Collective Endorsement and the Dissemination Problem in Malicious Environments[*]

Subramanian Lakshmanan   Deepak J. Manohar   Mustaque Ahamad   H. Venkateswaran

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332
{subbu,mjdeepak,mustaq,venkat}@cc.gatech.edu

Abstract

We consider the problem of disseminating an update known to a set of servers to other servers in the system via a gossip protocol. Some of the servers can exhibit malicious behavior. We require that only the updates introduced by authorized clients are accepted by non-malicious servers. Spurious updates, in particular those generated by compromised nodes, are not accepted by non-malicious servers. We take the approach of collective endorsement where each server endorses an accepted update by computing a list of message authentication codes with symmetric keys allocated to it. We use a novel key allocation scheme that allocates a set of symmetric keys to each participating server to minimize the total number of keys.

Our protocol is designed to minimize update diffusion time. In the absence of faulty nodes, its diffusion time is $O(log\ n)$, which is the best possible time achieved when nodes only suffer from benign faults. If the actual number of Byzantine faults experienced during an update's dissemination is f, the diffusion time increases to $O(log\ n + f)$. This is better than the latency of previously known protocols that take $O(log\ n + b)$ time, where b is the assumed threshold that defines the maximum number of malicious servers that can be tolerated rather than f, the actual number of failures. The buffer requirements and message sizes are higher in our protocol than other known protocols and thus it trades off memory and bandwidth resources to improve latency.

1 Introduction

Applications are becoming more and more distributed in nature in an increasingly networked world. Trust and security have been important concerns in such distributed applications. Byzantine fault tolerant algorithms [15] aim at tolerating a limited number of malicious servers in a system. Such algorithms are guaranteed to work correctly as long as the number of malicious servers in the system is less than a threshold. A desirable property in such algorithms is that performance of the algorithm in the absence of any malicious activity be comparable to that of corresponding ones in a benign environment. In a benign environment, a node may fail and stop its execution but the node does not disrupt the functioning of the system by behaving maliciously.

We consider the update dissemination problem in a Byzantine environment where some of the participating servers can exhibit arbitrary malicious behavior. An update may be a message that is sent by an authorized person, to be communicated to all the servers in the system, possibly during an emergency situation. An update could also be a new value of a data item that is replicated at the servers for high availability.

[*]This work was supported in part by NSF Trusted Computing program grant CCR-TC-0208655.

Servers communicate with each other in rounds of gossip to disseminate updates introduced at a subset of servers. Non-faulty servers should accept only those updates that are introduced by authorized clients and must reject others, in particular, the spurious ones generated by malicious servers. This is guaranteed only when the number of compromised servers does not exceed a threshold b. This threshold assumption relies on mechanisms that detect server compromises and fix the exploited vulnerabilities to limit the number of servers that can be compromised in a short period of time.

Using public key signatures to protect the disseminated data against data corruption and source spoofing reduces the dissemination problem to one in a benign setting where servers fail by crashing. However, such a scheme requires every client to hold a public key and servers to store public key of every client. Also public key signatures are computationally expensive [1, 13], particularly when large volumes of data are to be disseminated. Hence, eliminating usage of digital signatures for dissemination is desirable in a setting where the client set is large or the frequency at which updates are introduced is high.

Gossip protocols for malicious environments without using public key signatures for dissemination have been explored in the past [2, 3, 4, 5]. These protocols do not take advantage of a possibly smaller number of actual faults when compared to the assumed threshold b. The best known class of protocols [4, 5] disseminate an update in $O(log\ n) + b$ rounds, where b is the assumed maximum number of malicious servers, irrespective of the actual number of malicious servers in the system. We present a gossip style protocol for dissemination in malicious environments which takes $O(log\ n) + f$ rounds to disseminate an update, where f is the actual number of malicious servers in the system. In the absence of any malicious activity, our protocol takes only twice as long as the best possible gossip style protocol for benign settings. The buffer requirements and message sizes are higher in our protocol than other protocols and thus it trades off memory and bandwidth resources to improve latency. Since memory and communication resources available to server nodes continue to increase, our approach is viable and will be able to offer lower latency.

Our protocol uses a novel key allocation scheme that allocates a set of symmetric keys to each participating server. A client introduces an update at a randomly chosen set of servers, called the initial quorum. A server accepts an update when an authorized client introduces the update at the server. Each server endorses an accepted update by computing message authentication codes (MACs) for the update using the keys allocated to the server. We call such a list of MACs an endorsement. Endorsements are disseminated to other servers. When a server verifies that $b + 1$ other servers have endorsed an update, the server accepts the update. Our key allocation scheme reduces the total number of keys in the system and hence message lengths and buffer requirements, when compared to a naive node-to-node key sharing scheme.

The key allocation scheme and the collective endorsement technique presented here are quite general and can be used in other applications in Byzantine environments where a set of nodes has to vouch for the correctness of some information. We show, as an example application, how to use endorsements to render authorization tokens in a distributed system unforgeable. Authorization tokens are used for distributed authorization [6] and in a malicious environment, a token is valid only if at least $b + 1$ servers endorse the token.

Our contributions in this paper are: (1) a gossip-style dissemination protocol in a malicious environment where the diffusion time depends on the actual number of faults in the system and not on the assumed threshold b, and (2) a novel key allocation scheme and the endorsement technique that can be used in distributed applications. Rest of the paper is organized as follows. In section 2, we present the secure store we are building at Georgia Tech to motivate the problems we address in this paper. Section 3 discusses our key allocation scheme. Our dissemination protocol using endorsements and its performance is discussed in section 4. In section 5, we present our collective endorsement scheme to endorse authorization tokens. We discuss some of the related work in section 6 and conclude the paper in section 7.

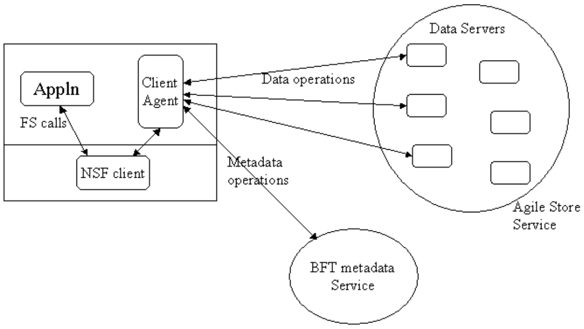


Figure 1: Secure store file system - schematic overview

2 Motivation

We present as a motivating application the secure store we are building at Georgia Tech. Figure 1 shows a schematic overview of our store which is accessed via a file system interface. The system is designed to tolerate a limited number of malicious servers at any time. A set of servers designated as metadata servers offer a threshold metadata service that manages all metadata related to the file system including access control lists. Data is stored at a separate set of servers designated as data servers. These data servers are more widely distributed and replicate the data stored at them to a certain extent. File system clients store and retrieve data from the data servers using a data access protocol. Both the metadata service and the data storage service are designed to tolerate a maximum of b malicious servers in total, at any given time.

