Intrusion Tolerance in Wireless Environments: An Experimental Evaluation

Henrique Moniz Nuno Ferreira Neves Miguel Correia António Casimiro Paulo Veríssimo
LASIGE, Faculdade de Ciências da Universidade de Lisboa *

Abstract

This paper presents a study on the performance of intrusion-tolerant protocols in wireless LANs. The protocols are evaluated in several different environmental settings, and also within the context of a car platooning application for distributed cruise control. The experimental evaluation reveals how performance is affected by the various environmental parameters such as the wireless standard, group size, and network topology. The distributed cruise control application demonstrates the practicability of such protocols, even when subjected to malicious faults.

Keywords: Intrusion Tolerance, Byzantine Agreement, Randomized protocols, Performance evaluation, Wireless networks

Corresponding author: Henrique Moniz – hmoniz@di.fc.ul.pt

The material has been cleared through the author affiliations.

1 Introduction

The evolution of wireless communications over the past decade – 1997 marks the release of the 802.11 standard – has had a profound impact on the way we look at distributed systems. The pace at which wireless standards have been progressing promises a global world of interconnected mobile devices where they can be constantly connected to large infrastructural networks (e.g., the Internet) or may spontaneously form ad-hoc islands of networks among them to meet some immediate need. The 802.11 family of Wireless LAN standards takes much of the responsibility for this 'wireless revolution'. Wireless LANs offer splendid opportunities but also create additional challenges. Notwithstanding the well-known problems regarding the protection of the communications [9, 15], there is also to consider the 'open' nature of the many services they promise to deliver, where malicious adversaries may wish to intrude.

This is where the concept of *intrusion tolerance* fits. Arising from the intersection of two classical areas of computer science – *fault tolerance* and *security* – its philosophy is to guarantee continuity in correct system

*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 NoE IST-4-026764-NOE (RESIST), and by the FCT through project POSI/EIA/60334/2004 (RITAS) and the Large-Scale Informatic Systems Laboratory (LASIGE)

behavior even if some of its components fail in unpredictable ways, perhaps even falling under the control of an intelligent adversary [12, 22, 17]. This approach contrasts with the typical way of dealing with intrusions which is to exclusively prevent them, i.e., to avoid successful attacks at all costs.

Intrusion tolerant protocols have been used in past for several applications, including reliable communication, consensus and voting, and state machine replication [20, 16, 5, 4]. A few of these protocols have been implemented, however, their evaluation has been limited to standard (wired) LANs and WANs. The system models under which they operate give an indication that these protocols were not designed with wireless needs in mind (e.g., they assume that nodes are fully-connected). Therefore, very little is known about the behavior of the protocols in wireless environments. Namely, it is unknown how well they will adapt to the distinctive characteristics of wireless networks, such as potentially lower bandwidths, higher failure rates and increased contention in communication medium. Additionally, the kinds of devices that are many times utilized in these environments also have distinctive characteristics, for instance limited power supply and lower CPU capabilities, which most probably will impact on the performance of the protocols.

In this paper, we want to understand how well existing protocols can support the execution of intrusion-tolerant applications in wireless LANs. To contextualize our research, we have considered a particular application with demanding requirements – a *car platooning* service for *distributed cruise control*. In simple terms, the service is responsible for exchanging information in order to automatically control a group of vehicles traveling close together to some common destination (e.g., a set of trucks transporting a load across country). This application is quite interesting because it has some requisites both in terms of time and security: for example, velocity should be adjusted within a reasonable interval if a car wants to increase or decrease speed; and, correct behavior needs to be preserved even if external or internal attacks occur.

We developed an algorithm which implements the *speed agreement* operations of the distributed cruise control application. This algorithm is built using the services of an existing stack of intrusion-tolerant protocols for group communication – RITAS [19]. The services provided by the RITAS are particulary interesting because first they tolerate arbitrary (or Byzantine) failures, and second they are built for an asynchronous system model. This synchrony model avoids all time dependencies, making the protocols immune to attacks in the domain of time (e.g., attacks which artificially delay some parts of the system). Moreover, it is well-suited for the considered environment because under certain conditions wireless networks are essentially untimely (e.g., noisy environments). RITAS offers several flavors of broadcast and consensus protocols, and copes with the FLP impossibility result for asynchronous systems by resorting to randomization[11].

Both the distributed control application and its supporting protocols are evaluated under several environmental

settings. With the intention of not losing sight of the broader goal of the paper (i.e., to measure the feasibility of intrusion tolerance in wireless networks), the experiments were made using hardware with capabilities similar to what is expected to be found in automobiles (low-end personal computers), and with hardware that is typically utilized in wireless networks in general (Personal Digital Assistant (PDA) devices). For purpose of this investigation a subset of the RITAS protocols had to be ported from Linux to the Windows Mobile platform. This task ended up being more difficult than expected due to differences in the thread management of Windows Mobile, and because of the limited support for some of the cryptographic operations.

To the best of our knowledge, there is no other study on intrusion-tolerant agreement protocols in wireless networks. There has been an important track of research on routing protocols for wireless ad-hoc networks that can be considered to be intrusion-tolerant, since they aim to tolerate routing misbehavior of some of the nodes. Two surveys are in [14, 1]. There is also one work on intrusion-tolerant broadcast in that kind of networks [10]. More recently, there has been research on crash fault-tolerant consensus protocols for wireless networks, presenting conditions for solvability, tradeoffs and efficient algorithms [6, 7, 23, 2, 13].

The paper has the following main contributions: it provides a general evaluation of intrusion-tolerant protocols in wireless networks; it proposes an algorithm for the speed agreement operations of a distributed cruise control, presenting the respective implementation and performance evaluation; it presents the first implementation and experimental evaluation of intrusion-tolerant protocols using mobile devices such as PDAs.

The rest of the paper is organized as follows. Section 2 explains distributed cruise control. Section 3 describes the used system model. A speed agreement algorithm for distributed cruise control is described in Section 4. Section 5 presents the experimental evaluation. Finally, Section 6 concludes the paper.

2 Distributed Cruise Control

Classical cruise control is a system designed to automatically control the speed of a vehicle. The driver sets a speed, and the system assumes automatic control of the throttle of the vehicle. The same speed is maintained despite changes in road conditions (e.g., inclinations).

