Optimizing Byzantine Consensus for Fault-Tolerant Embedded Systems with Ad-Hoc and Infrastructure Networks

Hans P. Reiser, António Casimiro
{hans,casim}@di.fc.ul.pt
LASIGE, Faculdade de Ciências da Universidade de Lisboa *

Abstract

Consensus algorithms are an important building block for fault-tolerant distributed systems. This paper investigates approaches to optimize solutions of distributed consensus to the properties of embedded systems. We discuss alternatives that allow constructing better practical solutions in realistic environments. For example, many networked embedded systems are equipped with both ad-hoc communication among collaborating actors and communication with a static infrastructure. Traditional consensus algorithms, however, are usually fully decentralized, and thus are unable to benefit from the additional infrastructure. Other existing approaches fully rely on the infrastructure, and thus fail to work if the infrastructure is not available. This paper sketches hybrid approaches that combine the advantages of both strategies.

1 Introduction

Consensus is an important problem in distributed systems. Many tasks, such as state-machine replication, atomic commitment, and total order multicast, can be reduced to consensus. In the past decades, many theoretical and practical algorithms have been proposed for a large variety of system models.

*Faculdade de Ciências da Universidade de Lisboa. Bloco C6, Campo Grande, 1749-016 Lisboa, Portugal. Tel. +(351) 21 750 0532. Fax +(351) 21 750 0533. Navigators Home Page: http://www.navigators.di.fc.ul.pt. This work was partially supported by the EU through project IST-FP6-STREP-26979 (HIDENETS) and by the FCT through the Large-Scale Informatic Systems Laboratory (LASIGE).

Practical solutions to the consensus problem play an important role for the construction of dependable networked embedded systems. For example, consensus can be used to coordinate the actions of distributed mobile actors, such as robots or cars. The properties of consensus guarantee that all correct participants make a common decision that cannot be disrupted by faulty entities.

The system environment of distributed embedded systems shows some differences to "traditional" distributed systems. Nodes have significant constraints on memory, CPU power, and communication capacity, and there may be a higher fluctuation of nodes. At the network level, many systems have hybrid communication facilities, composed of ad-hoc communication and infrastructure communication. The ad-hoc network allows direct communication between nodes within communication range. Sometimes, a multi-hop routing implementation allows decentralized communication between nodes on a larger scale on the basis of the ad-hoc network. In addition, access points provide connectivity to a static infrastructure.

Some recent research investigated consensus algorithms for mobile networks. These algorithms are either designed for autonomous operation using only an ad-hoc network, or are fully based on the support from a central infrastructure. In the ad-hoc network, the participants can fluctuate very fast. Nodes are easy to attack, as often there is no physical control over the participants. Fully decentralized algorithms must cope with these problems. Solutions that rely on the central server often are more efficient. However, a central server that is required for operation represents a single point of failure, a very undesirable design for dependable applications. Also,

moving nodes must handle hand-over between access points; with current standard technology, these hand-overs may cause lack of connectivity to the infrastructure for durations up to several seconds.

This paper discusses some approaches to better optimize distributed consensus algorithms to the system characteristics of distributed embedded systems. The main focus is on finding strategies that allow operation with and without connectivity to a central infrastructure. A system that supports both ways of operations can provide different liveness or timeliness guarantees, depending on whether there is connectivity to the infrastructure. For example, a central infrastructure can help to make stronger timing guarantees with high coverage on consensus termination. If the connection to the central infrastructure fails, a decentralized ad-hoc protocol could provide the same functionality with weaker timing guarantees.

This paper is structured as follows. The next section discusses related work. Section 3 describes our system model. Section 4 discusses approaches for designing Byzantine consensus algorithms for embedded systems with hybrid network infrastructures, and Section 5 concludes.

2 Related Work

Several papers aim at providing solutions for consensus in mobile environments. Badache et al. [3] provide a multi-valued consensus algorithm for crash-stop failures in an asynchronous system with $\diamond S$ failure detection. The authors assume that mobile hosts (MH) are connected to mobile support stations (MSS), and communication between MH is routed through MSSs (there is no direct ad-hoc communication between MHs). The main idea of the protocol is that each MSS acts as a representative of all MHs connected to it. Seba et al. [11] generalize this idea in a general framework for solving consensus in infrastructured wireless networks. This solution adds support for MSSs that dynamically join or leave the consensus protocol session due to hand-over of MHs.

