Byzantine Fault-Tolerant Consensus in Wireless Ad hoc Networks

Henrique Moniz, Nuno F. Neves, Miguel Correia

Abstract—Wireless ad hoc networks, due to their inherent unreliability, pose significant challenges to the task of achieving tight coordination amongst nodes. The failure of some nodes and momentary breakdown of communications, either of accidental or malicious nature, should not result in the failure of the entire system. This paper presents an asynchronous Byzantine consensus protocol - called Turquois - specifically designed for resource-constrained wireless ad hoc networks. The key to its efficiency is the fact that it tolerates dynamic message omissions, which allows an efficient utilization of the wireless broadcasting medium. The protocol also refrains from computationally expensive public-key cryptographic during its normal operation. The protocol is safe despite the arbitrary failure of $f < \frac{n}{3}$ nodes from a total of n nodes, and unrestricted message omissions. Progress is ensured in rounds where the number of omissions is $\sigma \leq \lceil \frac{n-t}{2} \rceil (n-k-t)+k-2$, where k is the number of nodes required to terminate and $t \leq f$ is the number of nodes that are actually faulty. These characteristics make Turquois the first consensus protocol that simultaneously circumvents the FLP and the Santoro-Widmayer impossibility results, which is achieved through randomization. Finally, the protocol was prototyped and subject to a comparative performance evaluation against two well-known Byzantine fault-tolerant consensus protocols. The results show that, due to its design, Turquois outperforms the other protocols by more than an order of magnitude as the number of nodes in the system increases.

Index Terms—Wireless ad hoc networks, Byzantine Fault Tolerance, Consensus, Distributed Algorithms.

1 Introduction

Wireless ad hoc networks have been the subject of considerable attention within the research community for the past few years. Unlike *managed* wireless networks, which have infrastructural support from fixed components (e.g., an access point), wireless ad hoc networks exist without any form of centralized control. There is no notion of infrastructure and every node, in principle, plays an equal role on the network operation. The decentralized nature of these networks makes them particularly suited for emergency situations like natural disasters and military conflicts, where the reliance on a single point of failure is not only inappropriate, but maybe even unattainable.

The ability for nodes to conduct coordinated activities is of paramount importance in many ad hoc networking applications. Nodes that are within communication range of each other often have to share a common physical resource. Hence, some form of coordination is required such that the access to that resource is fair and efficient. For example, it can be the electromagnetic spectrum, where nodes synchronize their broadcasting periods to prevent overlap [1], [2]; it can be a road, where a set of semi-automated vehicles standing at an intersection have to decide on the order in which they cross [3]; or it can even be the airspace, where aircraft coordinate their maneuvering in order to avoid collisions [4], [5]. All these activities require some sort of agreement among the nodes, which has to be performed in a dependable way.

The consensus problem is a fundamental abstraction of this necessity for agreement in a distributed system. Informally, the problem states that every node proposes a value, and then the nodes have to decide on a common result. Basically, any kind of coordinated activity amongst the nodes of a distributed system can be reduced to consensus. Consequently, a consensus primitive can be applied by the nodes of a wireless ad hoc network to coordinate their actions. Despite being simple to describe, consensus is far from being a trivial problem and has associated impossibility results in systems where nodes or communication links can fail [6], [7]. Wireless networks, in particular, are inherently unreliable. Environmental phenomena such as interference, fading, and collisions give rise to pervasive communication failures, and node mobility may result in momentary disconnection. In addition, wireless ad hoc networks are usually resource-constrained. They usually have less bandwidth than wired local networks, and the computational power of their nodes is often more restricted.

To make matters more challenging, ad hoc networks can be deployed in hostile environments in which both the nodes and the communication can be attacked. In particular, nodes may exhibit arbitrary (or *Byzantine*) behavior due to intrusions and actively strive to disrupt the correct operation of the system. Over the years, several Byzantine fault-tolerant protocols have been proposed for LAN settings, for example, to build replicated services and group communication protocol stacks (e.g., [8], [9]). However, very little work has been done in designing Byzantine fault-tolerant protocols for wireless ad hoc networks.

This paper aims at conciliating Byzantine fault tolerance

During this work, H. Moniz was with the Large-Scale Informatics System Laboratory (LASIGE) of the University of Lisbon and with Microsoft Research Cambridge. N. F. Neves is with the Large-Scale Informatics System Laboratory (LASIGE) of the University of Lisbon. M. Correia is with INESC-ID/IST.

This work was partially supported by the EC through project FP7-257475 (MASSIF), by the FCT through the Multiannual program, project PTDC/EIA-EIA/113729/2009 (SITAN), and project PEst-OE/EEI./LA0021/2011 (INESC-ID).

with the unreliable and resource-constrained nature of ad hoc networks. To achieve this, the paper focuses on the problem of Byzantine fault-tolerant binary consensus for single-hop wireless ad hoc networks, while assuming a system model that closely matches the reality of wireless environments. In particular, it is assumed that nodes are subject to transitory disconnection (due to mobility or unreliable communication) and permanent corruption by a malicious entity. The focus on the single-hop scenario is directly related to the fact that nodes within direct communication range of each other frequently have to synchronize their actions due to their physical proximity. While consensus may also be useful in multi-hop scenarios, any single-hop protocol can be adapted to a multi-hop scenario if supported by an adequate routing layer (e.g., [10], [11]), which is a well-studied problem in the context of wireless networks.

A system model that accurately captures the fundamental characteristics of the environment is crucial for efficiency. Since a wireless network provides a natural broadcasting medium, the cost of transmitting a message to multiple nodes can be just the same of sending it to a single one, as long as they are within communication range. Properly exploited, this property can have a profound impact on performance. However, to take advantage of this property one needs to depart from the traditional modeling assumptions of Byzantine fault-tolerant systems. Usually these assume a reliable point-to-point communication model, which hinders any possibility of taking advantage of the broadcasting medium (because it forces the implementation of end-to-end message delivery mechanisms, e.g., TCP). Thus, the underlying model should also embrace the inherent unreliability of radio communications.

To this end, we propose a model that derives from the *communication failure model* introduced by Santoro and Widmayer [7]. Their model assumes the existence of dynamic and transient transmission faults, meaning that any communication from one node to another can be faulty at one moment and be correct at another. In a wireless environment, this implies that any broadcast message may be delivered non-uniformly by the intended recipients. Some of them may deliver the message, while others may not. Under particularly harsh conditions, like a jamming attack, even all messages may be lost during a period of time.

More concretely, our model assumes an asynchronous system composed of n ad hoc nodes where a subset f of them may be compromised by a malicious adversary (with $f < \frac{n}{3}$). Compromised nodes can fail in an arbitrary manner, namely by sending messages with erroneous content or by simply becoming silent. Therefore, we will consider that potentially all transmissions from these nodes might be lost (or discarded), either due to network omission faults or bad behavior. Additionally, we will assume the existence of *dynamic omission transmission faults* that might affect the communications between correct nodes. Our consensus protocol will ensure progress towards a decision in rounds where the number of omissions of messages transmitted between correct nodes is bounded by $\sigma \leq \lceil \frac{n-t}{2} \rceil (n-k-t)+k-2$ (where k is the number of nodes required to decide and $t \leq f$ is the number of nodes that are actually faulty). If a higher number of faults occur, then the protocol always ensures safety, but progress might be stopped or continue at a slower pace until the network starts to lose less messages.

The paper has the following contributions:

(1) A binary consensus protocol, named Turquois[1], designed to tolerate a combination of Byzantine nodes and dynamic omission transmission faults.

(2) Since the system is asynchronous and can have both Byzantine nodes and dynamic omission faults, consensus is bound by the impossibility results of [6], [7]. Turquois circumvents these impossibility results by employing randomization, ensuring termination with probability 1.

(3) A novel mechanism for broadcast message authentication that resorts to an inexpensive hashing operation instead of typical public-key cryptography, preserving the computational restrictions usually associated with mobile nodes and increasing efficiency.

(4) An extensive performance evaluation of Turquois, both in a network simulator and in a real-world testbed, which includes a detailed comparison with two well-known Byzantine fault-tolerant binary consensus

The remainder of the paper is organized as follows. Section 2 discusses the related work. Section 3 presents the system model and defines the k-consensus problem, which is essentially the classic consensus problem adapted to our system model. Section 4 presents Turquois, our randomized algorithm to k-consensus that tolerates both dynamic omission failures and Byzantine nodes. Section 6 evaluates the performance of Turquois in both a real-world testbed and a simulation environment. Section 7 presents our conclusions. Finally, the complete correctness proof of Turquois is presented in Appendix A.

2 Related Work

Over the past decade, there have been some contributions to the solution of consensus in wireless ad hoc networks, however, almost all of them did not consider the presence of Byzantine nodes. Research on Byzantine fault-tolerant protocols for wireless environments has been practically restricted to broadcasting problems [12], [13], [14], [15], [16].

Consensus in wireless ad hoc networks. Concerning the problem of consensus, Badache et al. were the first to present a protocol specifically for wireless environments [17]. Their solution considers that mobile hosts (MHs) are connected to fixed mobile support stations (MSSs), which are assumed to be fully connected. To solve consensus, each MH communicates the initial proposal value to the respective MSS. The MSSs execute amongst themselves the Chandra-Toueg consensus protocol using a $\diamond S$ failure detector [18], and then communicate the decision value to the associated MHs. Later on, this work was extended by Seba et al. to take into consideration the dynamism in the set of MSSs executing consensus due to the handover of MHs [19].