Distributed cruise control is an extension of this functionality for platoons. A *platoon* is a tightly spaced vehicle group formation, and is aimed at improved highway efficiency (e.g., lane throughput, fuel consumption, etc.) and passenger comfort as much as possible. In the context of this work, a platoon is defined as a group of vehicles on a highway traveling at close distances from each other and heading to some common destination. This is the case, for instance, of platoons of heavy-duty freight trucks, which travel together from the departure to the arrival point [25].

The main difference between classical cruise control and distributed cruise control is that instead of having each vehicle defining its speed individually, the platoon members communicate in a wireless ad-hoc environment in order too agree on a common speed. Once chosen, this baseline speed is enforced by all vehicles, though not strictly. Vehicles are allowed to deviate from this baseline speed in order to maintain safe distances from each other.

Strict safety falls outside the scope of this work. Given the openness of the environment, and the uncertainty of communication delay, it must be ensured by other means (e.g., automatic breaking based on sensor information embedded in vehicles). Nevertheless, it is assumed that the priority of a vehicle is to keep safe distances from the other vehicles, and this overrides any speed value previously agreed upon. In fact, maintaining the baseline speed is best-effort only. These safety distances are expressed as the minimum distance (Δ_{min}) to the front vehicle and the maximum distance (Δ_{max}) to the follower vehicle, as shown in Figure 1. The former ensures that, given known values for the current speed of a vehicle and the message transmission delay, a vehicle will never collide with the preceding one. The latter ensures that a vehicle will not get too far apart from the follower one, as a means to ensure network connectivity. It is assumed that the vehicles know these distance values and are able to maintain them at all times during the platoon operation.

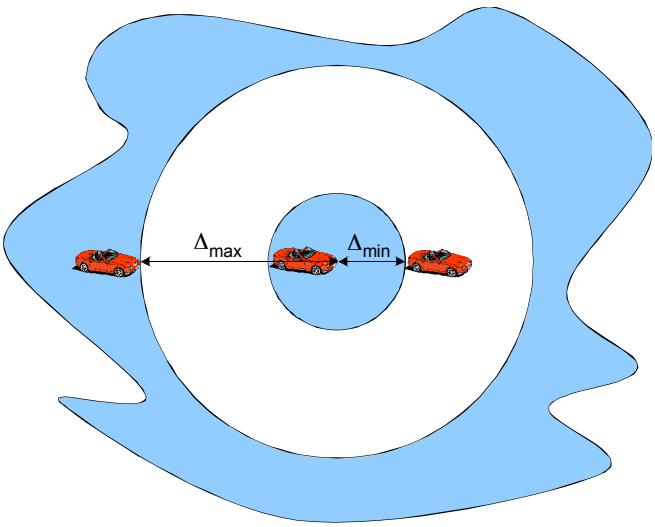

Figure 1: Safety distances between vehicles.

This paper presents an algorithm that solves the *speed agreement* problem. It is designed as a service that can be used by a sophisticated distributed cruise control application to reach agreement on a baseline speed.

The speed agreement algorithm is arranged at the top of a protocol stack as depicted in Figure 2. The protocols below the speed agreement are used as primitives by this algorithm. Vector consensus allows processes (or vehicles) to propose values and agree on vector composed by a subset of these values. Multi-valued consensus allows

processes to agree on a value from an arbitrary domain. Below is a binary consensus protocol. This protocol, being the simplest form of consensus, is where randomization is applied to circumvent the FLP result. With it, processes can agree on a binary value. At the bottom, there are two broadcast primitives. Reliable broadcast ensures that a broadcasted value is received by every process, and echo broadcast ensures that no two processes receive a different value.

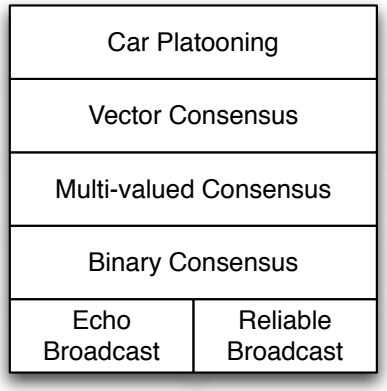

Figure 2: Protocol stack.

3 System model

This section defines the system model for the distributed cruise control application. It applies to the speed agreement protocol and to all the protocols below in the stack (see Figure 2).

A platoon is formed by a group of n vehicles (also called *processes* or *nodes*) $P = \{p_1, p_2, ...p_n\}$, where $n \geq 4$. It is assumed that a platoon is formed at some departure point and travels together to some arrival point. Split and merge maneuvers are outside the scope of our problem and are not considered for simplification.

Only vehicles belonging to the platoon can participate in the execution of the protocols. Nevertheless, some members of the platoon may be either ill-intentioned or corrupted by external entities. A vehicle is said to be *correct* if its follow the protocols until arrival. Otherwise, it is said to be *corrupt*. The only assumption on the behavior of corrupt vehicles/processes is that they follow the principle of no self-harm. This means that a vehicle cannot force its collision with other vehicles or take any action representing danger to its occupants, either by means of explicit car control (e.g., sudden breaking or acceleration) or transmission of incorrect information. They can, however, force collisions between other vehicles, or take any action they like to disrupt the correct operation of the platoon. There is a limit to the number corrupt vehicles that can exist in a platoon. In a platoon with n vehicles, at most $f = \lfloor \frac{n-1}{3} \rfloor$ of them can be corrupt.

Correct processes are assumed to be fully-connected by *authenticated reliable channels* that provide authenticity, reliability and integrity. Authenticity means that the recipient of a message knows who was the sender, reliability means that messages are eventually received, and integrity means that messages are delivered without modifications.

There are no assumptions about time whatsoever, meaning that there are no bounds to the processing times or communication delays. Hence, the system is completely asynchronous.

Each pair of vehicles (p_i, p_j) shares a secret key s_{ij}. It is out of the scope of this work to present a solution for distributing these keys, but it may require a trusted dealer or some kind of key distribution protocol based on public-key cryptography. Nevertheless, this is normally performed before the execution of the protocols and does not interfere with their performance.

Each process has access to a random bit generator that returns unbiased bits observable only by the process (if the process is correct).

