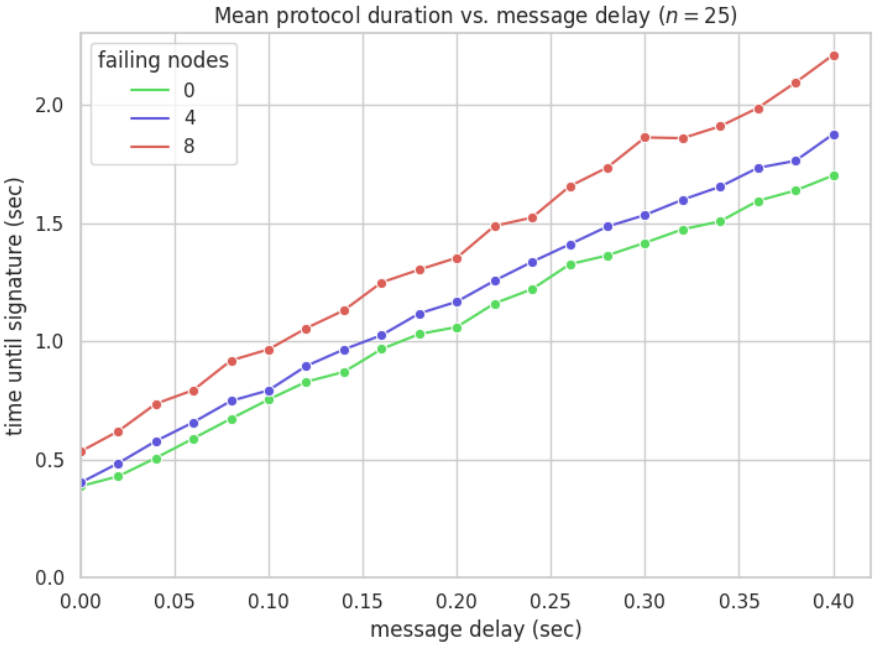


Figure 3: Timings until a cosignature was returned under varying mean message delays and failing node counts

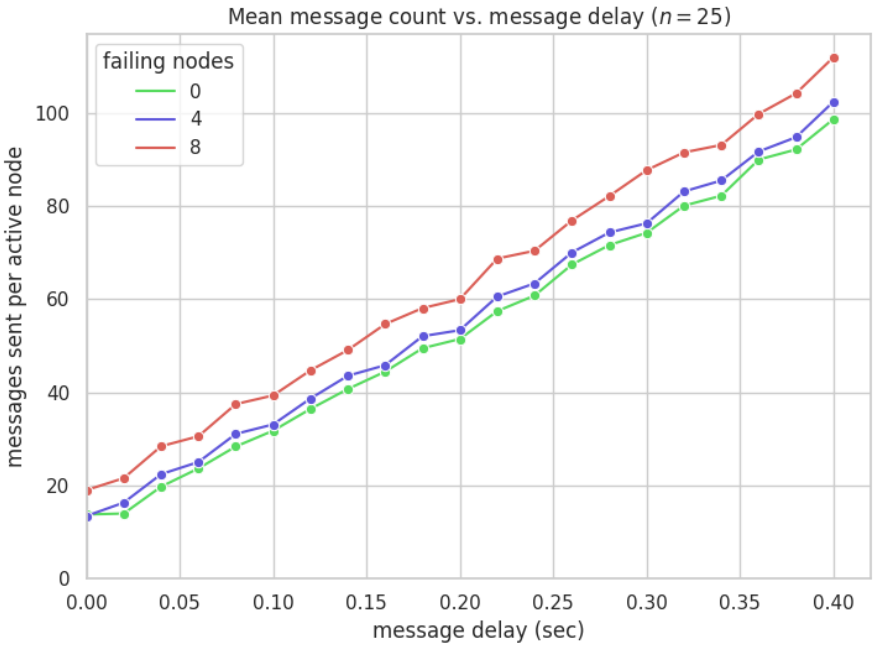


Figure 4: Number of messages sent per active node under varying mean message delays and failing node counts

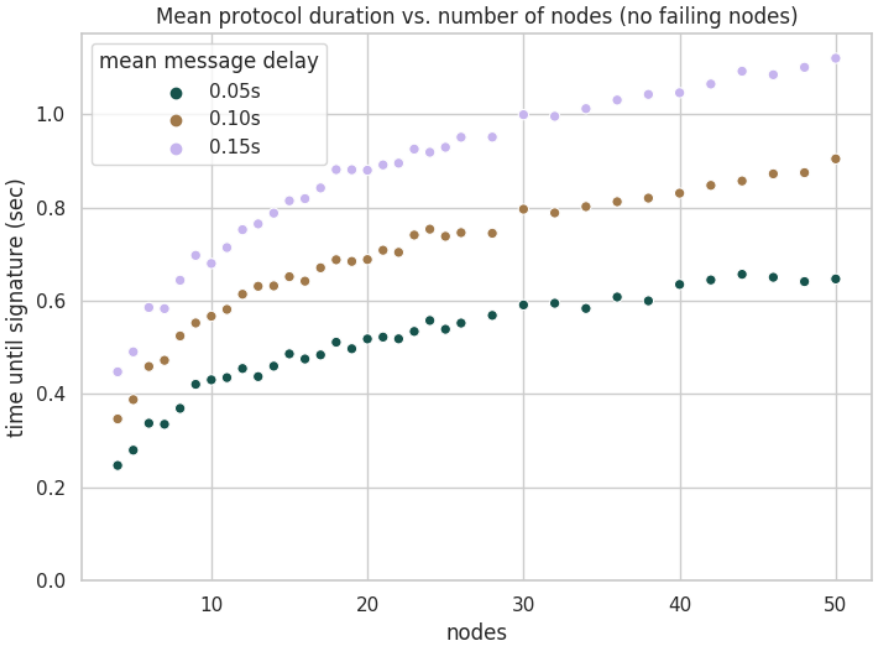


Figure 5: Timings until a cosignature was returned under varying mean message delays and node counts

Figure 5. We can see that the number of nodes has a similar importance as the mean message delay in its effects on protocol duration. The protocol duration grows approximately as $O(log(n))$ as predicted by the theory of gossip protocols described in Subsection 2.1. With $n = 50$ and a relatively pessimistic mean network delay of 0.15 seconds, the mean protocol duration is around 1.1 seconds.

As described in Subsection 3.3, the new protocol can run in two modes when it comes to signature aggregation: either there is no early aggregation, or tree-based aggregation. Figure 6 shows that tree-based aggregation causes a substantial improvement in the amount of data sent per active node. The advantage of this early aggregation is bigger the more nodes there are, with tree-based aggregation saving more than 50% in data transferred for $n = 50$. The aggregation setting had no observable effect on either the protocol duration or the number of messages sent. This is expected, because the setting only affects how the transferred data is represented and does not affect the gossip protocol otherwise.