Wu et al. [13] present a hierarchical consensus protocol for mobile ad hoc networks. This protocol works with crash-stop failures and assumes an asynchronous system with an unreliable failure detector of the $\diamond P$ class. It works fully decentralized without using any infrastructure. The algorithm introduces a hierarchy between at least $f + 1$ hosts that act as proxies, and the remaining hosts that are associated to a proxy. By merging messages in proxies, this approach reduces the total number of messages.

Another form of "hybrid" approach combines failure detection with randomization. Failure detectors and randomization are two ways of avoiding the famous FLP impossibility [9]. Aguilera and Toueg [1] and Mostefaoui et al. [10] both propose a binary consensus algorithm of this hybrid kind for the crash-stop model. If the unreliable failure detector works correctly, deterministic termination is guaranteed; otherwise, the algorithm terminates eventually with probability 1.

All work discussed so far only uses either ad-hoc communication or infrastructure communication, but not both, and only covers crash-stop failures. Some authors have previously addressed Byzantine consensus for mobile networks. Angluin et al. [2] present a weaker form called "stabilizing consensus", in which nodes do not commit to a final output at a certain point in time, but instead have outputs that eventually converge to a stable configuration. Drabkin et al. [8] discuss the related problem of efficient Byzantine broadcast in ad-hoc networks. Wang et al. [12] propose an algorithm for Byzantine consensus in mobile ad-hoc networks.

In this paper, we address the problem of Byzantine consensus. In open mobile ad-hoc networks, it is even easier than in traditional systems to inject malicious nodes. Thus, reaching consensus in spite of malicious nodes is an important mechanism. The main advantage of our hybrid approach is that it makes use of a hybrid network consisting of ad-hoc and infrastructure connections.

3 System Model

The system consists of a set of n processes $P = \{p_0, \ldots, p_{n-1}\}$ and a infrastructure service process p_s. The processes are said to be *correct* if they adhere to the protocol until termination. Otherwise, they are called *corrupt*. We assume a Byzantine fault model, i.e., there are no constraints on the actions of corrupt processes. Moreover, we assume that only f out of n processes can be corrupt, with $n \geq 3f + 1$.

In the multi-valued consensus problem, each processor

p_i proposes some initial value v_i, and then the processors must a agree on a common value. As we assume a randomized model in an asynchronous system, the termination of the protocol can be guaranteed only in a probabilistic way. Corrupt processes must not be able to force the correct processes to do something "bad". Formally, the consensus problem is specified by the following properties:

- *Validity 1:* If all correct processes propose the same value v, then any correct process that decides, decides v.
- *Validity 2:* If a correct process decides v, then v was proposed by a correct process, or $v = \perp$.
- *Agreement:* No two correct processes decide differently.
- *Termination:* All correct processes eventually decide with probability 1.

We further assume that the identity of all message senders can be verified. Without this assumption, an adversary could forge any message and consequently impersonate any node, and thus no solution would be possible at all. In practice the verification can be assured with cryptographic message authentication mechanism.

4 Design Options

Most binary and multi-valued consensus algorithms are constructed by a hierarchical composition. Figure 1 shows such a typical composition, in which multi-valued consensus is implemented on top of binary consensus and reliable multicast, with authenticated point-to-point links at the bottom.

In the following, we distinguish three alternatives for creating an infrastructure-assisted solution, given an existing decentralized implementation of consensus. We focus in solutions that gain some advantage if the infrastructure connection is available, and otherwise work as a traditional decentralized protocol. Infrastructure interaction can be integrated in the existing building blocks at the level of reliable multicast, at the level of binary consensus, and at the level of multi-valued consensus.