Some protocols use a *collision-resistant hash function* $H(m)$ that maps an arbitrarily length input m into a fixed length output. We assume that it is impossible (1) to find two values $m \neq m'$ such that $H(m) = H(m')$, and, (2) given a certain output, to find an input that produces that output. The output of the function is often called *a hash*.

4 Speed Agreement Algorithm

This section presents the speed agreement algorithm for intrusion-tolerant distributed cruise control, and the supporting protocols. It allows vehicles inside a platoon to agree on a common baseline speed even if some of them attempt to thwart the correct platoon operation. The chosen baseline speed can be obtained using two different approaches, which are not necessarily mutually exclusive: a leader-based strategy, and a decentralized strategy.

Leader-based strategy. One of the platoon members, which is not necessarily the car at the head of the platoon, is defined a priori to be the leader, and disseminates the target speed to the rest of the platoon.

For this strategy, the challenge, in terms of safety, is to ensure that the leader reliably disseminates the speed value. The leader itself can be malicious, this means that mechanisms must be in place that keep the leader from disseminating contradictory speed values to different members of the platoon. It also must be ensured that all members of the platoon eventually receive the target speed value and are not left in the dark since this could seriously affect convergence to a stable situation.

We want to guarantee that the information disseminated by the leader is delivered by all vehicles, it is not

contradictory, and it is actually originated by the leader. A *reliable broadcast* protocol solves this problem since it ensures that every process receives the broadcasted value.

Decentralized strategy. Every member proposes a target speed, and an agreement protocol is executed to decide, based on or among the proposed values, a value to be used by all platoon members.

This second approach is potentially more immune to the scenario where some vehicle cannot guarantee the chosen speed. Since the decision is based upon the proposals from all members, it can, for instance, be the slowest car (under certain limits) to determine the target speed. It has to be ensured that the vehicles actually decide on the same speed, even if some of them try to disrupt this procedure by, for instance, disseminating contradictory proposals.

There are two protocols that, in essence, solve this problem: multi-valued consensus and vector consensus. *Multi-valued consensus* allows processes to decide on a value among the proposed ones. *Vector consensus* allows processes to agree on a vector composed by a subset of the proposed values. The latter is preferable since it provides more flexibility to the platooning application. It is preferable for the application to have access to a common vector composed by the proposals from the vehicles where it can apply some deterministic function to obtain to target speed (e.g., the lowest value, or the most common value), instead of being tied down to the single value returned by a multi-valued consensus protocol. For this reason, vector consensus is used by this strategy.

The algorithm. The speed agreement algorithm will apply both strategies complementarily. There will be a predefined leader – the car at the head of the platoon – that will set the target speed. In case some vehicle is unable to apply the stipulated speed, it can demand a decentralized agreement by having the vehicles execute a vector consensus. It is assumed that if, after a decentralized decision, a vehicle is unable to enforce the target speed, some kind of mechanism (e.g., manual intervention) ejects the car from the platoon. This procedure is, however, outside the scope of our application. The speed agreement algorithm works as follows:

1. The leader reliably broadcasts the new target speed.
2. All vehicles that can so, assume the new target speed.
3. If any vehicle cannot assume the new target speed, it reliably broadcasts a message stating so.
4. Upon receiving this message, all vehicles initiate a vector consensus.
5. The new target speed is obtained by applying a deterministic function on the decision vector.
6. All vehicles assume this new target speed, no exceptions.

The following subsections describe the supporting protocols, bottom-up, for the speed agreement algorithm. These have been previously described in the literature, and are presented here briefly for self-containment.

4.1 Reliable and Echo Broadcast

The *reliable broadcast* protocol ensures that, upon a broadcast, all correct processes either deliver the same message or no message at all. *Echo broadcast* is a weaker, but more efficient, version of reliable broadcast. In the case where the sender is corrupt, it does not guarantee that all correct processes will deliver the message. It only ensures that the subset of correct processes that deliver, will do it for the same message.

The implemented reliable broadcast protocol was originally proposed in [3]. It starts with sender broadcasting an (INITIAL, m) message. When a process receives this message, it broadcasts an (ECHO, m) message, and waits for $\lfloor \frac{n+f}{2} \rfloor + 1$ such messages to arrive. It then transmits a (READY, m) message, and waits for $2f + 1$ such messages to deliver m. The implemented *echo broadcast* primitive was originally proposed in [21] and is similar to the reliable broadcast protocol. The difference is that it does not require the READY messages, and it waits for $\frac{n+f}{2}$ ECHO messages to deliver m.

4.2 Binary Consensus

Two distinct randomized binary consensus protocols were implemented. A local coin protocol that has a high time and communication complexity but avoids the use of expensive cryptography, and a shared coin protocol (SCP) that has a low time and communication complexity but uses several asymmetric cryptographic primitives. These protocols were subject to an experimental comparison in (wired) Ethernet LANs in a previous paper [18].

Bracha's Local Coin Protocol. This protocol was originally proposed in [3]. It proceeds in rounds and exchanges $O(n^3)$ messages per round. The expected number of rounds until termination is 2^{n-f}.

Each round starts with the processes reliably broadcasting their proposals. The processes wait for $n - f$ proposals and reliably broadcast a new proposal that reflects the majority of the values previously received. Again, they wait for $n - f$ proposals, and if more than half of the received values are equal, they reliably broadcast that value. Otherwise, they reliably broadcast an undefined value $\perp$. Finally, the processes wait for $n - f$ messages and make a decision if at least $2f + 1$ messages have the same value $v \neq \perp$. Otherwise, if at least $f + 1$ messages have the same value $v \neq \perp$, then a new round is initiated with v as the initial proposal. If none of the previous conditions apply, then a random bit is chosen and is used as the initial proposal for the new round.

ABBA Shared Coin Protocol. This protocol was proposed in [4]. It exchanges $O(n^2)$ messages per round and reaches a decision in 1 or 2 rounds with high probability. The protocol makes extensive use of asymmetric cryptography to ensure the correctness of the execution. It uses a different system model than the one presented in Section 3: a slightly different reliable channel definition, and two cryptographic primitives – dual threshold signatures, and a threshold coin-tossing scheme. The reliable channels only need to provide the reliability property (i.e., that messages are eventually received) between every pair of correct processes. The integrity of the messages is guaranteed by the use of public-key signatures inside the protocol itself, which also serves to justify the proposals of the individual processes.