Another important parameter is the rumor-sending interval t. Figure 7 shows that the protocol duration grows roughly linearly as a function of the rumor-sending interval. However, the number of messages sent and the amount of data transmitted skyrocket as the rumor-sending interval tends

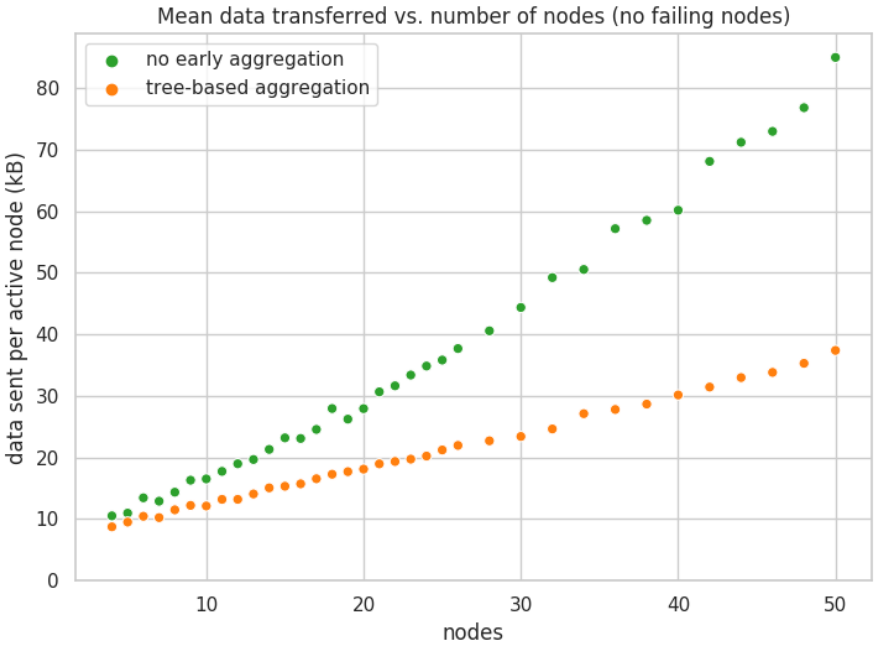


Figure 6: Amounts of data transferred per active node under varying node counts and aggregation settings

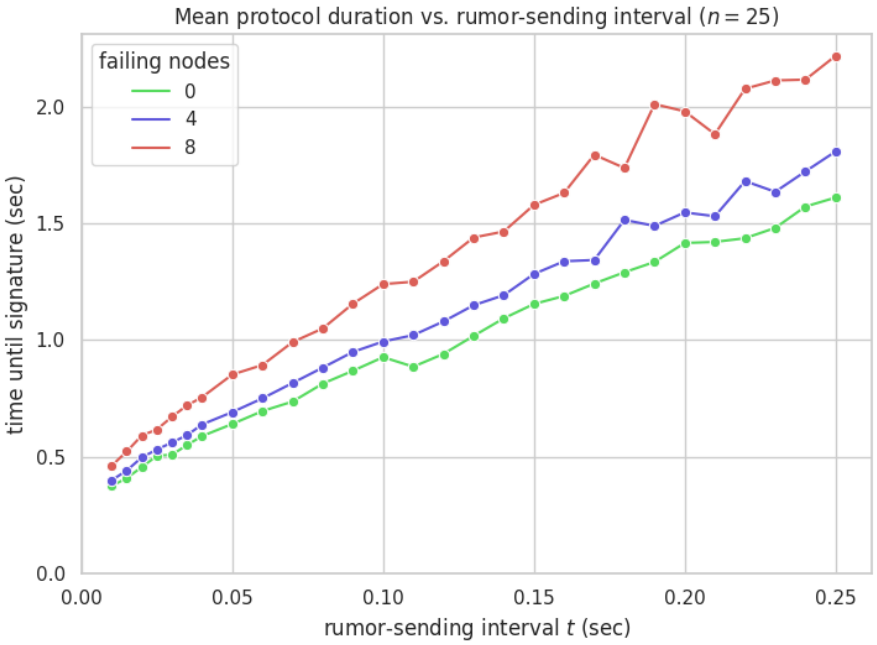


Figure 7: Timings until a cosignature was returned under varying rumor-sending intervals and failing node counts

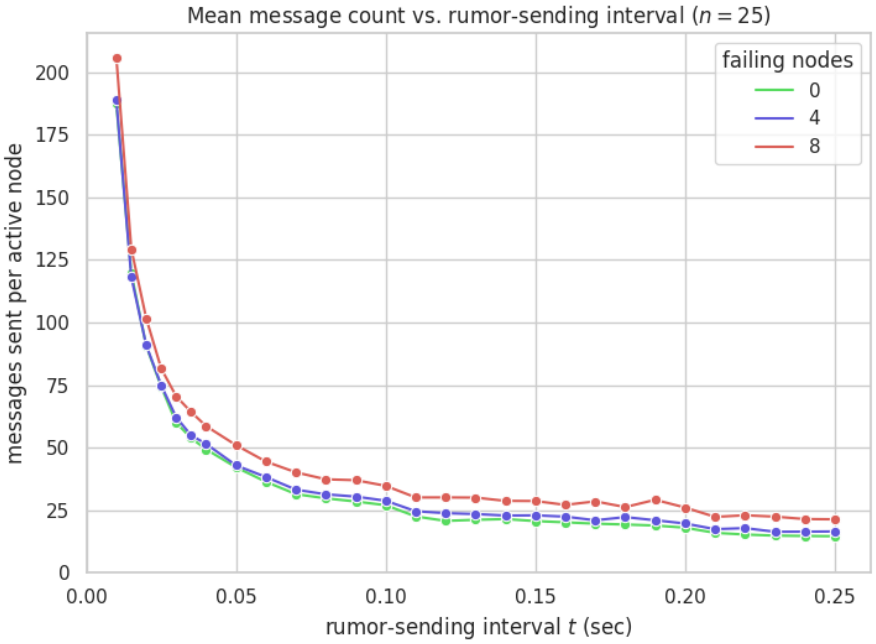


Figure 8: Number of messages sent per active node under varying rumor-sending intervals and failing node counts

towards zero. For the message, this is shown in Figure 8. Therefore, there is a trade-off in the choice of rumor-sending interval: a shorter interval decreases the protocol duration, but increases the number of messages and the amount of data transferred.

Some more experiments were run to test the effects of other less important parameters. Their results in the form of plots are found in Appendix B and will be summarized here. We found that the important quantity in the aforementioned trade-off is the number of rumor messages sent per node per second. This quantity depends on two parameters: the number of recipients for each rumor message r, and the rumor-sending interval t. The number of rumor messages sent per node per second is $\frac{r}{t}$. If the ratio between r and t is not changed, we found that the mean of the measured metric only varies slightly. Changing r therefore has a similar effect as changing t in the opposite direction. With $\frac{r}{t}$ constant, we found that the protocol duration is slightly lower when r is reduced. The effect is rather small and the difference between $r = 1$, $r = 2$ and $r = 3$ is hardly noticeable in our monitored metrics.

The number of initial shutdown-message recipients s had a small, but noticeable effect on the number of messages sent per active node: the total message count increases with s. We do not have enough data to tell what

effect s has on the amount of data transferred. As expected, s has no effect on the protocol duration.

Finally, we also tested the effect of the variance in message delays on our metrics (with the mean message delay fixed at 0.1 seconds). We found that the variance has no big impact on the three metrics we monitor. A higher variance slightly reduced the mean protocol duration in our experiments.

6 Analysis and discussion of the findings

The initial goal of improving fault tolerance over BLS CoSi is met by the new gossip cosigning protocol. The protocol takes a reasonably short time to produce a cosignature, although an even shorter duration may be desirable for some applications. Extreme outliers in the protocol duration almost never occur, which is an important property for many applications. Both the number of messages exchanged and the amount of data transferred are higher than with BLS CoSi. This is to be expected, as the redundancy inherent to gossip protocols is avoided in BLS CoSi. BLS CoSi also uses a lot more bandwidth on some nodes than on others. Taking these considerations into account, the higher measurements for the gossip protocol on these two metrics are reasonable. The amount of data transferred has been kept low through the optimization of aggregating signatures early. We have experimentally backed up some predictions of gossip protocol theory (as applied to this protocol), notably that the protocol duration grows roughly like $O(log(n))$ as a function of the number of nodes n.