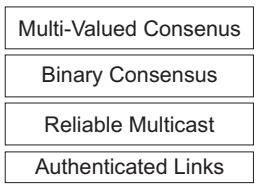

Figure 1: Hierarchical composition of multi-valued consensus

4.1 Reliable Multicast

The level of *reliable multicast* implements a mechanism that guarantees that if a message m is delivered to a correct node, this (and only this) message is eventually delivered to all correct nodes. A typical algorithm that is able to tolerate malicious corrupt nodes is the reliable broadcast algorithm described by Bracha [5].

A reliable multicast might be supported efficiently by a central infrastructure node if this central node is guaranteed to fail only by crashing. The infrastructure could distribute the broadcast value, and only in the case of unavailability of the central node, the distributed participants would have to execute a distributed protocol. However, if malicious corruptions of the infrastructure node are included in the failure model, it becomes obvious that in any case, the set of nodes needs to execute a distributed algorithm to ensure the delivery of a unique value. We expect that a Byzantine infrastructure cannot provide any significant benefit, and thus we do not discuss this option in more detail.

4.2 Binary Consensus

The level of *binary consensus* implements an algorithm that guarantees that all correct nodes agree on a common value, and if all correct nodes propose the same value, they decide on that value. Formally, the agreement and termination condition are the same as for multi-valued consensus as specified above, and the validity assumptions can be simplified to just *Validity 1*. Typical randomized binary consensus protocols are based on the random *coin-toss* operation, which returns the value 0 or 1 with equal probability. The coin tossing can be implemented with *local* or *shared* protocols. Typical examples

of local coin-tossing are the algorithms of Ben-Or [4] and Bracha [5]. Shared coins are based on cryptographic distributed algorithms that provide the same random value to all participants. All the randomized algorithms work in rounds, and the complexity of the algorithm (in terms of time and messages) correlates to the number of rounds. Shared coin algorithms, such as ABBA [6], typically have an expected small, constant number of rounds, but require the execution of computationally complex cryptographic protocols. In embedded distributed systems, in which the participants typically have only small computation power, the computational complexity can render them impractical. On the other hand, local coin algorithms in general require an expected exponential number of rounds.

The coin tossing is the point where an infrastructure can provide essential support. A central infrastructure server can provide consistent random number to all participants. Let us first assume a crash-stop infrastructure. In this case, all nodes try to contact the infrastructure to obtain the random value. The infrastructure reveals this value as soon as sufficiently many nodes have requested the value. This way, all nodes obtain the same random number, which results in the low expected number of rounds (as in existing shared coin algorithms), without the need of complex cryptography. If a node does not receive a timely response from the infrastructure, it creates a local random value, resulting in the behaviour of a normal local coin protocol.

The situation becomes more difficult with a corrupt infrastructure. In the worst case, the malicious infrastructure might distribute "bad" random values in a way that inhibits consensus termination. This problem can be mitigated in practice by limiting the infrastructure interaction to a small number of rounds. If the infrastructure behaves correctly, it is highly probable that the algorithm will terminate within this bound. Lack of termination within the limit is a strong indication of infrastructure corruption. The normal local-coin algorithm that is used after the bound will guarantee eventual termination with probability 1. The only adverse effect that a corrupted infrastructure can have is a delay of termination for a few rounds. In a system model in which the infrastructure is correct with high probability (but not guaranteed to be correct), this approach results in a low expected number of rounds.

4.3 Multi-valued Consensus

The third option for implementing infrastructure-assisted consensus is at the level of *multi-valued* consensus. This section presents such a hybrid multi-valued consensus algorithm that makes use of an infrastructure service if it is available. In addition to the properties defined in the system model, the algorithm provides the following two properties:

- *Fast Termination 1:* If all correct nodes have the same initial value, the algorithm terminates within a constant, small number of steps.
- *Fast Termination 2:* If the infrastructure is available, all correct nodes decide within a constant, small number of steps.

Figure 2 shows a pseudo-code definition of our consensus algorithm. The algorithm is inspired by the multi-valued consensus algorithm defined by Correia et al. [7]. The two main differences to the original algorithm are the interaction step with the infrastructure, and a reduction of message size. Due to space limitations, we do not present a rigorous correctness proof here, but only give an informal discussion of the algorithm.

