UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Byzantine Fault-Tolerant Agreement Protocols for Wireless Ad hoc Networks

Henrique Lícias Senra Moniz

DOUTORAMENTO EM INFORMÁTICA
ESPECIALIDADE CIÊNCIA DA COMPUTAÇÃO

2010

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

Byzantine Fault-Tolerant Agreement Protocols for Wireless Ad hoc Networks

Henrique Lícias Senra Moniz

DOUTORAMENTO EM INFORMÁTICA
ESPECIALIDADE CIÊNCIA DA COMPUTAÇÃO

2010

Tese orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves
e pelo Prof. Doutor Miguel Nuno Dias Alves Pupo Correia

Abstract

The thesis investigates the problem of fault- and intrusion-tolerant *consensus* in resource-constrained wireless ad hoc networks. This is a fundamental problem in distributed computing because it abstracts the need to coordinate activities among various nodes. It has been shown to be a building block for several other important distributed computing problems like state-machine replication and atomic broadcast.

The thesis begins by making a thorough performance assessment of existing intrusion-tolerant consensus protocols, which shows that the performance bottlenecks of current solutions are in part related to their system modeling assumptions. Based on these results, the *communication failure model* is identified as a model that simultaneously captures the reality of wireless ad hoc networks and allows the design of efficient protocols. Unfortunately, the model is subject to an impossibility result stating that there is no deterministic algorithm that allows n nodes to reach agreement if more than $n-2$ omission transmission failures can occur in a communication step. This result is valid even under strict timing assumptions (i.e., a synchronous system).

The thesis applies randomization techniques in increasingly weaker variants of this model, until an efficient intrusion-tolerant consensus protocol is achieved. The first variant simplifies the problem by restricting the number of nodes that may be at the source of a transmission failure at each communication step. An algorithm is designed that tolerates f dynamic nodes at the source of faulty transmissions in a system with a total of $n \geq 3f + 1$ nodes.

The second variant imposes no restrictions on the pattern of transmission failures. The proposed algorithm effectively circumvents the Santoro-Widmayer impossibility result for the first time. It allows k out of n nodes

to decide despite $\sigma \leq \lceil \frac{n}{2} \rceil (n-k)+k-2$ omission failures per communication step. This algorithm also has the interesting property of guaranteeing *safety* during arbitrary periods of unrestricted message loss.

The final variant shares the same properties of the previous one, but relaxes the model in the sense that the system is asynchronous and that a static subset of nodes may be malicious. The obtained algorithm, called Turquois, admits $f < \frac{n}{3}$ malicious nodes, and ensures progress in communication steps where $\sigma \leq \lceil \frac{n-f}{2} \rceil (n-k-f)+k-2$. The algorithm is subject to a comparative performance evaluation against other intrusion-tolerant protocols. The results show that, as the system scales, Turquois outperforms the other protocols by more than an order of magnitude.

Keywords: Distributed systems, dependability, security, fault tolerance intrusion tolerance, agreement, consensus, wireless ad hoc networks

Resumo

Esta tese investiga o problema do *consenso* tolerante a faltas acidentais e maliciosas em redes ad hoc sem fios. Trata-se de um problema fundamental que captura a essência da coordenação em actividades envolvendo vários nós de um sistema, sendo um bloco construtor de outros importantes problemas dos sistemas distribuídos como a replicação de máquina de estados ou a difusão atómica.

A tese começa por efectuar uma avaliação de desempenho a protocolos tolerantes a intrusões já existentes na literatura. Os resultados mostram que as limitações de desempenho das soluções existentes estão em parte relacionadas com o seu modelo de sistema. Baseado nestes resultados, é identificado o *modelo de falhas de comunicação* como um modelo que simultaneamente permite capturar o ambiente das redes ad hoc sem fios e projectar protocolos eficientes. Todavia, o modelo é restrito por um resultado de impossibilidade que afirma não existir algoritmo algum que permita a n nós chegaram a acordo num sistema que admita mais do que $n-2$ transmissões omissas num dado passo de comunicação. Este resultado é válido mesmo sob fortes hipóteses temporais (i.e., em sistemas síncronos)

A tese aplica técnicas de aleatoriedade em variantes progressivamente mais fracas do modelo até ser alcançado um protocolo eficiente e tolerante a intrusões. A primeira variante do modelo, de forma a simplificar o problema, restringe o número de nós que estão na origem de transmissões faltosas. É apresentado um algoritmo que tolera f nós dinâmicos na origem de transmissões faltosas em sistemas com um total de $n \geq 3f + 1$ nós.

A segunda variante do modelo não impõe quaisquer restrições no padrão de transmissões faltosas. É apresentado um algoritmo que contorna efectivamente o resultado de impossibilidade Santoro-Widmayer pela primeira

vez e que permite a k de n nós efectuarem progresso nos passos de comunicação em que o número de transmissões omissas seja $\sigma \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$. O algoritmo possui ainda a interessante propriedade de tolerar períodos arbitrários em que o número de transmissões omissas seja superior a σ.

A última variante do modelo partilha das mesmas características da variante anterior, mas com pressupostos mais fracos sobre o sistema. Em particular, assume-se que o sistema é assíncrono e que um subconjunto estático dos nós pode ser malicioso. O algoritmo apresentado, denominado *Turquois*, admite $f < \frac{n}{3}$ nós maliciosos e assegura progresso nos passos de comunicação em que $\sigma \leq \lceil \frac{n-f}{2} \rceil (n - k - f) + k - 2$. O algoritmo é sujeito a uma análise de desempenho comparativa com outros protocolos na literatura. Os resultados demonstram que, à medida que o número de nós no sistema aumenta, o desempenho do protocolo Turquois ultrapassa os restantes em mais do que uma ordem de magnitude.

Palavras Chave: Sistemas distribuídos, confiabilidade, segurança, tolerância a faltas, tolerância a intrusões, acordo, consenso, redes ad hoc sem fios

Resumo Alargado

As redes de comunicação sem fios assumem um papel cada vez mais preponderante na sociedade. As redes ad hoc sem fios, em particular, representam uma tecnologia emergente que, nos tempos recentes, tem sido um importante foco de investigação por parte da comunidade científica internacional. Este tipo de redes existe sem nenhuma forma de controlo centralizado. A estas não se aplica a noção de infra-estrutura e todos os nós da rede, em princípio, desempenham igual papel na sua operação. Esta natureza descentralizada das redes ad hoc sem fios torna-as muito apropriadas para situações de emergência, tais como desastres naturais ou conflitos militares, em que a dependência num único ponto de falha, não só não é desejável, como talvez inalcançável.

Um tema recorrente em muitos cenários de comunicação ad hoc é o da *coordenação local*. Dispositivos sem fios, que se encontrem a uma distância que permita a comunicação directa entre si, necessitam frequentemente de sincronizar as suas acções de modo a executarem uma qualquer tarefa distribuída. Esta situação não acontece por mero acaso. A proximidade entre os nós geralmente implica que estes tenham que partilhar um determinado recurso. Nesse caso, alguma forma de coordenação é necessária para que o acesso a esse recurso seja efectuado de forma justa e eficiente. O recurso pelo qual os nós do sistema competem pode ser, por exemplo: o espectro electromagnético, onde os nós tentam sincronizar os seus períodos de transmissão de maneira a que não hajam sobreposições (Clementi *et al.*, 2001; Kowalski, 2005); uma estrada, onde um conjunto de veículos semi-automatizados tenta decidir qual a ordem pela qual atravessam um cruzamento (Misener *et al.*, 2005); ou o espaço aéreo, onde vários aviões coordenam as suas manobras de modo a evitarem colisões (Brown, 2007; Moniz *et al.*, 2009).

Os nós de um sistema também necessitam muito frequentemente de se sincronizar de modo a maximizar os seus recursos individuais. Por exemplo, numa rede de sensores, os nós, tipicamente, organizam-se em grupos compostos por nós fisicamente próximos uns dos outros, em que cada grupo é gerido por um nó líder que coordena certas operações como o encaminhamento ou a agregação de dados (Dong & Liu, 2009;

Kuhn *et al.*, 2006). Esta tarefa de gestão de um grupo pode exigir um consumo de energia substancial no nó líder e/ou aumentar a sua probabilidade de falha. De modo a balancear a carga e garantir tolerância a faltas, os nós podem necessitar de se coordenar para eleger um novo líder periodicamente.

A complexidade relacionada com estes cenários de coordenação local pode ser abstraída através de uma primitiva de *consenso*. Este é um problema fundamental dos sistemas distribuídos. Informalmente, pode ser definido do seguinte modo: cada um dos processos que compõe o sistema propõe um valor e todos os processos têm que decidir um valor comum baseado nos valores propostos. Esta definição captura a essência da coordenação, que passa, fundamentalmente, pelos processos chegarem a acordo sobre uma determinada informação. Muitos tipos de actividades coordenadas num sistema distribuído podem ser reduzidos ao problema do consenso. Por exemplo, o consenso foi demonstrado ser equivalente a outras operações distribuídas como a replicação de máquina de estados (Schneider, 1990) ou a difusão atómica (Correia *et al.*, 2006).

Apesar da sua definição simples, o consenso em sistemas tolerantes a faltas está longe de ser um problema trivial. De facto, dependendo dos pressupostos que são feitos sobre o sistema, pode dar-se o caso de simplesmente não existir uma solução determinista para o consenso (Fischer *et al.*, 1985; Santoro & Widmeyer, 1989). Por exemplo, o resultado de impossibilidade FLP (Fischer *et al.*, 1985) mostra que não existe nenhuma solução determinista para o problema do consenso em sistemas assíncronos mesmo que se admita que apenas um nó do sistema pode falhar. Um sistema assíncrono é caracterizado por não assumir quaisquer pressupostos sobre o seu tempo de execução, de modo que não é possível recorrer, por exemplo, a mecanismos temporais para detectar falhas de nós.

A tese foca-se no problema do consenso tolerante a faltas acidentais e maliciosas em redes ad hoc sem fios. Estes ambientes apresentam variados desafios à computação de tarefas distribuídas. Os seus recursos são tipicamente restritos, tanto em termos de poder computacional como de comunicação. A falta de fiabilidade inerente às comunicações por rádio também representa outro desafio significativo. Fenómenos ambientais tais como condições atmosféricas adversas, interferência de rádio, ou colisões nas transmissões, contribuem para a perda de sinal. A própria natureza dos nós que

compõem estas redes, que são muitas vezes móveis, também pode levar a períodos arbitrários de perda de conectividade. Finalmente, a presença de elementos maliciosos, algo provável em certas aplicações críticas, intensifica ainda mais a dificuldade de projectar protocolos de consenso para estes ambientes. Idealmente, o sistema deve fornecer um serviço correcto mesmo que alguns dos seus nós sejam atacados e controlados por um adversário sofisticado. Neste caso, diz-se que o sistema é *tolerante a intrusões* (Fraga & Powell, 1985; Veríssimo *et al.*, 2003).

A concretização de protocolos de consenso que sejam simultaneamente eficientes e sustenham a sua correcta execução face a este tipo de cenários, representa uma contribuição inovadora e um importante passo no sentido da aplicação das redes ad hoc sem fios a cenários críticos.