6.1 Limitations

The new gossip protocol still has its limitations. It does not adapt to the properties of the network, such as the network message delay. Ideally, a gossip protocol would send fewer messages per second in a slower network. The rumor messages sent in the network can get quite large, depending on the number of nodes. This has not been a problem in our experiments with up to 50 nodes, but it could easily become a problem with thousands of nodes.

Due to the inherent randomness in the gossip protocol and the unpredictability of network message delays, there is no certain upper limit for the protocol duration. Membership needs to be closed for the duration of the protocol, and all nodes need to know each other's identities and public keys. To be safe against man-in-the-middle attacks, the protocol needs authenticated channels for messages.

6.2 Future work

The gossip protocol can be improved in a number of ways. Currently, the recipients of rumor messages are selected randomly at uniform, but there may be better ways of selecting peers. If this is changed, care needs to be taken not to introduce new vulnerabilities into the protocol.

The current strategy of aggregating signatures along a binary tree is relatively simple and shows some clear improvements over the alternative approach of not aggregating signatures until the end. It is possible that there are better ways of aggregating signatures. However, this is not straightforward because signatures cannot be aggregated if the sets of signatories are not disjoint.

The current implementation relies solely on push messages to construct a cosignature. Adding pull messages may be an improvement. For example, nodes might specifically pull from peers whose signature they are missing.

The protocol's properties have not been formally proven. This could become necessary if, for example, the statistical properties of the protocol duration are important for an application such as ByzCoin.

7 Conclusion

Using a gossip protocol has turned out to be a well-suited tool for designing a cosigning algorithm. The new cosigning protocol has an unusually short lifespan compared to most gossip protocols, but it fulfills the goals of being fault tolerant, not overwhelming any node and still being relatively fast. This has been found after testing the protocol experimentally under various circumstances and measuring three key performance metrics. In simulations, the new protocol was compared to the pre-existing BLS CoSi in terms of speed, reliability and efficiency. While BLS CoSi is slightly more efficient, using less bandwidth for the same, the new protocol is more reliable both in the time it takes to produce a cosignature and in the rate of protocol failures. For most applications, the new protocol is therefore an improvement over BLS CoSi.

The new gossip cosigning protocol is optimized for use in ByzCoin, but it can also be used in other applications including transparency mechanisms based on cosigning. Future improvements of the protocol are expected, both to expand its possible uses and to optimize the protocol's speed and efficiency.

A Appendix: Installing and running the program

The code for the program is published on GitHub [15]. It can be installed using Go 1.12 by running

```
env GO111MODULE=on go get \
    github.com/dedis/student_19_gossip_bls
```

and then navigating to the directory

```
student_19_gossip_bls/blscosi_bundle/blscosi_bundle/
```

and running this command:

```
env GO111MODULE=on go install
```

As always in the Cothority project, a cothority can be run with a command like `./run_nodes.sh -n 6 -v 1` in the conode directory after building the `conode` executable using `env GO111MODULE=on go build`. After a command such as `date > date`,

```
blscosi_bundle sign -o sig.json date
```

launches a signature request and

```
blscosi_bundle verify -o sig.json date
```

verifies the signature. To launch a simulation, a command like

```
simulation_bundle local.toml
```

can be used after installing the simulation program in the directory

```
student_19_gossip_bls/blscosi_bundle/simulation_bundle/
```

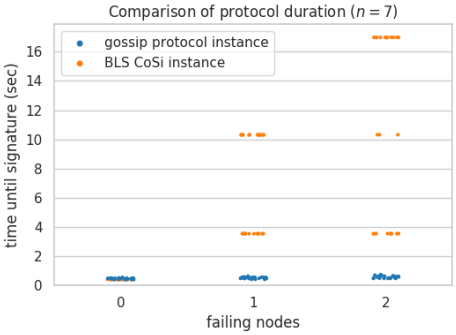
again with the same command as above:

```
env GO111MODULE=on go install
```

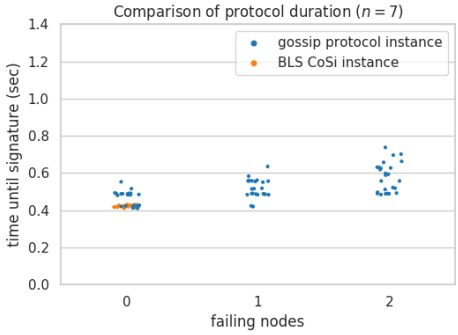
B Appendix: Full simulation results

This appendix shows some further uncommented simulation results that may be of interest. Results from Section 5 are repeated here for the sake of completeness. The y-axis starts at 0 in all graphs for better readability.

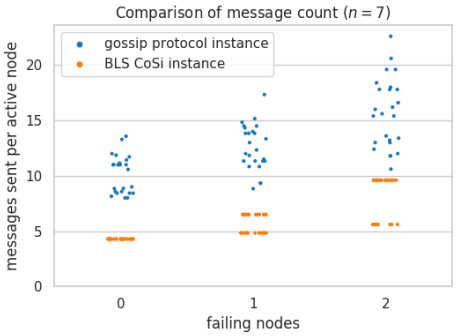
Comparison of BLS CoSi and the new gossip protocol



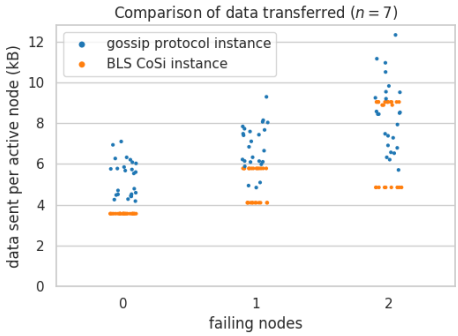
Protocol duration, $n = 7$



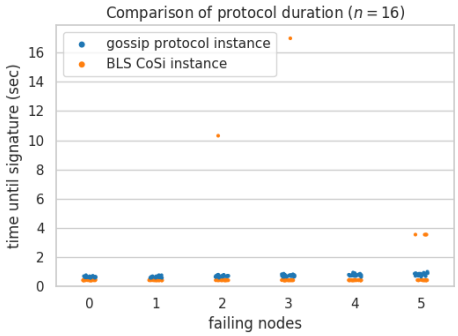
Protocol duration, $n = 7$, zoomed in to show short instances



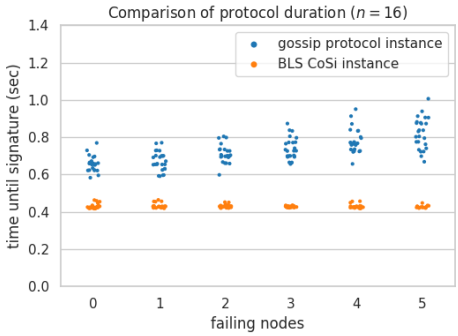
Message count, $n = 7$



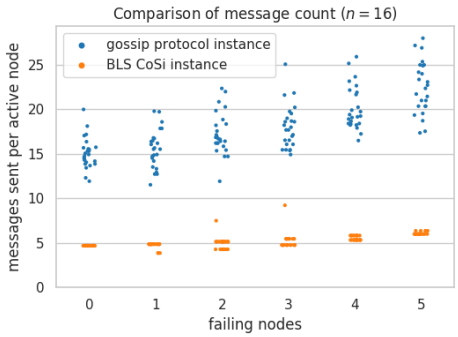
Data transferred, $n = 7$



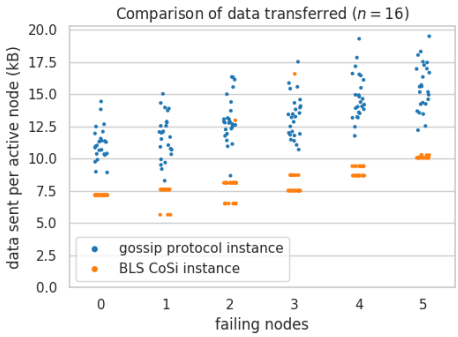
Protocol duration, $n = 16$



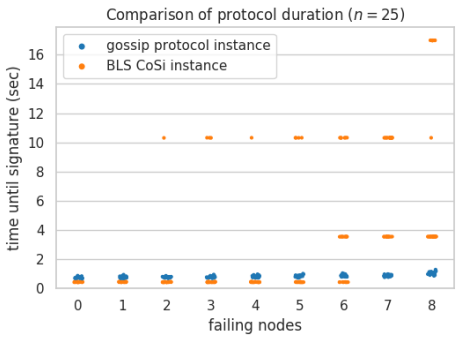
Protocol duration, $n = 16$, zoomed in to show short instances