Wu et al. describe a hierarchical consensus protocol for mobile ad hoc networks [20]. Their protocol selects a subset of predefined mobile nodes to act as *clusterheads*, which

1. Turquois: 1. a semiprecious stone, typically opaque and of a sky-blue color; 2. french for Turk, historic enemy of the Byzantine.

take essentially the same role of the MSSs in the protocol of Badache et al. [17]. The *clusterheads* gather the initial values of their associated nodes and execute consensus using a $\diamond P$ failure detector. The decision is then propagated from the *clusterheads* to the nodes. Vollset et al. present randomized consensus protocols that tolerate crashing nodes and arbitrary topological changes [21]. Their solution, however, requires a fairness condition where correct nodes are not permanently disconnected.

Consensus with dynamic omission failures. The research discussed so far in this section assumes reliable links. In this paper, we address the consensus problem under a model that extends the *communication failure model* of Santoro and Widmayer [7], [22]. The communication failure model has an associated impossibility result, stating that there is no deterministic solution to the problem of *k-agreement* (i.e., $k > \lceil \frac{n}{2} \rceil$ nodes decide the same value 0 or 1) if more than $n-2$ messages can be lost at each synchronous communication round. This is a very restrictive result because a single node crash causes n omission failures per round, thus preventing consensus.

Chockler et al. described a consensus algorithm for a system where nodes can fail by crashing and messages can be lost due to collisions [23]. Their protocol can solve consensus due to the additional power offered by collision detectors, which allow nodes to take measures to recover from message losses. Message omissions other than those due to collisions, however, are not covered by their model.

Borran et al. [24] address consensus under the *heard-of model* (HO) [25], which permits a fine-grained specification of the patterns of message delivery allowed for the problem to be solvable. This work uses the HO model to express the Paxos algorithm [26] and extend it with a communication layer for wireless networks, which provides a leader election service. The reliance on a leader may not be very appropriate in some ad hoc scenarios, and the problem of dynamic omission transmission failures is not taken in consideration because the protocol assumes periods of reliable and delay bounded message deliveries.

Biely et al. also employs the HO model to distinguish cases where the fault pattern exceeds the upper bound of Santoro and Widmayer, but not in a harmful way to the system (e.g., $n-1$ faults are harmful if they originate at the same node, but may not be if they originate each one at a different nodes) [27]. The work of Schmid et al. presents an analogous contribution in the sense that it limits the number of faults that each node may experience [28]. None of these two contributions, however, deal with the essence of the Santoro-Widmayer impossibility result because faults are artificially restricted.

Moniz et al. address this issue by presenting a randomized consensus algorithm that tolerates a new upper bound of $\lceil \frac{n}{2} \rceil (n-k) + k - 2$ dynamic omission failures per communication round, regardless of their pattern [29]. The protocol described in this paper tolerates not only dynamic transmission omission faults, but also a static, *a priori* unknown, subset of Byzantine nodes.

Consensus with message loss. Some classical protocols in the literature tolerate arbitrary message loss in their channels (e.g., Paxos [26], [30], BFT [31], and Fast Byzantine Paxos [32]). As with Turquois, these protocols maintain safety despite unrestricted message loss. There protocols, however, rely on a leader node for progress. Liveness is ensured only if correct nodes agree on the identity of the leader and can communicate with the leader in a reliable and timely manner. This assumption is what allows these protocols to circumvent the Santoro-Widmayer impossibility result.

More recently, Borran & Schiper introduced a protocol that, like ours, is leader-free, tolerates Byzantine nodes, and is always safe regardless of the number of omission faults [33]. To circumvent the Santoro-Widmayer impossibility result, their model assumes the existence of a Global Stabilization Round (GSR), after which communication between correct nodes is assumed to be reliable. In other words, there are no omission faults in the system involving correct nodes after a specific, but unknown, point in time (i.e., the GSR).

3 System Model and Problem Definition

System model. The system is composed by a fixed and known set of n nodes, each one running a single process belonging to $\Pi = \{p_0, p_1, ..., p_{n-1}\}$. The communication between processes proceeds in asynchronous broadcast rounds. At each round, every node $p_i \in \Pi$ transmits a message m to every node $p_j \in \Pi$, including itself, by invoking `broadcast(m)`. A round r is defined as the r^{th} time that nodes invoke the `broadcast()` primitive and is triggered by a clock tick local to each node.

The fault model assumes that up to f nodes can be Byzantine, and that these nodes may fail in an arbitrary way. For example, a Byzantine node can become silent, send messages with wrong values, or collude with other Byzantine nodes to disrupt the correct operation of the system. Such nodes are said to be *faulty*, while nodes that follow the algorithm are called *correct*.

The fault model also accommodates dynamic omission failures in message transmissions amongst correct nodes. A transmission between two correct nodes p_i and p_j is subject to an omission failure if the message broadcast by p_i is not received by p_j. The number of omission failures that can occur per round is unrestricted, in the sense that safety properties are always guaranteed. However, in order to ensure progress, we will make the following fairness assumption: given an unbounded number of rounds, there are infinitely many rounds in which the number of omission faults that affect correct nodes is bounded by a σ value (see protocol description). If a message m transmitted by node p_i to node p_j is not subject to a dynamic omission failure and both nodes are correct, then m is eventually received by p_j.

Cryptographic functions employed in the protocol are secure and can not be subverted by an adversary, and each node $p_i \in \Pi$ can call a local random bit generator to obtain unbiased bits observable only by p_i.

Problem definition. The paper addresses the *k-consensus* problem. This problem considers a set of n nodes where each node p_i proposes a binary value $v_i \in \{0, 1\}$, and at least k of them have to decide on a common value proposed by one of the nodes (with $\frac{n+f}{2} < k \leq n-f$). The remaining non-Byzantine nodes (at most $n-k$) do not necessarily have to decide, but if they do, they are not allowed to decide on a different value. Our problem formulation is designed to accommodate a randomized solution and is formally defined by the properties:

- *Validity.* If all correct nodes propose the same value v, then any correct node that decides, decides v.
- *Agreement.* No two correct nodes decide differently.
- *Termination.* At least k correct nodes eventually decide with probability 1.

The consensus problem in its binary variant can either be used in itself (e.g., Atomic Commitment [34]) or as a building block for other useful protocols. It is a well-studied problem of how to apply binary consensus as a black box to solve, for example, multi-valued consensus or atomic broadcast [35], [36], [37], [38], [39], [40].

4 Turquois: Byzantine k-Consensus

The Turquois algorithm allows k nodes out of n to reach consensus on a binary value $v \in \{0,1\}$ (see Algorithm 1). Correctness is maintained as long as the number of Byzantine nodes is bounded by $f < \frac{n}{3}$. Furthermore, the algorithm ensures safety (i.e., the validity and agreement properties) despite an unrestricted number of transmission omission faults. Progress towards termination is guaranteed in rounds where the number of omission faults is $\sigma \leq \lceil\frac{n-t}{2}\rceil(n-k-t)+k-2$, where $t \leq f$ is the number of nodes in the system that are actually faulty. Turquois is a randomized algorithm as it relies on each node p_i having access to a *local coin*[2] mechanism that returns random bits observable only by p_i (e.g., [43], [44]). The first local coin protocol was proposed by Ben-Or, of which our protocol is reminiscent [43].

Intuitively, the algorithm is structured around a cycle of three phases. The phases of the cycle are called CONVERGE, LOCK, and DECIDE. A node is in each one of these phases when its phase value is, respectively, $\phi_i \pmod 3 = 1$, $\phi_i \pmod 3 = 2$, and $\phi_i \pmod 3 = 0$. The cycle is repeated as long as it is necessary for the nodes to decide on a common value. Each phase in the cycle plays a specific purpose. In the CONVERGE phase, nodes try to converge their proposal values by updating their proposal value v_i to the value observed most times in that phase. Next, in the LOCK phase, nodes try to *lock* on a single value $v \in \{0,1\}$. Each node either sets its proposal value to this v or to a value $\perp$ indicating a lack of preference. Finally, in the DECIDE phase, nodes attempt to decide on the value *locked* on the previous phase. If a node is not able to decide at the end of a DECIDE phase, it may propose a random value at the beginning of the following cycle. This random step guarantees that eventually there is a cycle that starts with enough correct nodes proposing the same value. When this happens, k correct nodes necessarily decide by the end of that cycle.

Additionally, the algorithm resorts to a message validation procedure for every message received (lines 8-9), which is described in detail in Section 5. The validation procedure limits the power of Byzantine nodes by providing authentication and enforcing congruency in the proposal values. It is useful to analyze the algorithm being mindful that the validation procedure provides these properties. This design is akin to the contribution of Srikanth & Toueg on simulating authenticated broadcasts for Byzantine algorithms [45].