A primeira contribuição da tese passa por uma exaustiva avaliação de desempenho de protocolos tolerantes a intrusões existentes na literatura. Para este passo, procedeu-se à escolha de protocolos de acordo que possuam características que os tornem interessantes para um ambiente de comunicação sem fios. Em primeiro lugar, teriam que ser protocolos tolerantes a intrusões dado que, em última instância, o objectivo passa por concretizar protocolos que possuam esta característica. Em segundo lugar, a execução dos protocolos teria que ser descentralizada. Dada a necessidade de preservar recursos e lidar com a potencial mobilidade dos nós, a execução dos protocolos deverá evitar depender de um nó líder. Finalmente, os protocolos deveriam assumir um modelo de sistema assíncrono. Dada a pouca fiabilidade inerente às redes ad hoc sem fios, a independência face a qualquer tipo de pressupostos temporais garante uma execução correcta mesmo na presença de um comportamento temporal mais imprevisível do sistema.

Naturalmente, estas características implicam que os protocolos estejam sujeitos ao, já mencionado, resultado de impossibilidade FLP. Isto implica que o seu modelo de sistema tenha que ser estendido de alguma forma que permita uma solução para o consenso. Esta questão tem sido estudada exaustivamente na literatura da área, existindo algumas soluções. No entanto, a sua maioria exige a inclusão de hipóteses temporais mais fortes no modelo, seja de forma explícita (e.g., modelos de sincronia parcial (Dolev *et al.*, 1987; Dwork *et al.*, 1988)) ou implícita (e.g., detectores de falhas (Chandra & Toueg, 1996; Kihlstrom *et al.*, 2003)). A única excepção é a *aleatoriedade*, cuja solução passa por abordar o problema do consenso através de uma per-

spectiva probabilista. Por permitir manter o sistema totalmente livre de pressupostos temporais, a escolha recaiu sobre protocolos que recorram a esta técnica de modo a contornar o resultado FLP. Outro argumento a favor da aleatoriedade é que, ao contrário das outras técnicas, a execução dos seus algoritmos é inerentemente descentralizada.

Os resultados de avaliação obtidos neste passo levaram a importantes observações sobre o desempenho de protocolos com recurso a aleatoriedade em situações práticas. Em primeiro lugar, os resultados demonstram que este tipo de protocolos pode ser encarado como uma solução eficiente para sistemas tolerantes a intrusões em redes locais *com* fios. Esta conclusão vai contra a percepção generalizada sobre estes protocolos, cuja análise teórica determina que o seu tempo de execução é exponencial em relação ao número total de nós do sistema. A análise de desempenho expõe a distância que existe entre a teoria e a prática e prova que, na prática, os protocolos terminam tipicamente em uma ou duas rondas de comunicação. No entanto, apesar dos resultados positivos em redes locais com fios, o seu desempenho em ambiente sem fios não foi tão satisfatório, ficando patentes certas limitações. A principal conclusão que se retirou desta análise foi que o fraco desempenho está em parte relacionado com o modelo de sistema assumido por estes protocolos (que é virtualmente extensível a todos os protocolos tolerantes a intrusões) e, em particular, ao seu modelo de faltas, que força a concretização de canais fiáveis ponto-a-ponto entre todos os processos. Esta característica não permite aos protocolos um acesso eficiente ao meio de comunicação partilhado. Por exemplo, não permite que se recorra a primitivas de difusão não-fiável, mais alinhadas com a natureza das redes sem fios.

Esta conclusão levou à identificação do *modelo de falhas de comunicação*, introduzido em Santoro & Widmeyer (1989), como sendo um modelo que simultaneamente captura o ambiente das redes ad hoc sem fios e permite a criação de protocolos eficientes. Este modelo assume que as transmissões de mensagens entre os nós estão sujeitas a faltas de comunicações dinâmicas e transitórias, i.e., a comunicação entre quaisquer dois nós pode ser alvo de faltas num determinado momento e correcta noutro. Esta visão altamente dinâmica das faltas de comunicação encontra-se alinhada com a natureza das redes ad hoc sem fios porque captura a perda momentânea de comunicação relacionada com a mobilidade dos nós ou a pouca fiabilidade da rede. O

maior benefício desta aproximação ao problema é que a inexistência de hipóteses sobre a fiabilidade de qualquer comunicação entre dois nós liberta o sistema de ter que concretizar mecanismos de garantia de entrega de mensagens. Isto permite que os protocolos retirem proveito de todo o potencial de comunicação por difusão, tão natural num ambiente sem fios, onde o custo de transmitir uma mensagem para um nó pode ser o mesmo de transmitir para os n nós do sistema, desde que o raio de transmissão dos nós permita comunicação directa entre si.

Apesar da sua utilidade para representar uma rede ad hoc sem fios, a investigação no modelo de falhas de comunicação tem sido muito reduzida. A este facto está relacionado um importante resultado de impossibilidade associado ao modelo. O resultado, denominado *resultado de impossibilidade Santoro-Widmayer*, aplica-se a qualquer problema de acordo num sistema de n nós e determina que é impossível conseguir resolver o acordo deterministicamente se for assumido que se podem perder mais de $n - 2$ mensagens transmitidas num dado passo de comunicação. Este é um resultado muito desencorajador, dado que a falha por paragem de um só nó resulta necessariamente em $n - 1$ omissões de mensagens, levando a que o acordo seja impossível. Para tornar o problema mais difícil, este resultado aplica-se a sistemas totalmente síncronos, onde tanto os tempos máximos de processamento dos nós como os de transmissão de mensagens na rede são conhecidos e assumidos que se cumprem.

Uma das maiores contribuições desta tese prende-se com o desenho de uma solução capaz de contornar este resultado de impossibilidade recorrendo a técnicas de aleatoriedade. Neste sentido, a tese aborda o problema do consenso em variantes progressivamente mais fracas do modelo, até ser alcançado um protocolo eficiente e tolerante a intrusões.

A primeira variante do modelo restringe o número de nós que estão na origem de transmissões faltosas. É apresentado um algoritmo que tolera f nós dinâmicos na origem de transmissões faltosas em sistemas com um total de $n \geq 3f + 1$ nós. A segunda variante do modelo não impõe quaisquer restrições no padrão de transmissões faltosas. É apresentado um algoritmo que contorna efectivamente o resultado de impossibilidade Santoro-Widmayer, e que permite a k de n nós efectuarem progresso nos passos de comunicação em que o número de transmissões omissas seja $\sigma \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$. O algoritmo possui ainda a interessante propriedade de tolerar períodos arbitrários em que o número de transmissões omissas seja superior a

σ. A última variante do modelo partilha das mesmas características da variante anterior, mas com pressupostos mais fracos sobre o sistema. Em particular, assume-se que o sistema é assíncrono e que um subconjunto estático dos nós pode ser malicioso. O algoritmo apresentado, denominado *Turquois*, admite $f < \frac{n}{3}$ nós maliciosos e assegura progresso nos passos de comunicação em que $\sigma \leq \lceil \frac{n-f}{2} \rceil (n - k - f) + k - 2$. O algoritmo é sujeito a uma análise de desempenho comparativa com outros protocolos na literatura. Os resultados demonstram que, à medida que o número de nós no sistema aumenta, o desempenho do protocolo Turquois ultrapassa os restantes em mais do que uma ordem de magnitude.

Acknowledgements

Phew. This has been quite a journey.

First and foremost, I want to thank my advisors, Nuno Neves and Miguel Correia. With them, the delicate balance between indulgence and leadership that I needed for my research always prevailed. The creative freedom that they entrusted in me gave the inspiration necessary to produce something that I can call my own. Their tireless support during the most fundamental periods was the key to bring this thesis to a conclusion. Most importantly, it was their uncompromising nature to push only towards the most excellent work that, ultimately, instilled me with the same spirit.

A special acknowledgement also goes to Paulo Veríssimo for his relentless work in leading our research group. I would also like to thank the FCT for financially supporting me during a period of my graduate studies.

To my lab colleagues for their friendship, understanding, and technical discussions. A very special thank you goes to all the people who took their own time and energy to provide their invaluable support and feedback when I was preparing for the DISC 2009 presentation. By risking leaving someone out, I want to acknowledge João Antunes, Alysson Bessani, Luís Brandão, Mônica Dixit, João Leitão, and Giuliana Santos.

To my nuclear group of friends for being them, and for their unrelenting faith in me. You know who you are.

During this work, there were a few times where I felt transcending my abilities, and achieved heights that I previously thought to be beyond me. This was possible because I was being propelled by the inner strength of two people. My deepest acknowledgement goes to my fiancée, Tânia. This thesis was born with our love. Every uncertain period was endured only

with her passionate support and encouragement. Thank you, honey. You are my co-author in life.

Finally, to my parents for always letting me chose my own path.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Context and Motivation

Impetus. Wireless networks are becoming increasingly prevalent in society. Devices such as smartphones, PDAs, laptop computers, and GPS units, which rely on wireless technologies like cellular telephony, bluetooth, wi-fi, and satellite communications, are just a few examples of how pervasive this technology is in our lifes.

Wireless ad hoc networks, in particular, represent an emergent area that has been the subject of considerable attention within the scientific 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 archetypal scenario of wireless ad hoc networks is a collection of nodes, none of which possessing any distinguishing processing or communication capabilities, spread out in some geographical area and exchanging messages through radio waves.

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. In the near future, we can expect wireless ad hoc networking to invade many aspects of modern life, either in civil or military contexts, and from the mundane to the life-

critical. Sophisticated vehicular coordination (Hull *et al.*, 2006; Nadeem *et al.*, 2004), automated air traffic management (Brown, 2007; Moniz *et al.*, 2009), gaming (Claypool, 2005), or social networking (Motani *et al.*, 2005), are just a few examples of promising new applications of wireless ad hoc networks.

Research. Wireless ad hoc networks are an extremely active subject of research across many domains of computer science, and many interesting problems were motivated by these networking environments.

For example, there is the problem of how a node can learn its own localization (Aspnes *et al.*, 2006; Priyantha *et al.*, 2000, 2005) and locate other nodes (Abraham *et al.*, 2004; Awerbuch & Peleg, 1995; Li *et al.*, 2000). There is the study of the communication capacity of wireless networks, which allows the calculation of the theoretically achievable efficiency of many ad hoc networking scenarios (Grossglauser & Tse, 2002; Gupta & Kumar, 2000; Moscibroda & Wattenhofer, 2006). There is also the topology control problem, an interesting distributed coordination problem, which tries to find the optimal balance between transmission power and network connectivity (Heide *et al.*, 2004; Li *et al.*, 2005). Another common line of research is related to the clustering of nodes in the network, which involves solving graph theory problems, such as finding connected dominating sets and maximal independent sets, for forming a communication backbone (Luby, 1985; Moscibroda & Wattenhofer, 2005; Wan *et al.*, 2004; Wu & Li, 1999). A final example is the problem of finding local algorithms for distributed tasks, where a node communicates only with its closest neighbors and computes in time independent of the size of the network (Kuhn *et al.*, 2004; Naor & Stockmeyer, 1993).

The dynamic and unreliable nature of wireless ad hoc networks also raises new challenges for previously well-studied problems. Medium access control (Rajendran *et al.*, 2006; Ye & Heidemann, 2004), routing (Broch *et al.*, 1998; Draves *et al.*, 2004; Johnson & Maltz, 1996; Karp & Kung, 2000; Perkins & Belding-Royer, 1999), broadcasting (Lou & Wu, 2002; Sasson *et al.*, 2003; Stojmenovic *et al.*, 2002), and clock synchronization (Elson & Romer, 2003; Sundararaman *et al.*, 2005) are all classical problems that, within the context of wireless ad hoc networks, are faced with a greater degree of uncertainty stemming from frequent topological changes and unreliable communication.

