Secure Reliable Multicast Protocols in a WAN[*]

Dahlia Malkhi Michael Merritt Ohad Rodeh

AT&T Labs, Murray Hill, New Jersey
{dalia,mischu}@research.att.com

The Hebrew University of Jerusalem
tern@cs.huji.ac.il

abstract

A secure reliable multicast protocol enables a process to send a message to a group of recipients such that all honest destinations receive the same message, despite the malicious efforts of fewer than a third of them, including the sender. This has been shown to be a useful tool in building secure distributed services, albeit with a cost that typically grows linearly with the size of the system. For very large networks, for which such a cost may be too prohibitive, we present two approaches for bringing the cost down: First, we show a protocol whose cost is on the order of the number of tolerated failures. Secondly, we show how relaxing the consistency requirement to a selected probability level of guarantee can bring down the associated cost to a constant.

1 Introduction

Communication over a large and sparse internet is a challenging problem because communication links experience diverse delays and failures. Moreover, in a wide area network (WAN), security is most needed, since communicating parties are geographically dispersed and thus are more prone to attacks. The problem addressed in this paper is how to distribute messages among a large group of participants so that all the (correctly behaving) participants agree on messages' contents, despite the malicious cooperation of up to a third of the members.

Previous work on this problem suffers from message complexity and computation cost that do not scale to very large communication groups: Toueg's echo_broadcast [11] requires $O(n^2)$ authenticated message exchanges for each message delivery (where n is the size of the group). Reiter improved this message complexity in the ECHO protocol of the Rampart system [8] through the usage of digital signatures. The ECHO protocol incurs $O(n)$ *signed* message exchanges, and thus, message complexity is improved at the expense of increased computation cost. Malkhi and Reiter [6] extended this approach to amortize the cost of computing digital signatures over multiple messages through a technique called *acknowledgment chaining*, where a signed acknowledgment directly verifies the message it acknowledges and *idirectly*, every message it acknowledges. Nevertheless, $O(n)$ digital signatures sit in the critical path between message sending and its delivery. For a very large group of hundreds or thousands of members, this may be too prohibitive.

In this paper, we propose two approaches for bringing down the cost and delay associated with reliable broadcast. First, we show a protocol whose cost is on the order of the number of tolerated failures. Secondly, we show how relaxing the consistency requirement to a selected probability level of guarantee can bring down the associated cost to a constant. The principle underlying agreement on message contents in previous works, as well as ours, is the following: For a process p to send a message m to the group of processes, authentic validations are obtained for m from a certain set of processes. We call this validation set the *witness set* of m, denoted *witness*(m). Witness sets are chosen so that any pair of them intersect at an honest process, and such that some witness set is always accessible despite failures. More precisely, witness sets satisfy the *Consistency* and *Availability* requirements of *Byzantine dissemination quorum systems* (cf. [5]), as follows:

Definition 1.1 *A dissemination quorum system is a set of subsets, called quorums, satisfying:*

[*]Copyright 1997 IEEE. Published in the Proceedings of ICDCS'97, May 27-30, 1997 in Baltimore, Maryland. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE.

*For every set B of corrupt processes, and every two quorums Q_1, Q_2, $Q_1 \cap Q_2 \not\subseteq B$ (***Consistency***).*
*For every set B of corrupt processes, there exists a quorum Q such that $Q \subseteq \overline{B}$ (***Availability***).*

Using dissemination quorums as witness sets for messages, if a corrupt process generates two messages m, m' with the same sequence number and different contents (called *conflicting messages*), the corresponding witness sets $witness(m)$, $witness(m')$ intersect at an honest process (by Consistency). Thus, they cannot both obtain validations, and at most one of them will be delivered by the honest processes of the system. Further, dissemination quorums ensure availability, such that an honest process can always obtain validation from a witness set despite possible failures. For a resiliency threshold $t < \lfloor (n-1)/3 \rfloor$, previous works used quorums of size $\lceil (n+t+1)/2 \rceil$, providing both consistency and availability.

Our first improvement in the *3T* protocol drops the quorum size from $\lceil (n+t+1)/2 \rceil$ to $2t+1$. When t is a small constant, this improvement is substantial, since we need only wait for $O(t)$ processes, no matter how big the WAN might be. Briefly, the trick in bringing down the size of the witness sets is in designating for every message m a witness set of size $3t+1$, determined by its sender and sequence number. A message must get $2t+1$ acknowledgments, out of the designated set of $3t+1$, in order to be delivered.