All metadata operations are done with the metadata service. Whenever a client wants to access a file, it obtains an authorization token from the metadata service. A client accesses data by contacting a quorum of data servers. The size of a quorum is determined by the consistency and performance requirements for that particular file. Every server in the quorum authorizes the access request independent of other servers by validating the authorization token presented to it. Data written to a subset of data servers is disseminated to other servers in rounds of gossip in the background.

Both authorization tokens and disseminated data need to be endorsed by at least $b + 1$ servers for any non-faulty server to accept them as valid. Thus our file system presents two problems where a group of servers need to collectively endorse some information : (1) creation and use of authorization tokens, and (2) dissemination of updates that are introduced only at a subset of the servers. We use a novel key allocation scheme for endorsements, that allocates a set of keys to each server in the system as explained in the next section.

3 Key Allocation Scheme

In this section we present a key allocation scheme that will be used for endorsements in later sections. We do not address the key distribution problem in this paper. Each server in the system gets a set of symmetric keys from a universal set. These keys will be used by each server to either endorse some information or verify endorsements. An endorsement for some information is a set of MACs computed using that information and a subset of the universal set of keys. If more than one server participate in computing the MACs, we call the endorsement a collective endorsement. We assume the usual cryptographic properties of MACs.

Let the set of servers be indexed with two indices taken from $0\ to\, p - 1$, for a prime p greater than both $\sqrt{n}$ where n is the total number of servers in the system and b where b is the maximum number of servers

S_{31} – \$ S_{12} – #

k_{00}	k_{01}	k_{02} \$	k_{03}	k_{04}	# k_{05}	k_{06}
k_{10} \$	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}	# k_{16}
# k_{20}	k_{21}	k_{22}	k_{23}	k_{24}	k_{25} \$	k_{26}
k_{30}	# k_{31}	k_{32}	k_{33} \$	k_{34}	k_{35}	k_{36}
k_{40}	k_{41} \$	# k_{42}	k_{43}	k_{44}	k_{45}	k_{46}
k_{50}	k_{51}	k_{52}	# k_{53}	k_{54}	k_{55}	k_{56} \$
k_{60}	k_{61}	k_{62}	k_{63}	# k_{64} \$	k_{65}	k_{66}

k'_0	# k'_1	k'_2	k'_3 \$	k'_4	k'_5	k'_6

Figure 2: Key allocation for 2 servers : $S_{3,1}$ and $S_{1,2}$ for $p = 7$. Keys allocated to $S_{3,1}$ are marked by \$ and those allocated to $S_{1,2}$ are marked by #.

that can be malicious at any time. A server is denoted by $S_{\alpha,\beta}$ with $0 \leq \alpha, \beta \leq p-1$. The universal set of keys consists of $p^2 + p$ symmetric keys as follows:

$$U = \{k_{i,j} | i = 0 \; to \, p-1, \; j = 0 \; to \, p-1\} \cup \{k'_i | i = 0 \; to \, p-1\}.$$

A server $S_{\alpha,\beta}$ is allocated the following set of keys:

$$\{k_{i,j} | i = \alpha j + \beta \; mod \, p, j = 0 \; to \, p-1\} \cup \{k'_\alpha\}.$$

That is, server $S_{\alpha,\beta}$ is allocated p keys from the first set of keys along the straight line $i = \alpha j + \beta \; mod \, p$ and the key k'_α from the second set. Figure 2 shows key allocation for two servers, for $p = 7$. We now state two simple properties of this key allocation scheme.

Property 1: Any two servers share exactly one key.

If two servers have the same first index α, they are allocated the same key k'_α from the second set. But they do not share a key in the first set. If the first indices of two servers are different, they share exactly one key in the first set[1] and are allocated different keys from the second set. The following property which is used to validate an endorsement follows directly from property 1.

Property 2: If a server verifies m distinct MACs in an endorsement using the corresponding keys successfully, at least m servers should have participated in computing the MACs unless the verifying server itself generated those MACs.

The following acceptance condition for an endorsement directly follows from property 2.

Acceptance Condition: A server accepts an endorsement as valid if and only if the server verifies at least $b + 1$ MACs in the endorsement, none of which was generated by the server itself.

Since only a maximum of b servers can be faulty in the system, any endorsement computed by $b + 1$ servers is accepted as valid. We do not address the problem of key distribution since it is not the focus of this paper. Schemes that have been explored by others in other fields like multicast and sensor networks can be used in our system [16, 17]. In the next section, we use this key allocation scheme in our dissemination protocol.

[1] In a field mod p, if $\alpha_1 \neq \alpha_2$, two straight lines $i = \alpha_1 j + \beta_1$ and $i = \alpha_2 j + \beta_2$ intersect at only one point, where j at the intersection point is given by $(\beta_2 - \beta_1)(\alpha_1 - \alpha_2)^{-1}$.

4 The Dissemination Problem

4.1 Problem Statement and Assumptions

We consider the problem of update dissemination in a distributed system where the maximum number of malicious servers at any given time is not more than a threshold b. A client introduces an update at at least $2b + 1$ servers. Once introduced, the update is disseminated to other servers in the system securely. An update is said to be valid only if an authorized client introduces it. In particular, spurious updates introduced by malicious servers should not be accepted by non-faulty servers. Hence, a non-faulty server accepts an update as valid if the update is either introduced by an authorized client or accepted by at least $b + 1$ other servers. We assume that every server can communicate with every other server securely. In particular, our protocol uses a pull strategy and communication channels are assumed to be secure against impersonation and replay attacks. We assume a synchronous system since our protocol works in rounds of gossip.

Let the set of servers be $S_{i,j},\ i = 0\ to\ p-1,\ j = 0\ to\ p-1$ for a prime $p > 2b + 1$ [2]. Each server is allocated $p + 1$ keys as described in the previous section.

4.2 Gossip Protocol

Our dissemination algorithm works as described in figure 3. Initially a client introduces an update at m randomly chosen servers, for an m greater than $2b + 1$. Choice of m will be discussed in later sections. We call this set of m servers the initial quorum. Each of the m servers independently authenticates and authorizes the client request before accepting the update (see section 5 for a discussion of distributed authorization). Updates are timestamped to prevent replays. Each of the non-faulty servers that accepts the update directly from a client generates MACs for the update using all the $p + 1$ keys it has and stores the MACs in its buffer. The update itself is disseminated to other servers using a protocol meant for benign environments [7]. Dissemination of MACs is done in rounds of gossip. Each server keeps a counter of successfully verified MACs for each update. Every round, each server selects a partner randomly and requests MACs. Upon a request, a server sends all MACs in its buffer, generated by it or received from other servers, to the requesting server. The requesting server verifies the correctness of all the MACs for which it has the key. It discards the invalid ones and updates the counter for the newly verified MACs. If it has successfully verified at least $b + 1$ MACs, the server accepts the update as valid. Once accepted, the server generates the rest of the MACs for the update using the keys it has. The server stores all the verified or generated MACs and other received MACs (for which the server does not have the key to verify) in a buffer to disseminate to other servers in future rounds. If a received MAC for a key that the server does not have is different from the already stored one (received at an earlier round), it replaces the stored MAC with the just received one (see 4.4). All MACs are sent and stored accompanied by identifiers of the keys used to generate them. Figure 4 shows the progress of a typical run of the protocol in a system of 840 servers with a b of 10, for an update introduced at 12 non-malicious servers. The plot shows the number of servers that have accepted the update at the end of each round.