Local synchronization. A recurring theme in many wireless ad hoc networking scenarios is *local synchronization*. Nodes that are within direct communication range of each other frequently have to synchronize their actions in order to perform some task. This situation does not arise by mere chance. Proximity usually implies sharing a common physical resource and some form of coordination is required such that access to that resource is fair and efficient. There are many possibilities to what the shared resource can be. For example, it can be the electromagnetic spectrum, where nodes synchronize their broadcasting periods such that there is no overlap (Clementi *et al.*, 2001; Kowalski, 2005); 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 (Misener *et al.*, 2005); or it can even be the airspace, where aircraft coordinate their maneuvering in order to avoid collisions (Brown, 2007; Moniz *et al.*, 2009).

Nodes may also have to synchronize to maximize their own individual resources. For example, in densely-deployed sensor networks, nodes are often organized into clusters of nodes that are physically close to each other, with every cluster being managed by a cluster leader that coordinates operations such as routing and data aggregation (Dong & Liu, 2009; Kuhn *et al.*, 2006). The task of managing a cluster may require substantial energy consumption and/or may increase the chance of failure of the cluster leader. For purposes of load balancing and fault tolerance, the leader may need to be periodically re-elected.

The topology control problem mentioned above is another example where nodes synchronize to maximize their internal resources. In this problem, nodes calibrate their transmission power, keeping it as near as possible to the minimum value, while maintaining network connectivity. This has the effect of preserving energy and reducing the overall contention in the network.

Consensus. The complexity involved in each one of these local synchronization scenarios can be normally abstracted through a consensus primitive. Informally, it can be defined in the following way: each process in the system proposes some value and all processes have to decide on a common value obtained from the proposals. Albeit simple, this definition captures the essence of coordination, i.e., the agreement on some common piece of information.

For example, the connection of leader election and consensus has been widely studied (Malkhi *et al.*, 2005; Mostefaoui & Raynal, 2001; Sabel & Marzullo, 1995). Leader election can be achieved by having each node propose the ID of their preferred process for leader. A consensus execution outputs a single common node ID in every node, which becomes the new leader. In the k-selection problem (Clementi *et al.*, 2001; Kowalski, 2005), where nodes try to synchronize their broadcasting periods such that there is no overlap, a consensus instance can be run for each process to decide which is its broadcasting slot. The problem has also been shown to be a building block for several other important distributed computing problems like state-machine replication (Schneider, 1990) and atomic broadcast (Correia *et al.*, 2006). State-machine replication is a general method for replicating services, while atomic broadcast ensures the delivery of messages in the same order by all processes.

Challenges. Wireless ad hoc networks, however, present several challenges to distributed computing. They are typically resource-constrained both in terms of computing power and communication capacity. For example, they have less bandwidth than wired local-area-networks, their shared communication medium leaves them more vulnerable to denial-of-service attacks, and their nodes are usually mobile devices with limited energy and computing capabilities, whose very nature may lead to arbitrary periods of disconnection. The inherent unreliability of radio communications is another significant challenge. Pervasive physical phenomena - such as poor atmospheric conditions, fading, or interference - contribute to signal loss. On top of this, link-layer reliability is usually based on best-effort mechanisms (e.g., CSMA/CA). Collisions in wireless networks are hard to detect. The radio communication is usually half-duplex, which makes impossible for the transmitter to sense the channel for collisions while transmitting data. Some proposals for MAC layer protocols that support collision detection on the receiver side do exist (Polastre *et al.*, 2004; Whitehouse *et al.*, 2005), but so far the ability to provide reliable collision detection in wireless environments remains a major technical challenge. Finally, the presence of malicious elements, which is likely given the critical services potentially provided by wireless ad hoc networks, further intensifies the difficulty of designing agreement protocols for these environments.

A consensus protocol designed for wireless ad hoc networks must cope with the inherent unreliability of the environment. It is imperative that consensus is performed in a dependable way, especially given the suitability of ad hoc networks to critical situations. Neither the momentary breakdown of communications nor the failure of some nodes, either of accidental or malicious nature, should be synonymous to the failure of the entire system. Hence, nodes that do not fail need to be able to reach agreement even if others are uncooperative, either by not communicating, crashing, or plainly acting selfishly or maliciously. When correct system behavior is ensured despite the failure of some of its components, it is said that the system is *fault-tolerant*. Furthermore, if these components fail because they are compromised and controlled by an intelligent adversary, the system is *intrusion-tolerant* (Fraga & Powell, 1985; Veríssimo *et al.*, 2003).

Despite its simple definition, the solution to consensus in fault-tolerant distributed systems is far from trivial. In fact, depending on the assumptions that one makes about the system, it may very well be the case that there is simply no deterministic solution to consensus (Fischer *et al.*, 1985; Santoro & Widmeyer, 1989). For example, the widely-known FLP impossibility result (named after the initials of its authors) states that there is no deterministic solution to consensus in asynchronous systems (i.e., where no assumptions are made about timeliness) if only one process can crash (Fischer *et al.*, 1985).

1.2 Research Methodology

Goal. The overarching goal of this thesis is to devise efficient consensus protocols for wireless ad hoc networks. Our focus is on single-hop networks, which we believe to be an important scenario given the prevalent need for local synchronization in wireless environments. 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., Vahdat & Becker (2000)), which is a well-studied problem in the context of wireless networks.

The problem is addressed from both a theoretical and practical perspective. The work described in this thesis involves on one hand the definition of appropriate mod-

els and algorithms for distributed computing in wireless ad hoc networks and overcoming the associated lower bounds and impossibility results, and, on the other hand, the implementation and experimental evaluation of consensus protocols in real-word testbeds.

The designed protocols must be able to withstand a highly unreliable, and perhaps hostile, environment. This implies tolerating accidental faults such as message loss and node crashes, and, especially, arbitrary node behavior due to intrusions. Intrusion-tolerant protocols, when specifically tailored for efficient operation in wireless environments, represent a novel contribution and important step towards the deployment of critical services in these environments.

Performance Assessment of Existing Protocols. The first step in designing efficient consensus protocols for wireless ad hoc networks is to identify an appropriate system model for these environments. This becomes a fundamental step if one wishes to achieve good performance by exploiting the particular characteristics of wireless environments. For example, the openness of the network provides a natural broadcasting medium, where the cost of transmitting a message to multiple processes can be just the same of transmitting it to a single process, as long as they are within communication range.

Finding an appropriate system model requires a great deal of insight into the practical performance of these protocols. Consequently, our first step is to understand how current intrusion-tolerant agreement protocols perform in wireless ad hoc networks, when compared to traditional (wired) local-area networks. To this end, it was carried a thorough performance assessment of two families of (randomized) intrusion-tolerant protocols under several different environmental settings. This step significantly deepened our perception about the performance of these protocols, and, in particular about their shortcomings in wireless environments, which, as our results confirm, stem essentially from their classical modeling assumptions. The knowledge obtained in this step was then used to identify a more appropriate system model for wireless ad hoc networks.

Randomization. When surveying existing protocols to be evaluated, we looked for three main characteristics that are particularly interesting for wireless ad hoc networks.

First, we looked for intrusion-tolerant protocols because, ultimately, our goal was to design algorithms that sustained both accidental and malicious faults. Second, the execution of the protocols had to be decentralized (or leader-free). Wireless ad hoc networks are, by definition, decentralized. Given the need to preserve resources and deal with potential mobility of nodes, the progress of protocols should not depend on any single process. Third, the protocols would have to be designed for an asynchronous model of computation. Given the inherent unreliability of wireless networks, the independence of any kind of timing assumptions would ensure correctness in face of even the most unpredictable timing behavior by the network.

Of course, any protocol with such characteristics is subject to the aforementioned FLP impossibility result, which states that there is no deterministic solution to fault-tolerant consensus under the asynchronous model. This implies that the system model has, in some way, to be changed in order to circumvent this result and make consensus solvable. While this is a widely studied problem, most of the available approaches require the inclusion of stronger timing assumptions (e.g., partial synchrony models (Dolev *et al.*, 1987; Dwork *et al.*, 1988)), or the addition of devices that hide in their implementation these assumptions, such as failure detectors (Chandra & Toueg, 1996; Kihlstrom *et al.*, 2003) or wormholes (Neves *et al.*, 2005). The exception is *randomization*. This technique relies on a probabilistic approach to consensus (Ben-Or, 1983; Rabin, 1983). In a randomized protocol, there are certain steps in which the current proposal may take a random value, chosen according to a given probability distribution. This means that an adversary scheduler cannot determine the outcome of any disruptive strategy because that outcome is affected by a random element outside of its control. The main advantage of this approach is that it allows the system to remain completely asynchronous. To achieve this, one has to judiciously weaken the problem statement to allow a probabilistic termination of consensus.

Another strong argument in favor of randomization is that its algorithms are inherently decentralized, unlike other approaches to the FLP impossibility, which tend to produce algorithms that rely on a leader during normal operation and falling back to a leader election procedure in case the failure of the leader is detected. For example, until recently, designing a leader-free intrusion-tolerant consensus algorithm for partially synchronous systems was a problem that was yet to be approached (Borran & Schiper, 2010).

For these reasons, it was decided that the performance assessment would focus on randomized consensus protocols. By itself, this assessement represents one of the major contributions of the thesis. It involved the implementation of more than half a dozen agreement protocols in two different platforms (i.e., Linux and Windows Mobile), and their evaluation under several different settings including wired and wireless networks. The assessment was structured to reach two main goals: (1) to provide a deep insight into the performance of fault- and intrusion-tolerant protocols that could be used to identify a more appropriate system model for wireless ad hoc networks; and (2) to raise general awareness about the behavior and practical possibilities of randomization.

That second point was an important achievement because our results demonstrate that randomized protocols can indeed be efficient and should be regarded as a valid solution for practical fault- and intrusion-tolerant distributed systems. This realization is contradictory to the general intuition regarding randomized protocols, in particular those of the *local coin* class[1], which have been historically dismissed as being too inefficient due to their exponential expected time complexity. The obtained results, however, show otherwise. This was mainly because the adversarial model under which these protocols are designed (i.e., one that completely decides the order in which the processes receive the messages) is rarely, if ever, observed in practice. In fact, a real adversary possessing such power could probably perform much more devastating attacks such as blocking the communication entirely. Therefore, in practice, the network scheduling leads to a speedy termination (usually in one or two communication rounds). For example, in some scenarios with 4 processes, the local coin binary consensus protocol (or Bracha's protocol (Bracha, 1984)) terminated in under 1 ms in a wired local-area-network. Even with 10 processes, if every process proposed the same initial value, consensus was achieved in around 4 ms. These results roughly double if the proposals of processes diverged, which usually required an additional execution round.

[1]There are algorithms that are based on *local coin* mechanism that returns random bits observable locally by each process. This class exists in juxtaposition to algorithms based on a *shared coin*, which returns bits observable by all processes

A System Model for Wireless Ad hoc Networks. Despite the positive results obtained in wired LANs, the performance of the protocols in wireless settings, more specifically in 802.11a/b/g networks, was relatively poor. For example, in one set of experiments in a 802.11b wireless LAN, the Bracha's binary consensus (a local coin protocol) took an average of 35 ms to terminate with 4 processes, and this value increased to 415 ms with 10 processes. The ABBA binary consensus (a shared coin protocol) took 146 ms and 301 ms, respectively for 4 and 10 processes. These results demonstrate the difficulty of applying these protocols to wireless environments.