2. As opposed to a *shared coin* that returns bits observable by all nodes (e.g., [41], [42]).

Algorithm 1: Turquois: a Byzantine k-consensus algorithm

```
Input: Initial binary proposal value $proposal_i \in \{0,1\}$
Output: Binary decision value $decision_i \in \{0,1\}$
1  $\phi_i \leftarrow 1$;
2  $v_i \leftarrow proposal_i$;
3  $status_i \leftarrow undecided$;
4  $V_i \leftarrow \emptyset$;

   TASK T1:
5  when local clock tick do
6      broadcast($\langle i, \phi_i, v_i, status_i \rangle$);
7  end

   TASK T2:
8  when $m = \langle j, \phi_j, v_j, status_j \rangle$ is received do
9      $V_i \leftarrow V_i \cup \{m : m$ is valid$\}$;
10     if $\exists\langle *, \phi, v, status\rangle \in V_i : \phi > \phi_i$ then
11         $\phi_i \leftarrow \phi$;
12         if $\phi \pmod 3 = 1$ and $v$ is the result of a coin flip then
13             $v_i \leftarrow$ coin$_i$();
14         else
15             $v_i \leftarrow v$;
16         end
17         $status_i \leftarrow status$;
18     end
19     if $|\{\langle *, \phi, *, *\rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
20         if $\phi_i \pmod 3 = 1$ then                        /* phase CONVERGE */
21             $v_i \leftarrow$ majority value $v$ in messages with phase $\phi = \phi_i$;
22         else if $\phi_i \pmod 3 = 2$ then                       /* phase LOCK */
23             if $\exists v \in \{0,1\}$: $|\{\langle *, \phi, v, *\rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
24                 $v_i \leftarrow v$;
25             else
26                 $v_i \leftarrow \perp$;
27             end
28         else                                                  /* phase DECIDE */
29             if $\exists v \in \{0,1\}$: $|\{\langle *, \phi, v, *\rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ then
30                 $status_i \leftarrow decided$;
31             end
32             if $\exists v \in \{0,1\}$: $|\{\langle *, \phi, v, *\rangle \in V_i : \phi = \phi_i\}| \geq 1$ then
33                 $v_i \leftarrow v$;
34             else
35                 $v_i \leftarrow$ coin$_i$();
36             end
37         end
38         $\phi_i \leftarrow \phi_i + 1$;
39     end
40     if $status_i = decided$ then
41         $decision_i \leftarrow v_i$;
42     end
43 end
```

Each node p_i has an internal state comprised by three variables: (1) the *phase* $\phi_i \geq 1$, (2) the *proposal value* $v_i \in \{0,1\}$, and (3) the *decision status* $status_i \in \{decided, undecided\}$. Each node starts its execution with $\phi_i = 1$, $status_i = undecided$, while v_i is set to the initial proposal value indicated by the input parameter $proposal_i$ (lines 1-3). Each node also maintains a set V_i, initially set to empty (line 4), where it keeps the messages received throughout the execution of the algorithm.

The algorithm is run in parallel by tasks T1 and T2, which are activated by the respective **when** condition (lines 5 and 8). When activated, a task runs towards completion without any interruption from the other task. Task T1 defines a broadcasting round and is activated periodically upon a local clock tick (lines 5-7). A node p_i broadcasts a message of the form $\langle i, \phi_i, v_i, status_i \rangle$ containing its identifier i and the variables that comprise its internal state.

Task T2 is activated whenever a message arrives (lines 8-43). Some of the messages that a node is supposed to receive may be lost, or may carry invalid content if transmitted by a Byzantine node. Therefore, all arriving messages are subject to a validation procedure that constrains the wrongful actions of Byzantine nodes. Essentially, a message is considered *valid* if it could have been sent by a node that followed the algorithm (details in Section 5). Valid messages are accumulated in set V_i (line 9), while the others are discarded.

Based on its current internal state and the messages accumulated in set V_i, a node p_i performs a state transition, which happens when one of two conditions occur:

1) the set V_i holds some message whose phase value ϕ is *higher* than the current phase ϕ_i of p_i;
2) the set V_i holds more than $\frac{n+f}{2}$ messages whose phase is *equal* to the current phase ϕ_i of p_i.

The first case is as follows (lines 10-18). When the condition is met (line 10), node p_i updates the state to match the state of the received message, with a slight exception. The special instance is the following: if the phase value is $\phi \pmod 3 = 1$ and the value v was obtained from the result of a coin flip (which can be verified from the validation procedure described in Section 5), then p_i executes a local coin flip to determine v_i (lines 12-13). Since it is not possible to force Byzantine nodes into a fair coin flip, this step becomes necessary to guarantee that correct nodes assume a random value.

The second case involves more steps (lines 19-39). The way a node p_i updates its state depends on the value of its current phase number ϕ_i modulo 3. In CONVERGE phases ($\phi_i \pmod 3 = 1$) the proposal value is set to the majority value of all messages with phase value $\phi = \phi_i$ (lines 20-21).

In LOCK phases ($\phi_i \pmod 2 = 2$) the proposal value v_i is updated the following way (lines 22-27): if there are more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with $\phi = \phi_i$ and the same value v, then v_i is set to v (lines 23-24), otherwise it is set to a special value $\perp \notin \{0, 1\}$ indicating a lack of preference (lines 25-26). This step ensures that in the following phase $\phi_i + 1$ every node either proposes the same value $v \in \{0, 1\}$ or $\perp$. Furthermore, if there was unanimity amongst correct nodes at the previous phase $\phi_i - 1$, then every node must set its proposal value to the same value v (since messages with a different value are considered invalid). This will imply that in the next phase $\phi_i + 1$ every node receives the same value $v \in \{0, 1\}$ in all valid messages and decides.

In DECIDE phases ($\phi_i \pmod 2 = 0$), a node sets $status_i$ to $decided$ if there are more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with $\phi = \phi_i$ and the same value $v \neq \perp$ (lines 29-31). The proposal value v_i is set to v if there is at least one message of the form $\langle *, \phi, v, * \rangle$ in V_i with $\phi = \phi_i$ and a value $v \neq \perp$. Otherwise, v_i is set to the value of function `coin()`, which returns a random number 0 or 1, each with probability $\frac{1}{2}$ (lines 32-36). Regardless of the previous steps, the phase is always incremented by one unit (line 38).

At the end of each round, a node p_i checks if $status_i$ has been set to $decided$. If so, it decides by setting the output variable $decision_i$ to the current proposal value v_i (lines 40-42). Further accesses to this variable do not modify its value. Hence, they have no impact on the correctness of the algorithm. The full correctness proof can be found in Appendix A.

As stated before, the algorithm is safe despite unrestricted message omissions and ensures progress in rounds where the number of omissions is $\sigma \leq \lceil \frac{n-t}{2} \rceil (n-k-t) + k - 2$. It is worthy to stress that omissions of message transmissions between correct nodes are accounted in $sigma$. Any faulty behavior involving Byzantine nodes, including message omissions, is already fully captured by $t \leq f$, the number of Byzantine nodes actually present in the system. Furthermore, although $\sigma \leq \lceil \frac{n-t}{2} \rceil (n-k-t) + k - 2$ is a sufficient condition for progress, there may be instances where it is not necessary. For example, when some messages are omitted in one round and are the only ones received in a subsequent round, it may be the case where in both rounds the number of omissions is higher than $sigma$, but both rounds combined accumulate the sufficient messages in some set V_i such that p_i increments its phase number.

Regarding the storage requirements for the algorithm and the message size, these are both $O(n)$. The storage asymptotic upper bound is driven by vector V_i. This vector only needs to maintain messages with phase value $\phi >= \phi - 2$, which are necessary for semantic validation described in the following section. Since the phase value of a node always matches the message received with the highest phase value (lines 10-11), this implies the storage of $O(n)$ messages. The message size is $O(n)$ also because of the semantic validation.

5 Validation of Messages

A node p_j must check the validity of arriving messages before adding them to set V_j. This procedure is fundamental to the correct operation of the protocol because it limits the wrongful actions that a Byzantine node can accomplish. There are two types of validation that a message must pass: authenticity validation and semantic validation. The first guarantees that some of the fields of a message were actually generated by a node p_i, while the second ensures that the contents of a message are congruent with the current execution of the algorithm. A message is deemed *valid* if it passes both tests.

5.1 Authenticity Validation

This form of validation provides (partial) message authentication. More precisely, for any message $\langle i, \phi, v, status \rangle$, it provides to a receiving node p_j assurance that the values of ϕ and v originated at the alleged source node p_i. This statement deserves the following caveat. The authenticity of the $status$ variable is not protected by this mechanism. Consequently, it is possible for a malicious entity to replay a message

$\langle i, \phi, v, status\rangle$ with an arbitrary $status$ value. This, however, does not impact the correctness of the protocol because our semantic validation mechanism (see next section) requires nodes to justify their $status$ based on the received proposal values, therefore, making the attack ineffective.

Authentication is based on a mechanism for generating and verifying *one-time hash-based message signatures* that is particularly efficient for a round-based group communication protocol with a small domain of input values. In our case, the mechanism is devised for an input domain of three values (0, 1, and $\perp$), which represents the possible proposal values that a message can have. To the best of our knowledge, this is the first time such a mechanism is employed in an agreement protocol.

The mechanism is composed by a generic *message authentication* procedure for each phase of the k-consensus protocol, and by a *key exchange* procedure that has to be executed periodically. The message authentication resorts to an efficient one-way hash function H to generate hash values of length h (e.g., SHA-256 or RIPEMD-160) [46]. The key exchange procedure resorts to a more computationally expensive trapdoor one-way function F (e.g., RSA [47]) that is used to sign an array of verification keys. It is assumed that each node p_i has an associated public/private key pair to be used in F, where pu_i is the public key and pr_i is the private key. Every node knows the public key of all other nodes.