Our second improvement stems from relaxing the consistency requirement to a *probabilistic* requirement. This leads to a protocol that consists of two-regimes: The first one, called the *no-failure* regime, is applied in faultless scenarios. It is very efficient, incurring only a **constant** overhead in message exchanges and signature computing. The second regime is the *recovery* regime, and is resorted to in case of failures. The two regimes inter-operate by having the witnesses of the no-failure regime actively probe the system to detect conflicting messages. We introduce a probabilistic protocol combining both regimes, $active_t$, whose properties are as follows:

- For any desired guarantee level $\epsilon > 0$, and given a resiliency threshold t, $active_t$ can be tuned to guarantee agreement on messages contents by all of the honest processes with probability greater than $1 - \epsilon$, for appropriate system sizes. This probability is taken, at the limit, over the set of all potential messages exchanged in the protocol.

- The overhead of forming agreement on messages contents in $active_t$ in faultless circumstances is determined by two constants that depend on ϵ only (and not on the system size or t).

In this paper, we assume a static set of communicating processes. It is possible, however, to use known techniques (e.g., [8]) to extend our protocol to operate in a dynamic environment in which processes may leave or join the set of destination processes and in which processes may fail and recover.

The rest of this paper is organized as follows: In section 2 we formally present our assumptions about the system. Section 3 presents a precise problem definition, and shows a simple solution, which will be employed later. Section 4 contains the *3T* protocol description, and Section 5 contains the $active_t$ protocol description. We conclude in section 6.

2 Model

The system contains n participating processes, denoted $P = \{p_1, p_2, \ldots, p_n\}$. Processes interact solely through message passing. There is no limit on the relative speeds of different processes or a known upper bound on message transmission delays; hence, the system is asynchronous. Every pair of processes is connected via an authenticated FIFO channel, that guarantees the identity of senders using any one of well known cryptographic techniques.

The system may admit failures of up to $t \leq \lfloor (n-1)/3 \rfloor$ processes, where faulty processes may deviate from the behavior dictated by the protocol in an arbitrary way, subject to cryptographic assumptions. That is, faulty processes may share encryption and signature keys and send arbitrary messages (or no messages at all) constructed using these keys, but may not forge digital signatures or crypt-analyze messages between other processes. (Such failures are called authenticated *Byzantine* failures.) Live processes that follow the protocol correctly are called *honest*, whereas processes that exhibit any form of failure are termed *corrupt*.

We assume that every process possesses a private key, known only to itself, that may be used for signing data using a known public key cryptographic method (such as [10]). Let d be any data block. We denote $K_i(d)$ the signature of p_i on the data block d by means of p_i's private key. We assume that every process in the system may obtain the public keys of all of the other processes, such that it can verify the authenticity of signatures. Our protocols also make use of a cryptographically secure hash function H (such as MD5 [9]).

(In the following, we make the cryptographic assumption that $H(m) = H(m')$ implies that $m = m'$).

In practice, one way by which multiple processes may appear to coordinate a malicious attack is if the private keys of some processes are compromised by an intruder, who can then impersonate them and deviate from the protocol on their behalf. The security of our most efficient protocol assumes that during such an attack, the intruder is not able to capture or modify the communication among honest processes in any way, nor to choose the targets of its attack in a way that depends on the protocol execution.

More formally, a system that admits Byzantine failures is often abstracted as having an *adversary* working against the successful execution of protocols. We assume the following limitations on the adversary's powers: The adversary chooses which processes are corrupt at the beginning of the execution, and thus the choice of corrupt processes is non-adaptive. Corrupt processes cannot access the local memories of the honest processes, nor break their private keys, and thus cannot obtain data internal to computations in the protocols. Moreover, we assume that every message sent between two honest processes has a certain positive probability of reaching its destination within some known timeout period (thus, network failures are not adaptive to the protocol execution, either). We note that the last assumption is required only for the $active_t$ protocol.

3 The Problem Definition

A reliable multicast protocol provides each process p with two operations:

WAN-multicast(**m**): p sends the message m to the group.

WAN-deliver(**m**): p delivers a message m.