In part, these results can be attributed to the particular characteristics of the evaluated protocols - Bracha's protocol exchanges too many messages and ABBA relies on computationally expensive cryptographic operations, which were taxing even for Pentium III PCs, and much more for resource-constrained mobile devices. Nevertheless, while analyzing the results it became clear that a more systemic bottleneck affects performance, which is directly related to the employed system model, in particular the fault model.

Generally, faults can be assumed to occur in two components: processes and links. The model used by virtually every intrusion-tolerant protocol (randomized or not) assumes that faults occur only in processes and that these are connected by reliable point-to-point communication links. Furthermore, these faults are commonly assumed to be static, i.e., a fault is associated to a particular process that is forever considered faulty. While it is true that are a few protocols (e.g., Paxos (Lamport, 1998) and PBFT (Castro & Liskov, 1999)) that explicitly assume arbitrary messages loss in their channels, these are constrained by a fair loss property. This property states that if a correct process p sends a message m to another correct process q an infinite number of times, then q receives m from p an infinite number of times. In practice, this has the same effect of assuming asynchronous reliable point-to-point channels because the correctness of the protocols is dependent on the fact that every message sent from one correct process to another must be delivered (as long as it keeps being retransmitted). From the perspective of wireless ad hoc networks, the downside of this approach is that it forces the implementation of end-to-end message delivery mechanisms (e.g., TCP). Unfortunately, implementing point-to-point communication on top of a shared communication medium can be a very inefficient approach because it does not allow a natural use of the wireless broadcasting medium.

In order to fully exploit the natural characteristics of wireless ad hoc networks, the unreliability of wireless communications has to be explicitly assumed. Any message can be lost at any time and previous faulty behavior is not an indication of future behavior, i.e., communication faults are dynamic and transient. Any protocol that can tolerate this kind of uncertainty is free to efficiently access the shared broadcasting medium of wireless ad hoc networks.

This realization led to the identification of the *communication failure model* as a more appropriate model for wireless ad hoc networks. This model was introduced by Nicola Santoro and Peter Widmayer in a seminal paper entitled *'Time is not a Healer'* (Santoro & Widmeyer, 1989). Under the communication failure model, the system is affected by transmission faults that can occur anywhere in the system, i.e., the set of links affected by failures may change at every clock cycle. This precisely captures the unpredictability that naturally occurs in wireless ad hoc networks.

This fault model is very general. Despite assuming dynamic failures, it can also capture permanent and localized failures. For example, the crashing of a process can be described as a series of transmission omission faults with the crashed process as sender.

Santoro-Widmayer Impossibility Result. Research in the communication failure model, however, has been limited. This is mainly due to an associated impossibility result, dubbed the Santoro-Widmayer impossibility result. Basically, it states that there is no deterministic algorithm that that solves some form of non-trivial agreement (consensus included) if more than $n - 2$ omission failures can occur in a communication step. This is a very discouraging result, in the same line of the classic FLP result, because, in practice, the crashing of a single process necessarily results in $n - 1$ transmission failures, rendering any form of agreement impossible. Moreover, this result is produced under strong timing assumptions where both the processes' relative processing times and communication delays are bound by known constants (i.e., a synchronous system). Hence the motto, *time is not a healer*.

This result has been a major obstacle to the development of distributed algorithms under this model, despite its potential usefulness. Unlike the FLP impossibility result for asynchronous systems, which has been widely studied by the scientific community (e.g., Ben-Or (1983); Chandra & Toueg (1996); Dwork *et al.* (1988); Neves *et al.*

(2005); Rabin (1983)), the Santoro-Widmayer result has eluded researchers for the past 20 years. Since the fault model in which the FLP result applies (i.e., process crash failures) can be emulated in the communication failure model, it is easy to see how the uncertainty that renders agreement impossible in the FLP result is similar to the uncertainty that gives rise to the Santoro-Widmayer impossibility.

Most of the existing approaches to circumvent the FLP result cannot be directly applied to the Santoro-Widmayer result. These usually rely on time or failure detection. Unfortunately, strengthening the timing assumptions clearly does not help because the result is valid under the synchronous model. While failure detection is possible (because of the assumed timing model), it cannot be relied upon because faults are dynamic, and therefore past behavior is not indicative of future behavior. Randomization, however, is an approach that can be applied to circumvent this impossibility result, as shown in this thesis.

Exploration of the Communication Failure Model. The thesis explores the communication failure model in increasingly weaker variants until an efficient intrusion-tolerant consensus protocol is devised, which is then subject to a comparative performance evaluation.

While many formulations of consensus exist, the problem is studied in its binary variant where each process proposes a value 0 or 1, and then they have to decide on one of the proposed values. This simple problem definition allows us to focus on what is essential. First, this problem embodies the most difficult challenge in designing agreement protocols, which requires circumventing the associated impossibility results. Developing extensions from binary consensus to other variants is often an easier task because the underlying limitations of agreement have already been dealt with (Correia *et al.*, 2006). Second, in this work, the pursuit for efficiency is considered a major objective. Performance often stems primarily from the assumed system model, which can then be complemented with particular heuristic optimizations employed by the algorithms (some of these are later explored in Chapter 7). Any performance gains obtained through solving binary consensus in the communication failure model will naturally percolate to other higher-level protocols. Third, focusing on binary consensus simplifies the comparison of our algorithms to those already evaluated (see Chapter 3) and provides empirical data that can justify the system assumptions.

Consensus with communication faults of restricted source. The problem is first approached under a simplified variation of the communication failure model that restricts the number of processes where transmission faults can originate. The system is assumed to be synchronous and faults can be of the omission (i.e., messages can be lost) and corruption (i.e., messages can be modified) types. This essentially captures both unreliable communication and malicious behavior of nodes. Although the fault pattern that can occur at each time step is restricted to originate at most in f processes, these processes can arbitrarily change from one time step to another. The interest of this model is purely theoretical because it is possibly unrealistic to restrict the pattern of faults in this way. Nevertheless, despite being a restricted version of the communication failure model, this model is still constrained by the Santoro-Widmayer impossibility result. A randomized protocol is presented that can tolerate f process at the source of transmission faults per step, with $n \geq 3f + 1$.

Circumventing the Santoro-Widmayer impossibility result. The following step focused on truly circumventing the Santoro-Widmayer impossibility result, i.e., admitting more than $n-2$ omission faults regardless of their pattern. This was accomplished through a randomized algorithm under a synchronous system model. This step is a milestone in the thesis and represents a major theoretical contribution because it is the first algorithm that effectively achieves this objective. The algorithm allows for k processes to decide on a binary value in a system with n processes such that $k > \frac{n}{2}$. The safety properties of consensus (i.e., validity and agreement) are ensured even with an unrestricted number of faults, while progress is attained in steps where the number of faults is $\sigma \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$. These properties are adequate for wireless ad-hoc networks because they allow the algorithm to tolerate arbitrary periods of unrestricted message loss without compromising safety.

Byzantine consensus in wireless ad hoc networks. The final step in our exploration of the communication failure model was directed at obtaining an efficient intrusion-tolerant protocol. To this end, we designed, implemented and evaluated Turquois - a binary consensus protocol for wireless ad hoc networks that tolerates both dynamic omission faults and Byzantine processes. It fills an important gap in the distributed

computing literature by being the first intrusion-tolerant consensus algorithm specifically designed for wireless ad hoc networks. Its characteristics are close to ideal for wireless communication. It assumes an asynchronous system model subject to dynamic communication failures and a static subset of Byzantine processes. This makes it subject to both the FLP and the Santoro-Widmayer impossibility results. For this reason, Turquois represents a very challenging design from a theoretical perspective. It is possibly the weakest model on which Byzantine consensus has been solved. Nevertheless, the algorithm is optimal in the number of Byzantine processes that tolerates (i.e, $f \leq \lfloor \frac{n-1}{3} \rfloor$), and, like in the previous iteration, it ensures safety despite arbitrary periods of unrestricted message omissions. Progress is ensured in communication steps where the number of omissions is $\sigma \leq \lceil \frac{n-f}{2} \rceil (n - k - f) + k - 2$. Fundamental to the operation of Turquois is a novel mechanism for message authentication based on computationally inexpensive cryptographic hash functions, which avoids the use of public-key cryptography during normal execution.

Turquois is subject to a thorough comparative performance evaluation with the protocols of Bracha (Bracha, 1984) and ABBA (Cachin *et al.*, 2000). This evaluation validates our modeling claims. The results show that, as the number of the nodes in the system increases, Turquois outperforms the other protocols by more than an order of magnitude in many scenarios. For example, in executions with 16 processes and no process failures, Turquois took on average 88 ms to reach consensus, while ABBA took 1914 ms and Bracha's took 7321 ms.

1.3 Contributions

The core contributions of this thesis are summarized into the following points, along with the related publications:

1. **A detailed experimental evaluation of existing intrusion-tolerant agreement protocols.** Our first contribution involved the implementation of several agreement protocols in two different platforms (i.e., Linux and Windows Mobile), and their evaluation under different environmental settings including wired and wireless networks. It was the first time that randomized intrusion-tolerant protocols were analyzed with respect to their practical performance, providing revealing

results. Notably, the results show that there is a significant gap between theory and practice, in the sense that the theoretical analysis of these protocols (in particular, those of the local coin class) is highly pessimistic when compared to what is observed in practice. The results obtained in this step were also fundamental into identifying the performance bottlenecks associated with intrusion-tolerant protocols in wireless environments, which lead to the development of a more appropriate system model, based on the communication failure model of Santoro and Widmayer.

- **Experimental Comparison of Local and Shared Coin Randomized Consensus Protocols**
  Henrique Moniz, Nuno Neves, Miguel Correia, and Paulo Veríssimo *25th IEEE Symposium on Reliable Distributed Systems, October 2006*
- **Intrusion Tolerance in Wireless Environments: An Experimental Evaluation**
  Henrique Moniz, Nuno Neves, Miguel Correia, António Casimiro, and Paulo Veríssimo
  13th IEEE Pacific Rim Dependable Computing Conference, December 2007
- **RITAS: Services for Randomized Intrusion Tolerance**
  Henrique Moniz, Nuno F. Neves, Miguel Correia, and Paulo Veríssimo
  IEEE Transactions on Dependable and Secure Computing, accepted for publication, 2010

2. **Circumvention of the Santoro-Widmayer impossibility result.** The obtained system model for wireless ad hoc networks is bound by a 20-year old impossibility result. This result states that there is no deterministic solution to any non-trivial agreement problem if more than $n - 2$ messages can be lost at any communication step in a system with n processes. This important theoretical lower bound was circumvent for the first time by resorting to randomization. A binary consensus algorithm is presented that preserves safety despite an unbounded number of omission failures, and guarantees progress in communication steps where the number of omission failures is $\sigma \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$.

Additionally, these characteristics are very useful for wireless environments because they allow the algorithm to cope with periods of arbitrary message loss, while ensuring progress in periods with better reliability (every time the upper bound σ holds).