The correctness of the gossip protocol can be demonstrated by showing that the following properties hold.

Safety: No spurious update introduced by a group of malicious servers will be accepted by any non-faulty server as long as the number of malicious servers is less than or equal to b.

Since no non-faulty server will generate any MAC for an update before accepting it, not more than b servers will generate MACs for a spurious update. This implies, from property 2 of the key allocation

[2] Number of servers can be less than p^2 but each server receives two indices i, j between 0 and $p - 1$, chosen randomly and without repetition.We only require that each server share at least $2b + 1$ keys with other servers.

```
At server $S_{\alpha,\beta}$:

1.  If an update is introduced by an authorized client, accept
    the update and generate MACs using the allocated keys. Store
    the generated MACs in buffer to be disseminated to other servers.

2.  Each round :

        2.1.  Choose a random partner.
        2.2.  Ask for updates and collect MACs.
        2.3.  For each received MAC
              2.3.1. If the key to verify is present,
                            Verify the MAC using the allocated key,
                            store in buffer if successfully verified, reject otherwise.
                     Else,
                            Accept the incoming MAC and store in buffer, possibly replacing
                            an already stored MAC for the same key and update with
                            the received MAC(see  4.4).

3.  If some server asks for updates, forward all the stored MACs.

4.  Accept an update as valid if the update is verified by $b + 1$ MACs using distinct
    keys. If accepted, generate the rest of the MACs for the update and store in buffer.
```

Figure 3: Gossip Protocol

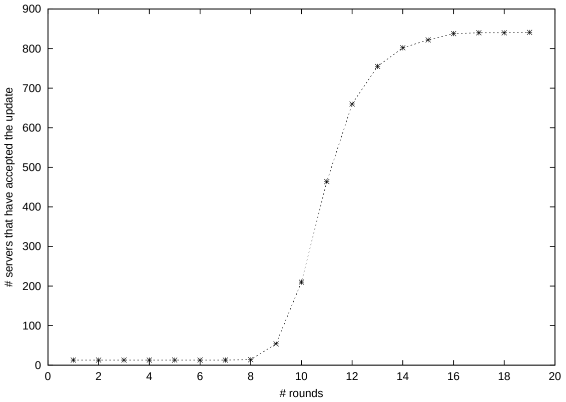


Figure 4: Number of servers that have accepted the update as a function of the round number in a typical run for n=840,b=10 for an update injected at 12 non-malicious servers.

scheme, that no server will be able to verify $b + 1$ MACs for a spurious update. Thus, a spurious update will never be accepted by any non-faulty server.

Liveness: If an update is introduced by a client at a sufficient number of servers, the update will eventually be accepted by all servers.

A server has to verify at least $b + 1$ MACs to accept an update. For a sufficiently large initial quorum,

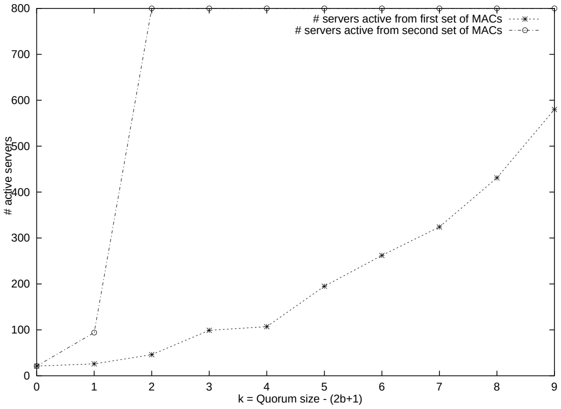


Figure 5: Number of servers that accept the update from first and second set of MACs for different sizes of initial quorum, k - difference between quorum size and optimal quorum size, 2b+1, for $n = 800$ servers and $b = 10$.

there will be a number of servers that would accept the update as valid from MACs generated by the servers in the initial quorum. The size of the initial quorum is $2b + 1 + k$ for some small value of k (see section 4.3). All those servers whose key allocation lines intersect those of the servers in the initial quorum at at least $2b + 1$ distinct points will accept the update. Once a fraction of servers accept the update, these servers generate MACs in turn, which are disseminated to other servers. This would lead to remaining servers accepting the update. Malicious servers cannot stop valid MACs from reaching non-faulty servers. They can only delay the flow of MACs reaching non-faulty servers. The pull strategy we use further limits the power of malicious servers to stop the flow of valid MACs. Every generated MAC will eventually reach every server. Hence all non-faulty servers would accept the update eventually.

4.3 Choice of Initial Quorum

When introducing an update, a client chooses an initial quorum of servers to introduce the update into the system. The size of this quorum is determined by the requirement that a sufficient number of servers outside the initial quorum should each share at least $2b + 1$ keys with the initial quorum. After MACs generated by the initial quorum are disseminated to all the servers in the first phase, a fraction of servers accept the update and generate more MACs. Other servers will accept the update after receiving the MACs generated in the second phase. In appendix A, we show that if $p \geq q \geq 4b + 3$, all servers will accept the update in two phases for any choice of initial quorum of size q. This is only a theoretical upper bound and in practice we have found that we require a much smaller initial quorum. If the servers in the initial quorum have keys allocated along parallel lines from the first set, then the size of the initial quorum can be $2b + 1$. If the initial quorum is chosen randomly, our simulations show that the size of the initial quorum is $2b + 1 + k$ for a small k. Figure 5 shows the average number of servers that directly accept the update in the first phase from the MACs generated by the initial quorum of servers as a function of k for randomly chosen initial quorum and the total number of servers that will accept the update at the end of second phase. As can be seen from the plots, for a system of about thousand servers with a maximum of ten being malicious, a small k equal to two or three serves our purpose.