For convenience, we assume that a message m contains several fields:

sender(m): The identity of the sending process.

seq(m): A count of the messages originated by sender.

payload(m): The (opaque) data of the message.

The protocol should guarantee that all of the honest members of the group agree on the delivered messages, and furthermore, that messages sent by honest processes are (eventually) delivered by all of the honest processes. More precisely, the protocol should maintain the following properties:

Integrity: Let p be an honest process. Then p performs *WAN-deliver*(m) at most once, and if *sender*(m) is honest, then only if *sender*(m) executed *WAN-multicast*(m).

Self-delivery: Let p be an honest process. If p executes *WAN-multicast*(m) then eventually p delivers m, *i.e.*, eventually p executes *WAN-deliver*(m).

Reliability: Let p_i and p_j be two honest processes. If p_i delivers a message m from r (via *WAN-deliver*(m)), then (eventually) p_j delivers m.

(Probabilistic) Agreement: Let p_i and p_j be two honest processes. Let p_i deliver a message m, p_j deliver m', such that *sender*(m) = *sender*(m') and *seq*(m) = *seq*(m'). Then (with very high probability) p_i and p_j delivered the same message, *i.e.*, *payload*(m) = *payload*(m'). (Probability is taken assuming uniform choice of *seq*(m), and taken at the limit as *seq*(m) goes to infinity.)

The problem statement above is strictly weaker than the Byzantine agreement problem [4], which is known to be unsolvable in asynchronous systems [2]. This statement holds even if we use the unconditional Agreement requirement. The reason is that only messages from honest processes are required to be delivered, and thus messages from corrupt processes can "hang" forever. Note that there is no ordering requirement among different messages, and thus the problem statement is weaker than the totally ordered reliable broadcast problem, which can be solved only probabilistically [7]. The reliable multicast problem is solvable in our environment, as is demonstrated by the E protocol depicted in Figure 1 (which borrows from the Rampart ECHO multicast protocol [8]).

It is easy to verify that the echo protocol satisfies Integrity, Self-delivery, Reliability and Agreement. This protocol is resorted to when the $active_t$ protocol below fails to guarantee Reliability. However, it is inefficient in faultless runs, and incurs an intolerable overhead for very large groups. The protocols below attempt to minimize the overhead in faultless scenarios.

4 The 3T Protocol

In this section we introduce the *3T* protocol. The improvement in this protocol over the E protocol

1. For a process p_i to *WAN-multicast* m, process p_i sends $\texttt{<E}, \texttt{regular}, m\texttt{>}$ to P to obtain a set of $\lceil (n+t+1)/2 \rceil$ signed acknowledgments $A = \{\texttt{<E}, \texttt{ack}, K_j(m)\texttt{>}\}$. It then sends $\texttt{<E}, \texttt{deliver}, m, A\texttt{>}$ to P.

2. When p_i receives a message $M = \texttt{<E}, \texttt{regular}, m\texttt{>}$ from p_j, and no conflicting message was previously received, p_i sends a signed acknowledgment $\texttt{<E}, \texttt{ack}, K_i(m)\texttt{>}$ back to p_j.

3. When p_i receives a message $M = \texttt{<E}, \texttt{deliver}, m, A\texttt{>}$, such that A contains a valid set of acknowledgments for m, and such that m was not previously delivered, p_i performs *WAN-deliver*(m), and forwards M to P.

Figure 1: The E protocol

above comes from computing witness sets for messages based on their identification, i.e., on the pair $<$*sender id*, *sequence*$>$. The protocol makes use of a deterministic $(3t+1)$-valued function $I(p_i, k)$, whose range is the set of $(3t+1)$-tuples of distinct process id's, to determine the witness set for p_i's k'th message. Thus, $\mathit{witness}(m) = I(\mathit{sender}(m), \mathit{seq}(m))$. For efficiency, I could be chosen to distribute the load of witnessing over distinct sets of processes for different messages. Figure 2 provides the details of the *3T* protocol.

The propositions below maintain that the *3T* protocol maintains Integrity, Self Delivery, Reliability and Agreement. The overhead incurred is $3t+1$ signature generations and message exchanges per delivery.