5.1.1 Key Exchange

The key exchange procedure generates m secret keys, which are essentially random bit strings of length h, and distributes the corresponding verification keys. These are valid for m phases of the k-consensus protocol. If m is equal to or larger than the number of phases required to reach consensus, then the key exchange procedure only needs to be executed once, at the beginning of the k-consensus protocol. Potentially, this scheme can be further optimized so that a single key exchange can span multiple instances of the k-consensus. Nevertheless, for clarity purposes, we describe the scheme assuming only a single instance.

For each node p_i, the key exchange $e \geq 1$ consists of the following steps. Node p_i generates a two-dimensional array SK_i of secret keys, such that each element $SK_i[\phi][v]$ is a random bit string of length h, with $(e-1)m+1 \leq \phi \leq em$ and $v \in \{0, 1, \perp\}$[3]. It then creates an equivalent two-dimensional array VK_i of verification keys, such that each element $VK_i[\phi][v] = H(SK_i[\phi][v])$. Finally, the verification keys array VK_i is signed using the trapdoor one-way function F and the private key pr_i, and then both the VK_i and the signature are disseminated to the other nodes using an out-of-band reliable channel.

When VK_i arrives to a node, the correctness of the keys is confirmed by verifying the signature with the public key of p_i, and then the array is stored for future use. For efficiency purposes, the first VK_i array can be distributed offline along with the public keys. Subsequent arrays may be transmitted during idle periods of the system such that interference with normal execution is kept to a minimum.

5.1.2 Message Authentication

For any phase ϕ, a message $\langle i, \phi, v, status\rangle$ broadcast by node p_i is authenticated by attaching $SK_i[\phi][v]$. When a node p_j receives the message, it applies the hash function to $SK_i[\phi][v]$ and verifies if $H(SK_i[\phi][v])$ is equal to $VK_i[\phi][v]$. If they are equal, then by the properties of cryptographic hash functions ϕ and v originated at p_i.

5.2 Semantic Validation

The semantic validation ensures that the values carried by the three states variables within a message are congruent with the execution of the algorithm. For example, if, at phase $\phi = 1$, every correct node broadcasts the same value 0, then it is not possible for a node that is executing the protocol to send a proposal value of 1 at phase $\phi + 1$. Therefore, if such proposal arrives, then it must have been sent by a Byzantine node, and it can be discarded without impacting the protocol. In practice, this validation mechanism restricts the way that Byzantine nodes may lie.

There are two ways for the congruency of messages to be verified: one is *implicit* and the other is *explicit*. The implicit way is based on whenever a node receives a message, it checks if enough messages have arrived to justify the values carried by the message just received. For example, if a node has in set V_i more than $\frac{n+f}{2}$ messages with phase ϕ, then, for any message of the form $\langle *, \phi{+}1, *, *\rangle$, its phase value is implicitly valid.

The explicit way is based on broadcasting, along with the message, the previous messages that justify the values of the state variables. For example, a message with phase $\phi + 1$ can be justified by having appended more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, *, *\rangle$ (and, naturally, the appended messages must also pass the validity checks).

Our current implementation of the algorithm resorts to both techniques. First, a node tries an implicit validation, which is optimistic by nature, and is much more efficient because messages are allowed to be kept small. However, if, for the following clock tick, a node is forced to broadcast the same message, then explicit validation is employed by appending the justifying messages.

Each of the state variables carried by a message are validated independently. A message passes this validation test if all three variables pass in their individual test. The messages required to validate each variable may sometimes overlap. We explain in more detail how to perform the validations. The pseudocode is in Algorithm 2. The pseudocode applies to both implicit and explicit validation. For the explicit validation, we assume any appended messages are also present in set V_i.

5.2.1 Phase value

The phase value ϕ of a message of the form $\langle *, \phi, *, *\rangle$ requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, *, *\rangle$ to be considered valid (lines 1-3).

3. In practice, $SK_i[\phi][\perp]$ only needs to be generated if $\phi \pmod 3 = 0$ because $\perp$ is an acceptable proposal value only in such phases.

Algorithm 2: Semantic validation procedure

Input: Incoming message of the form $\langle j, \phi_j, v_j, status_j \rangle$
Output: A boolean value indicating if the message is valid

```
// phase value
if not $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_j - 1\}| > \frac{n+f}{2}$ then
    return false;
end
// proposal value
if $\phi_j \pmod 3 = 2$ then
    if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 1 \wedge v = v_j\}| > \frac{n+f}{2}/2$ then
        return false;
    end
else if $\phi_j \pmod 3 = 0$ then
    if $v_j \in \{0, 1\}$ then
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 1 \wedge v = v_j\}| > \frac{n+f}{2}$ then
            return false;
        end
    else if $v_j = \bot$ then
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 2 \wedge v = 0\}| > \frac{n+f}{2}/2$ then
            return false;
        end
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 2 \wedge v = 1\}| > \frac{n+f}{2}/2$ then
            return false;
        end
    end
else if $\phi_j \pmod 3 = 1 \wedge \phi_j > 1$ then
    if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 2 \wedge v = v_j\}| > \frac{n+f}{2}$
    and not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_j - 1 \wedge v = \bot\}| > \frac{n+f}{2}$ then
        return false;
    end
end
// status value
if $\phi_j \leq 3$ then
    if not $status_j$ = undecided then
        return false;
    end
else
    if $status_j$ = decided then
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi \pmod 3 = 0 \wedge v = v_j\}| > \frac{n+f}{2}$
        then
            return false;
        end
    else
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi$ is the highest $\phi$ such that $\phi$
        $\pmod 3 = 2 \wedge \phi < \phi_j \wedge v = 0\}| > \frac{n+f}{2}/2$ then
            return false;
        end
        if not $|\{\langle *, \phi, v, * \rangle \in V_i : \phi$ is the highest $\phi$ such that $\phi$
        $\pmod 3 = 2 \wedge \phi < \phi_j \wedge v = 1\}| > \frac{n+f}{2}/2$ then
            return false;
        end
    end
end
return true;
```

5.2.2 Proposal value

The validation of the proposal value varies according to the phase carried in the message. Messages with phase value $\phi = 1$ are the only that do not require validation and are immediately accepted.

- *Messages with phase* $\phi \pmod 3 = 2$: The proposal value v is valid if there are more than $(\frac{n+f}{2})/2$ messages with phase $\phi - 1$ and proposal value v (lines 4-7).
- *Messages with phase* $\phi \pmod 3 = 0$: If the proposal value is $v \in \{0, 1\}$, then it requires more than $\frac{n+f}{2}$ messages with phase $\phi - 1$ and proposal value v. If the proposal value is $\bot$, then it requires more than $(\frac{n+f}{2})/2$ messages of the form $\langle *, \phi - 2, 0, * \rangle$ and more than $(\frac{n+f}{2})/2$ messages of the form $\langle *, \phi - 2, 1, * \rangle$ (lines 8-20).
- *Messages with phase* $\phi \pmod 3 = 1$: The validity of proposal value v in these messages depends if it was obtained deterministically (Line 33) or randomly (Line 35). If obtained deterministically, it requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 2, v, * \rangle$. If set randomly, then it requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, \bot, * \rangle$ (lines 21-25).

5.2.3 Status value

For the $status$ variable, any message with phase $\phi \leq 3$ must necessarily carry value $undecided$ because no node can decide prior to phase 3 (lines 27-30). For messages with $\phi > 3$, a $status = decided$ (and value v) requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ where $\phi \pmod 3 = 0$. A $status = undecided$ requires more than $(\frac{n+f}{2})/2$ messages of the form $\langle *, \phi', 0, * \rangle$ and more than $(\frac{n+f}{2})/2$ messages of the form $\langle *, \phi', 1, * \rangle$, where ϕ' must be the highest $\phi' \pmod 3 = 2$ lower than ϕ (lines 37-42).

6 Performance Evaluation

This section analyzes the performance of Turquois in 802.11b wireless ad hoc networks under both a real-world network testbed and a simulation environment. It is composed by two main sections.

Section 6.1 compares primarily the performance of Turquois against the Bracha's and ABBA protocols. The protocols are executed in the Emulab platform [48], on a testbed composed of up to 16 rack-mounted and Wi-Fi enabled hosts. Like Turquois, both Bracha's and ABBA protocols are leader-free randomized algorithms that achieve optimal resilience on the number of Byzantine nodes. Unlike our protocol, they were not designed with a wireless environment in mind, and employ the typical asynchronous model with reliable point-to-point links.

These protocols were chosen essentially because they are leader-free, a characteristic we deem fundamental for wireless ad hoc networks. While there are faster protocols in the literature (e.g., BFT [31], Fast Byzantine Paxos [32]), their liveness is only guaranteed when there is a unique correct leader, all correct nodes agree on its identity, and the leader can communicate in a timely manner with sufficiently many nodes. These conditions are too strong for a wireless ad hoc network. This is specially true if the environment is considered hostile - an implicit assumption when there are Byzantine failures - and some omission failures can be caused by a malicious attacker. For instance, suppose this attacker has just enough power to jam the communication of a single node. With a leader-based protocol, the attacker can follow a strategy of chasing the designated leader, severely thwarting the protocol execution. For this reason, we considered only leader-free protocols for evaluation.

The protocol of Bracha does not resort to any kind of cryptographic operations, apart from a computationally efficient hash function to authenticate the point-to-point channels, but requires many message exchanges (in complexity order of $O(n^3)$), and the expected worst-case number of rounds to terminate is $O(2^n)$. The ABBA protocol, on the other hand, has message complexity of $O(n^2)$ and terminates in a constant expected number of steps (at most two rounds of three steps each), but relies heavily on expensive public-key cryptography.

