Partially Authenticated Algorithms for Byzantine Agreement

Malte Borcherding

Institute of Computer Design and Fault Tolerance, University of Karlsruhe
76128 Karlsruhe, Germany
email: borcher@ira.uka.de

Abstract

Byzantine agreement is a fundamental issue in fault-tolerant and secure distributed computing. Protocols solving Byzantine agreement guarantee that a sender can transmit a value to a group of receivers consistently, even if some of the nodes, including the sender, are arbitrarily faulty.

In the past, protocols for Byzantine agreement were generally either authenticated or non-authenticated. Non-authenticated protocols make no use of signatures, while in authenticated protocols, all messages have to be signed.

Authenticated protocols can tolerate more faults and are more message-efficient than non-authenticated protocols, but they have the disadvantage of time-consuming signature generation. In this paper, we introduce techniques to reduce the amount of signatures by combining mechanisms from authenticated and non-authenticated protocols.

Keywords: Byzantine agreement, fault tolerance, distributed systems, secure systems, authentication

1 Introduction

The problem of Byzantine agreement arises when a set of nodes in a distributed system needs to have a consistent view of a message sent by one of them, despite the presence of arbitrarily faulty nodes. More precisely, a protocol solving Byzantine agreement must satisfy the following conditions:

(B1) All correct nodes decide for the same value.

(B2) If the sender is correct, all nodes decide for the value of the sender.

(B3) Each correct node eventually decides for a value.

Protocols solving Byzantine agreement are generally divided into two classes: authenticated protocols and non-authenticated protocols. In authenticated protocols, all messages are signed digitally in a way that the signatures cannot be forged, and a signed message can be unambiguously assigned to its signer. This mechanism allows a node to prove to others that it has received a certain message from a certain node. Authenticated protocols can tolerate an arbitrary number of faulty nodes. In non-authenticated protocols, no messages are signed. These protocols require more than two thirds of the participating nodes to be correct ([LSP82]). For both paradigms of authenticated and non-authenticated protocols, there exist customized protocol techniques.

Although message authentication allows for an optimal fault tolerance, the generation of signatures is a very time-consuming task. For this reason, the fault tolerance of protocols which require message authentication only in *certain rounds* has been investigated in [Bor95]. The protocols given there were maximally fault-tolerant for a given number of authenticated rounds, but not very message-efficient.

A different approach has been taken in in [ST87]. There, authenticated messages are simulated by non-authenticated subprotocols. This allows to transform authenticated protocols easily into non-authenticated protocols while keeping some of their properties. But with this technique, the good fault-tolerance properties of authenticated protocols are lost.

Hence, the question arises whether it is possible to combine techniques for authenticated and non-authenticated protocols for use in *partially* authenticated protocols. The resulting protocols should balance the low message complexity and high fault tolerance of authenticated protocols with the fast message generation on non-authenticated protocols.

In this paper, we answer this question to the positive. We give protocols which make use of mixed techniques and identify situations in which they are applicable.

In *Proceedings of the 9th International Conference on Parallel and Distributed Computing Systems, Dijon, France, 1996, pp. 8–11, ISCA.*

2 System Model

Our world consists of n nodes connected by a complete network. We assume that t of the nodes may behave in an arbitrary manner, while $c = n - t$ always behave correctly. The number of nodes that actually behave faulty during a given protocol execution will be denoted with f ($f \leq t$).

The nodes operate at a known minimal speed, and messages are transmitted reliably in bounded time. The receiver of a message can identify its immediate sender, and we assume the existence of an authentic signature scheme such that a signature cannot be forged and each node knows whom a signature on a message belongs to.

During a protocol execution, the nodes communicate in successive *rounds*. In each round, a node may send messages to other nodes, receive the messages sent to it in the current round and perform some local computation. m of the rounds are distinguished as *authenticated rounds*. In these rounds, all messages are to be signed. As a convention, the first round of a protocol will be called round 1.