At the beginning, all nodes broadcast their local value to all nodes including the central server p_s. Next, p_s selects a value (which matches the value of the correct nodes if all of them propose the same value) and broadcasts this value with a *justification* (lines 14/15). The *justification* of a value v is the set of $f + 1$ nodes from which an INIT message for v has been received. The *justification* of $\perp$ is a set of $n-f$ nodes, in which no $f+1$ nodes propose the same value. The justification set of a value can efficiently be encoded by n bits (1 bit per node). This compact justification is a significant difference to the original algorithm [7], which instead of using a vector of $n - f$ received messages for justifying values. The bit encoding is sufficient to validate messages, and is more appropriate for embedded systems, as it significantly reduces the amount of network traffic and the demand for local memory.

If all correct nodes are able to timely interact with a correct p_s, they all obtain an identical, justified COORD value. A COORD value is *justified* if it contains a justification set such that an INIT message has been

Function MVConsensus(est_i)
(1) RBcast(INIT(est_i));
(2) wait until (valid COORD(lv) has been delivered or TIMEOUT expired);
(3) if (justified COORD(lv) has been delivered) then $est_i \leftarrow lv$;
(4) else wait until (($n - f$) INIT messages have been delivered);
(5) if ($\exists v$: #INIT(v)$\geq f + 1$) then $est_i \leftarrow < v, \textit{justification} >$ else $est_i \leftarrow < \perp, \textit{justification} >$;
(6) RBcast(PRE(est_i));
(7) wait until (($n - f$) justified PRE(x) messages have been delivered);
(8) if (#PRE($x_1 \neq \perp$)$\geq f + 1$, #PRE(x_2)$= 0$ for $x_2 \notin (x_1, \perp)$) then $b_i \leftarrow 1$ else $b_i \leftarrow 0$;
(9) $c_i \leftarrow$ BinaryConsensus(b_i);
(10) if $c_i = 0$ then return $\perp$;
(11) wait until (($f + 1$) PRE($< v \neq \perp, \textit{justification} >$) have been received);
(12) return v;

Function Deliver (INIT(v)) at Coordinator p_s
(13) if (($n - f$) INIT(v) messages or $f + 1$ identical INIT(v) messages have been delivered):
(14) if ($\exists v$: #INIT(v)$\geq f + 1$) then $lv :=< v, \textit{justification} >$ else $lv :=< \perp, \textit{justification} >$
(15) RBcast(COORD(lv));

Figure 2: Infrastructure-assisted consensus algorithm

received from all of its members, and the messages from the justification set proof the validity of the value chosen by the coordinator (i.e., for a value $v \neq \perp$, there is a set of $f + 1$ identical messages, and for $v = \perp$, there is a set of $n - f$ message without a subset of $f + 1$ identical messages). If the interaction with p_s fails, a node calculates its own justified value.

Next, every node broadcast a PRE message with a value and its justification obtained in the step before. In line (7) the justification of all PRE message can be verified as described for the COORD value. Note that it may happen that a PRE message is not justified when it is delivered (as a required INIT message might not yet have been received), but it may later become justified through the reception of the missing INIT message. The justification ensures that if all correct nodes propose the same value v_1, then no other value $v_2 \neq \perp$ may get a justification (as this would require the justification from $f + 1$ nodes), nor may $v_2 = \perp$ get a justification (as any $n-f$ nodes contain at least $n - 2f$ correct nodes (i.e., with $n \geq 3f + 1$, at least $f + 1$ correct nodes).

Line (8) ensures that if a node selects $b = 1$ because of PRE messages for a value x_1, no other node selects $b = 1$ for a different value x_2. If one node selects x_1, there are at least $f + 1$ PRE messages for this value, so all other nodes receive at least one of these messages. If all propose the same value, there will be only PRE messages for that value, causing all correct nodes to propose $b = 1$.

We assume the use of a randomized binary consensus such as that of Bracha [5], which guarantees that if all correct nodes start a round with identical values, they decide in the same round. The two fast termination properties follow directly from this property and the observation that if either all correct nodes have identical initial values or all of them successfully interact with the infrastructure, they all propose the same value ($b = 1$) to binary consensus.