4.4 Dealing with Conflicting MACs

In the gossip protocol shown in figure 3, a server that receives a MAC for an update for a key that it does not have cannot verify the correctness of the MAC. However, the server stores the MAC to forward to other

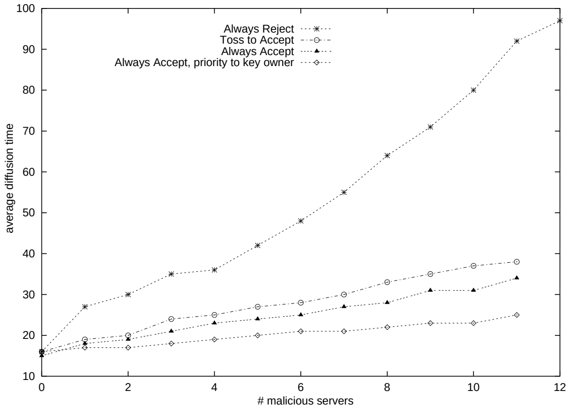


Figure 6: Average diffusion time against actual number of faults for $b = 11$ and $n = 1000$ servers, for various policies on resolving conflicts between MACs.

servers. A malicious server may generate invalid MACs for a valid update, to mount denial of service attacks on other servers' buffers and hence delay the acceptance of an update. A receiving server, when it receives a MAC that is different from the already stored MAC for the same update and the same key, has to decide which MAC to keep. There are three different possible strategies for handling this situation. One strategy is for the receiving server to always reject the incoming MAC. That is, the first received MAC stays in the server buffer and all others are rejected. Second strategy is to accept the incoming MAC with a certain probability. Third strategy is to always accept the incoming MAC, replacing any previously accepted MAC. Figure 6 compares the performance of each of these strategies. Our simulations show that the third strategy is the most effective while the first is the least effective. This is because the always-accept strategy gives all generated MACs a chance to reach every server quickly.

The protocol could be further optimized by requiring the receiving server to give preference to MACs received from servers that have the keys for those MACs over those from servers that do not have the keys for the MACs. The last plot in figure 6 shows diffusion times for this policy. To implement this policy, each server has to know what keys are allocated to other servers.

4.5 Key Consensus

We do not consider the problem of how each server receives the keys allocated to it. The problem of secure key distribution has been addressed by others [16, 17]. In this section we discuss an issue that may be of concern in key distribution, namely consensus on shared keys. Each key in our key allocation scheme is shared by p servers. Some of these servers may be malicious. Hence, some servers that share a key may not have identical copies of the key unless a Byzantine fault tolerant consensus protocol is used for key distribution. In the file system we build, our metadata service is a single central threshold service that can be used to distribute the keys to the data servers. Even if this service is not available, we point out that a strict consensus on all keys is not necessary. Any distribution algorithm that distributes the keys correctly when no participating server is malicious would work for our purpose. As long as each server shares $2b + 1$ keys with other servers, there will be at least $b + 1$ good keys that will be useful in the dissemination process. Hence, as long as keys that are not allocated to any malicious server are correctly shared, our dissemination algorithm works correctly. As an example, a simple key distribution scheme could be used where, for each key a designated key leader distributes keys to other servers. All our simulations and experiments were run by making invalid all keys that are allocated to at least one malicious server.

Metric	Tree Random([3])	Short-Path (Malkhi et.al. [5])	Youngest-Path Path Verification (Minksy et.al. [4])	Collective Endorsements
Diff. Time	$\Omega(b.log(n/b))$	$O(log\ n + b)$	$O(log\ n) + b + c$	$O(log\ n) + f$
Mesg. Size	$O(1)$	$\psi(n, b)$	$30(b+1).\ O(log\ n)$	$d.\ O(p^2)$
Storage	$O(b)$	$\psi(n, b)$	$30(b+1).\ O(log\ n)$	$d.\ O(p^2)$
Comp. Time	$O(log\ b)$	$\Omega((\psi(n, b)/log(n/b))^{b+1})$	$O(b^{b+1} + b.log\ n)$	$O(p/(log\ n))$

Figure 7: Performance comparison of different gossip protocols. n - number of servers, d - size of a MAC, c - a small constant, $\psi(n, b) = ((n/b + 2))^{O(log(b+2+log\ n))}$. All measures are per host per round except diffusion time

4.6 Performance

In this section, we analyze the performance of our protocol and compare it with known protocols for dissemination in Byzantine environments. We consider four performance metrics : (1) diffusion time, (2) average message length per host per round, (3) average buffer size required per host per round and (4) average computation time at a server every round. The other performance metric, host load which is defined as the average number of messages received per round is one, which is same for all the protocols discussed here.

We ran simulations and experiments with an implementation of our protocol to validate the claims we make here about the performance of our protocol. We implemented both our protocol and a version of path verification protocol suggested by Minsky and Schneider [4] for a cluster of thirty machines running Linux on 300MHz Pentium processors. For both implementations, round length was set to fifteen seconds. A typical experiment involved starting a randomly chosen set of servers in malicious mode and the rest in normal mode and injecting updates at a randomly chosen set of $b + 2$ non-malicious servers at a chosen frequency. Most effective malicious behavior for our protocol is simply sending random bits for MACs to other servers upon every request. This is easy to see since if a malicious server sends a correct MAC for an update upon a request, it will only possibly reduce the diffusion time of the protocol run. For the path verification protocol, we made malicious servers simply fail benignly, replying with empty list of proposals for requests from other servers. For both protocols, updates were discarded twenty five rounds after they were injected, which was well over the diffusion time for most of the updates. For the path-verification protocol, the diffusion strategy chosen was promiscuous youngest diffusion [4] with an age-limit of 10 rounds for a proposal and the sampling strategy chosen was bundle sampling with a maximum bundle size of 12. A value of 11 was chosen for p for our protocol. The following characteristics were studied: (1) diffusion time as a function of the threshold b (2) diffusion time as a function of the actual number of malicious servers f (3) average message size, buffer size and computation time at each server as a function of the frequency at which updates were injected. Last three metrics were measured when the system achieved a steady state and updates were being dropped at the same rate at which fresh updates were being injected.

Figure 7 compares the performance of our protocol and other known protocols for Byzantine dissemination. It should be noted here that other protocols are information-theoretically secure while ours is only computationally secure since we rely on cryptographic properties of MACs. As can be seen from the table, our protocol trades off bandwidth and memory resources to improve latency. In subsequent sections, we discuss each of these metrics in some detail, presenting our simulation and experimental results. Although protocols presented in [2] and [3] require considerably less bandwidth and memory, their latency is high ($\Omega(b.log(n/b))$) compared to other protocols. The protocol described in [4] offers the best latency among other protocols. Since our primary focus is on reducing latency, we will compare our protocol to [4] in the following.