The protocol proceeds in rounds where each rounds begins with the processes transmitting their preferred proposal to the others. Upon received $n - f$ such messages, processes set their new proposal to $v \in \{0, 1\}$ if the received messages were unanimous on v. Otherwise the new proposal is set to *abstain*. Processes transmit this new proposal and wait for $n - f$ such messages. If they are unanimous, the processes can make a decision. Otherwise, each process generates a share of a threshold coin, transmits it to all other processes, and a new round is initiated. The preferred proposal for the new round is set as follows: if a message with value v distinct from *abstain* has been received, the preference is set to v; otherwise, the threshold coin is assembled from a quorum of shares and its value set as the new preference.

4.3 Multi-valued Consensus

Multi-valued consensus allows processes to propose and decide on a value with an arbitrary domain. Depending on the proposals, the decision is either one of the proposed values or a default value $\perp \notin \mathcal{V}$. The implemented protocol is the from the RITAS protocol stack [19] which in turn is adapted from [8]. It uses the the underlying *reliable broadcast*, *echo broadcast*, and *binary consensus* protocols.

The protocol start with every process reliably broadcasting its initial proposal v in a (INIT, v) message. The processes then wait for $n - f$ INIT messages and store the received values in a vector V. If a process receives $n - 2f$ messages with the same value v, it echo-broadcasts a (VECT, v, V) message (the vector V serves to justify the proposed value v). Otherwise, it echo-broadcasts a default value $\perp$. They wait for $n - f$ messages, and if $n - 2f$ messages have the same value $v \neq \perp$ and all remaining messages are $\perp$, then it proposes 1 for a binary consensus execution. Otherwise it proposes 0. If the binary consensus returns 0, the process decides on the default value $\perp$. If the binary consensus returns 1, the process waits until it receives $n - 2f$ *valid* VECT messages (if it has not done so) with the same value v and then it decides on that value.

4.4 Vector Consensus

Vector consensus allows processes to agree on a vector with a subset of the proposed values. It ensures that every correct process decides on the same vector V of size n; if a process p_i is correct, then the vector element $V[i]$ is either the value proposed by p_i or the default value $\perp$, and at least $f + 1$ elements of V were proposed by correct processes. The implemented protocol is the one described in [8], which uses *reliable broadcast* and *multi-valued consensus* as underlying primitives.

The protocol starts by reliably broadcasting a message containing the proposed value by the process and setting the round number r_i to 0. The protocol then proceeds in up to f rounds until a decision is reached. Each round proceeds as follows. A process waits until $n - f + r_i$ messages have been received and constructs a vector W_i of size n with the received values. The indexes of the vector for which a message has not been received have the value $\perp$. The vector W_i is proposed as input for the *multi-valued consensus*. If it decides on a value $V_i \neq \perp$, then the process decides V_i. Otherwise, the round number r_i is incremented and a new round is initiated.

5 Experimental Evaluation

This section evaluates the performance of the protocol stack of Figure 2 in 802.11 wireless LANs. First, the supporting protocols – reliable and echo broadcast through vector consensus – are evaluated independently via a set of micro-benchmarks. The experiments gauge the protocols in a number of environmental settings that are modeled as system parameters. This gives an insight into the impact of the various wireless environments on the performance of the protocols. Second, the speed agreement algorithm is evaluated under various types of faults that are injected into the stack to assess the resistance of the application in hostile environments.

5.1 Testbeds

The experiments were carried out on two different testbeds, *tb-emulab* and *tb-pda*.

The first, *tb-emulab*, was formed by 11 nodes from the Emulab network testbed [24]. Each node was a Pentium III PC with 600 MHz of clock speed and 256 MB of RAM, and contained a 802.11 a/b/g D-Link DWL-AG530 WLAN interface card, which was able to operate as an access point (AP). The operating system running on these nodes was Redhat Linux 9 with kernel version 2.3.34. The nodes were located on the same physical cluster and were, at most, a few meters distant from each other.

The second testbed, *tb-pda*, was formed by 7 HP hw6915 PDAs. These PDAs were equipped with an Intel PXA270 416 MHz processor, 64 MB of SDRAM, and integrated 802.11b WLAN. The operating system was

Windows Mobile 5. The experiments were taken with the PDAs placed on the same table a few centimeters apart from each other. The access point used in this testbed was an Asus WL-320gE 802.11 b/g.

5.2 Implementation

The broadcast and consensus protocols were taken from the RITAS suite, which provides an implementation for Linux. These protocols were then ported to the Windows Mobile platform. The car platooning application was then developed for both platforms on top of the existing protocols. The reliable channels were implemented by the use of TCP for reliability, and the IPSec Authentication Header protocol for integrity.

RITAS was originally implemented in C and compiled in gcc in Linux. Although only wired LANs were assumed, there was no need of changes to run it with wireless networks in PCs with Linux (testbed *tb-emulab*). However, to test it with PDAs (testbed *tb-pda*), we had to port most RITAS code to Windows mobile. These protocols were developed in Visual Studio using C++, and were debugged with one process running in Visual Studio's PocketPC emulator and the remaining processes in Linux virtual machines running the Linux version of RITAS. This allowed a more seamless debugging process and ensured interoperability between both implementations. The main challenge in porting the protocols was concerned with the mutual exclusion code, since the Windows Mobile semantics for multi-threading is considerably different from POSIX threads in Linux.

5.3 Experimental Methodology

This section describes the methodology used in the experiments to evaluate the performance of the protocols. It defines the performance metric, the environmental system parameters, and the conduction of the experiments.

The performance metric utilized in the experiments was the *latency*. This metric is always relative to a particular process p_i. In the case of the consensus protocols, 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 consensus value. In the case of the broadcast protocols (and the speed agreement algorithm), it is the interval of time between the moment the sender (or leader) process, say p_i, initiates the execution of the protocol, and the moment p_i delivers the broadcasted value (or accepts a speed value).