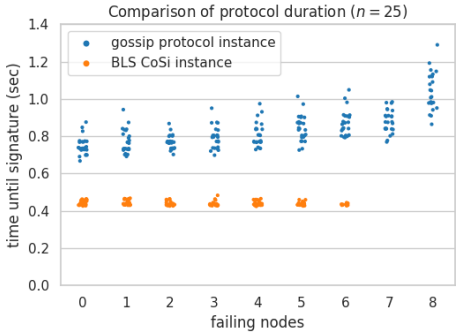
Message count, $n = 16$



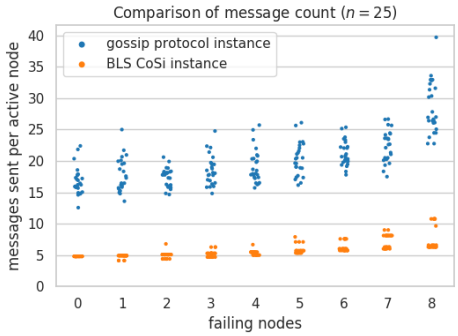
Data transferred, $n = 16$



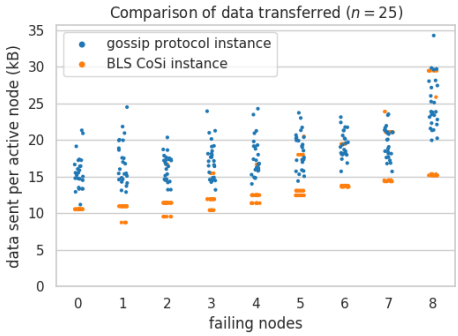
Protocol duration, $n = 25$



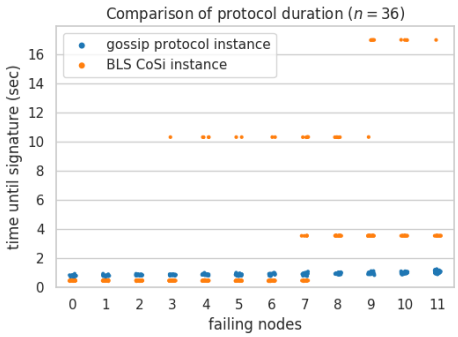
Protocol duration, $n = 25$, zoomed in to show short instances



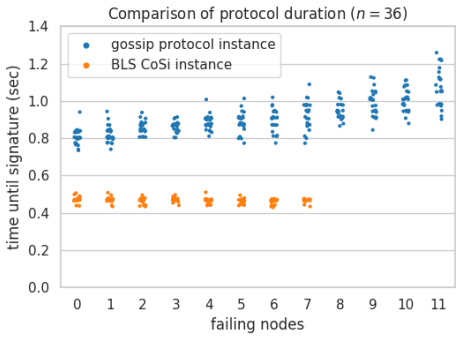
Message count, $n = 25$



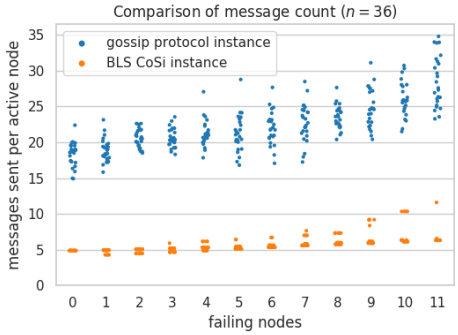
Data transferred, $n = 25$



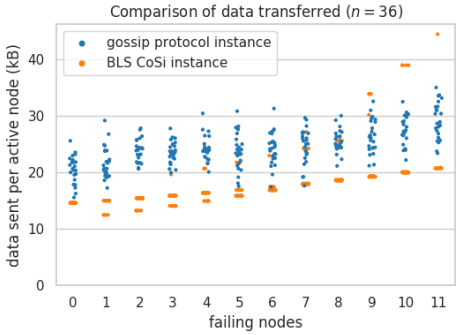
Protocol duration, $n = 36$



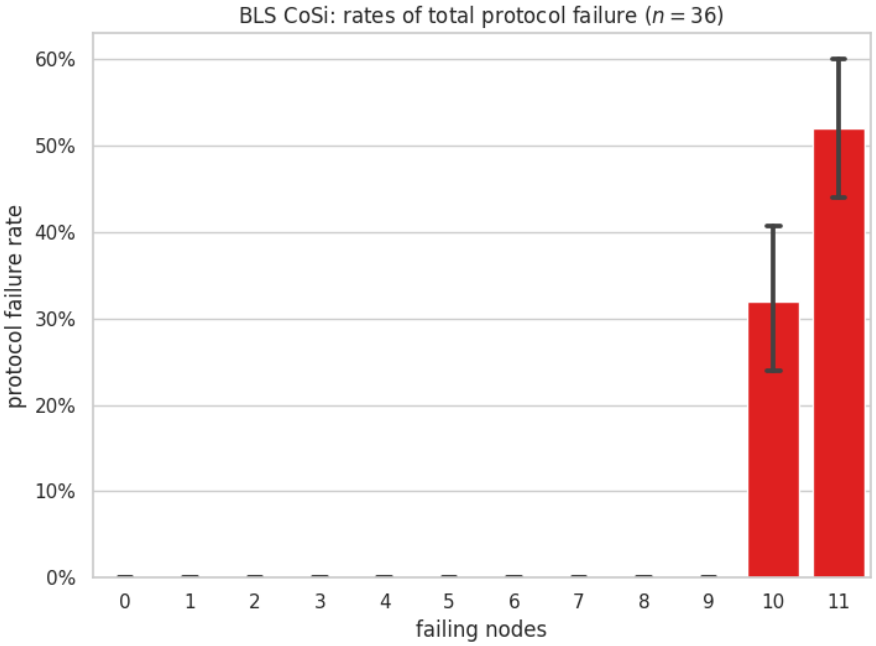
Protocol duration, $n = 36$, zoomed in to show short instances