Section 6.2 analyzes the performance of Turquois in the ns-3 network simulator [49] and evaluates the impact of several additional parameters - not possible to evaluate under the experimental setting of the previous section - such as a significantly higher number of nodes.

6.1 Protocol Comparison in Emulab

This section evaluates the latency of the Turquois, Bracha's and ABBA protocols under several parameters such as the number of nodes, types of faults in the system, and distribution of the initial proposal values. Other aspects such as the clock tick mechanism of Turquois and the impact of its message authentication mechanism are also evaluated.

6.1.1 Testbed and Implementation

The experiments were carried out on the Emulab testbed [48]. A total of 16 nodes were used, each one with the following hardware characteristics: Pentium III CPU, 600 MHz of clock speed, 256 MB of RAM, and 802.11 a/b/g D-Link DWL-AG530 WLAN interface card. The operating system was the Fedora Core 4 Linux with kernel version 2.6.18.6. The nodes were located on the same physical cluster and were, at most, a few meters distant from each other.

All the protocols were implemented in C. In Turquois, nodes communicate using UDP broadcast. A local clock tick is triggered if one of the following conditions is true: (1) 10 ms have passed since the last broadcast, or (2) the phase value was changed. In both Bracha's and ABBA, the nodes use TCP to communicate because of their requirement of reliable point-to-point links. While the reliable point-to-point links do not have to necessarily be implemented as TCP channels, we use TCP for both the convenience and the maturity of the protocol. Bracha's protocol requires authenticated channels. To this end, we use the IPSec Authentication Header with security associations being established between every pair of nodes before the execution of the protocol. Both Turquois and ABBA employ their own authentication mechanisms. For these protocols, the cryptographic keys were generated and distributed before the execution of the protocols.

6.1.2 Methodology

The performance metric utilized in the experiments is the *latency*. This metric is always relative to a particular node p_i, and it is denoted as the interval of time between the moment p_i proposes a value to a consensus execution, and the moment p_i decides.

The average latency for the whole set of nodes is obtained in the following manner. A signaling machine, which does not participate in the execution of the protocols, is selected to coordinate the experiment. It broadcasts a 1-byte UDP message to the n nodes involved in the experiment. When a node receives such a message, it starts a consensus execution. Nodes record the latency value as described above, and send a 1-byte UDP message to the signaling machine indicating the termination of the execution of the protocol. The signaling machine, upon receiving n such messages, waits five seconds, and recommences the procedure. The *average latency* is obtained by repeating this procedure 50 times, and then by averaging the latencies collected by all nodes. The confidence interval for the average latency is calculated for a confidence level of 95%.

The experiments were carried out for combinations of group size, proposal distribution, and fault load. The group size defines the number of nodes in the system. In our experiments, the values are 4, 7, 10, 13, and 16 nodes. The proposal distribution defines the initial values to be proposed by the nodes. In the *unanimous* proposal distribution all nodes propose the same initial value 1. In the *divergent* distribution nodes with an odd node identifier propose 1, while the others propose 0. The *fault load* defines the type of faults that are injected in the system. In the *failure-free* fault load, all nodes behave correctly. The *fail-stop* fault load makes $f = \lfloor \frac{n-1}{3} \rfloor$ nodes crash before the measurements are initiated. In the *Byzantine* fault load, $f = \lfloor \frac{n-1}{3} \rfloor$ nodes try to keep the correct nodes from reaching a decision by attacking the execution of the protocol. This is accomplished as follows. In both Bracha's and Turquois, a Byzantine node in phase 1 and 2 proposes the opposite value that it would propose if it were behaving correctly, and in phase 3 it proposes the default value $\perp$. This strategy is followed even if messages are potentially considered invalid. In ABBA, since the protocol terminates in a constant number of steps, a Byzantine node does not have much room to delay the execution of the protocol by proposing incorrect values. Instead, it transmits messages with invalid signatures and justifications in order to force extra computations at the correct nodes. Finally, the value of the parameter k in Turquois is set to $k = n - f$ in all fault loads, with $f = \lfloor \frac{n-1}{3} \rfloor$.

6.1.3 Failure-free fault load

Table 1 and Figure 1 show the average latency for every tested combination of group size and proposal distribution, in executions without node failures. By observing the results, it becomes apparent that Turquois performs significantly better than the other two protocols. The difference becomes wider as the number of nodes increases, exceeding an order of magnitude in some cases.

The performance of Turquois stems naturally from its design. Two fundamental reasons contribute to its efficiency. First, the use of UDP broadcast takes full advantage of the shared communication medium. This was only possible because the protocol is able to tolerate dynamic transmission faults. Second, the use of a novel hash-based signature scheme for message validation allows for computational efficiency. The impact of these features is clearly reflected in the results.

Bracha's protocol is the worst contender, showing serious performance degradation due to the $O(n^3)$ message complexity. In addition to being a shared medium, wireless ad hoc networks are restricted in their speed and capacity, and,

Group Size	Average Latency ± Confidence Interval (ms)					
	Turquois		ABBA		Bracha	
	unanimous	divergent	unanimous	divergent	unanimous	divergent
$n = 4$	14.90 ± 4.74	28.67 ± 9.99	74.70 ± 7.93	135.39 ± 28.04	101.06 ± 8.15	127.39 ± 22.99
$n = 7$	26.85 ± 6.18	54.38 ± 12.20	125.81 ± 6.22	253.66 ± 37.93	552.77 ± 31.36	715.15 ± 112.90
$n = 10$	43.15 ± 10.05	71.75 ± 25.05	277.90 ± 12.47	547.42 ± 81.94	1361.90 ± 33.17	2282.23 ± 315.53
$n = 13$	60.94 ± 14.15	128.07 ± 42.51	693.39 ± 103.45	1722.44 ± 295.05	3459.10 ± 100.34	6276.91 ± 734.11
$n = 16$	87.57 ± 22.34	236.31 ± 77.27	1914.54 ± 283.18	4309.51 ± 750.20	7321.41 ± 110.69	10420.00 ± 2640.11

Table 1
Average latency and confidence interval in a 802.11b network with no node failures (latency in milliseconds and confidence level of 95%).

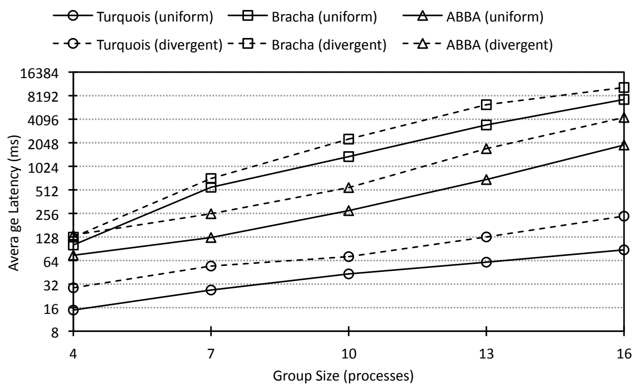

Figure 1. Average latency in a 802.11b network with no node failures (logarithmic scale of base 2).

therefore, a higher number of message transmissions is bound to have a severe cost. The ABBA protocol performs better than Bracha's, but still much worse than Turquois. Despite its $O(n^2)$ message complexity, the fact that, like Bracha's, it still requires the use of TCP channels combined with heavy cryptography proves to be too much of a burden.

The relative difference between proposal distributions was approximately the same across all protocols, with the latency roughly doubling from an unanimous to a divergent proposal distribution. The reason for this is that when nodes propose different values, the protocols usually need to execute for an additional cycle of steps. For example, in Turquois, nodes decide by the end of phase 3 with unanimous proposals, but with divergent proposals they typically decide by the end of phase 6. Under the divergent scenario, the first cycle of steps is usually not enough for nodes to decide, but is sufficient for a significant number of them to converge into the same proposal value, which leads to a decision by the end of the following cycle.

6.1.4 Fail-stop fault load

Table 2 and Figure 2 show the performance of the protocols when $f = \lfloor \frac{n-1}{3} \rfloor$ nodes crash before the execution of the protocols begins. Two observations are clear from these results. First, for all three protocols, there is practically no difference between the two proposal distributions. Since f nodes crash, for every group size tested, exactly $n - f = \lfloor \frac{n+f}{2} \rfloor + 1$ nodes are left in the system. This means that, as the nodes make progress, they necessarily have to receive the same set of messages. Thus, never diverging in their proposal values after the first phase.

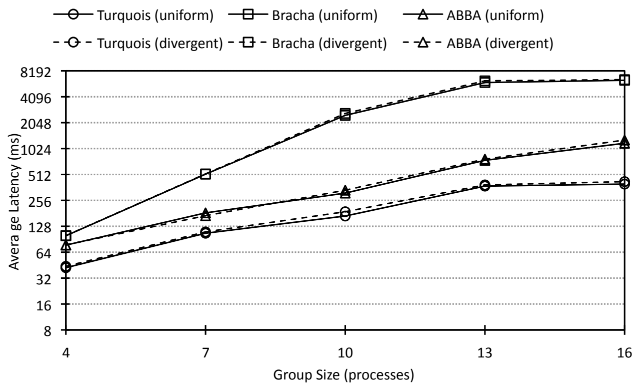

Figure 2. Average latency in a 802.11b network with fail-stop node failures (logarithmic scale of base 2).