The environmental system parameters are configurable parameters of the system that define specific execution environments. These are the *group size*, the *wireless standard and network bandwidth*, and the *network topology*. The *group size* defines the number of processes n in the system, and in our case it can take three values: 4, 7, and 10. The *wireless standard and network bandwidth* defines the amendment to the 802.11 WLAN standard used, and intrinsically defines the amount of bandwidth available in the network. Three WLAN standard are used: 802.11a,

802.11b, and 802.11g. The 802.11a and 802.11g standards provide 54 Mb/s bandwidth, and 802.11b provides 11 Mb/s. The *network topology* defines the way the network nodes communicate with each other. There are two types of network topology: ad-hoc and infrastructure. In the ad-hoc network topology the nodes communicate directly with each other with no access points. In the infrastructure network topology, all nodes communicate through an access point (AP). This kind of topology can also make sense in the context of car platooning since the vehicles may also communicate through a roadside infrastructural network.

The conduction of the experiments was carried out the following way. A signaling machine, which does not participate in the execution of the protocols, is selected to conduct the experiment. It repeats the following procedure m times: it broadcasts a 1-byte UDP message to the n processes involved in the experiment. When a processes receives one of these messages, it executes whatever protocol is relevant for the current experiment (the information about which protocol to execute is carried within the 1-byte UDP message). Processes 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 taking the mean value of the sample of measured values.

5.4 Micro-benchmarks

The micro-benchmarks are a set of experiments that aim to access the impact of the several environmental settings on the performance of the protocols. This section presents four micro-benchmarks that evaluate specific settings. The first analyzes the impact of the wireless standard with different group sizes. The second, the effects of the computational capability of the processes. The third, the impact of the network topology. Finally, the fourth focuses on the binary consensus protocols. These protocols are interesting to look in more detail since they are the only ones that employ randomization, and do so using different strategies. In all the experiments, the message payload size of the broadcast and consensus protocols was set to 100 bytes. The only exception is binary consensus where 1-byte payloads are used (since the protocol only deals with binary values it does not make sense to have larger payloads). Except for Section 5.4.4, where is presented an explicit comparison between the two binary consensus protocols, the evaluated protocol is always Bracha's binary consensus.

5.4.1 Wireless Standard and Group Size

This micro-benchmark evaluates the performance impact of both the wireless standard and the group size. The used testbed was *tb-emulab*. The network topology was set to infrastructure with one of the nodes acting exclusively

	802.11b (ms)			802.11g (ms)			802.11a (ms)		
	$n = 4$	$n = 7$	$n = 10$	$n = 4$	$n = 7$	$n = 10$	$n = 4$	$n = 7$	$n = 10$
Reliable broadcast	20.2	72.4	219.2	8.3	24.7	62.5	7.7	23.4	50.6
Binary Consensus	42.6	292.7	832.8	18.2	92.6	326.3	17.1	73.3	310.8
Multi-valued consensus	178.4	1581.7	4678.9	79.9	564.6	1608.1	73.1	438.8	1340.5
Vector consensus	259.9	2078.3	6234.1	109.3	879.9	2504.6	94.8	720.4	1828.5

Table 1: Latency measurements in milliseconds for the various protocols given different wireless standards and group sizes in testbed *tb-emulab* in infrastructure mode.

as an access point. All possible wireless standard and group size settings were tested. The tested protocols were reliable broadcast, Bracha's binary consensus, multi-valued consensus, and vector consensus.

Table 1 shows the obtained measurements. The relative cost of the protocols can be easily observed. It is completely congruent with their interdependencies within the stack. The greatest gap is from the binary consensus to the multi-valued consensus and is justified by the large messages the multi-valued consensus has to reliably broadcast to justify the proposed values [8]. The gaps from the reliable broadcast to the binary consensus, and from the multi-valued consensus to the vector consensus are smaller and directly related with the overhead incurred from the respective upper-layer protocols.

The performance impact of the wireless standard is mainly a consequence of the available bandwidth. The experiments with 802.11b (11 Mb/s) were significantly slower than the ones with 802.11g and 802.11a (54Mb/s). A reliable broadcast with four processes takes 8.3 ms on a 802.11g network, while on a 802.11b network this more than doubled to 20.2 ms. This pattern is roughly observed for all the experiments, while the difference becomes slightly more accentuated with larger group sizes.

Another interesting result is that the values obtained in the 802.11a experiments were consistently lower than the ones obtained in 802.11g despite both standards being capable of achieving the same bandwidth. This difference is modest for the cheaper protocols and smaller group sizes. For instance, a reliable broadcast with four processes costs 8.3 ms in 802.11g, and 7.7 ms in 802.11a, an almost negligible difference. However, as the protocols become more expensive and the group size increases (i.e., the network becomes more stressed), this difference becomes substantial. For vector consensus with ten processes, the cost is 2504.6 ms in 802.11g, and 1828.5 ms in 802.11a, which is a considerable difference.

5.4.2 Computational Capability

The second set of experiments measure how the computational capability of the individual nodes affects the performance of the protocols. To attain this goal, both testbeds were configured with the same system parameters: the wireless standard was set to 802.11b, the group size to 4 and 7 processes, and the network topology to ad-hoc

mode. The computational capability in this contexts refers not just to the processing power of the CPU, but the whole local environment where the protocols are executed (e.g., hardware, drivers, operating system).

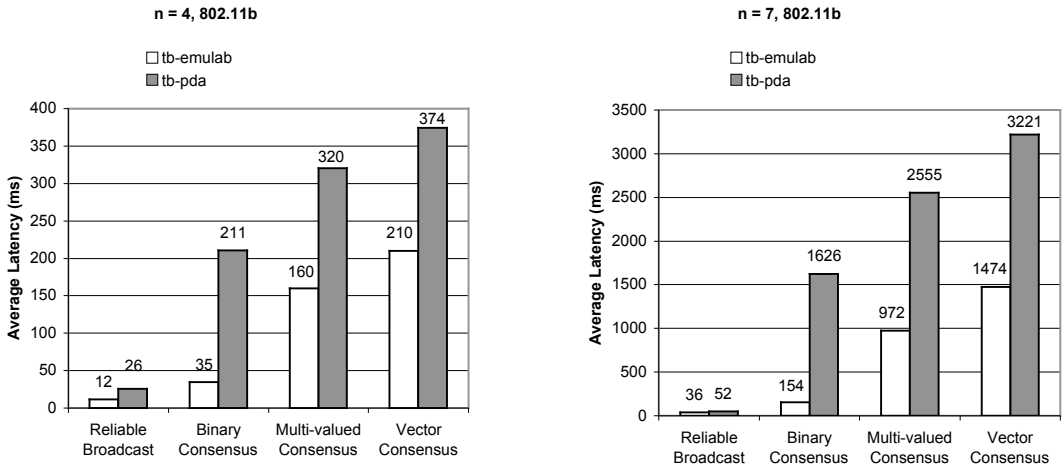

Figure 3: Average latency in 802.11b ad-hoc network for both testbeds.