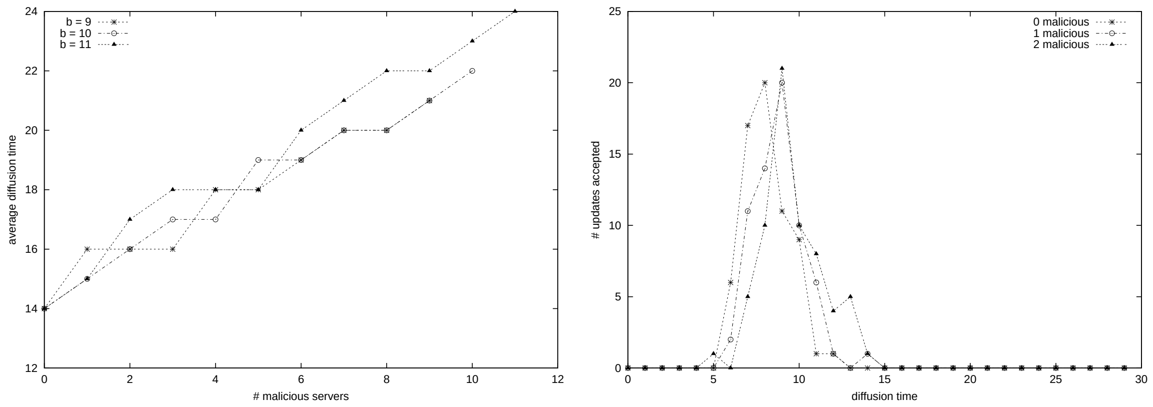


Figure 8: (a) Average diffusion time in number of rounds as a function of f for different values of b for collective endorsement protocol for $n = 1000$ servers, results from simulation, (b) Distribution of diffusion times of updates as a function of f for fixed $b = 3$ for $n = 30$ servers for collective endorsement protocol, experimental result.

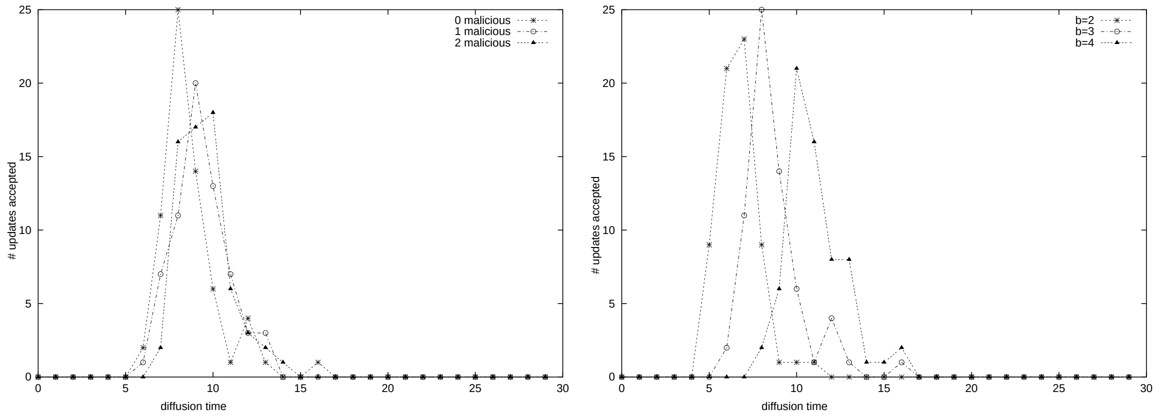


Figure 9: Distribution of diffusion times of updates as a function of f for fixed $b = 3$ and as a function of b for $f = 0$, $n = 30$ servers, for path verification protocol, experimental results

4.6.1 Diffusion Time

When no server acts maliciously, our protocol takes not more than twice the diffusion time of the best protocol for benign environments to diffuse an update. In this case, every generated MAC takes only $O(log\ n)$ rounds (time taken by best-possible benign-case protocol) to reach every server. A fraction of servers accept the update and generate MACs using other keys in turn. The newly generated MACs take another $O(log\ n)$ rounds to reach every server. If the size of the initial quorum is appropriately chosen, every server will verify at least $b + 1$ MACs either generated by the initial set of servers or generated by servers that accept the update based on MACs generated from initial set of servers. We obtain an upper bound for the size of initial quorum in appendix A. Both in our simulation and experimental results, we observed quite often the diffusion time to be less than twice best possible time for a benign setting since (1) some servers accept the update even before the first $log\ n$ rounds and (2) MACs generated in second phase need not reach a lot of servers.

When f servers are acting maliciously, diffusion time of our protocol is $O(log\ n) + f$, independent of the assumed threshold b. We justify this claim by showing in appendix B that in the presence of f malicious servers, each MAC takes $O(log\ n) + f$ rounds to reach a constant fraction of servers. After $O(log\ n)$ rounds,

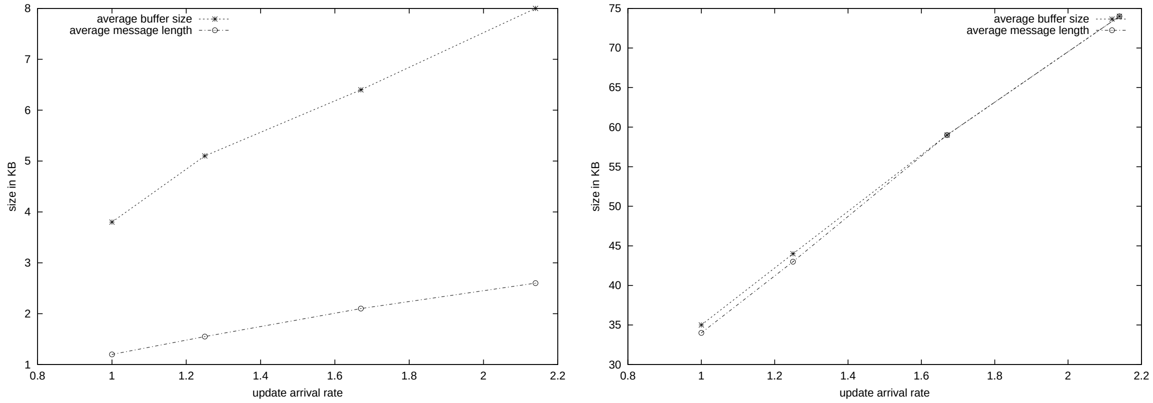


Figure 10: Message size and buffer size in KB as functions of update arrival rate for (a) path verification and (b) collective endorsement protocols for $b = 3$ and $n = 30$ servers, experimental results

for a particular key, a constant fraction of the servers hold some MAC, whether valid or spurious. Fraction of these servers that hold a valid MAC can be shown to be $1/(f + 1)$. Thus after the first $log\ n$ rounds, the probability that a server would obtain a valid MAC is close to $1/(f + 1)$. Among the servers that have the key to verify a MAC, the fraction of servers that have not seen a valid MAC decreases by a factor of $f/(f + 1)$ every round on the average. It takes about $O(f)$ rounds for this fraction to reduce to a small constant.

Our proof in appendix B holds only when malicious servers are not allowed to disseminate spurious MACs for a key before some non-faulty server starts disseminating the corresponding valid MAC. Since this is true for the first set of MACs generated by the initial quorum, these MACs take $O(log\ n) + f$ rounds to reach a constant fraction of servers. Some of the servers accept an update at the end of this phase. Remaining servers would need only a few more MACs to accept the update and these are obtained in a very short period of time, after having been generated by the servers that accept the update in the first phase. Second set of MACs may take a longer time to reach all the servers. However, this does not affect the latency because at the end of first phase, most of the remaining servers need only a few more MACs and the MACs generated in second phase do not have to reach a lot of servers.