The selection of the TIMEOUT value (in line 2) has a direct impact on the efficiency of the algorithm. A TIMEOUT value too short will cause the failure of infrastructure interaction. In this case, the algorithm will work only with the weaker guarantees of distributed consensus without infrastructure. A large TIMEOUT value will delay consensus execution for a large period of time in case that the infrastructure is unavailable or corrupt.

4.4 Final remarks

Comparing the three approaches, the integration of infrastructure interaction at the multi-valued consensus level seems to be the most promising. If the infrastructure is available in this step, all correct nodes will propose the same value to binary consensus, ensuring fast termination. This variant could be combined with infrastructure-assisted binary consensus. The combination ensures fast termination in case that the infrastructure is unavailable at the start of the consensus, but becomes available later during the binary consensus phase. In addition, the infrastructure interaction at the reliable multicast level would help to reduce the total number of message, but only if the infrastructure does not show malicious behaviour.

5 Conclusion

In this paper, we have discussed approaches for efficiently solving the consensus problem in distributed embedded systems in realistic environments in which an ad-hoc network and an infrastructure network are simultaneously available. The general idea is that if the infrastructure is not available, the participants execute a randomized consensus algorithm with probabilistic termination guarantees using the ad-hoc network. If the infrastructure is available, the participants take advantage of this and achieve consensus with better termination guarantees.

This paper has presented on-going work. The presented ideas still lack an experimental validation, which should provide real data about the relative behaviour of the proposed approaches and of previously published consensus algorithms. An extended version of this document will include a formal correctness proof. An issue to be investigated in the future is using distributed access points instead of a central server to support the consensus progress. Furthermore, the memory consumption of the protocol, for example for communication buffers, is an important issue in embedded systems. This aspect of the protocol should be accurately examined, and the worst-case memory consumption of the protocols should be minimized. We strongly believe that tailored consensus solution will help to construct dependable distributed embedded systems.

References

[1] M. K. Aguilera and S. Toueg. Failure detection and randomization: A hybrid approach to solve consensus. *SIAM J. Comput.*, 28(3):890–903, 1999.

[2] D. Angluin, M. J. Fischer, and H. Jiang. Stabilizing consensus in mobile networks. In *DCOSS*, pages 37–50, 2006.

[3] N. Badache, M. Hurfin, and R. J. de Araújo Macêdo. Solving the consensus problem in a mobile environment. In *IPCCC*, pages 29–35. IEEE, 1999.

[4] M. Ben-Or. Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In *PODC '83*, pages 27–30. ACM Press, 1983.

[5] G. Bracha. An asynchronous [(n - 1)/3]-resilient consensus protocol. In *PODC '84: Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM Press, 1984.

[6] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology*, 18(3):219–246, July 2005.

[7] M. Correia, N. F. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Computer Journal*, 41(1):82–96, Jan 2006.

[8] V. Drabkin, R. Friedman, and M. Segal. Efficient byzantine broadcast in wireless ad-hoc networks. *dsn*, 00:160–169, 2005.

[9] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[10] A. Mostefaoui, M. Raynal, and F. Tronel. The best of both worlds: A hybrid approach to solve consensus. In *DSN '00: Proc. of the 2000 Int. Conf. on Dependable Systems and Networks*, pages 513–522, 2000.

[11] H. Seba, N. Badache, and A. Bouabdallah. Solving the consensus problem in a dynamic group: An approach suitable for a mobile environment. In *ISCC '02: Proc. of the 7th Int. Symp. on Computers and Communications*, page 327. IEEE Computer Society, 2002.

[12] S. C. Wang, W. P. Yang, and C. F. Cheng. Byzantine agreement on mobile ad-hoc network. *2004 IEEE Int. Conf. on Networking, Sensing and Control*, 1:52–57, Mar. 2004.

[13] W. Wu, J. Cao, J. Yang, and M. Raynal. A hierarchical consensus protocol for mobile ad hoc networks. In *PDP '06: Proc. of the 14th Euromicro Int. Conf. on Parallel, Distributed, and Network-Based Processing*, pages 64–72. IEEE Computer Society, 2006.