Message count, $n = 36$

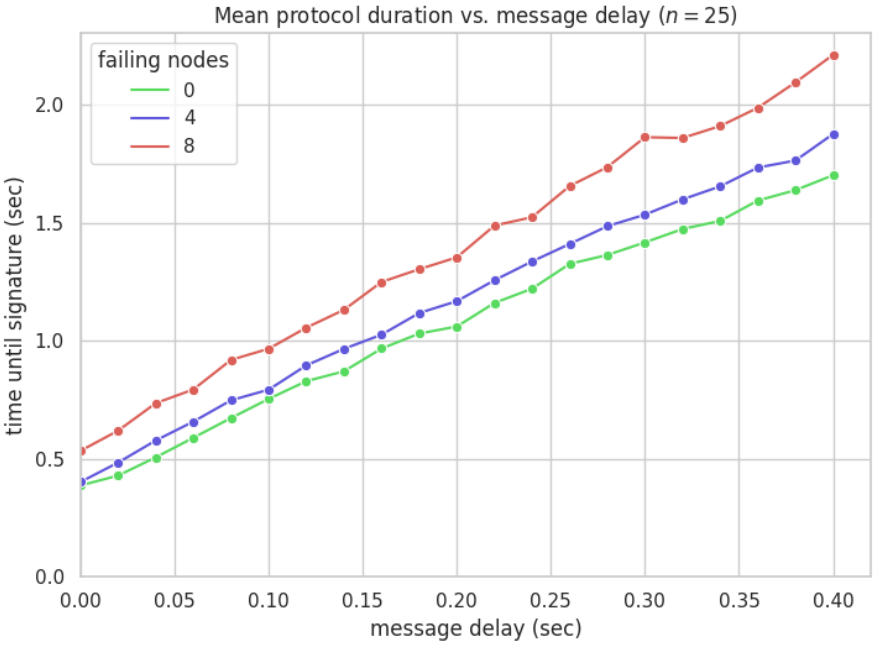


Data transferred, $n = 36$

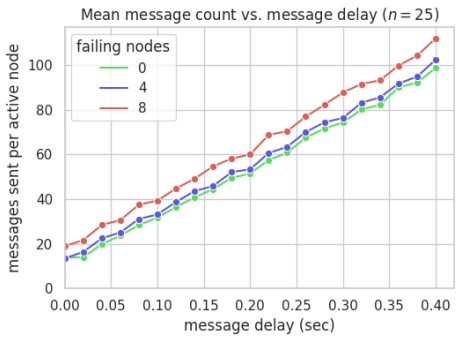


Rates of total protocol failure in BLS CoSi, $n = 36$

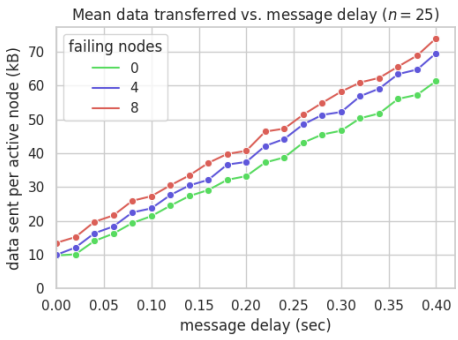
Varying message delay times



Protocol duration vs. mean message delay. The difference between minimum and maximum delay is fixed at 0.01 seconds (except for the special case where the minimum and maximum delay are 0.0 seconds).

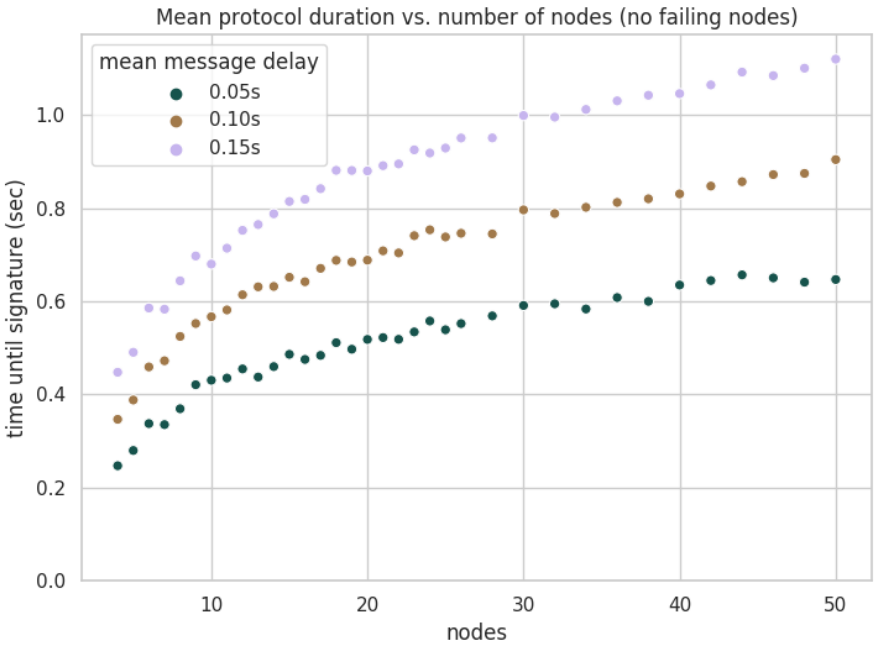


Message count vs. mean message delay

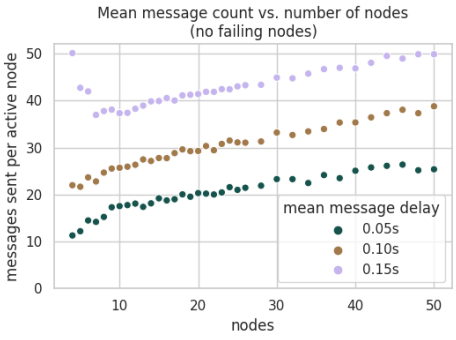


Data transferred vs. mean message delay

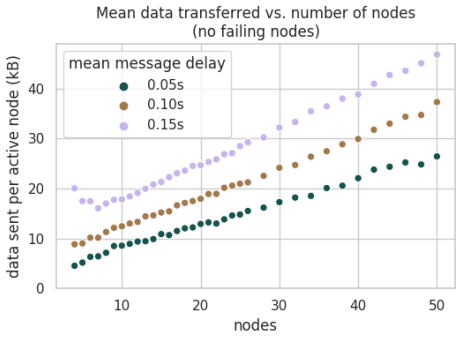
Varying numbers of nodes and message delays



Protocol duration vs. number of nodes by mean message delay. The difference between minimum and maximum delay is fixed at 0.01 seconds.