Our claim is justified by the results we obtained from our simulations and experiments. Our simulations show that as we increase the number of malicious servers by one every time, starting from no-malicious-servers case, diffusion time is increased only by one round every time. Figure 8(a) shows the average diffusion time as a function of f for various values of b as generated by our simulations. This is validated by the diffusion times we observed in our experiments with our implementation. Distribution of diffusion time across updates for various values of f for a b of 3, as seen in our experiments is shown in figure 8(b). The corresponding plot for our implementation of path verification protocol as measured in our experiments is shown in figure 9. It can be seen from the figures that average diffusion time for path-verification protocol depends on b and f while that of our protocol depends only on f.

4.6.2 Message Size, Buffer Size and Computation Time

Average message size and storage requirements per round per host for our protocol is number of keys times the size of a MAC. Number of keys is $p^2 + p$ where p is greater than $2b + 1$ and $\sqrt{n}$. This is expensive when compared to that of the path-verification protocol, which is $(O((b.log\ n)$. However we believe our buffer requirements and message size are scalable to a moderate number of servers. In our implementation, we chose 128bit MACs. Figure 10 shows average buffer size and message size measured in our experiments for our protocol and path verification protocol, as a function of the frequency of arrival of updates. The figures show that our resource requirements are only an order of magnitude bigger than those of path-verification protocol for the chosen number of servers (30). However, our protocol would need only the same amount of message size and buffer size even when n is increased to 100 as long as b is less than 6. A higher value of b can be chosen for the same $p(11)$ if consensus on shared keys is guaranteed in the presence of malicious faults. Further optimization of message and buffer sizes is possible by making servers generate MACs for multiple updates in a combined fashion. We did not include this feature in our implementation.

We observed slightly smaller average computation time for implementation of our protocol over our implementation of path verification protocol. However, for higher values of b, we expect our protocol to take much shorter time compared to path verification protocols. Our protocol requires only about $p+1$ MAC operations at each server for an update in the whole of an update's dissemination time. Path verification protocols require $O(b^{b+1})$ [4] computation time at each server every round. This is because path-verification protocols involve checking for $b+1$ non-intersecting paths from a set of paths, which is known to be NP-complete.

5 Collective Endorsement of Authorization Tokens

In the dissemination protocol, a server needs to determine if a client that wishes to submit an update is authorized to do so. In the secure store we build (figure 1), metadata service maintains access control lists. Metadata service is a threshold service with the non-faulty servers replicating ACLs. File system clients get an authorization token from the metadata service to access file data from data servers. The authorization token issued must be unforgeable and verifiable by every data server.

With reference to our key allocation scheme, every metadata server is allocated keys along vertical straight lines $j = constant, i = 0\ to\ p-1$ from the first set of p^2 keys $\{k_{i,j}\}$. We do not need the other p keys k'_i . Prime p is chosen to be greater than the number of metadata servers, which is at least $3b+1$. Data servers are allocated keys along other straight lines, but not vertical ones.

Before issuing an authorization token, each metadata server refers to its copy of ACLs to see if an access is allowed. After checking access, each non-faulty metadata server endorses the same authorization token with a list of MACs computed using the set of symmetric keys it has. The file system client collects all such MACs from every metadata server. The list of all such MACs constitutes a valid endorsement that will be accepted by any data server.

Total number of keys is about the number of servers in the system. Hence, the length of an endorsement that consists of all MACs is $O(n)$. Still, we believe size is reasonable for a moderate number of servers and is justified for the amount of savings in computation when compared to public key schemes. Total size of the endorsement can be reduced by reducing the size of each MAC [13, 1], trading off security against forgeability for size. Furthermore, complete list of MACs need not be sent to every data server. For a chosen data server, appropriate MACs alone can be sent. However, as the number of servers in the system increases, public key signatures and threshold public key schemes become more efficient than collective endorsements.

6 Related work

Dissemination protocols have been extensively studied in benign environments in the past [8, 7, 9, 10, 11]. Malkhi, Mansour and Reiter [2] were the first to consider dissemination in malicious environments without using public key signatures. In [3], Malkhi et. al. presented a class of protocols that took advantage of a well defined logical structure of the system. In all these earlier protocols, a server accepts an update only if $b+1$ other servers inform the server that they have accepted. These protocols are conservative in nature, where a participating server cannot help in dissemination until it accepts the update.

Malkhi and others [5] and Minsky and Schneider [4] independently suggested a class of gossip protocols where a server accepts an update if it receives the same update via $b+1$ non-intersecting paths. Diffusion times for these protocols were $O(log\ n + b)$ and $O(log\ n) + b$ respectively. Message size and buffer requirements for Minsky's protocol were particularly low. Both these protocols and the earlier ones [3, 2] do not rely on any cryptographic primitive and are information theoretically secure as opposed to our protocol which uses MACs.

Schemes allocating sets of symmetric keys to participating nodes have been used in multicast key management applications [12, 14]. Naor and others also suggested using a list of MACs to replace public key signatures in [13]. They used a key allocation technique which gave a probabilistic guarantee against forgery by a limited coalition of malicious servers. Liskov and Castro [1] eliminated public keys in Byzantine fault tolerance by replacing signatures with a vector of MACs. An exclusive symmetric key was shared between every pair of servers. This pairwise key sharing can be looked at as a special case of the key allocation scheme we presented here, when b and n are of same order and the chosen prime p is about n.

In contrast to earlier published gossip protocols, our protocol uses a new technique called collective endorsement that can be used in other applications as well. Although our protocol has higher bandwidth and buffering requirements, it offers a desirable feature that the diffusion time increases only when there are actual faults. In normal case, when there are no or few faults, its diffusion time is comparable to a system with benign faults. In other protocols, the diffusion time depends on b which is the threshold for worst-possible number of faults.

7 Conclusion

This paper presented a gossip style protocol that disseminates an update in $O(log\ n) + f$ rounds in the presence of f faults. The diffusion time does not depend on b, the assumed maximum number of faults and when there are no faults, the diffusion time is comparable to that of best-possible benign case protocol. Our protocol trades off bandwidth and memory resources to achieve lower latency. A novel key allocation scheme is used to keep message and buffer sizes low. We implemented our protocol and presented simulation and experimental results validating our claims. We have derived an analytical upper bound of $4b+3$ for the quorum size required for a random choice of initial quorum. We have also shown theoretically that each MAC takes $O(log\ n)+f$ rounds to reach a constant fraction of servers, justifying our claim on dissemination time.

We are exploring using higher degree polynomials for key allocation in our key allocation scheme. For small values of b, the total number of keys can be reduced to a large extent by using higher degree polynomials. However, the size of initial quorum for higher degree polynomials is an issue that needs to be addressed. Tightening the analytical upper bound for initial quorum size and reducing the message and buffer sizes without increasing the diffusion time are some interesting problems to be explored in future.