- **A Distributed Systems Approach to Airborne Self-Separation**
  Henrique Moniz, Alessandra Tedeschi, Nuno F. Neves, and Miguel Correia
  Computational Models, Software Engineering and Advanced Technologies in Air Transportation, L. Weigang, A. Barros, and I. Oliveira (eds.), IGI Global, 2009
- **Randomization can be a Healer: Consensus with Dynamic Omission Failures**
  Henrique Moniz, Nuno F. Neves, Miguel Correia, and Paulo Veríssimo
  23rd International Symposium on Distributed Computing, September 2009
- **Randomization can be a Healer: Consensus with Dynamic Omission Failures**
  Henrique Moniz, Nuno F. Neves, Miguel Correia, and Paulo Veríssimo
  Distributed Computing, accepted for publication, 2010

3. **Turquois, a Byzantine binary consensus protocol for wireless ad hoc networks.** This protocol was designed over an asynchronous model that combines dynamic transmission omission failures with Byzantine process failures. It is bound by the impossibility results of FLP (Fischer *et al.*, 1985) and Santoro-Widmayer (Santoro & Widmeyer, 1989). Turquois employs a novel message authentication mechanism that avoids the use of computationally expensive public-key cryptography, this way sparing computational resources and keeping the authentication payload within a fixed length. The protocol is extremely efficient in wireless ad hoc networks and, in some experiments, outperformed existing protocols by more than an order of magnitude.

   - **Turquois: Byzantine Consensus in Wireless Ad hoc Networks**
     Henrique Moniz, Nuno F. Neves, and Miguel Correia
     40th IEEE/IFIP International Conference on Dependable Systems and Networks, July 2010

- **Efficient Intrusion-Tolerant Consensus in Wireless Ad hoc Networks**
  Henrique Moniz, Nuno F. Neves, and Miguel Correia
  journal paper, under preparation, 2010

1.4 Thesis Overview

Chapter 2: Background. This chapter provides background on the problem at hand. In particular, it presents an historical perspective on the problem of consensus and other related agreement problems. It also provides an overview of group communication systems, which are related to the evaluated intrusion-tolerant protocols of Chapter 3, and surveys related research on wireless ad hoc networks.

Chapter 3: Assessment of Intrusion-Tolerant Protocols. This chapter carries out a detailed performance assessment of existing intrusion-tolerant protocols under several environmental settings - both wired and wireless. The knowledge obtained from this work brings insight into the various tradeoffs involved in the implementation of intrusion-tolerant protocols, which later supports the design of new protocols specifically tailored for wireless ad hoc networks.

Chapter 4: Consensus with Faulty Transmissions of a Restricted Source. From the insights obtained in the previous chapter, this chapter introduces a system model more adapted to wireless environments than the ones that are typically employed in the design of intrusion-tolerant protocols: the *communication failure model*, which considers the existence of ubiquitous message transmission failures. This model is bound by the Santoro-Widmayer impossibility result, which rules out deterministic practical solutions to any non-trivial agreement problem. This chapter also presents the first of an incremental series of steps into attaining an efficient intrusion-tolerant consensus protocol under the communication failure model. A consensus protocol is obtained by restricting the number of processes per round at the source of faulty transmissions. This makes the protocol resilient to failures of dynamic origin, but limits the pattern of transmission faults that may occur in each step.

Chapter 5: Consensus with Dynamic Omission Failures. This chapter shows how to circumvent the Santoro-Widmayer impossibility result by resorting to randomization. It presents a binary consensus protocol whose characteristics make it interesting for wireless ad hoc networks. The protocol ensures safety in the presence of an unrestricted number of transmission omission failures, and guarantees progress in communication rounds where the number of omissions is within a certain bound σ.

Chapter 6: Consensus with Byzantine Processes and Dynamic Omission Failures. This chapter presents Turquois, a binary consensus protocol for wireless ad hoc networks that tolerates Byzantine processes. The protocol circumvents two impossibility results: the FLP and the Santoro-Widmayer results. It assumes an asynchronous model and is optimal in the number of Byzantine processes that tolerates. Like the previous iteration, ensures safety despite arbitrary periods of unrestricted message omissions.

Chapter 7: Performance Evaluation of Turquois. This chapter presents a comparative performance evaluation of Turquois. The protocol was prototyped and compared with the binary consensus protocols evaluated in Chapter 3. The protocols are tested both in real-world 802.11b wireless ad hoc networks and in a wireless network simulator. Amongst other conclusions, the results show that, as the system scales, Turquois outperforms the other protocols by more than an order of magnitude.

Chapter 8: Conclusions and Future Research Directions. This chapter reviews the work described throughout this thesis and highlights its most important contributions. Additionally, it identifies future research directions to further extend this work, e.g., to multi-hop environments and/or dynamic process groups.

Chapter 2

Background

This chapter provides a background of previous work related with the thesis. It is organized in the following way. Section 2.1 discusses the role of agreement in distributed systems. In particular, it focuses on *consensus*, which is the specific agreement problem addressed in this work. It defines the problem formally, discusses the models in which it can be addressed and the associated computational limitations. Section 2.2 overviews other agreement problems related to consensus such as leader election, broadcasting problems, quorum systems, and others. Section 2.3 presents an overview of group communication protocol stacks. These are suites of protocols that provide coordination services like group membership and atomic broadcast, of which consensus is a fundamental building block. Two of the stacks described in this section are subject to a performance assessment in Chapter 3. Finally, Section 2.4 discusses consensus and related agreement problems in the context of wireless ad hoc networks.

2.1 Consensus in Distributed Systems

Agreement has a prominent role in distributed computing. It is essential that a set of processes, belonging to a distributed system, is able to agree on one or more values. In particular, it is important that this task is attained in the presence of failures. So even if some processes behave in arbitrary ways, either due to accidental of malicious causes, those processes that operate correctly must be able to reach an agreement.

The most common way of encapsulating the problem of reaching agreement is through the *consensus* abstraction (Turek & Shasha, 1992), which is the main focus of this work. The problem of consensus is usually defined along the following lines: each process from the group proposes a value, and then they have to decide on a common result, obtained from the original proposals. Basically, any kind of coordinated activity can be carried out by resorting to consensus as a primitive. For instance, this problem has been shown to be related to state-machine replication (Schneider, 1990) and atomic broadcast (Correia *et al.*, 2006; Hadzilacos & Toueg, 1993).

This section provides an overview of the consensus problem in systems where processes communicate by exchanging messages with each other. It starts by defining the correctness properties of consensus, and then it presents the system models in which consensus is often approached. Specifically, it explains how time and faults are modeled. It then discusses the conditions in which consensus is unsolvable and explains two important results in this area: the FLP and the Santoro-Widmayer impossibility results.

2.1.1 Problem Statement

The consensus problem usually assumes a distributed system composed by a known set of n processes. A process is deemed *correct* if it does not fail, i.e., it correctly follows the protocol until conclusion. Otherwise, the process is considered to be *faulty*. A consensus execution is initiated when every correct process p_i proposes an input value v_i, and terminates after every correct process decides on a common value v.

More formally, consensus is defined by three properties: *validity*, *agreement*, and *termination*. The definition of these properties may vary slightly, depending on the underlying system model, but the differences are usually only a matter of adjusting the definitions to the particular semantics of the model. Below is the typical formulation of these properties:

Validity. If every correct process proposes the same value v, then any process that decides, decides v.

Agreement. No two correct processes decide differently.

Termination. Every correct process decides.

The *agreement* and *termination* properties are self-explanatory. *Agreement* ensures consistency, i.e., if some process decides a value v, then no other process can decide a different value. *Termination* ensures that processes *must* decide. The purpose of *validity* is not so obvious. Essentially, it exists to rule out trivial solutions to consensus. Without this property, processes could simply decide on some predetermined value v', even without exchanging any message, because this does not violate the other two properties.

These properties can be broken down in two different categories: *safety* and *liveness*. Safety restricts the *bad* things that can happen in the system. Liveness ensures that *good* things eventually happen. Both *validity* and *agreement* are safety properties, while *termination* is a liveness property.

2.1.2 System Models

The consensus problem can be formulated under several different system models. The two main dimensions of a distributed system model are the timing model and the fault model. These are of particular importance in order to understand the thesis and they are discussed in the following two subsections.

2.1.2.1 Timing Model

The timing model defines how the delays required to perform specific actions are bounded. Examples of such delays are the time required for a message to be sent from one process to another and the time required for a process to compute something. The timing model can vary between two extremes: synchronous and asynchronous.

Synchronous Systems

It is said that a system is *synchronous* if there is a known bound on the message transmission delays and relative speeds of processes. Hadzilacos and Toueg define a synchronous system as one where the following properties are true (Hadzilacos & Toueg, 1993):

- there is a known upper bound on the time required by any process to execute a computational step;
- every process has a local clock with a known bounded rate of drift with respect of real time;
- there is a known upper bound on message delay; this consists of the time it takes to send, transport, and receive a message over any link.

These properties are essential for processes to benefit from timing information in their execution, e.g., they allow the detection of process crashes or message omissions by the use of timeouts. A common consequence of these properties is that in many synchronous systems it is assumed that processes execute in lockstep. The execution consists of synchronous rounds. In each round every process can send a message to other processes, receive messages transmitted by other processes, and perform a computation based on its state and the set of messages just received.

Asynchronous Systems

A system is *asynchronous* if there are no timing assumptions whatsoever. Process execution speeds, message transmission delays, and clock drift rates are completely arbitrary. For example, the time required for a message to be sent from one process to another is unknown. As a result, it becomes impossible to use timeouts to detect a message loss or a process failure. The asynchronous model is, in principle, preferable to model systems deployed in hostile environments. This is because the independence from timing assumptions makes a system more resilient to attacks in the domain of time (e.g., denial of service attacks). For example, a consensus protocol designed for an asynchronous system will be able to preserve its safety properties regardless of how long messages take to travel the network, and, as long as messages eventually arrive, it will terminate correctly. On the other hand, since algorithms are unable to incorporate any timing information in their execution, the asynchronous model is much more restrictive in the problems it allows to solve.

2.1.2.2 Fault Model

The fault model defines the types of faults that are assumed to occur in the system. A fault occurs when a system component (i.e., a process or a link) deviates from its correct behavior. Within the context of this work, faults will be classified in two main classes: omissive faults and arbitrary (or Byzantine) faults. A fault is of the omissive class when the affected component does not perform some action it was expected to. Relevant examples of this class of faults are the crash of a process and a message loss in a communication link. A fault is of the arbitrary class when the affected component performs an action in a manner that is arbitrarily different from its specification. An example is a compromised process that acts with malicious intend, and purposely sends incorrect values to force the other processes to deviate from the properties of the protocol, e.g., by forcing correct processes in a consensus protocol to decide different values.

2.1.3 Impossibility of Consensus

Despite its simple definition, the solution of consensus in fault-tolerant distributed systems is far from trivial. These systems are supposed to operate correctly even if some of their individual components are subject to failures. It is, then, when one starts to consider the possibility of failures that the problem of consensus becomes much more complex. In fact, depending on the assumptions that are made about the system, it may happen that there is simply no deterministic solution to consensus.