The second observation is that, for the unanimous proposal distribution, in most cases the performance of the protocols is worse in the fail-stop scenario than in the fault-free experiments. At a first glance this result seems counterintuitive because when some nodes crash there is less contention on the network and, in principle, the protocols can run faster. The problem is that protocols become more sensitive to message loss when only $n - f$ nodes are present in the system. More retransmissions are needed to ensure that nodes receive enough messages to make progress. Turquois is particularly sensitive to this fact. There are two reasons that explain this: (1) since Turquois uses UDP broadcast, a single collision can result in up to $n - 1$ nodes not receiving a message, while in the protocols that employ TCP one collision results in just one node not receiving the message; (2) furthermore, the timeout mechanism in the current implementation of Turquois is crude when comparing to the sophistication of TCP, and is not adaptable to network conditions nor to the number of nodes involved in the communication. This also explains its proportionally wider confidence interval. An optimization of the retransmission mechanism could significantly improve the performance of Turquois in these scenarios. Nevertheless, Turquois still performs significantly better than the other two protocols with this fault load.

There are two exceptions to the observation that protocols perform better in the failure-free fault load when compared with the fail-stop fault load. They occur in Bracha's and ABBA

Group Size	Average Latency ± Confidence Interval (ms)					
	Turquois		ABBA		Bracha	
	unanimous	divergent	unanimous	divergent	unanimous	divergent
$n = 4$	42.26 ± 30.29	43.84 ± 31.27	77.31 ± 9.17	77.88 ± 9.34	99.29 ± 3.05	99.61 ± 3.17
$n = 7$	106.28 ± 37.98	110.18 ± 22.00	183.20 ± 15.96	169.90 ± 6.18	516.26 ± 26.70	519.76 ± 37.63
$n = 10$	168.45 ± 39.46	188.95 ± 35.05	310.97 ± 15.61	335.93 ± 24.09	2488.75 ± 52.53	2619.35 ± 75.43
$n = 13$	375.00 ± 56.03	387.22 ± 60.06	747.56 ± 44.77	771.68 ± 52.71	5992.63 ± 143.00	6267.88 ± 355.51
$n = 16$	395.96 ± 55.11	422.65 ± 82.41	1180.03 ± 109.18	1284.83 ± 103.64	6362.68 ± 136.64	6469.38 ± 159.40

Table 2
Average latency and confidence interval in a 802.11b network with fail-stop node failures (latency in milliseconds and confidence level of 95%).

when $n = 16$. This indicates that there may be a turning point where the group size becomes more stringent to performance than sensitivity to message loss, although experiments with higher numbers of nodes would be necessary to confirm this.

6.1.5 Byzantine fault load

Table 3 and Figure 3 show the performance of the protocols when $f = \lfloor \frac{n-1}{3} \rfloor$ nodes act according to a malicious strategy. It is interesting to note that the *relative* difference between the unanimous and divergent proposal distributions is similar to the scenario with no node failures, with the latency very roughly doubling in the divergent distribution. Like in the failure-free scenario, this is due to divergent proposal values forcing nodes to execute for extra rounds to reach a decision.

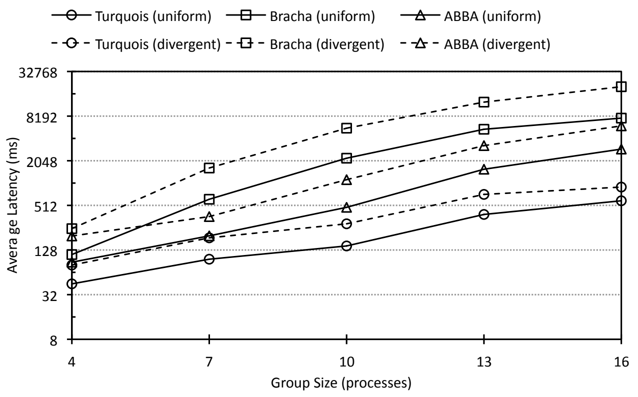

Figure 3. Average latency in a 802.11b network with Byzantine node failures (logarithmic scale of base 2).

When compared directly to the failure-free scenario, this fault load suffers from a performance degradation that becomes increasingly noticeable with a higher group size, specially with a divergent proposal distribution. The reason for this is that many messages broadcast by Byzantine nodes carry values that fail to pass the validation mechanisms of the protocols. The result is that, similarly to the fail-stop scenario, protocols become sensitive to message loss with the added burden of a higher contention (with n nodes broadcasting messages). As for Turquois, despite its non-optimized timeout mechanism making it more sensitive to this issue, it is still the faster protocol.

6.1.6 Timeout Mechanism of Turquois

Under the experiments carried out so far, Turquois employed a constant local clock tick value of 10 ms. Here we analyze how an optimized local clock tick can impact the performance of the protocol. To this end, Turquois was executed with a variable number of nodes - 4 to 16 - and local clock tick value - 2, 5, 10, 20, 50, and 100 ms. Every other parameter was fixed. The protocol was run in a 802.11b network with the failure-free fault load and a unanimous proposal distribution.

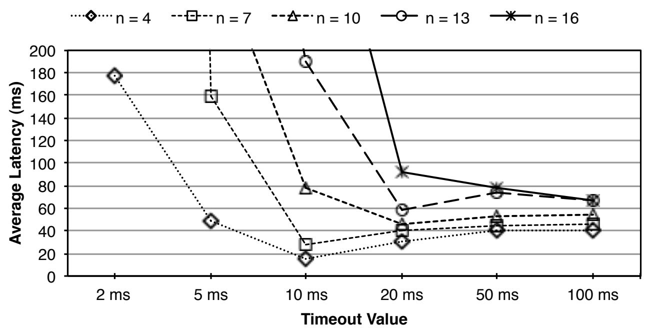

Figure 4. Average latency in a 802.11b network with varying timeout value.

The graph from Figure 4 shows how the latency of the protocol can be affected by the local clock tick value. As it can be observed, a poorly chosen timeout value can severely impair the performance. A low timeout value can be particularly harmful. It generates too much contention in the network, which results in considerable message loss, severely degrading performance. It can also be observed that as the number of nodes increases, the optimal timeout value tends to be higher in order to alleviate the contention created by having extra nodes in the system. These observations indicate that Turquois and other protocol sharing a similar structure could benefit from having an adaptive timeout that self-adjusts in reaction to the network conditions. Some of the authors are currently involved in ongoing investigation regarding this possibility [50].

6.1.7 Cryptography

A key aspect for the performance of Turquois is the novel message authentication it employs. This section measures the impact of our hash-bashed signature mechanism as compared to classical RSA signatures. Figure 5 compares the performance of Turquois using the hash-bashed message authentication mechanism of Section 5 with the alternative of employing

Group Size	Average Latency ± Confidence Interval (ms)					
	Turquois		ABBA		Bracha	
	unanimous	divergent	unanimous	divergent	unanimous	divergent
$n = 4$	44.74 ± 30.16	80.18 ± 33.93	87.65 ± 22.38	197.78 ± 25.25	111.16 ± 6.99	248.66 ± 38.80
$n = 7$	96.20 ± 37.88	186.74 ± 60.54	198.69 ± 17.72	361.53 ± 48.41	619.09 ± 23.40	1634.17 ± 236.21
$n = 10$	145.22 ± 23.21	288.94 ± 64.04	481.83 ± 31.10	1137.94 ± 37.78	2216.42 ± 54.17	5633.47 ± 668.64
$n = 13$	386.39 ± 38.57	719.79 ± 72.57	1573.46 ± 110.70	3276.53 ± 211.76	5445.93 ± 114.10	12656.41 ± 1572.59
$n = 16$	590.95 ± 76.14	904.27 ± 83.48	2940.68 ± 426.93	6045.06 ± 533.52	7698.29 ± 180.10	20412.36 ± 2271.55

Table 3
Average latency and confidence interval in a 802.11b network with Byzantine node failures (latency in milliseconds and confidence level of 95%).

instead RSA signatures of 1024, 2048, and 4096 bits in length. For this experiment, the protocol was executed in 802.11b network with the failure-free fault load and a unanimous proposal distribution.

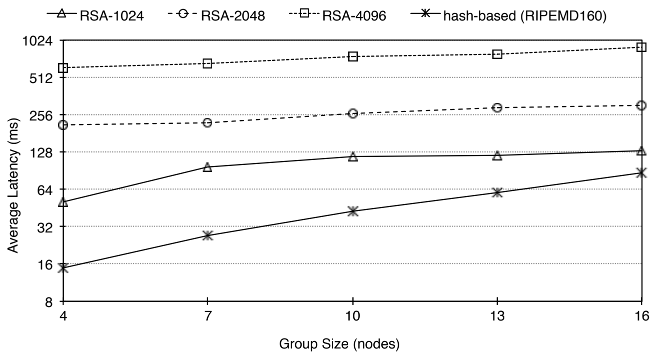

Figure 5. Average latency in a 802.11b network with RSA and hash-based message authentication mechanisms (logarithmic scale of base 2).

From the graph, it can be observed that Turquois benefits immensely from the hash-based authentication mechanism. It is important to establish that an increase in group size does not translate to the generation of more signatures by each node (each node generates exactly one signature per communication round regardless of the group size). It does, however, imply the verification of more signatures, but verification is at least an order of magnitude faster than generation. The fact that the curves for the RSA-based executions grow little with the group size is an indicator that the bottleneck lies more with the computation and less with the communication. This justifies the employment of efficient cryptographic techniques, specially because it significantly improves performance with practical group sizes.