References

[1] Liskov B., Castro M., "Authenticated Byzantine Fault Tolerance Without Public-Key Cryptography", Technical Report /LCS/TM-595, MIT, 1999.

[2] Malkhi D., Mansour Y., Reiter M., "On diffusing Updates in Byzantine Environment", *Proc. 18th IEEE Symp. on Reliable Distributed Systems*, Lausanne, 1999.

[3] Malkhi D., Reiter M., Rodeh O., Sella Y., “Efficient Update Diffusion in Byzantine Environments”, *Proceedings of the 20th IEEE Symposium on Reliable Distributed Systems*, pages 90-98, New Orleans, USA, 2001.

[4] Minsky Y., Schneider F., “Tolerating Malicious Gossip”, *The Distributed Computing Journal*, 16(1):49-68, February 2003.

[5] Malkhi D., Pavlov E., Sella Y., “Optimal Unconditional Information Diffusion”, *Proceedings of the 15th International Symposium on Distributed Computing*, pages 63-77, Lisbon, Portugal, 2001.

[6] Thomas Y.C. Woo, Simon S. Lam, “A Framework for Distributed Authorization”, *Proc. ACM Conference on Computer and Communications Security*, Fairfax, Virginia, November 1993.

[7] Demers A., Greene D., Hauser C., Irish W., Larson J., Shenker S., Sturgis H., Swinehart D., Terry D., “Epidemic Algorithms for replicated database maintenance”, *Operating Systems Review*, vol. 22, 1988, pp. 8-32.

[8] Petersen K., Spreitzer M.J., Terry D., Theimer M., Demers A., “Flexible Update Propagation for Weakly Consistent Replication”, *Proc. of ACM SOSP*, pages 288 - 301, October 1997.

[9] Golding R. A., “Weak-Consistency Group Communication and Membership”, PhD thesis, UC Santa Cruz, December 1992, published as technical report UCSC-CRL92 -52.

[10] Renesse R., “Scalable and secure resource location”, *Proceedings of the Hawaii International Conference on System Sciences*, January 4-7, 2000.

[11] Ozkasap O., Renesse R., Birman K., Xiao Z., “Efficient buffering in reliable multicast protocols”, *Proceedings of the First International Workshop on Networked Group Communication*, pages 188-203, Pisa, Italy, November 1999. Springer-Verlag.

[12] C. K. Wong, M. Gouda and S. Lam, “Secure Group Communication Using Key Graphs”, *Proceedings of ACM SIGCOMM*, Vancouver, British Columbia, September 1998.

[13] Canetti R., Garay J., Itkis G., Micciancio D., Naor M, Pinkas B., “Multicast Security: A Taxonomy and Some Efficient Constructions”, *Proceedings of INFOCOM*, New York, March 1999.

[14] Fiat A., Naor M., “Broadcast Encryption”, *Proc. Crypto ’93*, pp. 480-491.

[15] Lamport L., Shostak R., Pease M., “The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[16] Carlo Blundo, Alfredo De Santis, Amir Herzberg, Shay Kutten, Ugo Vaccaro, and Moti Yung. “Perfectly Secure Key Distribution for Dynamic Conferences”,*Information and Computation*, 146(1):1-23, 10 October 1998.

[17] Chan H., Perrig A., and Song D., ”Random key predistribution schemes for sensor networks.” *IEEE Symposium on Security and Privacy*, pp 197-213, Berkeley, California, May 11-14 2003.

APPENDIX

A All servers accept an update in two phases when the initial quorum size $q \ \geq \ 4b+3$.

In the following, we prove that if $p \ \geq \ q \ \geq \ 4b+3$, all servers will accept an update in two phases of MAC generation for any random choice of initial quorum of size q.

Model : Let p be a prime. In the field Z_p define a straight line $L = (\alpha, \beta)$ as the set of points (i, j) such that $i = \alpha j + \beta$. Any such line has p points. Define the universal set of lines U to be the set of all possible lines, $0 \leq \alpha \leq p-1, 0 \leq \beta \leq p-1$. Define two integer constants b and q such that $p >= q >= 4b+3$.

For any two lines L_1 and L_2, we define their intersection to be the point that is on both lines. If two lines are parallel, their α being the same, we define their intersection to be a special point at infinity along the direction of the two lines. For a line L and a set of lines S, we define the intersection of L and S as the union of points of intersection between L and every line in S.

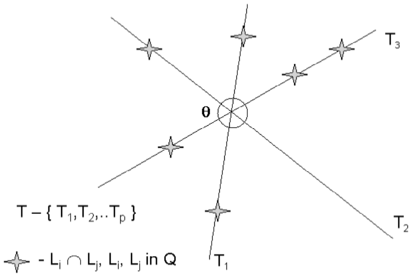
For a set of lines S, we define $\mathcal{D}(S)$ as the set of all lines L in U such that $\mid L \cap S \mid$ is at least $2b+1$. S is contained in $\mathcal{D}(S)$.

Claim 1 : If $p >= q >= 4b+3$, for a randomly chosen set of lines Q of size q, $U \ = \ \mathcal{D}(\mathcal{D}(Q))$.

Before we prove claim 1, we will prove the following claim.

Claim 2 : Let θ be a point through which none of the lines in Q pass. Then there exists a line L passing through θ that is contained in $\mathcal{D}(Q)$.

Proof for claim 2 :



Define $\mathcal{C}$ to be a multiset of points containing all intersection points between every pair of lines in Q. Some of these intersection points may be the same, if more than two lines in Q are concurrent. We will however consider these as different elements in $\mathcal{C}$. Thus $\mathcal{C}$ has exactly $\binom{q}{2}$ points. Since θ is not on any line in Q, $\theta \notin \mathcal{C}$.

Define T to be the set of lines that pass through the point θ. Size of T is p. Since none of the lines in Q pass through θ, Q and T do not overlap.

We claim that there exists at least one line L in T such that not more than $\binom{q}{2}/p$ points in $\mathcal{C}$ are on L. This is because, even assuming the worst case that all points in $\mathcal{C}$ are on some line of T (none of them can be on more than one line in T), there are $\binom{q}{2}$ points distributed among p lines. At least one line in T should get fewer or same as the average number of points per line.

Let us get a lower bound on the number of points L shares with Q. L shares q points with Q. Some of these may be the same (concurrent points) and one of them could be the point at infinity. Concurrent points are those that belong in $\mathcal{C}$. Subtracting one from q for every concurrent point, the distinct number of points shared between L and Q should be at least $q - \binom{q}{2}/p$. This expression is at least $2b+2$ if $p >= q >= 4b+3$. Even if one point is the point at infinity, we have $2b+1$ distinct and valid points shared between L and Q. Thus L is contained in $\mathcal{D}(Q)$.