3 Main Techniques

Depending on the environment and the goals, one can choose between several different possible techniques for solving Byzantine agreement. In the following sections, we will identify some of the main ideas. The described techniques all share the same fundamental structure:

1. In the first round, the sender sends its value to all others.
2. In the following rounds, the nodes report to the others what they have received in the previous round, provided some condition is met.
3. Based on the messages received, each node computes its decision value. This decision is made after the $t + 1$st round at the latest.

The techniques described differ in the condition in step 2 and the way of computing the decision value in step 3.

3.1 Exponential Information Gathering (EIG)

This technique is essentially simple fault masking. Correct nodes echo each message they receive, as long as they have not reported about that particular message in a previous round. Hence, the messages are of the form "A said B said ... the sender said X", and no node appears twice in such a chain. As a result, for a message which was sent in round k, there should be $n - k$ echoes, one from each node which is not yet listed.

After round $t + 1$, the nodes apply a recursive majority voting to the messages received. For each message x received in round t, a node takes the majority of the echoes about this message received in round $t + 1$. This majority value replaces the actual message x. Then, the same procedure is applied to the messages received in round $t - 1$, using the new values of the messages of round t. This voting is repeated until the agreed-upon value of the original message of round 1 is determined.

In a nonauthenticated EIG protocol (e.g., in [LSP82]), there has to be always a majority of correct echoes about messages from correct nodes. If a set of nodes reports about a message from a faulty node, there is either an agreement about *all* reports (including those from faulty nodes), or the message itself is not important for the agreement (i.e., a message from a faulty node about a message from a correct node.)

Authentication can be used in this context for restricting the behaviour of the faulty nodes in that they cannot report wrong values signed by correct nodes. This technique relaxes the requirements for the number of correct nodes echoing messages. If every message in a protocol has to be signed, any number of faulty nodes can be tolerated. If no messages are signed, it is required that $n \geq 3t + 1$. [Bor95] shows how to determine the necessary number of authenticated rounds if a given number of faulty nodes is to be tolerated. One result is that in order to tolerate $t = n/2$ faulty nodes, only $\lceil \log_2(n/2 + 1) \rceil$ authenticated rounds are necessary.

3.2 Early Stopping

If in a protocol execution there are less than t faulty nodes (i.e., $f < t$), it is not necessary to run the protocol for $t + 1$ rounds. In [BGP92], a variation of the EIG protocol is presented that reaches agreement and stops after $\min(t + 1, f + 2)$ rounds, which is optimal (cf. [DRS90]). This result is obtained by a more complex reasoning about the received messages than the simple majority voting in the plain EIG protocol.

3.3 Sender Fault Detection

Sender fault detection (SFD) relies on signatures. The main principle is to discover whether the sender behaves inconsistently towards other nodes. If a correct node sees exactly one value signed by the sender during protocol execution, it decides for that value. Otherwise, it decides for a default value. The protocol has to guarantee that either all correct nodes see exactly one signed value or all correct nodes detect that the sender is faulty.

The following (completely authenticated) protocol from [DS83] has that property: Each node keeps a local set which contains the values the node has seen so far. In the first round, the sender sends its value to all other nodes. If in the next t rounds, a node sees a new value signed by as many nodes as there have been rounds (starting with the sender's signature), it adds that value to its set. If it is the first or second value in the set, the node adds its signature and sends it to all nodes which have not yet signed.

The proof of correctness is not difficult: If a correct node sees two different signed values during the first t rounds, it forwards them to all correct nodes which have not seen them before. Hence, all correct nodes see that the sender is faulty and will decide for a default value. If a correct node sees a second signed value in the last round, this value carries $t+1$ signatures and has hence been seen and distributed by at least one correct node before. With a similar argument, it can be shown that if a correct node does not see a correctly signed value during the protocol execution, then no correct node has seen one.

This protocol has the property that if f faulty nodes behave faulty ($f < t$), then no valid messages are sent after round $f+2$: Suppose there is a valid message sent at round $f+3$. Then it carries at least 3 signatures from correct nodes. The third correct node must have seen the value for the first time one round before adding its signature. This is not possible, since the first correct signer has broadcast the value at least two rounds earlier. As a consequence, no valid message is signed by more than two correct nodes.