6.2 Simulation

This section analyzes the performance of Turquois in the ns-3 network simulator [49]. It tests Turquois under some additional parameters that could not be captured by the Emulab testbed, such as a higher number of nodes - up to 100 - and the physical distribution of the nodes. More specifically, this section (1) complements the previous analysis of how the timeout value affects performance, and (2) it evaluates Turquois considering the physical distribution of the nodes. Every experiment is carried out in a simulated 802.11b ad-hoc network with a failure-free fault load and unanimous value proposals.

6.2.1 Timeout Value

Figures 6 and 7 plot the average latency and average rounds to termination of Turquois as a function of the timeout value. Figure 6 shows the curves for 4, 10, and 25 nodes, and Figure 7 show the curves for 50, 75, and 100 nodes.

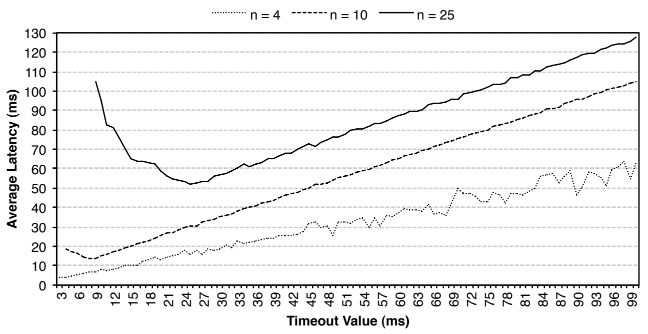

Figure 6. Average latency of Turquois with varying timeout value for 4, 10, and 25 nodes.

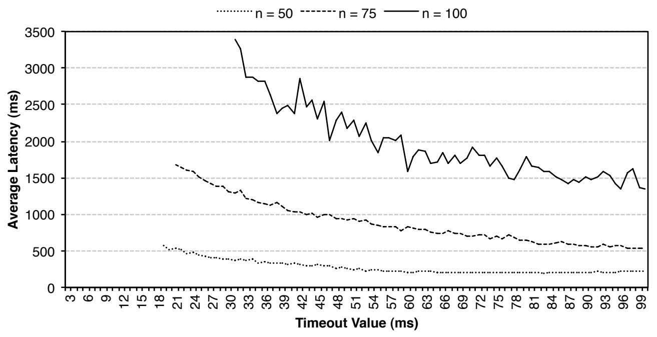

Figure 7. Average latency of Turquois with varying timeout value for 50, 75, and 100 nodes.

The results obtained via simulation are congruent with the observed trend of the experimental evaluation of Section 6.1.6. A relatively low timeout value generates too much contention, which significantly affects performance. As the timeout value increases, the performance becomes better until it reaches a sweet spot (which seems to be roughly around n ms). After that, the latency increases linearly with the timeout value. This pattern is clearly observable with $n = 25$ and, to a lesser extent, with $n = 10$. With $n = 4$, the number of nodes is too low to cause any significant contention, even with very low timeout values. With $n = 50$, $n = 75$, and $n = 100$, it can only be observed the latency approaching its sweet spot (actually, in $n = 50$, it can still be observed an increase in latency towards the higher timeout values, but very slightly). Notice that the curves begin being drawn for a progressively higher timeout value with increasing n. For example, with $n = 50$ the curve only starts at $timeout = 19$, with $n = 75$ it starts at $timeout = 21$, and with $n = 100$ at $timeout = 31$. This is because with lower timeout values the contention generated was so much that the algorithm did not terminate in a reasonable number of steps (i.e., within 1000 rounds).

6.2.2 Node Density

This section analyzes the performance of Turquois by varying the node density (i.e., the physical area for a given number of nodes). For this simulation, the nodes were uniformly distributed within a disc of varying radius - 10, 90, and 130 meters. Figure 8 plots the average number of rounds until termination and Figure 9 plots the average percentage of nodes that a node is *connected*[4] to, which we call the average node connectivity.

The first observation is that the number of rounds increases linearly with the number of nodes. The second is that the number of rounds also increases with higher disc radius. The latter exposes a tradeoff between performance and connectivity and shows the practical tolerance of the protocol to reduced connectivity, i.e., the algorithm can still terminate with an average connectivity that implies more transmission failures than those admitted by the liveness threshold. For example, with $n = 100$, the assumed parameters were $f = 0$ (we assumed no node failures) and $k = 67$. This gives $\sigma \leq 1715$. According to Figure 9, for $n = 100$, the average connectivity was slightly less than 0.75. This means that, on each communication round, at least 25% of the system messages were lost, yielding a total count of at least $0.25n^2 = 2500$ omission failures per round, which is considerably higher than σ. These results are in no way contradictory with the liveness threshold. While a number of omissions equal or below σ guarantees progress, this does not imply that every omission pattern above σ must block the progress of the protocol. In practice, many of these patterns are benign and still allow for the protocol to make progress.

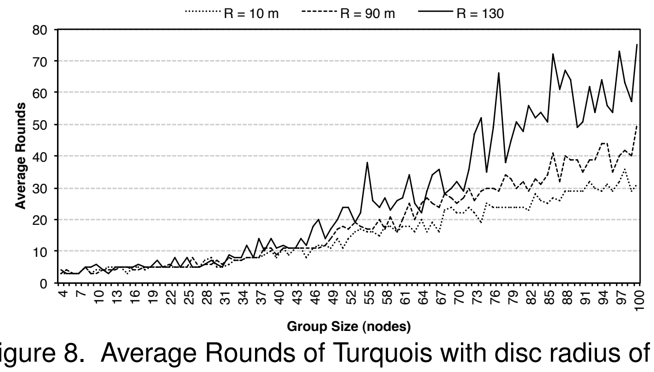

Figure 8. Average Rounds of Turquois with disc radius of 10, 90, and 130 meters.

4. We consider a node p_i to be connected to another node p_j for an execution of the protocol if, during that execution, p_i receives some message from p_j. This relationship is not symmetric.

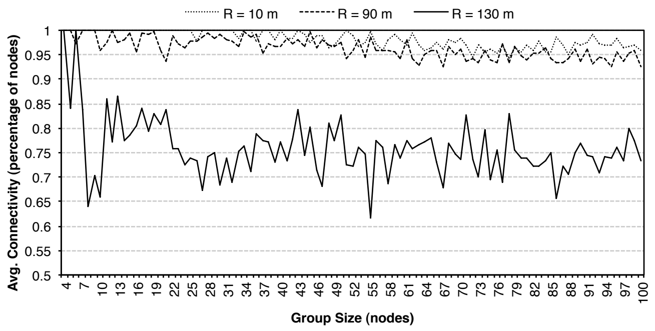

Figure 9. Average Node Connectivity of Turquois with disc radius of 10, 90, and 130 meters.

7 Conclusions

The paper presented Turquois, an Byzantine fault-tolerant binary consensus protocol specifically designed for wireless ad-hoc networks. Its design takes into account the typically constrained resources of wireless ad-hoc environments, while aiming for optimal resilience parameters. The protocol tolerates $f < \frac{n}{3}$ Byzantine nodes. Furthermore, it assumes communication to be inherently unreliable by incorporating the *communication failure model* [7]. *Safety* is maintained despite unrestricted message omissions, and *liveness* is ensured in rounds where the number of omissions is bounded by $\sigma \leq \lceil\frac{n-t}{2}\rceil(n-k-t)+k-2$, where k is the number of nodes required to decide, and $t \leq f$ is the number of nodes that are actually faulty. The timing assumptions are also very weak, requiring only a local timeout on each node to ensure these keep sending messages.

The key to its performance was the decision to assume unreliable communication, which allows the protocol to take full advantage of the broadcasting medium, where the cost of transmitting a message to multiple nodes can be just the same of sending it to a single one. Furthermore, the protocol avoids the use of public-key cryptography during its normal operation in order to preserve the computational power of mobile nodes, which is usually limited. The protocol was subject to a comparative performance evaluation against two well-known intrusion-tolerant consensus protocols. The results showed that, regardless of the type of faults present in the system, Turquois significantly outperforms the other protocols, in particular as the number of nodes in the system increases.

References

[1] A. Clementi, A. Monti, and R. Silvestri, “Selective families, superimposed codes, and broadcasting on unknown radio networks,” in *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2001, pp. 709–718.

[2] D. R. Kowalski, “On selection problem in radio networks,” in *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 2005, pp. 158–166.

[3] J. A. Misener, R. Sengupta, and H. Krishnan, “Cooperative collision warning: Enabling crash avoidance with wireless technology,” in *12th World Congress on ITS*, 2005.

[4] H. Moniz, A. Tedeschi, N. F. Neves, and M. Correia, “A distributed systems approach to airborne self-separation,” in *Computational Models, Software Engineering and Advanced Technologies in Air Transportation*, L. Weigang, A. Barros, and I. Oliveira, Eds. IGI Global, 2009.

[5] M. D. Brown, “Air traffic control using virtual stationary automata,” Master’s thesis, Massachusetts Institute of Technology, 2007.

[6] M. J. Fischer, N. A. Lynch, and M. S. Paterson, “Impossibility of distributed consensus with one faulty process,” *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, 1985.

[7] N. Santoro and P. Widmeyer, “Time is not a healer,” in *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science*, 1989, pp. 304–313.