Proof for claim 1 : From claim 2, it follows that if a point does not belong to any line in Q, then it belongs to some line in $\mathcal{D}(Q)$. Thus, for any line L, the number of points of intersection between L and $\mathcal{D}(Q)$ is p which is greater than $2b+1$. This proves that every line in U belongs to $\mathcal{D}(\mathcal{D}(Q))$.

B A valid MAC takes $O(log\ N) + f$ rounds to reach a constant fraction of servers

Model : Among a group of N servers, f servers are malicious. G of the N servers share a common symmetric key k, not overlapping with the f malicious servers. In round 0, one of the G servers starts with an update (that no other server knows), calculates digest(update) and computes MAC(digest,timestamp,k). In synchronous rounds of gossip, each server chooses a gossip partner for each round and pulls (1)digest of the update, (2)timestamp of the update and (3)MAC of the update. The f faulty servers always send a spurious MAC for the update to any requesting server. Servers other than the f faulty ones that do not have the key to verify the MAC ($N - G - f$ servers) will simply accept any incoming MAC with a timestamp in the past and store it in its buffer, to be sent to other servers in future rounds of gossip. On the other hand, G servers that have the key k reject an update if either timestamp is in the future or if MAC is not valid.

Assumption : For the sake of this proof, we assume that all servers have their clocks perfectly synchronized and make their gossip at the same time. Synchrony assumption prevents the faulty servers from spreading spurious MACs for an update even before the actual source gets a chance to disseminate.

Claim : In $O(log\ N) + O(f)$ rounds, a constant fraction of the G servers that have the key get the valid MAC that the source started with in round 0.

Proof :

We will divide the set of N servers into three groups : (1) Group A containing the G servers that know the key k, (2) Group B containing f faulty servers and (3) Group C containing the remaining $N - G - f$ servers. With some misuse of notation, we will use C to also denote the constant $N - G - f$, the number of servers in group C.

Notations :

- $b[r]$ - the number of servers in C in r^{th} round that have a spurious MAC. (b - bad)
- $l[r]$ - the number of servers in C in r^{th} round that have a valid MAC. (l - lucky)
- $g[r]$ - the number of servers in A in r^{th} round that have a valid MAC. (g - good)

Observations :

1. $g[r] \ \geq \ 1$ for all r.
2. A server in C which has a valid MAC in r^{th} round will continue to have a valid MAC in $(r+1)^{st}$ round unless the server chooses as its gossip partner for round $r+1$ a server that has a spurious MAC in that round. In the latter case, the server accepts a spurious MAC in that round.
3. A server in C which has a spurious MAC in r^{th} round will continue to have a spurious MAC in $(r+1)^{st}$ round unless the server chooses as its gossip partner for round $r+1$ a server that has a valid MAC in that round. In the latter case, the server accepts a valid MAC in that round.

Last two points can be summarized in the following equations characterizing the expected growth of $b[r]$ and $l[r]$:

$$l[r+1] = l[r] * (1 - \frac{b[r]+f}{N}) + (C - l[r]) * \frac{l[r]+g[r]}{N}. \tag{1}$$

$$b[r+1] = b[r] * (1 - \frac{l[r]+g[r]}{N}) \ + \ (C - b[r]) * \frac{b[r]+f}{N}. \tag{2}$$

We will replace $g[r]$ in equations 1 and 2 by the constant 1. Since $g[r]$ is always at least 1, this would give us a lower bound on the expected value for $l[r]$ and an upper bound on the expected value of $b[r]$.

$$l[r+1] \ = \ l[n] * (1 - \frac{b[r]+f}{N}) \ + \ (C - l[r]) * \frac{l[r]+1}{N}. \tag{3}$$

$$b[r+1] = b[r] * (1 - \frac{l[r]+1}{N}) \ + \ (C - b[r]) * \frac{b[r]+f}{N}. \tag{4}$$

From equations 3 and 4, if $l[r]/b[r] \ = \ 1/f$, it can be shown that $l[r+1]/b[r+1] \ = \ 1/f$. Since $l[0]/b[0] \ = \ 1/f$, by induction,

$$\frac{l[r]}{b[r]} \ = \ \frac{1}{f}, \ \forall r. \tag{5}$$

The system reaches a state of dynamic equilibrium when when $l[r] = C/(f+1)$ and $b[r] = f*C/(f+1)$, and $l[r]$ and $b[r]$ remain the same for all future rounds.

Let's denote by $T[r]$, the expected number of servers in round r that have some MAC, either valid or spurious. $T[r]$ is the sum of $b[r]$,$l[r]$,f and $g[r]$ which we will approximate to one. By adding equations 3 and 4 and adding $f+1$ on either side, it can be seen that

$$T[r] = T[r] * (1 + C/N - \frac{T[r]}{N}).$$

Approximating C/N to one, this equation can be rewritten as

$$T[r+1] = T[r] * (2 - \frac{T[r]}{N}). \tag{6}$$

Equation 6 is a typical equation that characterizes the traditional pull-based gossip protocol for benign settings. It can be shown that

$$1 - (\frac{T[r]}{N}) \ = \ (1 - \frac{T[0]}{N})^{2^r}.$$

Thus, after $O(log\ N)$ rounds, a constant fraction of the N servers would have some MAC or the other, 1/(f+1) of which will have valid MACs.

Let us assume a minimum of $\alpha * (1/(f+1)) * C$ servers in group C have valid MACs after $O(log\ N)$ rounds, for some constant α. Let's look at the growth of $g[r]$ from this point on.

Growth of g[n] is characterized by the following equation.

$$g[r+1] = g[r] \ + \ (G - g[r]) * (\frac{l[r] + g[r]}{N}). \tag{7}$$

Assuming $G, f << N$, we can approximate $(l[r] + g[r])N$ to $l[r]/C$ which is $\alpha/(f+1)$. Equation 7 becomes

$$g[r+1] = g[r] \ + \ (G - g[r]) * (\frac{\alpha}{f+1}).$$

or

$$1 - (\frac{g[r+1]}{G}) \ = \ (1 - \frac{g[r]}{G}) * (1 - \frac{\alpha}{f+1}).$$

For a constant fraction β among the G servers to get the valid MAC, it takes $O(log\ N) + k$ rounds where k is given by

$$(1 - \frac{1}{G}) * (1 - \frac{\alpha}{f+1})^k = 1 - \beta.$$

Taking log on both sides and approximating $log(1 - \alpha/(f+1))$ to $-\alpha/(f+1)$, we get

$$k = \frac{f+1}{\alpha} \ * \ (log(1 - 1/G) - log(1 - \beta)). \tag{8}$$

The constant β can be adjusted to yield a constant factor of 1 for $O(f)$ on the right hand side. The β thus obtained would be sufficiently large to guarantee that all servers will accept an update in $O(log\ N) + f$ rounds. This is because each MAC does not have to be disseminated to all the servers. A server would accept an update even if it misses some MACs provided it receives $b + 1$ other valid MACs. This justifies our claim that each MAC takes $O(log\ N) + f$ rounds to reach a constant fraction of servers.