This section discusses two important impossibility results of consensus: the FLP result (Fischer *et al.*, 1985), which states that consensus cannot be deterministically solved in asynchronous systems if just one process can crash; and the Santoro-Widmayer result (Santoro & Widmeyer, 1989), which states that consensus is impossible in systems with ubiquitous communication failures. As we will see from reading the following two subsections, these two results occur in diametrically opposed models, which, nevertheless, capture the same kind of uncertainty: the inability to detect permanent failures.

2.1.3.1 FLP Impossibility

The FLP impossibility is a classic result in distributed computing (Fischer *et al.*, 1985). It applies to asynchronous systems where communication is assumed to be reliable and processes can fail. Essentially, it states that there is no deterministic algorithm that can solve consensus in a system with n processes if only one process can potentially crash. This precludes solutions to harder problems like Byzantine consensus, where processes are also allowed to fail arbitrarily. Intuitively, the result comes from the fact that in an asynchronous setting, it is impossible for a process that has not received a message to know if the sender is faulty or just too slow. Since it is impossible to identify faulty processes, the correct processes can find themselves, for example, in a state in which they are waiting indefinitely for messages from a crashed process but have no way of knowing that a failure has occurred. By formalizing this intuition, it was proved that there is no deterministic consensus protocol that can guarantee the termination property. This is a very well studied problem and through the years several solutions have been proposed to circumvent it (see the following sections).

Partial Synchrony

Partially synchronous models were introduced as a way to circumvent the FLP result (Dolev *et al.*, 1987; Dwork *et al.*, 1988). Although consensus was shown to have no deterministic solution in asynchronous systems, it does not necessarily mean that complete synchrony is needed to reach consensus. The goal is to identify the set of synchrony assumptions that are both necessary and sufficient to solve consensus.

Dolev *et al.* (1987) break down the system into five parameters: processes, communication, message order, transmission mechanism, and receive/send. Each parameter can then be either synchronous or asynchronous. For processes, it means that a computational step can take arbitrarily long (asynchronous), or be bounded by a known interval of time (synchronous). For communication, the situation is analogous. For message order, it means that messages can be either delivered out of order (asynchronous), or delivered according to their real time transmission times (synchronous). For the transmission mechanism, it means that a process can send a message to at most one process in an atomic step (asynchronous), or it can send a message to all processes

in an atomic step (synchronous). For receive/send, it means that in an atomic step a process cannot both send and receive (asynchronous), or it can it can receive and send as part of the same atomic step (synchronous). The contribution of this work lies on determining exactly on which combination of parameters consensus can and cannot be deterministically solved. Their results are summarized in Table 2.1.

		mt							
		00	**01**	**11**	**10**	**00**	**01**	**11**	**10**
pc	**00**	0	0	0	n	0	0	n	0
	01	0	0	n	0	1	n	n	1
	11	n	n	n	n	n	n	n	n
	10	0	0	n	n	0	0	n	n
		s = 0				**s = 1**			

Table 2.1: The number crashed processes tolerated for each combination of system parameters: (p) processes, (c) communication, (m) message order, (t) transmission mechanism, and (s) receive/send. Synchrony is indicated by 1, and asynchrony by 0. The corresponding table entries can be 0 (no process crash tolerated), 1 (only 1 process crash tolerated), or n (every process in the system can crash).

Dwork *et al.* (1988) present a concept of partial synchrony of particular practical utility. The intuition of this work is that, while the existence of bounds on communication and processing delays is necessary to solve consensus, the knowledge of their exact values is not. They used the concept of partial synchrony in a distributed system, which lies between the completely synchronous and completely asynchronous models. There are two cases where the communication (or the computation) can be partially synchronous. One is to assume that there exists an upper bound on communication (or computation) time, but its value is unknown. The other is to consider that exists an upper bound which is known but it only holds after some unknown time.

From these scenarios, they devise three models of partial synchrony: partially synchronous communication and synchronous processors, partially synchronous communication and processors, and partially synchronous processors and synchronous communication. For each of these three timing models, they prove that consensus can be solved with $n \geq 2f + 1$ processes with f processes being allowed to fail by crashing, and with $n \geq 3f + 1$ with f processes allowed to fail arbitrarily.

Failure Detectors

The failure detector approach was introduced by Chandra & Toueg (1996). Since the impossibility to solve consensus in asynchronous systems arises from being unable to distinguish a crashed process from a very slow one, Chandra and Toueg augment the asynchronous model with a failure detection mechanism that can provide this information to processes. It is assumed that these failure detectors can make mistakes (i.e., they are not necessarily perfect), and that processes only fail by crashing.

In the failure detector model, each process has access to a local *unreliable failure detector* module that is responsible for detecting and maintaining a list of processes suspected to have crashed. There are several classes of failure detectors. These are specified according to two properties: *completeness* and *accuracy*. Completeness requires the failure detector to eventually suspect every process that crashes, and accuracy restricts the mistakes a failure detector can make when suspecting processes. More formally, two types of completeness properties and four types of accuracy properties were defined:

Strong completeness. Eventually every process that crashes is permanently suspected by every correct process.

Weak completeness. Eventually every process that crashes is permanently suspected by some correct process.

Strong accuracy. No process is suspected before it crashes.

Weak accuracy. Some correct process is never suspected.

Eventual strong accuracy. There is a time after which correct processes are not suspected by any correct process.

Eventual weak accuracy. There is a time after which some correct process is never suspected by any correct process.

A class of failure detectors is obtained by combining one of the two completeness properties with one of the four accuracy properties. This gives a total of eight classes

summarized in Table 2.2. Basically, while this model makes possible the use of asynchronous protocols, timing assumptions must still be made to implement the failure detector module.

Completeness	**Accuracy**			
	Strong	Weak	Eventually Strong	Eventually Weak
Strong	*Perfect* $\mathcal{P}$	*Strong* $\mathcal{S}$	*Eventually Perfect* $\diamond\mathcal{P}$	*Eventually Strong* $\diamond\mathcal{S}$
Weak	$\mathcal{Q}$	*Weak* $\mathcal{W}$	$\diamond\mathcal{Q}$	*Eventually Weak* $\diamond\mathcal{W}$

Table 2.2: The eight classes of failure detectors.

An example implementation of a failure detector is one where every process q sends a periodic *q-is-alive* message to all other processes. When a process p does not receive this message after a given time, it adds q to a list of suspected processes. If later p receives a *q-is-alive* message, it is because it has erroneously suspected q. Process p can then remove q from the list of suspects and, by increasing the timeout for the *q-is-alive* message, it may avoid making the same mistake in the future. Although this example fails to implement a failure detector with even the eventual weak accuracy property, in practice it can hold this property for a 'long enough' period of time. In this context, long enough means the necessary time for the consensus protocol to reach a decision.

While it is possible to extend the failure detector approach to systems subject to Byzantine faults (Doudou *et al.*, 2002; Kihlstrom *et al.*, 1997; Malkhi & Reiter, 1997), there is a downside in doing so. Unlike crash failure detectors, arbitrary failure detectors cannot be designed independently of the protocol that will use them. A strong association is required between arbitrary failure detectors and the specific protocols that use them. As such, the design of such detectors has to be made on a case-by-case basis depending on the supported protocol (Baldoni *et al.*, 2008; Doudou *et al.*, 2002).

Wormholes

Despite their usefulness, failure detectors are not particularly adequate for systems subject to Byzantine faults. For such a fault to be detected it would be necessary for

the failure detector to perfectly understand the semantics of whatever protocols are using its service. It would have to closely monitor every message transmission that occurs in the system and be aware of all the badness that could occur in these message exchanges. Additionally, there can be protocols were an invalid message does not necessarily translate into a malicious action. In such cases, it would be extremely difficult to distinguish a corrupt process from a correct one that made a honest mistake.

Wormholes are another way of augmenting the basic system model in order to solve consensus (Correia *et al.*, 2005; Neves *et al.*, 2005; Veríssimo, 2002). They are materialized through architectural hybridization (Veríssimo *et al.*, 2003), which captures the notion that certain parts of the system can be enhanced to offer stronger properties otherwise not guaranteed by the "normal" environment. In such a setting, the problem of solving consensus can be tackled by having a few of its critical steps executed inside the wormhole which, by design, can be immune to Byzantine faults and/or offer timely behavior.

Neves et al. solved the consensus problem with wormholes (Neves *et al.*, 2005). In their model the system is divided in two parts: a payload system which is the normal environment where processes carry out their execution, and a wormhole which is a distributed component with local parts on each node and a private network with enough synchrony guarantees to ensure that its services eventually terminate. While the payload system is subject to Byzantine failures, it is assumed that the wormhole is a secure component only subject to crash failures. This approach, however, can be difficult to implement in certain environments. For instance, in a wireless or wide-area network it would be difficult to obtain a private network with the necessary synchrony guarantees. Even if only a local wormhole is assumed, it would be hard to implement in nodes with limited computational power, such as a wireless sensor device.

Randomization

Randomization is fundamentally different from other solutions to circumvent the FLP result. The approaches described so far rely, either implicitly or explicitly, on incorporating timing assumptions into the system model in order to guarantee termination. Those protocols are deterministic: given a certain input and message scheduling, the same output will always be produced at a certain step of the protocol. Rather than

deterministic, randomized protocols are probabilistic. There are certain steps of the protocol that may use a random value, chosen according to a probability distribution. This means that even a computationally unbounded adversary cannot completely determine the result of a disruptive strategy because the outcome is affected by a random element outside of its control. Consequently, any strategy employed by the adversary is dependent on 'luck' in order to prevent correct processes from reaching agreement. The same reasoning applies for correct processes. They also rely on a 'lucky' event to reach a decision. The difference is that in a multi-round protocol, the correct processes only need that lucky event to happen once, while the adversary needs its lucky event to happen an infinite number of times to prevent correct processes from reaching a decision. If the correct processes are unable to reach agreement in a given round, they just need to carry out the execution for another round. No matter how small the probability for correct processes to reach agreement on any individual round, given an infinite number of rounds, the probability that correct processes decide has asymptotic value 1.

The biggest advantage of randomization is that the system model requires no additional timing assumptions. The only change is a slight modification to the termination property of consensus. Instead of stating that the correct processes *must* decide, it is stated that the correct processes eventually decide with probability 1. This allows the system to remain completely asynchronous and safeguards the protocols from attacks in the domain of time.

Randomized consensus protocols are based on a virtual *coin toss* operation that returns a random value to the processes that execute it. Depending on the implementation, this value might return 0 or 1 with the same probability (i.e., a fair coin flip), or it might return a value chosen from a larger domain. The protocols can be categorized in one of two classes, depending on how they implement the coin toss operation. There are those based on a *local coin* operation that is performed independently by each process, and there are those based on a *shared coin* operation that returns the same value to all processes, which is typically implemented by resorting to cryptographic techniques. Both classes of protocols were introduced independently in 1983 by Ben-Or (1983) (local coin) and Rabin (1983) (shared coin). Ben-Or presented two algorithms, tolerating respectively crash and Byzantine process failures. The crash protocol tolerated f faulty processes out of $n \geq 2f + 1$, and the Byzantine protocol tolerated f out

of $n \geq 5f + 1$. Both protocols terminate in an expected exponential number of rounds. Rabin's protocol tolerates Byzantine faults and the resilience was f faulty processes out of $n \geq 10f + 1$. The protocol achieves termination in a constant number of rounds.