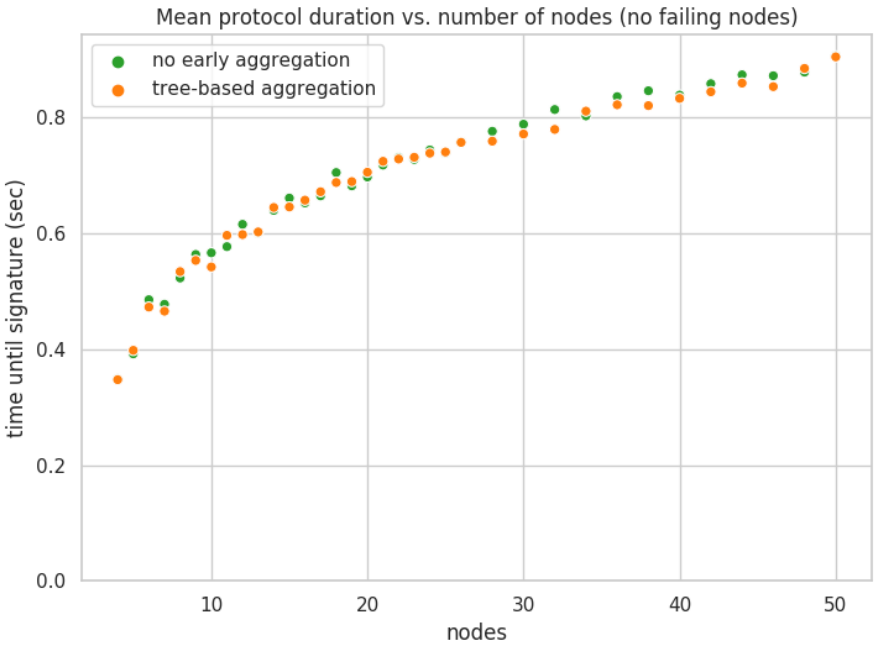


Message count vs. number of nodes by mean message delay

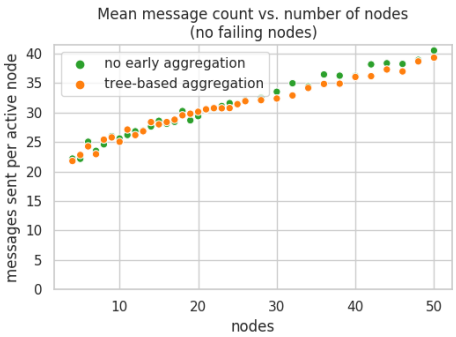


Data transferred vs. number of nodes by mean message delay

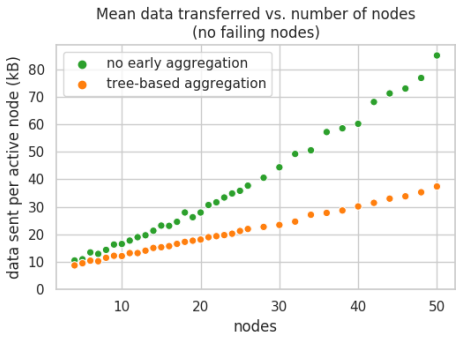
Varying numbers of nodes and aggregation settings