This does *not* imply that the protocols stop after round $f+2$, since no correct node can be sure that no further value will arrive in the next rounds. Hence, if the sender is correct and has signed exactly one value, all nodes have to wait until round $t+1$ before they can make their final decision. On the other hand, if a correct node has seen and forwarded two different signed values, it can decide for the default value immediately. Furthermore, it can be sure that all other correct nodes will decide after the next round.

4 Combination of Different Techniques

As we have seen, for both paradigms of authenticated and non-authenticated algorithms there are suitable techniques. This raises the question whether it is possible to combine these techniques for protocols which use authentication only in certain rounds.

The answer is not obvious at first sight: When in an SFD protocol a node receives a (valid) message in the last (t+1st) round, it knows that all correct nodes have seen the transmitted value because it has been signed by $t+1$ nodes. If not all rounds are authenticated, it cannot draw this conclusion.

On the other hand, in EIG protocols it is expected that correct nodes echo the messages from a correct node such that there is a majority of correct echoes. This is not done in SFD protocols, hence this case has to be taken care of when switching from an SFD protocol to an EIG protocol.

In the rest of the section, we will use the following notation: The authenticated rounds after non-authenticated rounds will be denoted a_i, and the non-authenticated rounds after authenticated rounds will be denoted b_i, as shown in Fig. 1. If the first round is authenticated, it is called round a_0, and b_1 otherwise (then a_0 is undefined). So a_1 can never be 1, and b_i is always smaller than a_i. Furthermore, the last round is regarded as authenticated round, so for each b_i there is always a defined a_i. The number of pairs (a_i, b_i), $i > 0$, will be denoted with s. Subprotocols starting at a_i and b_i will be called A_i and B_i, respectively.

Figure 2 shows protocols A_i and B_i. As can be seen, A_i starts with an SFD part and switches to an EIG part, while B_i does the inverse. The complete protocol (starting in round 1) will be called *Protocol C*. Protocol C is A_0 if the first round is authenticated, and B_1 otherwise.

The following theorem gives the requirements for Protocol C being correct.

Theorem 1. *Protocol C reaches Byzantine Agreement for*

$$n \geq \max_{i=1}^{s}(2t + a_i - 2b_i + 2).$$

Before we prove this theorem, we need some definitions and lemmas. Lemma 5 deals with subprotocols of type A_i, while Lemmas 6 and 7 describe properties of protocols of type B_i. Finally, we will give the proof of Theorem 1.

Definition 2 (All-faulty Message). A message of the form "A said B said ... the sender said X", where all listed nodes (including the sender) are faulty, is called *all-faulty message*.

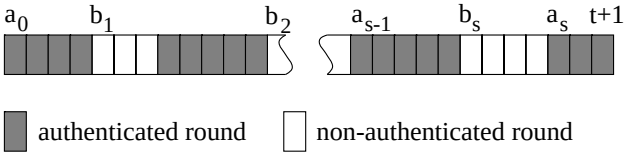

Fig. 1 Positions of the a_i and b_i

Protocol A_i

0. For $i = s$, A_i is an SFD protocol, as described in 3.3. Otherwise, 1. to 3. are applied.
1. In the authenticated rounds a_i to $b_{i+1} - 1$, the nodes proceed as in an SFD protocol.
2. If a node receives a message in round $b_{i+1} - 1$ which it would usually sign and distribute to other nodes in the next round, it broadcasts this message using protocol B_{i+1} to the recipients.
3. The results of the subprotocols started in round b_{i+1} are used together with the values received in rounds a_i to $b_{i+1} - 1$ for the decision. If there is only one value, this will be the result, otherwise the nodes decide for a default value.

Protocol B_i

1. In rounds b_i to $a_i - 1$, the nodes apply an EIG protocol as described in 3.1.
2. Each echo about a value received in round $a_i - 1$ is transmitted via protocol A_i to its $n - (a_i - 1)$ receivers.
3. The results of these protocols are then subject to the recursive majority voting as described in the EIG protocol.