In the years that followed, several randomized binary consensus protocols were presented, both following Ben-Or's local coin-style protocol (Bracha, 1984) and Rabin's shared coin approach (Canetti & Rabin, 1993; Toueg, 1984). A detailed survey on early work can be found in Chor & Dwork (1989). Among the several goals pursued, we emphasize the quests for optimal resilience in the number of Byzantine processes, constant expected round complexity, and practical coin sharing. Protocols with optimal resilience, i.e., that tolerate f out of $3f + 1$ faulty processes, were presented quite early, both for local coins (Bracha, 1984) and for shared coins (Toueg, 1984). Constant expected round complexity with optimal resilience was for long an objective of this line of research. The first protocol to attain this goal was presented by Canetti & Rabin (1993). This protocol is based on shared coins, but has constant expected round complexity at cost of an enormous number of transmitted messages (many thousands even for low numbers of processes). The ABBA protocol also terminates in a small constant expected number of rounds and sends a much lower number of messages than the protocol of Canetti and Rabin (Cachin *et al.*, 2000). It does so, however, at the cost of incurring in computationally expensive cryptographic operations. In relation to coin sharing, the original protocol of Rabin requires previous distribution of data among the processes for each coin toss operation (Rabin, 1983), something that can be quite impractical for real applications. This requirement was later removed, and more recent protocols do not need this previous distribution of data (Cachin *et al.*, 2000; Canetti & Rabin, 1993).

2.1.3.2 Santoro-Widmayer Impossibility

The Santoro-Widmayer impossibility result affects distributed systems subject to failures on the communication links (Santoro & Widmayer, 2007; Santoro & Widmeyer, 1989). It was formalized for a variant of consensus called k-*agreement*, in which $k > \lceil n/2 \rceil$ out of n processes must agree on a binary value $v \in \{0, 1\}$. It states that there is no finite time deterministic algorithm that allows a system with n processes

to reach k-*agreement* if more than $n - 2$ messages can be lost during a communication step. This is a very discouraging result because the crashing of a single process necessarily results in $n - 1$ transmission failures, rendering this form of agreement impossible. Moreover, this result is produced under strong timing assumptions where both the processes' relative processing times and communication delays are bound by known constants (i.e., a synchronous system) and holds regardless of processes failures.

Therefore, on one hand we have asynchronous systems bound by the FLP impossibility result (Fischer *et al.*, 1985), where agreement is impossible even if communication is perfectly reliable. On the other hand, due to the Santoro-Widmayer result, we have systems that are completely synchronous, but where agreement is also impossible because communication is unreliable. While several solutions have been proposed over the years to circumvent the FLP impossibility, the result of Santoro and Widmayer has not received comparable attention. The reason for this is likely to be related with some lack of practical interest in this model prior to the emergence of wireless ad hoc communication. For distributed systems based on wired networks, it was safe and convenient to assume end-to-end reliable delivery mechanisms, since the implementation of such mechanisms does not represent a significant performance overhead. Furthermore, except for randomization, the techniques that circumvent the FLP result are based, implicitly or explicitly, on strengthening the timing assumptions. These do not transpose well to the Santoro-Widmayer impossibility result because the system model is already completely synchronous.

The problem of reaching agreement with unreliable communication links goes back to 1975, when Akkoyunlu *et al.* (1975) pointed out that an agreement between two processes connected by unreliable communication paths leads to an infinite exchange of messages. In 1978, Gray identified essentially the same problem by formulating the *generals paradox* (Gray, 1978). He showed that there is no deterministic protocol that allows agreement between two processes connected by an unreliable communication link. This problem is often referred to as the *coordinated attack problem* from the formalization of Lynch (1997). Varghese and Lynch proposed a randomized solution to the coordinated attack problem where the protocol runs for a fixed number of rounds and agreement is reached with a probability proportional to the number of rounds (Varghese & Lynch, 1996).

The work of Chockler *et al.* (2008) presents algorithms that solve consensus in systems where nodes fail only by crashing and messages can be lost due to collisions. Their solution assumes that processes have access to a collision detector that determines when message collisions occur, which allows nodes to take recovery measures. Message omissions other than those due to collisions, however, are not covered by their model.

Two other works also study consensus with dynamic communication failures. The work of Biely *et al.* (2007) does so by addressing the problem in the context of the *heard-of model* of Charron-Bost & Schiper (2007). This model permits a fine-grained specification of the fault patterns allowed in the system, thus being able to distinguish the cases where the fault pattern exceeds the lower bound of Santoro and Widmayer but in a way that is not harmful to the system (e.g., $n - 1$ faults are harmful to the system if they originate at the same process, but may be acceptable if they occur at different process).

The work of Schmid *et al.* (2009) presents an analogous contribution in the sense that it restricts the number of faults that each process may experience, such that the harmful fault patterns are avoided. None of these two contributions, however, deal with the essence of the Santoro-Widmayer impossibility result, where for instance the failure of every transmission from a single process renders consensus impossible. This implies that consensus remains unsolvable if, in a wireless ad-hoc network, a single node falls out of range of every other node for an unknown period of time.

2.2 Related Problems of Consensus

While consensus is a central problem in distributed computing, there are several other problems that are related and that have been extensively studied. This section provides an overview of some of the most representative of these problems, and it is organized in three categories: mutual exclusion and leader election, broadcasting, and replication.

2.2.1 Mutual Exclusion and Leader Election

Mutual Exclusion. Mutual exclusion ensures that concurrent processes access shared resources in a serialized manner, i.e., one process at a time. This is often referred as the *critical section* (CS) problem in the context of operating systems. In a distributed system, however, processes have to coordinate their access to the CS solely through message passing. The problem is formalized through two properties. The *safety* property ensures that at most one process enters CS at a time. The *liveness* property enforces fairness by ensuring that any request to enter the CS is eventually granted. This prevents both deadlock and starvation.

Algorithms for distributed mutual exclusion can be *permission-based* or *token-based*. In permission-based algorithms, introduced by Ricart & Agrawala (1981), a process that wishes to enter the CS asks the other processes for permission, and enters the CS when a certain quorum of permissions is granted. If two or more processes are simultaneously interested in accessing the CS, then the conflict is resolved through some priority-based mechanism.

In token-based algorithms, exemplified by token ring networks (Bux *et al.*, 1983), the right to enter the CS is materialized by a *token*, a special object unique in the system. The token is then passed around in a manner that each process is granted the token infinitely often and there is only one token in the system at any given time, thus ensuring both safety and liveness.

Distributed Mutual Exclusion has been shown to be strictly harder than consensus in the sense that the weakest failure detector that can solve mutual exclusion is stronger than the weakest failure detector that can solve consensus (Delporte-Gallet *et al.*, 2005). The weakest failure detector T for mutual exclusion lies between the eventually perfect failure detector $\diamond P$ and the perfect failure detector P. For consensus, a failure detector of the $\diamond W$ class is sufficient.

Leader Election. This problem considers the issue of electing one process within a system to perform a special role. This is considered a more general problem than mutual exclusion because, for example, a leader election algorithm can be used to determine the process that enters the critical section. In the leader election problem, at any given moment, at most one process is considered to be the leader by every

correct process in the system (including the leader itself). The algorithm of Chang & Roberts (1980) for ring networks and the bully algorithm of Garcia-Molina (1982), which considers process crashes in a synchronous system, represent initial seminal work in this problem.

Like mutual exclusion, the leader election problem has been proved to be strictly harder than consensus. The weakest failure detector with which consensus can be solved is not sufficient to solve leader election. For example, although a strong failure detector $\mathcal{S}$ is sufficient to solve consensus, it is not sufficient to solve leader election (Sabel & Marzullo, 1995).

A useful relaxation of leader election is *eventual leader election*, which guarantees that *eventually* at most one process is recognized as the leader. This is abstracted through the Ω oracle (Chandra *et al.*, 1996b). Queries to the Ω oracle return a process ID and satisfy the eventual leadership property: there is a point in time after which all queries return the same process ID. The usefulness of this abstraction is that it has been shown to be sufficient to solve consensus. For example, the Paxos protocol is based on such an idea of eventual leadership (Lamport, 1998).

2.2.2 Broadcasting

Broadcasting in distributed systems requires coordination amongst processes to ensure they receive consistent information. For example, if a sender process crashes in the middle of a broadcast operation or if the network is unreliable, then it might be possible that some processes receive the message while others do not. An in-depth overview of broadcasting and related related problems in distributed systems can be found in (Hadzilacos & Toueg, 1993).

Reliable Broadcast. This problem is the weakest amongst broadcasting primitives discussed in this section. In essence, reliable broadcast requires that (1) all correct processes deliver the same set of messages, and (2) if the sender is correct then its message is delivered.

Reliable Broadcast in strictly weaker than consensus. The fact the reliable broadcast does not require a strong termination property like consensus (i.e., processes might

never deliver a message if the sender is not correct), makes this problem solvable in asynchronous systems.

Atomic Broadcast. This form of broadcasting imposes a total ordering on the delivery of messages. It can be seen as a reliable broadcast with the additional property that all correct processes deliver the messages by the same order. More formally, if two correct processes p and q deliver messages m and m', then p delivers m before m' if and only if q delivers m before m'. Atomic broadcast has been shown to equivalent to consensus in asynchronous systems in the sense that one problem can be reduced to another (Chandra & Toueg, 1996; Correia *et al.*, 2006).

Terminating Reliable Broadcast. With respect to reliable broadcast, this problem imposes the additional property that some message must be delivered even if the sender is faulty. This assumes that processes know when a broadcasting operation is impending. So, in the case that the sender is faulty, the problem specification allows the delivery of a special message F_s stating that the sender s is faulty. Otherwise, if the sender s is correct, then correct processes have to deliver the message m broadcast by s. Terminating reliable broadcast is strictly harder than consensus. It requires a perfect failure detector $\mathcal{P}$, according to Chandra & Toueg (1996).

2.2.3 Replication

State Machine Replication. If two nodes, starting with the same state, execute exactly the same deterministic sequence of commands, then they will evolve to an equal state. State machine replication exploits this principle in order to implement fault-tolerant services in distributed systems (Schneider, 1990). It is a well-known paradigm in distributed computing. At the heart of state-machine replication is a protocol that totally orders the messages carrying the commands to be executed by the replicas. This can be achieved, for instance, through an atomic broadcast protocol, or through a consensus protocol that allows processes agree on the order of message processing.

State machine replication was first proposed by Lamport (1978, 1984). The approach was later elaborated by Schneider (1990). Oki & Liskov (1988) were the first to create a practical implementation of state machine replication that tolerated process

crashes, while one of the early Byzantine fault-tolerant implementations, coined BFT, was by Castro & Liskov (1999). In BFT, there are clients and servers. The clients issue requests to the servers, then requests are processed by the servers in total order, and a reply is returned to the clients. The servers are either primary or backup. There is only one primary at any given moment in the system. The client requests are issued directly to the primary, which in turn multicasts the request to the backups. The replies are transmitted to the client by all servers. The client waits for a certain number of replies with the same result in order to obtain the response. This comprises the normal operation of the algorithm. In case a primary fails, a view change must occur and the servers must agree on a new primary. View changes are triggered by timeouts. After a view change the service resumes to its normal operation. Several other systems have been proposed over the years that improve in one way or another the original BFT protocol, these include the BASE (Castro *et al.*, 2003), Q/U protocol (Abd-El-Malek *et al.*, 2005), HQ (Cowling *et al.*, 2006), Steward (Amir *et al.*, 2010), Prime (Amir *et al.*, 2008), Zyzzyva (Kotla *et al.*, 2008), Aardvark (Clement *et al.*, 2009), and Spin (Veronese *et al.*, 2009).