Theorem 4.1 *Let p_i be an honest process participating in the 3T protocol. Then p_i executes WAN-deliver(m) at most once, and if* sender(m) *is honest, then only if* sender(m) *executed WAN-multicast*(m)*.*

Proof : That p_i supresses duplicate deliveries is immediate from the protocol. It is left to show that if *sender*(m) is honest, it must have sent m. This follows from the fact that for p_i to deliver m, p_i must have obtained a set A of valid acknowledgments for m from $2t+1$ processes in *witness*(m), (at least) one of which is honest, and therefore must have received m directly from its sender. $\square$

Theorem 4.2 *Let p_i be an honest process participating in the 3T protocol. If p_i executes WAN-*

1. For a process p_i to perform *WAN-multicast*(m), p_i sends $\texttt{<3T}, \texttt{echo}, p_i, \mathit{seq}(m), H(m)\texttt{>}$ to each $p_j \in \mathit{witness}(m)$ to obtain a set of $2t+1$ signed acknowledgments $A = \{\texttt{<3T}, \texttt{ack}, K_j(H(m))\texttt{>}\}$. (The choice of the $2t+1$ threshold is significant, since it guarantees that a majority of the honest members of *witness*(m) acknowledge the message m and thus ascertains its uniqueness.) Process p_i then sends $\texttt{<3T}, \texttt{deliver}, m, A\texttt{>}$ to P.

2. When p_i receives a message $\texttt{<3T}, \texttt{echo}, p_j, \mathit{cnt}, h\texttt{>}$ from p_j, such that no conflicting message was previously received, p_i sends a signed acknowledgment $\texttt{<3T}, \texttt{ack}, K_i(h)\texttt{>}$ to p_j.

3. When p_i receives a message $\texttt{<3T}, \texttt{deliver}, m, A\texttt{>}$, such that A contains valid signatures from $2t+1$ members in *witness*(m), and such that m was not previously delivered, p_i performs *WAN-deliver*(m). If a timeout period has passed and p_j is not known to have delivered m, p_i sends $\texttt{<3T}, \texttt{deliver}, m, A\texttt{>}$ to p_j.

Figure 2: The *3T* protocol

multicast(m) then p_i executes WAN-deliver(m).

Proof : According to the 3T protocol, if p_i sends $\texttt{<3T}, \texttt{echo}, p_i, \mathit{seq}(m), H(m)\texttt{>}$ to the $3t+1$ processes in *witness*(m), then at least $2t+1$ honest processes p_j among them will send back $\texttt{<3T}, \texttt{ack}, K_j(H(m))\texttt{>}$ messages to p_j, thereby enabling delivery of m by p_i. $\square$

Theorem 4.3 *Let p_i and p_j be two honest processes participating in the 3T protocol. If p_i performs WAN-deliver(m), then p_j performs WAN-deliver(m).*

Proof : For p_i to deliver m, p_i must have obtained a set A of valid acknowledgments for m from $2t+1$ processes in *witness*(m). If p_i does not learn that p_j delivered m within some timeout period, then p_i sends $\texttt{<3T}, \texttt{deliver}, m, A\texttt{>}$ to p_j, causing p_j to perform *WAN-deliver*(m). $\square$

Theorem 4.4 *Let p_i and p_j be two honest processes participating in the 3T protocol. Let p_i deliver a message m, p_j deliver m', such that* sender$(m) =$ sender(m') *and* seq$(m) =$ seq(m')*. Then p_i and p_j delivered the same message,* i.e., *payload(m) = payload(m').*

Proof : For p_i to deliver m, p_i must have obtained a set A of valid acknowledgments for m from $2t+1$ processes in *witness*(m). Likewise, p_j must have obtained a set A' of $2t+1$ valid acknowledgments for m'. By assumption, *witness*(m) = *witness*(m'). Therefore, the sets of processes represented in A, A' intersect in at least $t+1$ processes, at least one of which must be honest. Since honest processes never acknowledge conflicting messages, we have that $H(m) = H(m')$, and (by the cryptographic assumption,) *payload*(m) = *payload*(m'). □

5 The *active*$_t$ Protocol

In this section, we relax our requirements and provide a protocol that guarantees (only) Probabilistic Agreement. Thus, we allow the possibility that some selected small fraction of all attempts to send conflicting messages to the system will succeed.