The obtained measurements are presented in Figure 3. From the charts it is clear that the computational characteristics of the individual nodes greatly affects the performance of the protocols. There is always a significant gap between the two testbeds for all the experiments. The average latency roughly doubles from *tb-emulab* to *tb-pda*, except for binary consensus where the difference is much greater.

In both testbeds it is observed that, at some point, a larger gap exists from one protocol to another. In *tb-pda* this happens from reliable broadcast to binary consensus, and in *tb-emulab* from binary consensus to multi-valued consensus. Being the system parameters equal for both testbeds, one must assume that it is the limited computational capability of the nodes in *tb-pda* that is responsible for the gap observed from reliable broadcast to binary consensus.

For testbed *tb-emulab*, the gap from binary consensus to multi-valued consensus is congruent with the messages the latter protocol has to exchange, and was already observed in the benchmark from Section 5.4.1. What is interesting to note is that the average latency measured in *tb-pda* gets progressively closer to that measured in *pda-emulab* as the cost of the protocols increase. What happens is that another degradation factor comes into play: the network bandwidth. So, as the communication cost of the protocols increase, the performance bottleneck shifts from the computational capability of the nodes to the network bandwidth. This is why the gap between the two testbeds tends to decrease as more stress is put on the network.

5.4.3 Network Topology

This section looks at how the network topology of wireless networks impacts the performance of the protocols. For this experiment measurements were taken in testbed *tb-pda* with 802.11b networks for both ad-hoc and infrastructure. The group size was set to 4 and 7 processes.

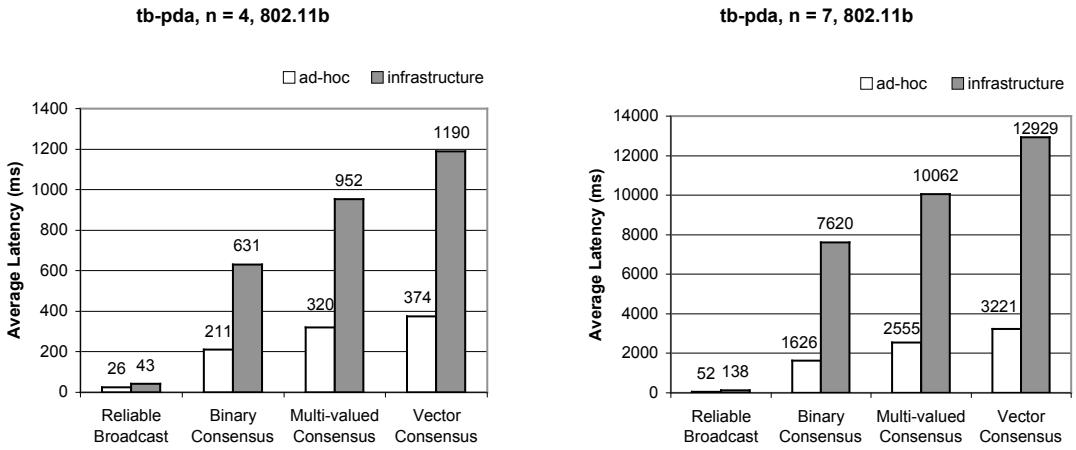

Figure 4: Average latency for *tb-pda* with 4 and 7 processes.

The measurements are depicted in Figure 4. The observation is that the operation in infrastructure mode does have a significant impact on performance. It introduces an additional delay into the communication between the processes since all data must be relayed through the AP.

The performance penalty in infrastructure mode remains essentially the same across all protocols despite their relative cost. Around 3 times with four processes, and 4 times with seven processes. For instance, in testbed *tb-pda*, a multi-valued consensus with four processes took 320 ms on average in ad-hoc mode, and 952 ms in infrastructure mode. With seven processes, it took 2555 ms in ad-hoc mode, and 10062 ms in infrastructure mode. So, a larger group also emphasizes a bit the degradation brought up by the presence of the AP.

These results demonstrate the high sensitivity these protocols have to the network latency, even more than the bandwidth, because of the large number of communication steps involved.

5.4.4 Binary Consensus Comparison

This section performs a more in-depth analysis of a key protocol in the stack: binary consensus. It compares the two different implementations of the protocol and presents some considerations about the performability of the two strategies employed by the protocols – one depends on the heavy use of public-key cryptography, and the other on abundant message exchanges. For these experiments testbed *tb-emulab* was used, and the following system

parameters were tested: the group size for 4, 7, and 10 processes; the wireless standard for 802.11b, and 802.11g; and the network topology for ad-hoc, and infrastructure.

Given the features of the protocols it was expected for the Bracha protocol to outperform the ABBA protocol given more favorable network conditions, and to exist a certain point, as network conditions degrade, where the ABBA strategy would pay off to the point of being faster than the Bracha protocol. Figure 5 presents the measurements observed for the various environmental settings tested.