Fig. 2 Protocols A_i and B_i in the mixed model

Definition 3 (Partial Correctness A). A protocol A_i is called *partially correct* if it reaches agreement on the sender's message, provided the sender is correct or the message is an all-faulty message.

Definition 4 (Partial Correctness B). A protocol B_i is called *partially correct* if it reaches agreement on the sender's message, provided that
(a) it is *an echo* of an all-faulty message or
(b) it is sent in round 1 (i.e., $i = 1$ and $b_1 = 1$).

In these definitions, the *sender* is meant to be the sender of the respective subprotocol, while the term *all-faulty message* takes *all* previous rounds into account.

Lemma 5. *A protocol A_{i-1} $(i = 1, \ldots, s)$ is partially correct, provided there is agreement on the echoes in round b_i concerning all-faulty messages.*

Proof. Values signed by correct nodes in rounds a_{i-1} to $b_i - 1$ are seen by all correct nodes and used for the final decision. The only problem are messages which are only signed by faulty nodes. If the protocols starting at round b_i guarantee agreement on these messages, the protocol started in round a_{i-1} is correct. □

Lemma 6. *From round $b_i + 1$ to round a_i there is always a majority of correct echoes about messages received in the previous round, provided that at most $t - b_i + 1$ nodes are faulty and $n \geq 2t + a_i - 2b_i + 2$.*

Proof. For each message transmitted in round $a_i - 1$, there are $n - a_i + 1 \geq 2t - 2b_i + 3$ echoes. At most $t - b_i + 1$ of these are from faulty nodes, as opposed to at least $2t - 2b_i + 3 - (t - b_i + 1) = t - b_i + 2$ correct echoes. Hence, the correct echoes are a majority. In the rounds before round a_i, the majority can only be

stronger, since the number of echoes is larger, while the maximum number of faulty nodes remains the same. □

Lemma 7. *A protocol B_i $(i = 1, \ldots, s)$ is partially correct, provided that*

(a) $n \geq 2t + a_i - 2b_i + 2$,
(b) there is agreement on messages sent by correct nodes in round a_i, and
(c) there is agreement on all-faulty messages sent in round a_i.

Proof. Partial correctness is only concerned with sender's messages constituting echoes of all-faulty messages (or $b_i = 1$). Hence, $b_i - 1$ faulty nodes do not participate, leaving $t - b_i + 1$ faulty participants. Given (a), Lemma 6 can be applied. Hence, if (b) holds, then all correct nodes will agree on the correct messages (due to the recursive majority voting).

If the sender of B_i is faulty, it is necessary that *all* echoes be agreed upon. As we have shown, this is true for the correct echoes. The faulty echoes sent in round $b_i + 1$ will be agreed upon if all echoes for these messages are agreed upon. This is again true for the correct echoes. Continuing this argument, we arrive at the requirement that there be agreement on the all-faulty messages sent in round a_i, which is stated in (c). □

Proof of Theorem 1
Lemmas 5 to 7 are now used to prove Theorem 1 by reverse induction on the A_i and B_i. The base is that A_s is partially correct. We will then show by induction that the whole protocol is partially correct. Finally, we prove that a partially correct protocol starting in round 1 solves Byzantine Agreement.

Proof (of Theorem 1). We will prove (i) to (iv) for $n \geq \max_{i=1}^{s}(2t + a_i - 2b_i + 2)$:

(i): A_s is partially correct.
(ii): A_i is partially correct $\Rightarrow$ B_i is partially correct $(i = 1 \ldots s - 1)$.
(iii): B_i is partially correct $\Rightarrow$ A_{i-1} is partially correct $(i = 1 \ldots s)$.
(iv): A partially correct protocol started in round 1 reaches Byzantine Agreement.

(i): Messages signed by correct nodes are always agreed upon (they cannot be forged and are seen by everyone). A protocol A_s started for an all-faulty message has one more round that there are faulty participants. Such a protocol is correct (see proof of SFD protocol). Hence, A_s is partially correct.