The idea of the *active*$_t$ protocol is as follows: The witness set *witness*(m) for a message m is a set of k processes, chosen uniformly from P by a pseudo-random function of the pair <*sender*(m), *seq*(m)>. The size of a witness set, k, is set so that only a vanishingly small fraction of the messages can have a witness set that contains only corrupt processes (who may be collaborating with the sender), as the number of messages (and hence *seq(m)*) goes to infinity. If only $t \leq \lfloor (n-1)/3 \rfloor$ members are corrupt, then because the pseudo-random function is effectively uniform, the expected fraction of messages with a 'corrupt' witness set is $\left(\frac{t}{n}\right)^k \leq \left(\frac{1}{3}\right)^k$. Relatively small values of k are sufficient for this to be negligable. (As with the size of cryptographic keys, k is effectively a constant.) This idea of forming distributed trust in a cooperation-resilient way borrows from the time-stamping mechanism of Haber et al. [3].

The motivation for this protocol is to choose witness sets significantly smaller than $3t+1$. As a result, assuring both safety and availability is a problem: Since t of the witnesses could be faulty, for availability one might want to wait for only $k-t$ replies. But this would be too few replies to assure safety: a faulty sender and faulty witnesses could force different messages to be delivered. Therefore, to guarantee availability of a witness set for every message, *active*$_t$ incorporates the E protocol as a recovery-regime. This is done as follows: For a process to send a message, it attempts to obtain signed acknowledgments from *witness*(m). We name this the *no-failure* regime. After a timeout period, if p_i has not obtained acknowledgments from all the members of *witness*(m), p_i reverts to the E protocol and re-sends m to P to obtain signed acknowledgments from any set of $\lceil (n+t+1)/2 \rceil$ processes. This is called the *recovery* regime.

The integration of the two protocols can potentially create an opportunity for a corrupt process to obtain signed acknowledgments for conflicting messages, since a carefully chosen set of $\lceil (n+t+1)/2 \rceil$ processes may not intersect *witness*(m). To decrease the possibility of delivering conflicting messages in combining the two regimes, we turn the witnesses of the no-failure regime into more active participants. The (honest) members of a witness set probe the system at ℓ randomly chosen *peer* processes before acknowledging a message, in order to detect the possible existence of a conflicting message. In order to allow witnesses to probe their peers on behalf of *sender*(m), we require every process to sign its own messages, where a signature on a message m, denoted *sign*(m), is a part of m and authenticates the entire contents of m. (The peer processes record the message and do not reply if it conflicts with a previous message, but do not need to sign the reply–hence knowledge of the message propagates randomly among honest processes, without incurring additional signature overhead.) In this way, if a message m' conflicting with m has been sent to a set S in the recovery-regime, then with high probability the *peers* chosen on behalf of m intersect S at an honest process. More precisely, the probability that ℓ random probes cross an honest member of a recovery set containing $\lceil (n+t+1)/2 \rceil$ processes is at least $1 - \left(\frac{n+t-1}{2n}\right)^\ell \geq 1 - \left(\frac{2}{3}\right)^\ell$. Therefore, the parameter ℓ can be chosen to achieve the desired level of probabilistic guarantee.

The details of the protocol are given in Figure 3.

---

1. For a process p_i to *WAN-multicast*(m), p_i sends <`AV`, `echo`, p_i, *seq*(m), *sign*(m), $H(m)$> to each $r \in$ *witness*(m) to obtain the set of k acknowledgments $A = \{$<`AV`, `ack`, $K_r(H(m))$>$\}$, from every $p_k \in$ *witness*(m).

   If a timeout period has passed and p_i does not obtain some acknowledgments in A, then p_i sends <`E`, `echo`, p_i, *seq*(m), $H(m)$> to P to obtain a set of $\lceil (n+t+1)/2 \rceil$ acknowledgments $A = \{$<`E`, `ack`, $K_j(H(m))$>$\}$.

   In either case, p_i then sends <`AV`, `deliver`, m, A> to P.

2. When p_i receives a message <`AV`, `echo`, p_j, *cnt*, *sign*, *h*>, where *sign* is a valid signa-

ture, it performs the active phase of secure message transmission: If no conflicting message was previously received, p_i randomly selects ℓ target processes, denoted peers_i. It sends $\texttt{<AV}, \texttt{inform}, p_j, \mathit{cnt}, \mathit{sign}, h\texttt{>}$ to every $p_k \in \mathit{peers}_i$ to obtain a message $\texttt{<AV}, \texttt{verify}, p_j, \mathit{cnt}, h\texttt{>}$ from p_k. It then sends a signed acknowledgment $\texttt{<AV}, \texttt{ack}, K_i(h)\texttt{>}$ to p_j. Note that p_i does not send back to p_j any information about peers_i.