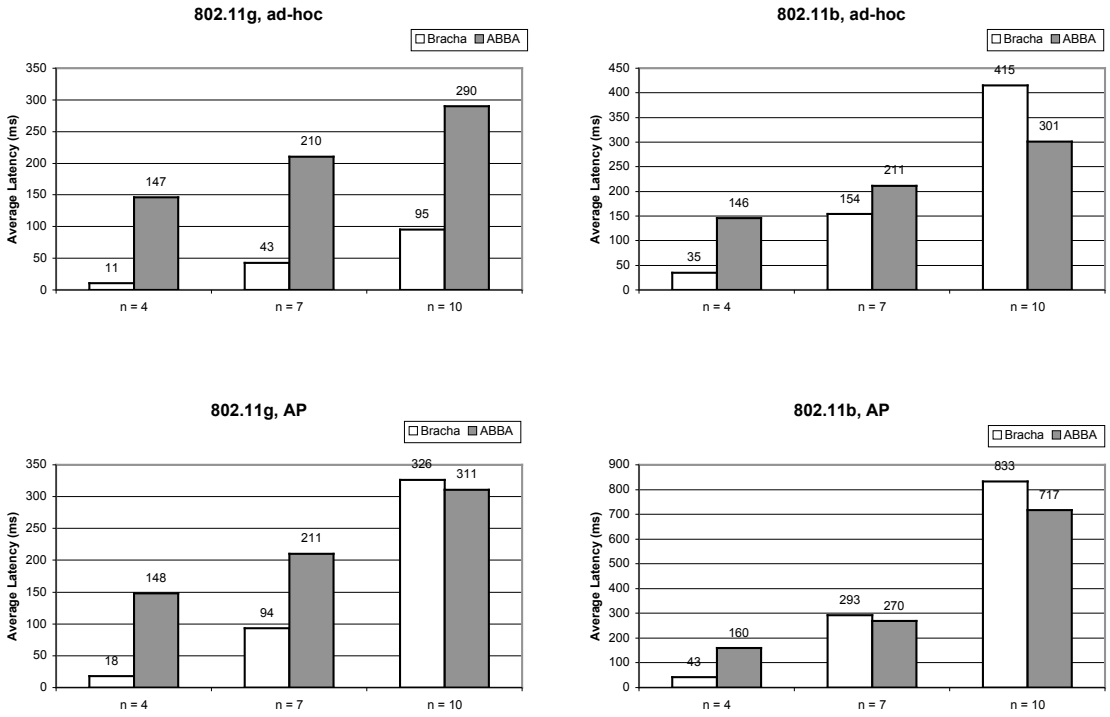

Figure 5: Average latency for binary consensus protocols in *tb-emulab*.

The results confirm the expectations. The upper-left chart shows the measurements for the best network configuration (802.11g and ad-hoc), where the Bracha protocol is clearly faster than ABBA, even for a group size of ten processes – 95 ms against 290 ms, respectively. In remaining charts where the network conditions are not so good – either there is an AP or the standard is 802.11b, or both – there is a point from which ABBA outperforms Bracha. For the upper-right (802.11b and adhoc) and lower-left (802.11g and AP) charts this happens when the group size is ten, and for the lower-right chart (802.11b and AP), which is the worst network configuration, this happens at $n = 7$. The conclusion is that Bracha is much faster with few processes and "good" network conditions, but it quickly degrades with the network capacity up to a point where the ABBA protocol, being more resilient to the network limitations, becomes faster.

5.5 Speed Agreement Algorithm

This section evaluates the speed agreement algorithm from Section 4. The application is evaluated in both testbeds and several scenarios are considered where the application is subject to various types of faults and different behaviors from the vehicles.

For testbed *tb-emulab*, two configurations are used, both with four nodes: one with a 802.11g ad-hoc network, and another with a 802.11g ad-hoc network. For *tb-pda*, the same number of nodes is used, but only on a 802.11b ad-hoc network. On both testbeds, the chosen binary consensus protocol is the local coin protocol. The application is evaluated under three types of faultloads, and two distinct vehicle behaviors.

The three faultloads are normal, silent, and Byzantine. In the *normal* faultload, the vehicles follow the protocol correctly until termination. In the silent faultload, one of the vehicles does not communicate with the others. Finally, in the Byzantine faultload, f of the vehicles try their best to disrupt the execution of the platooning application. This is done at the binary consensus and multi-valued consensus protocols. At the binary consensus, they always propose zero, trying to force the processes to decide on this value. This has the effect of the multi-valued consensus to return a default value $\perp$, which in turn forces the vector consensus to execute another round. At the multi-valued consensus layer, they always propose the default value both the INIT and VECT messages, trying to force the processes to decide on the default value. The goal is also to force the vector consensus into another round.

The vehicle behavior is related to the way vehicles react to the speed agreement algorithm, and it can be either *cooperative* or *contentious*. In the cooperative behavior, all the vehicles accept the proposal of the leader. In the contentious behavior, one of the vehicles disagrees with the value proposed by the leader and forces a decentralized agreement.

Table 5.5 presents the results for all the possible combinations between faultload and vehicle behavior. The first observation is that the contentious behavior imposes a significant performance penalty. For instance, in testbed *tb-pda* (802.11b), an execution with cooperative behavior takes an average of 41.30 ms, and 715.45 ms with contentious behavior. This result was expected given that the contentious behavior forces extra reliable broadcast and vector consensus executions. Even with this performance penalty, it is very interesting to note that the protocol can be executed quite fast in typical 802.11g network with moderately aged hardware. The executions with cooperative behavior in *tb-emulab* took around 3-4 ms in average, and around 60 ms with contentious behavior. These are promising results, specially considering that none of the protocols were optimized for wireless networks.

The observations about the wireless standard and the testbed impact are just confirmations of the results obtained in the previous sections. The testbed *tb-emulab* is faster than *tb-pda* under the same environmental conditions (802.11b and ad-hoc), and the 802.11g results are much better than the 802.11b.

Testbed/Standard	Faultload	Vehicle behavior	
		Cooperative (ms)	*Contentious* (ms)
tb-pda/802.11b	normal	41.30	715.45
	silent	29.24	501.11
	Byzantine	42.07	714.98
tb-emulab/802.11g	normal	4.21	64.31
	silent	3.81	57.09
	Byzantine	4.64	64.47
tb-emulab/802.11b	normal	32.71	579.54
	silent	24.80	407.72
	Byzantine	31.99	581.21

Table 2: Average latency for the speed agreement protocol in different testbeds with different vehicle behaviors and faultloads (ad-hoc mode).

The measurements about the faultload yield some interesting results. The *silent* faultload made the executions faster than the *normal* faultload. When one of the vehicles ceases communication, it alleviates the transmission medium, leaving more bandwidth available, and making the executions faster. Another interesting result is that the Byzantine failures did not had a noticeable impact on the performance of the algorithm. The resilience of the protocols extended to their performance, and confirms that the randomization techniques employed are not affected by Byzantine faults in terms of liveness, unlike most other approaches.

5.5.1 Feasibility of the Speed Agreement Algorithm

Despite the fact that many parameters would have to be taken into account to build a more complete control-based solution for a platooning application, we can reason about the possible impact of the communication delays affecting the behavior of the platoon. In particular, it is important to understand if the typical magnitude of protocol execution delays are adequate for this application.