(ii): We have to show that the proviso of Lemma 7 is fulfilled. (a) is part of the theorem's assumptions. (b) and (c) are fulfilled by the partial correctness of A_i.

(iii): Follows directly from Lemma 5 and B_i's partial correctness.

(iv): If the first round is authenticated, we can regard the whole protocol as A_0. Then A_0's partial correctness guarantees agreement (note that a faulty sender's message is an all-faulty message). If the first round is non-authenticated, the whole protocol is B_1, and B_1's partial correctness guarantees agreement. □

5 Discussion

The techniques described in the previous sections can be applied in different ways, depending on the requirements of the application. Here are some examples:

- If one strives for low message complexity and expects only few faulty nodes f_e (with regard to a higher number t which should be tolerated in the worst case), one can use the following construction: Let the first $m = f_e + 2$ rounds be authenticated. For the second (non-authenticated) part, choose an early-stopping EIG protocol. If only f_e nodes are faulty, then no messages will be sent in the EIG-part of the protocol. This can be shown as follows:

  A valid message in the subsequent EIG protocol has to carry $f_e + 2$ signatures. In 3.3, it has been shown that at most two of these signatures belong to correct nodes, so all faulty nodes must have signed. Hence, only a correct node could be the sender of such a message. But all correct nodes have seen the message before, so no valid message will be sent. As a consequence, all nodes behave (implicitly) correctly and the EIG protocol will stop after two rounds without any message overhead.

  For $f > f_e$, all messages sent after round m carry at least $m - 2$ signatures from faulty nodes. Hence, only $f - m + 2$ faulty nodes can participate in an early-stopping EIG protocol initiated in round $m + 1$. Under this condition, the EIG-protocol stops after $f - m + 4$, so that the complete protocol stops after $\max(f, f_e) + 4$ rounds. It is an open question whether this can be optimized in the hybrid model.

- If the goal is to avoid the time-consuming signature generation and to have early stopping

when only few faults occur, one can start with a non-authenticated early-stopping EIG-protocol and switch to an SFD protocol when it becomes more expensive to handle the exponentially growing number of messages than to sign the messages of the SFD-protocol.

- If the cost for signature generation always dominates the cost for message handling, and one wishes to find a trade-off between worst-case protocol time and fault tolerance, one should use the mixed model. This model allows to choose a minimal number of rounds as authenticated rounds in order to achieve a desired degree of fault tolerance. These rounds will generally not be consecutive. In [Bor95] it has been shown that for EIG protocols only $\log_2(n/2+1)$ authenticated rounds are necessary to tolerate $n/2$ faulty nodes. The results given above allow an immediate transfer to the mixed model.

6 Summary

In this paper, we have investigated partially authenticated protocols which make use of a combination of different techniques. We have shown that it is possible to switch between certain techniques for authenticated and non-authenticated environments within a single protocol. With these hybrid protocols, it is possible to find a trade-off between the high fault tolerance and message efficiency of authenticated protocols on the one hand and the early-stopping properties and fast message generation of non-authenticated protocols on the other hand.

References

[BGP92] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus. In *Proceedings of the 6th International Workshop on Distributed Algorithms (WDAG)*, LNCS 647, pages 221–237. Springer-Verlag, 1992.

[Bor95] Malte Borcherding. On the number of authenticated rounds in Byzantine agreement. In *Proceedings of the 9th International Workshop on Distributed Algorithms (WDAG)*, LNCS 972, pages 230–241, Le Mont Saint-Michel, France, 1995. Springer-Verlag.

[DRS90] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in Byzantine agreement. *Journal of the ACM*, 37(4):720–741, 1990.

[DS83] Danny Dolev and Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM Journal of Computing*, 12(5):656–666, November 1983.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

[ST87] T.K. Srikanth and Sam Toueg. Simulating authenticated broadcasts to derive simple fault-tolerant algorithms. *Distributed Computing*, 2:80–94, 1987.