Protocol duration vs. number of nodes by aggregation setting

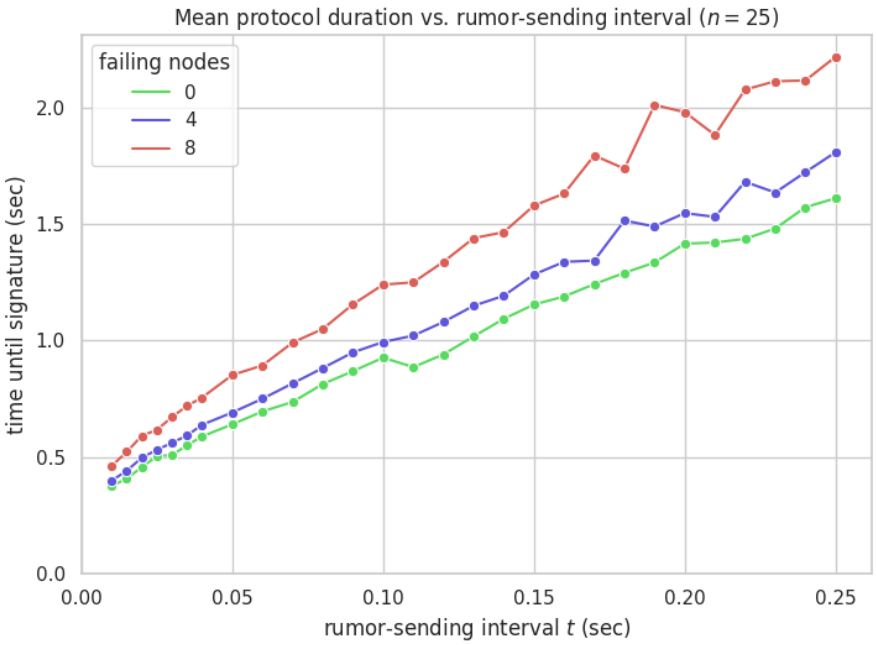


Message count vs. number of nodes by aggregation setting

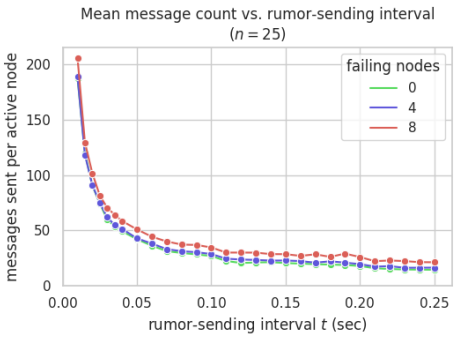


Data transferred vs. number of nodes by aggregation setting

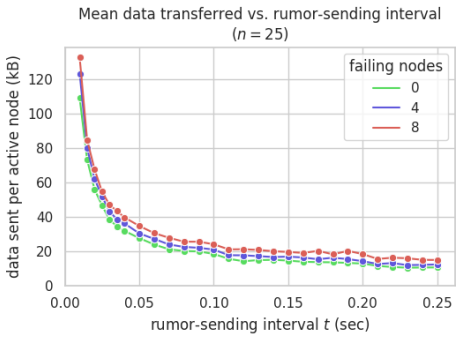
Varying rumor-sending intervals



Protocol duration vs. rumor-sending interval t

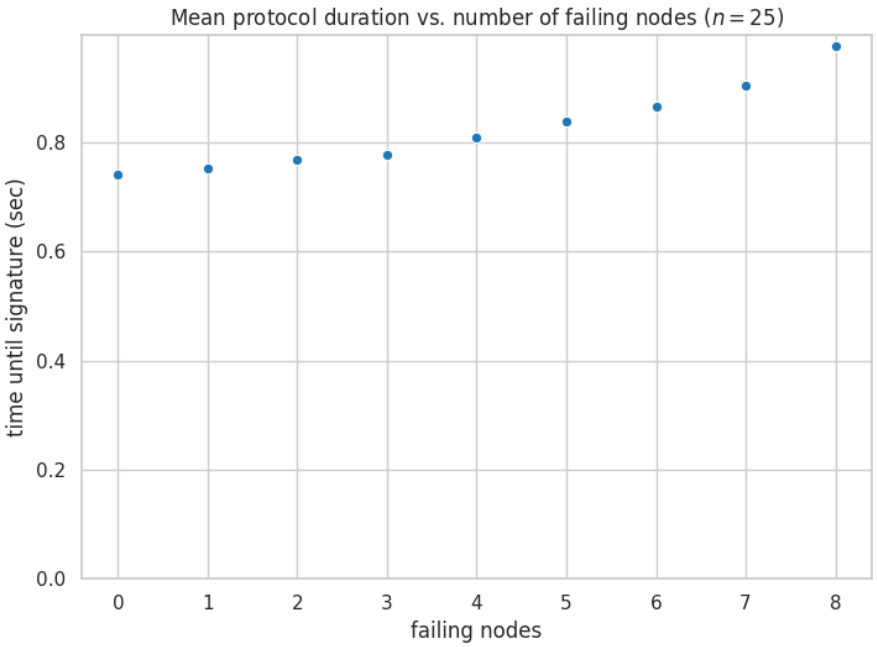


Message count vs. rumor-sending interval t

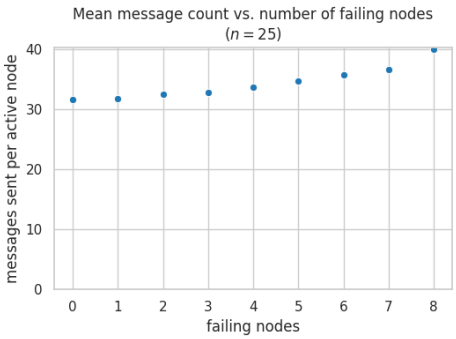


Data transferred vs. rumor-sending interval t

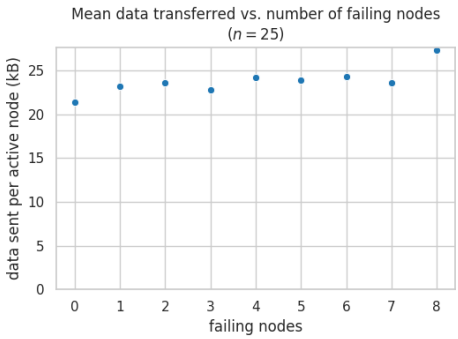
Varying numbers of failing nodes



Protocol duration vs. number of failing nodes

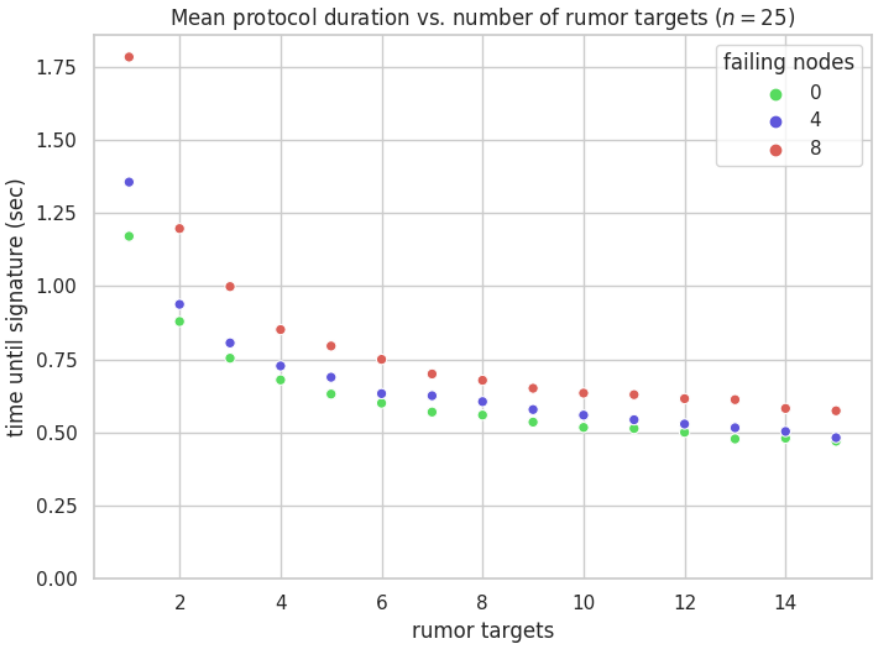


Message count vs. number of failing nodes

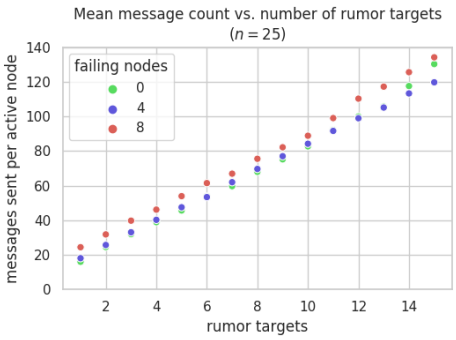


Data transferred vs. number of failing nodes

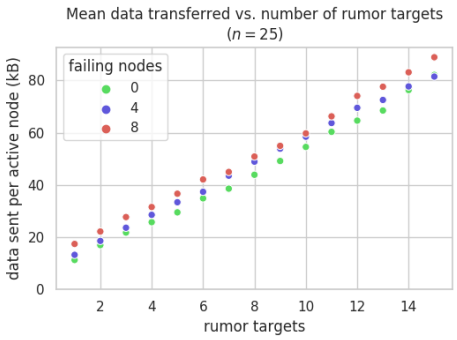
Varying numbers of rumor-message recipients



Protocol duration vs. number of rumor targets

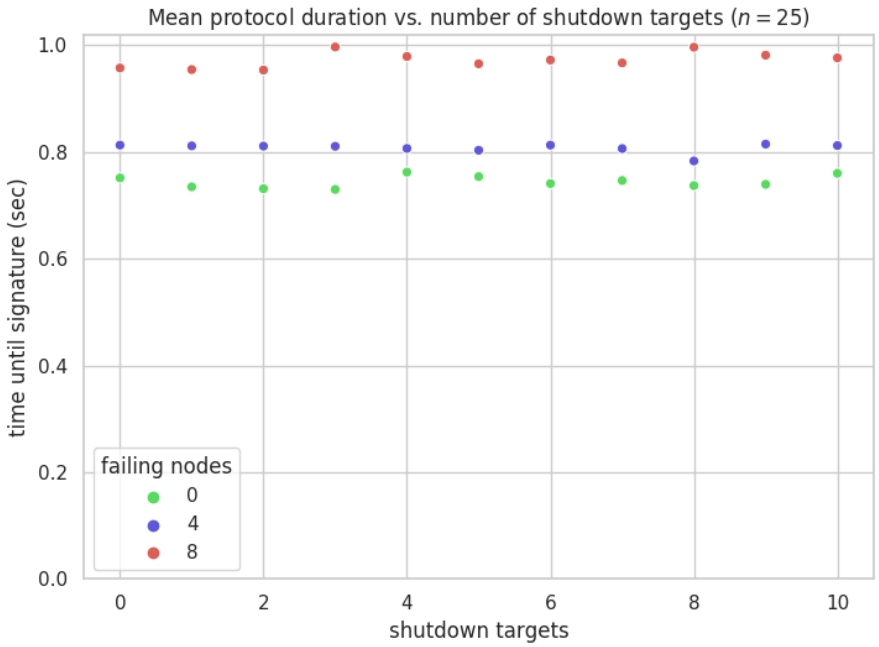


Message count vs. number of rumor targets

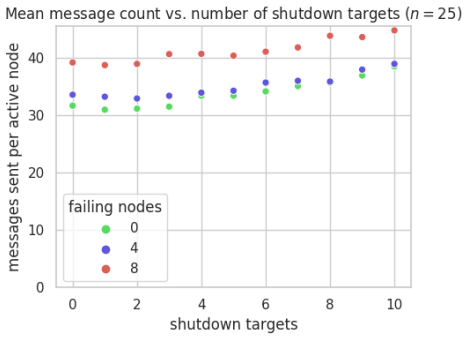


Data transferred vs. number of rumor targets

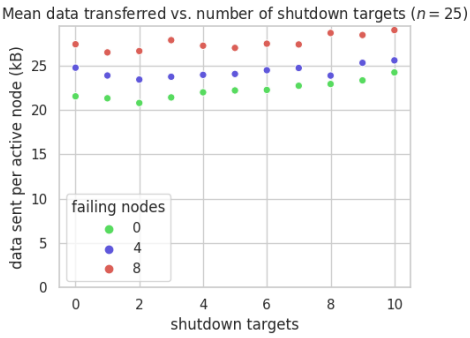
Varying numbers of shutdown-message recipients