[8] A. Bessani, P. Sousa, M. Correia, N. F. Neves, and P. Veríssimo, “The CRUTIAL way of critical infrastructure protection,” *IEEE Security and Privacy*, vol. 6, no. 6, pp. 44–51, 2008.

[9] M. K. Reiter, “The Rampart toolkit for building high-integrity services,” in *Theory and Practice in Distributed Systems*. Springer-Verlag, 1995, vol. 938, pp. 99–110.

[10] A. Vahdat and D. Becker, “Epidemic routing for partially-connected ad hoc networks,” Duke University, Tech. Rep. CS-2000-06, 2000.

[11] B. Awerbuch, D. Holmer, C. Nita-Rotaru, and H. Rubens, “An on-demand secure routing protocol resilient to byzantine failures,” in *Proceedings of the 1st ACM workshop on wireless security*, 2002, pp. 21–30.

[12] C. Koo, “Broadcast in radio networks tolerating Byzantine adversarial behavior,” in *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, 2004, pp. 275–282.

[13] A. Pelc and D. Peleg, “Broadcasting with locally bounded byzantine faults,” *Information Processing Letters*, vol. 93, no. 3, pp. 109–115, 2005.

[14] V. Drabkin, R. Friedman, and M. Segal, “Efficient byzantine broadcast in wireless ad-hoc networks,” in *Proceedings of the International Conference on Dependable Systems and Networks*, 2005, pp. 160–169.

[15] V. Bhandari and N. Vaidya, “On reliable broadcast in a radio network,” in *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 2005, pp. 138–147.

[16] C.-Y. Koo, V. Bhandari, J. Katz, and N. H. Vaidya, “Reliable broadcast in radio networks: the bounded collision case,” in *Proceedings of the 25th annual ACM symposium on Principles of distributed computing*. ACM, 2006, pp. 258–264.

[17] N. Badache, M. Hurfin, and R. Macedo, “Solving the consensus problem in a mobile environment,” in *Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference*, 1999, pp. 29–35.

[18] T. Chandra and S. Toueg, “Unreliable failure detectors for reliable distributed systems,” *Journal of the ACM*, vol. 43, no. 2, pp. 225–267, 1996.

[19] H. Seba, N. Badache, and A. Bouabdallah, “Solving the consensus problem in a dynamic group: an approach suitable for a mobile environment,” in *Proceedings of the 7th IEEE International Symposium on Computers and Communications*, 2002, pp. 327–332.

[20] W. Wu, J. Cao, J. Yang, and M. Raynal, “Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks,” *IEEE Transactions on Computers*, vol. 56, no. 8, pp. 1055–1070, August 2007.

[21] E. Vollset and P. D. Ezhilchelvan, “Design and performance-study of crash-tolerant protocols for broadcasting and reaching consensus in MANETs,” in *Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems*, 2005, pp. 166–175.

[22] N. Santoro and P. Widmayer, “Agreement in synchronous networks with ubiquitous faults,” *Theoretical Computer Science*, vol. 384, no. 2-3, pp. 232–249, 2007.

[23] G. Chockler, M. Demirbas, S. Gilbert, C. Newport, and T. Nolte, “Consensus and collision detectors in wireless ad hoc networks,” in *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 2005.

[24] F. Borran, R. Prakash, and A. Schiper, “Extending Paxos/LastVoting with an adequate communication layer for wireless ad hoc networks,” in *Proceedings of the 27th IEEE International Symposium on Reliable Distributed Systems*, 2008, pp. 227–236.

[25] B. Charron-Bost and A. Schiper, “The heard-of model: Computing in distributed systems with benign failures,” *Distributed Computing*, vol. 22, no. 1, pp. 49–71, 2009.

[26] L. Lamport, “The part-time parliament,” *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp. 133–169, 1998.

[27] M. Biely, J. Widder, B. Charron-Bost, A. Gaillard, M. Hutle, and A. Schiper, “Tolerating corrupted communication,” in *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, 2007, pp. 244–253.

[28] U. Schmid, B. Weiss, and I. Keidar, “Impossibility results and lower bounds for consensus under link failures,” *SIAM Journal on Computing*, vol. 38, no. 5, pp. 1912–1951, 2009.

[29] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo, “Randomization can be a healer: Consensus with dynamic omission failures,” in *Proceedings of the 23rd International Symposium on Distributed Computing*, 2009, pp. 63–77.

[30] R. Boichat, P. Dutta, S. Frolund, and R. Guerraoui, “Deconstructing paxos,” *SIGACT News*, vol. 34, no. 1, pp. 47–67, March 2003.

[31] M. Castro and B. Liskov, “Practical Byzantine fault tolerance,” in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 1999, pp. 173–186.

[32] J. P. Martin and L. Alvisi, “Fast byzantine consensus,” *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 202–215, 2006.

[33] F. Borran and A. Schiper, “A leader-free Byzantine consensus algorithm,” in *Proceedings of the 11th International Conference on Distributed Computing and Networking*, 2010, pp. 67–78.

[34] L. Lamport and J. Gray, “Consensus on transaction commit,” *ACM Transactions on Database Systems*, vol. 31, no. 1, pp. 133–160, 2006.

[35] V. Hadzilacos and S. Toueg, “Fault-tolerant broadcasts and related problems,” in *Distributed Systems*, S. Mullender, Ed. ACM Press / Addison-Wesley, 1993, ch. 5, pp. 97–146.

[36] M. Correia, N. F. Neves, and P. Veríssimo, “From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures,” *Computer Journal*, vol. 41, no. 1, pp. 82–96, Jan. 2006.

[37] J. Zhang and W. Chen, “Implementing uniform reliable broadcast with binary consensus in systems with fair-lossy links,” *Information Processing Letters*, vol. 110, no. 1, pp. 13–19, 2009.

[38] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo, “RITAS: Services for randomized intrusion tolerance,” *IEEE Transactions on Dependable and Secure Computing*, vol. 8, no. 1, pp. 122–136, 2011.

[39] C. Cachin, K. Kursawe, and V. Shoup, “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography,” in *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, Jul. 2000, pp. 123–132.

[40] R. Guerraoui and A. Schiper, “The generic consensus service,” *IEEE Transactions on Software Engineering*, vol. 27, no. 1, pp. 29–41, 2001.

[41] M. O. Rabin, “Randomized Byzantine generals,” in *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, 1983, pp. 403–409.

[42] C. Cachin, K. Kursawe, and V. Shoup, “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography,” *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[43] M. Ben-Or, “Another advantage of free choice: Completely asynchronous agreement protocols,” in *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, 1983, pp. 27–30.

[44] G. Bracha, “An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol,” in *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, 1984, pp. 154–162.

[45] T. Srikanth and S. Toueg, “Simulating authenticated broadcasts to derive simple fault-tolerant algorithms,” *Distributed Computing*, vol. 2, no. 2, pp. 80–94, 1987.

[46] A. J. Menezes, P. C. V. Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[47] R. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, February 1978.

[48] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar, “An integrated experimental environment for distributed systems and networks,” in *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, 2002, pp. 255–270.

[49] T. R. Henderson, S. Roy, S. Floyd, and G. F. Riley, “ns-3 project goals,” in *Proceedings from the 2006 Workshop on ns-2: the IP Network Simulator*, 2006, p. 13.

[50] M. Dixit, H. Moniz, and A. Casimiro, “Timeout-based adaptive consensus: improving performance through adaptation,” in *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, 2012, pp. 492–497.

Henrique Moniz obtained his PhD from the University of Lisbon in 2010. He spent two semesters at the same institution as an Invited Assistant Professor and joined Microsoft Research in 2011 as a Post-Doctoral Researcher. He is currently an associate researcher at INESC-ID. His research interests lie in distributed systems and security. He has contributions in the areas of fault tolerance, security, wireless and mobile networks, and theory of distributed computing. Since joining Microsoft Research he has become interested in data center architectures and large-scale storage systems.

Nuno Neves is Associate Professor at the Department of Informatics, Faculty of Sciences of the University of Lisboa. He is also a member of the LaSIGE - Large Scale Informatic Systems Laboratory, and the Navigators group, where he co-leads the fault and intrusion tolerance research area. His main research interests are in distributed and parallel systems, especially in the areas of security and dependability. Currently, he is principal investigator at the MASSIF European project, and coordinates the DIVERSE and SITAN national projects. In the past years, he contributed to several other security-related projects at European level, such as CRUTIAL, MAFTIA, and RESIST, and nationally, RITAS, AJECT and COPE. His work has been recognized in several occasions, for example with the IBM Scientific Prize and the William C. Carter award. He currently has more than 75 international publications in journals and conferences. He is on the editorial board of the International Journal of Critical Computer-Based Systems, and has served on the program committee of more than 50 conferences.

Miguel Correia is an Associate Professor at Instituto Superior Técnico of the Universidade Técnica de Lisboa. He is a researcher at INESC-ID, in the Distributed Systems Group. Until recently he was with the University of Lisboa Faculty of Sciences, LASIGE and the Navigators. He has a PhD in Computer Science from the University of Lisboa Faculty of Sciences. He has been involved in several international and national research projects related to intrusion tolerance and security, including the TCLOUDS, MAFTIA and CRUTIAL EC-IST projects, and the ReSIST NoE. He has more than 60 publications in international journals, conferences and workshops. His main research interests are: security, intrusion tolerance, distributed systems, distributed algorithms, computer networks, cloud computing, and critical infrastructure protection.