To do that, let us consider a simple case, in which the platoon is stable and, at some point, the leader vehicle slows down, reducing its speed by 40 Km/h (we can assume this is done instantaneously). At this point, a new agreement would be started to propagate the new speed to all other vehicles. Clearly, there is a time bound after which the agreement result is useless, because the follower car would have approached too much and some safety mechanism would have been triggered to prevent a possible collision. Therefore, to ensure a smooth adaptation, the agreement protocol must terminate reasonably fast. But how fast?

For the mentioned speed reduction, the follower car will approach the leader at about 40 Km/h, that is, 11 m/s. Therefore, if some control algorithm to be executed by the vehicles is configured to stabilize with an inter-vehicle distance of about 15 meters, this would mean that a one second delay would be allowed for the agreement protocol execution, while keeping a safety distance margin of about 4 meters. This provides a rough idea of the values that

would be in place when considering this platooning application, showing that our results are in the general case well within these margins. Consequently, it seems possible to consider intrusion-tolerant solutions in this context.

6 Conclusions

This paper presents a performance evaluation of randomized intrusion-tolerant agreement protocols in 802.11 wireless networks, and the application and evaluation of these protocols in a car platooning application scenario subject to various types of faultloads. The following points summarize some important results obtained in these evaluations:

- The measurements taken in 802.11a/g networks (54 Mb/s) were considerably better than the ones taken in 802.11b (11 Mb/s), showing how the available bandwidth can affect the performance of the protocols.
- The execution of the protocols is slightly but consistently faster in 802.11a against 802.11g. Despite both being capable of achieving the same maximum data rate, the typical data rate is higher in 802.11a networks.
- The computational capability of the individual processes can represent a significant performance bottleneck. Nevertheless, as the cost of the protocols increase and more stress is put on the network, this bottleneck tends to shift from the computational capability to the network bandwidth.
- The introduction of an access point, and the consequential relay of all communication through it, imposes a general performance penalty on the protocols. The protocols are highly sensitivity to the network delays.
- Bracha's binary consensus is faster than ABBA binary consensus when the network conditions are better (i.e., higher bandwidth, lower latency). Nevertheless, there is a point, as network conditions degrade, at which ABBA outperforms Bracha's.

The car platooning application in particular shows that these protocols can be applied in practical settings. Given relatively old hardware (Pentium III PCs) and a typical 802.11g ad-hoc network, the speed agreement algorithm can be executed between four vehicles in a matter of a few milliseconds (3-4 ms) in normal conditions, and around 60 milliseconds if some car forces a decentralized agreement. Other interesting results to retain from the car platooning application pertain to the protocol behavior with faults. If some car does not communicate, the protocols actually execute faster. Additionally, the performance is unaffected even if some vehicle attempts to actively disrupt the execution of the protocols.

The results obtained in this paper will serve as a reference point for future work. In particular, they will help in the design of randomized agreement protocols specifically tailored for wireless networks, where the characteristics of these environments should be used to the advantage of the protocols to the maximum possible extent.

References

[1] P. G. Argyroudis and D. O'Mahony. Secure routing for mobile ad hoc networks. *IEEE Communications Surveys*, 7(3):2–21, 2005.

[2] F. Bonnet, P. Ezhilchelvan, and E. Vollset. Quiescent consensus in mobile ad-hoc networks using eventually storage-free broadcasts. In *Proceedings of the 21st Annual ACM Symposium on Applied Computing*, Apr. 2006.

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

[4] C. Cachin, K. Kursawe, and V. Shoup. Random oracles in Contanstinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, pages 123–132, July 2000.

[5] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov. 2002.

[6] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proceedings of the 3rd International Conference on Ad-Hoc Networks & Wireless*, pages 135–148, 2004.

[7] 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*, July 2005.

[8] 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.

[9] D. Djenouri, L. Khelladi, and N. Badache. A survey of security issues in mobile ad hoc and sensor networks. *IEEE Communications Surveys*, 7(4):2–28, 2005.

[10] V. Drabkin, R. Friedman, and M. Segal. Efficient Byzantine broadcast in wireless ad-hoc networks. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks*, June 2005.

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

[12] J. S. Fraga and D. Powell. A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd International Conference on Computer Security*, pages 203–218, Aug. 1985.

[13] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. Technical Report DCC-UFBA 12-2006, Universidade Federal da Bahia, 2006.

[14] Y.-C. Hu and A. Perrig. A survey of wireless ad hoc routing. *IEEE Security and Privacy*, 2(3):28–39, May./Jun. 2004.

[15] J.-P. Hubaux, S. Capkun, and J. Luo. The security and privacy of smart vehicles. *IEEE Security and Privacy*, 2(3):49–55, May./Jun. 2004.

[16] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith. The SecureRing group communication system. *ACM Transactions on Information and System Security*, 4(4):371–406, Nov. 2001.

[17] J. H. Lala, editor. *Foundations of Intrusion Tolerant Systems*. IEEE Computer Society Press, 2003.

[18] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Experimental comparison of local and shared coin randomized consensus protocols. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems (SRDS'06)*, pages 235–244, oct 2006.

[19] H. Moniz, N. F. Neves, M. Correia, and P. Veríssimo. Randomized intrusion-tolerant asynchronous services. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*, pages 568–577, jun 2006.

[20] M. K. Reiter. The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume 938 of *Lecture Notes in Computer Science*, pages 99–110. Springer-Verlag, 1995.

[21] S. Toueg. Randomized Byzantine agreements. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, pages 163–178, Aug. 1984.

[22] P. Veríssimo, N. F. Neves, and M. Correia. Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek, and A. Romanovsky, editors, *Architecting Dependable Systems*, volume 2677 of *Lecture Notes in Computer Science*, pages 3–36. Springer-Verlag, 2003.

[23] 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*, pages 166–175, Oct. 2005.

[24] 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 *Procedings of the Fifth Symposium on Operating Systems Design and Implementation*, pages 255–270. USENIX, Dec. 2002.

[25] D. Yanakiev and I. Kanellakopoulos. Nonlinear spacing policies for automated heavy-duty vehicles. *IEEE Transactions on Vehicular Technology*, 47(4):1365–1377, Nov. 1998.