3. When p_i receives a message $\texttt{<AV}, \texttt{inform}, p_j, \mathit{cnt}, \mathit{sign}, h\texttt{>}$ from p_k, such that no conflicting message was previously received, it sends $\texttt{<AV}, \texttt{verify}, p_j, \mathit{cnt}, h\texttt{>}$ to p_k.

4. When p_i receives a message $\texttt{<E}, \texttt{echo}, p_j, \mathit{cnt}, h\texttt{>}$ from p_j, such that no conflicting message was previously received, it sends a signed acknowledgment $\texttt{<E}, \texttt{ack}, K_i(h)\texttt{>}$ to p_j.

5. When p_i receives $\texttt{<AV}, \texttt{deliver}, m, A\texttt{>}$, such that A contains a valid set of AV-acknowledgments from every member in $\mathit{witness}(m)$, or a set of $\lceil (n+t+1)/2 \rceil$ E-acknowledgments, and m was not previously delivered, p_i performs *WAN-deliver*(m). If a timeout period has passed and r is not known to have delivered m, p_i sends $\texttt{<AV}, \texttt{deliver}, m, A\texttt{>}$ to r.

Figure 3: The active_t protocol

We note that, an honest process receiving conflicting messages m and m' from a sender p_j identifies without doubt a failure in p_j due to the signatures on m, m'. Once p_j is known to have failed, all the honest processes are informed of the failure and no more messages are exchanged with p_j. Typically, since the presence of conflicting messages is attributable to their sender, it may deter malicious attempts, as their originators stand the risk of being detected, with their conflicting signatures as proof.

The next three theorems argue Integrity, Self-Delivery and Reliability of the active_t protocol. The proofs are very similar to the ones given in Theorem 4.1, 4.2 and 4.3, respectively.

Theorem 5.1 *Let p_i be an honest process participating in the active_t protocol. Then p_i executes WAN-deliver(m) at most once, and if $\mathit{sender}(m)$ is honest, then only if $\mathit{sender}(m)$ executed WAN-multicast(m).*

Theorem 5.2 *Let p_i be an honest process participating in the active_t protocol. If p_i executes WAN-multicast(m) then p_i executes WAN-deliver(m).*

Theorem 5.3 *Let p_i and p_j be two honest processes participating in the active_t protocol. If p_i performs WAN-deliver(m), then p_j performs WAN-deliver(m).*

We now prove that active_t maintains Probabilistic Agreement:

Theorem 5.4 *Let p_i and p_j be two honest processes participating in the active_t protocol. Let p_i deliver a message m, p_j deliver m', such that $\mathit{sender}(m) = \mathit{sender}(m')$ and $\mathit{seq}(m) = \mathit{seq}(m')$. Then with very high probability (determined by the protocol parameters) p_i and p_j delivered the same message,* i.e., *$m = m'$, where the probability is taken assuming $\mathit{seq}(m)$ is chosen uniformly (and at the limit as $\mathit{seq}(m)$ goes to infinity), and over all random choices made by honest processes.*

Proof : As argued in Theorem 4.4 above, for p_i, p_j to deliver m, m' respectively, they must have each delivered corresponding sets of valid acknowledgments A, A'. Denote $P(A)$ the set of processes represented in A, and likewise $P(A')$. If $P(A), P(A')$ intersect in an honest process, then we argue that $m = m'$ as in Theorem 4.4. It remains to compute the probability that $P(A)$ does not intersect $P(A')$ at any honest member.

Case 1: $P(A) = P(A') = \mathit{witness}(m)$. Thus, $\mathit{witness}(m)$ contains corrupt members only. Since $\mathit{witness}(m)$ randomizes the choice of processes as a function of $\mathit{sender}(m)$,$\mathit{seq}(m)$, the probability p_k for this event is at most $p_k \le \left(\frac{t}{n}\right)^k \le \left(\frac{1}{3}\right)^k$, assuming $\mathit{seq}(m)$ is chosen uniformly, and taken at the limit as $\mathit{seq}(m)$ goes to infinity.