Protocol duration vs. number of shutdown targets

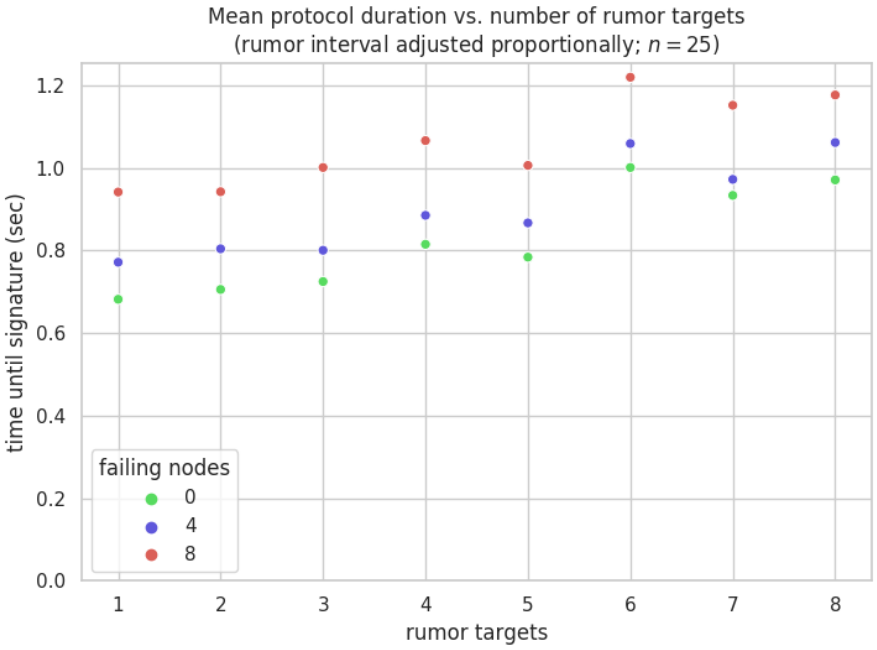


Message count vs. number of shutdown targets

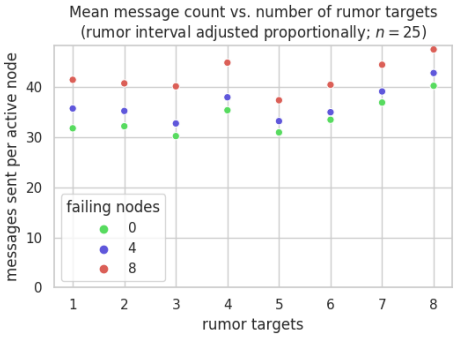


Data transferred vs. number of shutdown targets

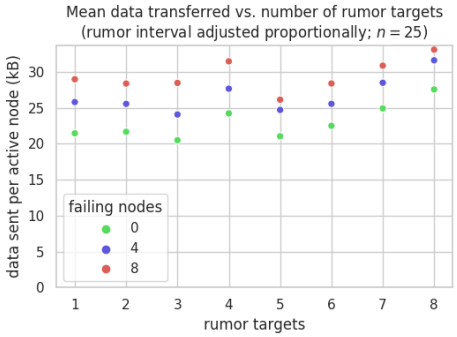
Varying numbers of rumor-message recipients, rumor-sending interval adjusted



Protocol duration vs. number of rumor targets. The rumor-sending interval t is adjusted proportionally to the number of rumor targets so that the number of messages sent per node per second is always 42.9.

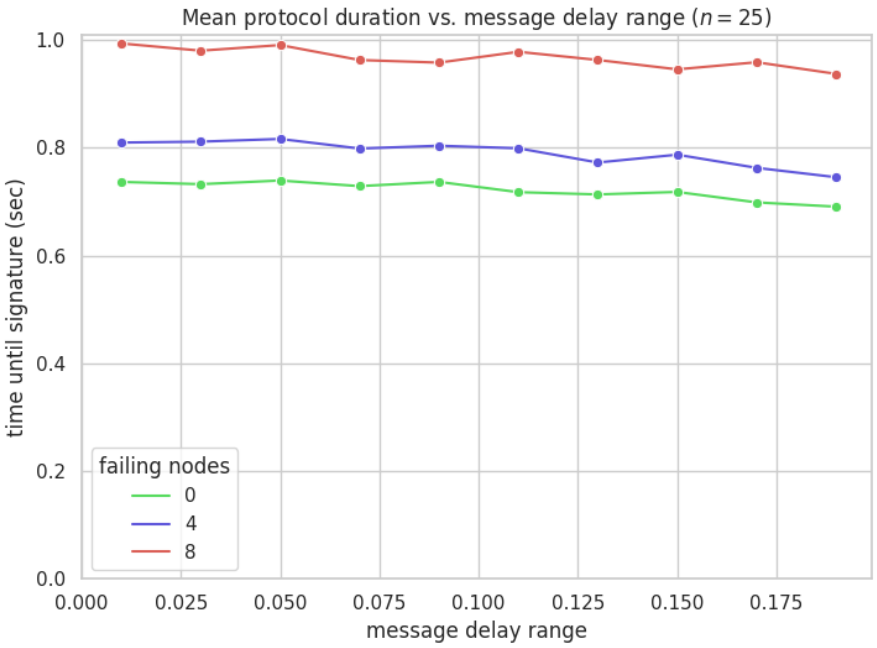


Message count vs. number of rumor targets. The rumor-sending interval t is adjusted proportionally.

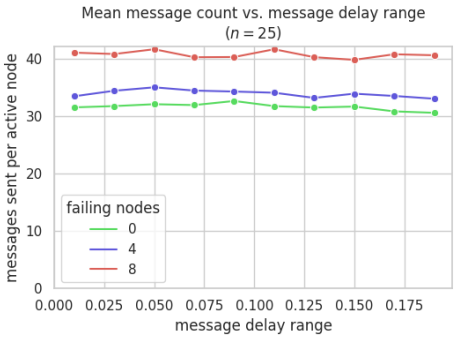


Data transferred vs. number of rumor targets. The rumor-sending interval t is adjusted proportionally.

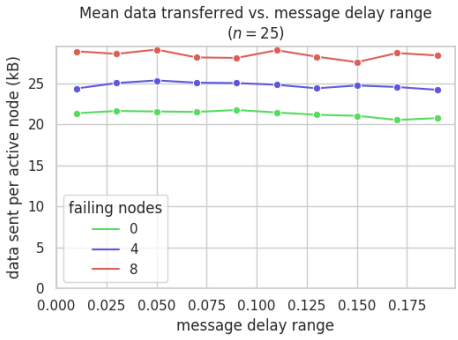
Varying message delay variance



Protocol duration vs. the difference between the minimum and maximum message delay. The mean message delay is fixed at 0.1 seconds.



Message count vs. the difference between the minimum and maximum message delay



Data transferred vs. the difference between the minimum and maximum message delay

References

[1] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. Keeping authorities "honest or bust" with decentralized witness cosigning. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 526–545, 2016.

[2] Kirill Nikitin, Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Justin Cappos, and Bryan Ford. CHAINIAC: Proactive software-update transparency via collectively signed skipchains and verified builds. In *26th USENIX Security Symposium (USENIX Security 17)*, pages 1271–1287, 2017.

[3] Bryan Ford. Apple, FBI, and software transparency. `http://bford.info/2016/03/10/apple/`, 2016. Accessed: June 7, 2019.

[4] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296, 2016.

[5] DEDIS lab. BLS collective signing. `https://github.com/dedis/cothority/tree/master/blscosi`, 2019. Accessed: June 7, 2019.

[6] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532, 2001.

[7] Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432, 2003.

[8] Dan Boneh, Manu Drijvers, and Gregory Neven. Compact multi-signatures for smaller blockchains. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 435–464, 2018.

[9] Ken Birman. The promise, and limitations, of gossip protocols. *ACM SIGOPS Operating Systems Review*, 41(5):8–13, 2007.

[10] Anne-Marie Kermarrec and Maarten Van Steen. Gossiping in distributed systems. *ACM SIGOPS Operating Systems Review*, 41(5):2–7, 2007.

[11] Márk Jelasity. *Gossip-based Protocols for Large-scale Distributed Systems*. PhD thesis, University of Szeged, 2013.

[12] DEDIS lab. Kyber – advanced crypto library for the Go language. `https://github.com/dedis/kyber`, 2019. Accessed: June 7, 2019.

[13] Stepan Snigirev. BLS signatures: better than Schnorr. `https://medium.com/cryptoadvance/bls-signatures-better-than-schnorr-5a7fe30ea716`, 2018. Accessed: June 7, 2019.

[14] DEDIS lab. Cothority – scalable collective authority. `https://github.com/dedis/cothority`, 2019. Accessed: June 7, 2019.

[15] Lukas Gelbmann. Code repository for this semester project. `https://github.com/dedis/student_19_gossip_bls`, 2019. Accessed: June 7, 2019.