Case 2: $P(A) \ne P(A')$. W.l.o.g., $P(A) = \mathit{witness}(m)$, and so $|P(A')| = \lceil (n+t+1)/2 \rceil$. To distinguish from case 1, assume that $P(A)$ contains at least one honest member p_h. Note that the honest member p_h chooses peers randomly, and does not disclose the composition of peers_h to $\mathit{sender}(m')$. Moreover, by assumption there is a positive probability for each message from $\mathit{sender}(m')$ to an honest process to reach its destination before the timeout (independent of the choice of peers_h). Thus, the choice of peers_h is independent from the choice of any process in $P(A')$. Therefore, the probability that p_h does not reach any honest member in $P(A')$ in ℓ probes is at most $\left(\frac{n - \lceil (n+t+1)/2 \rceil + t}{n}\right)^\ell \approx \left(\frac{n+t-1}{2n}\right)^\ell \le \left(\frac{2}{3}\right)^\ell$.

Thus, the overall probability for conflicting message to be deliverable is bounded by $\left(\frac{1}{3}\right)^k + (1 - \left(\frac{1}{3}\right)^k)\left(\frac{2}{3}\right)^\ell$. Obviously, this expression can be made as small as desired by appropriate choice of k, ℓ, for appropriate system sizes. □

Analysis

The active_t protocol aims to maximize performance in faultless cases by minimizing the number of signed messages and the number of overall message exchanges. Stress is placed on minimizing digital signatures (to effectively a constant number), since the cost of producing digital signatures in software is at least one order of magnitude higher than message-sending, for typical message sizes.

The overhead in forming agreement on message contents in faultless scenarios is k signatures and k message exchanges for collecting *witness* acknowledgments and $\ell \times k$ authenticated message exchanges with peers. We note that all of the overhead messages are small (containing fixed size hashes, signatures, and the like), signatures may be computed concurrently at all of the witnesses, and likewise, all pairs of message exchanges with peers may be done concurrently.

The overhead in case of failures can reach, in the worst case scenario, n signatures and message exchanges with witnesses of both the no-failure regime and the recovery regime, and additionally, $\ell \times k$ authenticated message exchanges between witnesses and their peers.

The level of guarantee achieved by active_t depends on the parameters k and ℓ. If the system contains as many as $\lfloor (n-1)/3 \rfloor$ corrupt processes that know about each other (the worst case scenario), then one out of 3^k messages, on average, will have a completely corrupt witness set. Since $\mathit{witness}(m)$ is a function of $\mathit{sender}(m)$ and $\mathit{seq}(m)$, whenever $\mathit{witness}(m)$ has a completely corrupt witness set, $\mathit{sender}(m)$ has the opportunity to collude with the corrupt witnesses, and convince honest processes to *WAN-deliver* conflicting messages. Moreover, since the $\mathit{witness}()$ function is known to all participants, once the corrupt processes and the $\mathit{witness}()$ function are determined, the adversary can predict the sequence number and sender of messages for which it can so collude and cause conflicting *WAN-deliver* events. Nevertheless, this is the case for only an exponentially-small fraction of the messages that are sent–by proper choice of the parameter k, and restricting the adversary from corrupting processes with knowledge of $\mathit{witness}(m)$, the likelihood of such a message occurring in the lifetime of the system can be made appropriately small.

There is also a chance of obtaining acknowledgments signed by honest members for conflicting messages, by having non-intersecting sets of honest processes participate in the two protocol regimes, as follows: If a corrupt process p_i generates conflicting messages m, m', sent to $\mathit{witness}(m)$ and S respectively, where $|S| = \lceil (n+t+1)/2 \rceil$ and $S \cap \mathit{witness}(m) = \emptyset$. However, if $\mathit{witness}(m)$ contains at least one honest member p_h, then the probability that peers_h does not intersect S at an honest member is at most $\left(\frac{2}{3}\right)^\ell$, which can be made a small as desired by choosing ℓ appropriately.

For example, in a network of 100 processes, and assuming the number of corrupt processes $t \leq 10$, choosing $k = 3$, $\ell = 5$ will guarantee that conflicting messages are detected with probability at least 0.95, whereas in a network of 1000 processes with $t \leq 100$, we can achieve 0.998 guarantee level with $k = 4$, $\ell = 10$.

Optimizations

It is possible to improve the fault tolerance of the active_t protocol family by allowing any subset of $k - C$ witnesses, where C is some constant, to validate a message. Unfortunately, while this improves resiliency to benign failures, it increases the probability of a corrupt witness set. Nevertheless, suppose that $t = \lfloor (n-1)/3 \rfloor$. The probability $p_{k,C}$ of a corrupt set of $k - C$ out of k randomly chosen processes is bounded by:

$$p_{k,C} \approx \sum_{j=0}^{C} \frac{\binom{\frac{n}{3}}{k-j}\binom{\frac{2n}{3}}{j}}{\binom{n}{k}} \leq \left(\frac{kn}{C(n-k)}\right)^C \left(\frac{1}{3}\right)^{k-C}$$

This probability tends to zero if we choose $C \ll k$. Therefore, this allows us to increase the fault tolerance while preserving safety to any desirable degree.

Another improvement can be achieved by employing the 3T protocol as the recovery protocol, instead of the E protocol. This would be done as follows: For every message m, the protocol determines, in addition to the active witness set $\mathit{witness}(m)$, a recovery witness set $\mathit{recovery}(m) = I(\mathit{sender}(m), \mathit{seq}(m))$, whose size is $3t+1$. A sender obtains acknowledgements from either $\mathit{witness}(m)$, or – for recovery – from $2t+1$ processes in $\mathit{recovery}(m)$. In the active phase of the protocol, an active witness in $\mathit{witness}(m)$ probes members within $\mathit{recovery}(m)$ only. It is easy to see that the correctness claims made for active_t still hold. The overhead of the optimized protocol in case of failures is, in the worst case scenario, $k + 3t + 1$ signatures and message exchanges with witnesses of both the no-failure regime and the recovery regime, and additionally, $\ell \times k$ authenticated message exchanges between witnesses and their peers. When $t \ll \lfloor (n-1)/3 \rfloor$, this significantly reduces the failure-overhead of n signatures and message exchanges incurred in the simple $active_t$ protocol.

6 Conclusions

Our experience in constructing robust distributed systems [1, 6] shows that (secure) reliable broadcast is an important tool for distributed applications. Implementing reliable broadcast in an insecure environment with arbitrary failures incurs inevitable overhead required for maintaining consistency. However, a price that may be acceptable in a small network becomes intolerable for a very large system.

In this paper, we have shown two approaches in which the requirements on the system may be weakened in order to allow for more efficient implementations of reliable broadcast: The first one is suitable for environments in which failures are rare, and where therefore, it is reasonable to assume a low threshold on the number of failures. The second one relaxes the consistency requirement to allow a marginal fraction of the messages to be delivered inconsistently. This approach is practical when reversing the effects of (just a small number of) bad message deliveries is possible, albeit not desired. In both cases, we have devised protocols that meet the requirements and incur costs that do not grow with the system size, in the normal faultless scenarios.

References

[1] Danny Dolev and Dalia Malki. The Transis Approach to High Availability Cluster Communication. *Communications of the ACM*, 39, April 1996.

[2] M. Fischer, N. Lynch, and M. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *J. ACM*, 32:374–382, April 1985.

[3] Stuart Haber and W. Scott Stornetta. How to time-stamp a digital document. *J. Cryptol.*, 3(2):99–111, January 1991.

[4] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, July 1982.

[5] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 569–578, New York, NY, USA, 1997. ACM.

[6] Dahlia Malkhi and Michael Reiter. A high-throughput secure reliable multicast protocol. *J. Comput. Secur.*, 5(2):113–127, March 1997.

[7] Louise E. Moser and P. M. Melliar-Smith. Total ordering algorithms for asynchronous byzantine systems. In *Proceedings of the 9th International Workshop on Distributed Algorithms*, WDAG '95, pages 242–256, London, UK, UK, 1995. Springer-Verlag.

[8] Michael K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2Nd ACM Conference on Computer and Communications Security*, CCS '94, pages 68–80, New York, NY, USA, 1994. ACM.

[9] R. Rivest. The md5 message-digest algorithm, 1992.

[10] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[11] Sam Toueg. Randomized byzantine agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC '84, pages 163–178, New York, NY, USA, 1984. ACM.