Quorums. A quorum system is a collection of subsets of processes such that any two subsets intersect in one or more elements. It is this intersection property that makes quorum systems interesting for coordinating actions in a distributed system and led to their application to a multitude of scenarios, including mutual exclusion (Barbara & Garcia-Molina, 1986; Cohen, 1993; Garcia-Molina & Barbara, 1985) and replication (Fu, 1990; Herlihy, 1986). For example, in quorum-based shared memory, a data item is timestamped and written to some quorum of servers. A client that wishes to access this data item invokes a read operation on some (possibly different) quorum. The intersection property of quorums ensures that the client has access to the most recent write, which is identified by comparing the timestamps from each server.

Malkhi & Reiter (1998) were the first to study these quorum systems in the context of Byzantine faults. They introduce the concept of *masking* quorums along with two additional variations: *dissemination* quorums and *opaque* masking quorums. Dissemination quorums are a specialization tailored for the storage of self-verifying information, i.e., information to which clients can independently verify its integrity, for example, through the use of digital signatures. Their intersection property requires that

the intersection of any two quorums contains at least one non-faulty server. The (normal) masking quorums require that the intersection of any two quorums contains more non-faulty servers than the faulty ones in either quorum. This way, the responses from non-faulty servers outnumber those from faulty ones. Opaque masking quorums have a stronger intersection property. In the context of a write operation in some quorum Q_1 and a subsequent read from quorum Q_2, they require that the number of servers in the intersection of the two quorums is greater than the number of faulty servers in Q_2 plus the number of servers in Q_2 not in Q_1. This is to avoid a scenario were the faulty servers behave as the (outdated) servers in $Q_2 \setminus Q_1$, and thus could outnumber the correct up-to-date servers and return an obsolete value to the client.

The quorum systems introduced by Malkhi & Reiter (1998) are *confirmable* in the sense that the completion of a write operation can be locally determined by a server. Martin *et al.* (2002b) later introduced the concept of *non-confirmable* quorums were the completion of a write cannot be locally determined by a server, but the operation is still guaranteed to eventually complete. For both types of quorums they proved tight lower bounds on the number of faulty servers (Martin *et al.*, 2002a).

This body of work on Byzantine quorum systems had a significant impact on subsequent work in Byzantine fault-tolerant state machine replication. In particular, the application of quorum-based techniques led to remarkable improvements in the performance of protocols for consensus and state machine replication by reducing the number of required steps, at least for non-faulty or optimistic executions (Abd-El-Malek *et al.*, 2005; Cowling *et al.*, 2006; Kotla *et al.*, 2008; Martin & Alvisi, 2006).

View Synchrony. View or virtual synchrony is a model that supports dynamic process groups (Birman, 1993; Birman & Joseph, 1987a). It is an essential component in many group communication systems (reviewed in Section 2.3). Basically, it is a way to serialize the events in a distributed system such that membership changes are atomic with respect to message flow. In particular, it is said that a message m is delivered to a process p in some system view [1] V^i if it is delivered *after* view V^i and *before* view V^{i+1}. View synchrony thus requires that if a message m is delivered to a process p in view V^i, then it is also delivered in view V^i to every other process $q \in V^i$.

[1] A system view contains essentially information about the group membership.

View synchrony is essentially an agreement problem and can be solved using consensus as a building block (Guerraoui & Schiper, 2001). As processes join or leave the group, the processes in the current view V^i execute a consensus algorithm that (1) agrees on a new view V^{i+1} that reflects the new membership, and (2) also agrees on the messages to be delivered on view V^i. A process p only changes to view V^{i+1} after delivering all messages that were agreed to belong to view V^i. View synchrony is subject to essentially the same computational limitations of consensus (Chandra *et al.*, 1996a).

2.3 Group Communication Systems

This section discusses Group Communication Systems (GCSs). These systems typically provide a set of fault-tolerant communication and agreement primitives to a group of processes. Examples of these are a membership service, which maintains a consistent view of the group amongst all members (e.g., view synchrony described in Section 2.2.3), or a reliable broadcast service, which guarantees that all members of a group receive the same messages. The communication primitives provided by GCSs are used as building blocks to aid in the development of fault-tolerant distributed applications. For example, database replication (Amir *et al.*, 1994) or large-scale distributed storage (Kubiatowicz *et al.*, 2000).

The communication primitives are based on implementations of agreement protocols (e.g., atomic broadcast) or are implemented on top of them (e.g., group membership on top of consensus). In fact, some GCSs present themselves as stacks of several agreement protocols. GCSs usually represent the most mature implementations of agreement protocols, and their study can provide interesting insights into the behavior of these protocols in practical settings. This is why our starting point in designing new protocols for wireless ad hoc networks is the performance assessment of two existing GCSs in wired and wireless environments (see Chapter 3). This section focuses on intrusion-tolerant GCSs, but it also covers some fault-tolerant and secure stacks (where the group is secure against external attacks, although not intrusion-tolerant in the sense that it does not tolerate malicious members).

Early work in group communication considered only crash failures. The most well-kown GCS of this line of work is the Isis toolkit (Birman & Joseph, 1987b). It was followed by several others such as xAmp (Rodrigues & Verissimo, 1992), Transis (Amir *et al.*, 1992), and Totem (Moser *et al.*, 1996).

Eventually, the work on GCSs evolved to support a stronger model, in which communication can be attacked from entities external to the group. These attacks can be passive (e.g., listening to all communication) or active (e.g., denial of service). In any case, processes within the group are still assumed to behave correctly at all times and indeed invulnerable to intrusions. The first of these systems was Horus (Reiter *et al.*, 1992, 1994; van Renesse *et al.*, 1996), which is an extension of Isis (Birman & Joseph, 1987b). The security features of Horus are essentially the ability to provide mutual authentication between processes and groups, and ensuring integrity and confidentiality of communication within the group. Horus relies on a trusted third party for initial key distribution. Ensemble is an evolution of Horus (Rodeh *et al.*, 2001a,b). It addresses two important limitations of the latter. First, it handles group partitions by allowing groups to be re-merged, even if they eventually do not share anymore the same keys. Second, it provides a rekeying protocol for group membership changes, i.e., every time a process joins or leaves the group, the communication keys are renegotiated in order to ensure confidentiality in relation to past and new members of the group. Secure Spread is another popular GCS within this body of work (Amir *et al.*, 2000, 2001). Its main difference from Horus and Ensemble is that it does not rely on a trusted third party for initial key distribution. Instead, it employs decentralized key management, where keys are generated cooperatively by group members.

More recently, the work on GCSs has evolved to support models where the individual processes may be compromised and exhibit arbitrary behavior, including of malicious nature. These GCSs implement agreement protocols more aligned with the research of this thesis (i.e., they are intrusion-tolerant). Some of these systems are classical GCSs, providing reliable communication primitives along with a group membership service (e.g., Rampart (Reiter, 1995), SecureRing (Kihlstrom *et al.*, 2001), and Worm-IT (Correia *et al.*, 2007)). Others, like SINTRA (Cachin & Poritz, 2002) and RITAS (Correia *et al.*, 2006; Moniz *et al.*, 2006b, 2010), while not providing a service for dynamic group membership, focus on implementing a broad range of agreement

and broadcast protocols, organized in a stack, which can then be used to build sophisticated applications.

The first mature implementation of a set of Byzantine agreement protocols was made for the Rampart toolkit (Reiter, 1994, 1995). Rampart implements the echo broadcast and atomic broadcast protocols. Echo broadcast ensures that, upon a broadcast, no two processes receive a message with different values. Atomic broadcast ensures that messages are delivered by all processes in the same order. The atomic broadcast problem has been proven to be equivalent to consensus, while echo broadcast is considered to be a weaker form of agreement because it does not require termination (i.e., processes may not deliver any message at all). The message order in the atomic broadcast protocol is defined by a leader process that echo-broadcasts the order information. In Rampart, however, if a process does not echo-broadcast a message to all or if a malicious leader performs some attack against the ordering of the messages, these events have to be detected and the corrupt process must be removed from the group. This implies liveness is dependent on this detection, and synchrony assumptions are required about the network delay, allowing attacks where malicious processes delay others in order to force their removal (Ramasamy *et al.*, 2002). For this reason, Rampart relies on a group membership protocol not only to handle voluntary joins and leaves from the group, but also to detect and remove corrupt processes. The group membership protocol uses a decentralized consensus primitive (i.e., one that does not resort to a leader) in order for processes to agree on the elements of the group. The consensus execution has to be decentralized to avoid malicious leaders from abusing the system.

Like Rampart, SecureRing is an intrusion-tolerant group communication system (Kihlstrom *et al.*, 2001). It relies on a token that rotates among the processes to decide the order of message deliveries. This signed token carries message digests, a solution that allows a lower number of signatures and an improvement in performance when compared to Rampart. In SecureRing, malicious behavior also has to be detected for the protocols to make progress, which means that it suffers from similar problems as Rampart.

Worm-IT uses the wormhole abstraction to provide a group membership service and a view-synchronous atomic multicast primitive (Correia *et al.*, 2007). It is designed under an hybrid system model. The system is considered to be asynchronous and

subject to Byzantine failures with the exception of a small subset, the wormhole, that is assumed to be secure (i.e., can only crash) and synchronous. Critical steps of the protocols that require stronger environmental properties (such as agreement tasks) are executed inside the wormhole. Worm-IT has the main advantage of being completely decentralized on its execution, not relying on leader processes.

SINTRA is a randomized protocol stack that provides a number of communication primitives for the construction of intrusion-tolerant distributed services (Cachin & Poritz, 2002). From bottom to top, the protocols provided by SINTRA are: reliable broadcast, echo broadcast, binary consensus (agreement on a binary value); multi-valued agreement (agreement on an arbitrary value); atomic broadcast; and secure causal atomic broadcast (an atomic broadcast that ensures privacy until the messages are received by a threshold of processes). Like Worm-IT, the SINTRA protocols are completely decentralized. Due to its randomized approach, SINTRA uses a completely asynchronous system model, not relying on any time assumptions for either liveness or safety. It uses a static group, and attains optimal resilience in the presence of arbitrary faults, tolerating $f \leq \lfloor \frac{n-1}{3} \rfloor$ Byzantine processes out of a total of n processes. The protocols in SINTRA also rely heavily in public-key cryptography, more specifically in threshold signatures and a shared coin tossing scheme based on secret sharing. These characteristics impose a significant negative impact on the performance of the protocols.

RITAS is another protocol stack that uses a randomized approach to circumvent the FLP impossibility result and retain a completely asynchronous system model (Correia *et al.*, 2006; Moniz *et al.*, 2006b, 2010) and decentralized execution. The protocols it provides are from bottom to top: reliable broadcast and echo broadcast; binary consensus; multi-valued consensus; vector consensus (agreement on a vector composed by some of the proposed values) and atomic broadcast. Like SINTRA, it assumes a static group of n processes and achieves optimal resilience, tolerating $f \leq \lfloor \frac{n-1}{3} \rfloor$ Byzantine processes. RITAS, however, differs significantly from SINTRA in its approach to randomization because it does not need any kind of public-key or otherwise expensive cryptographic operations. The downside is that it relies heavily on message exchanges, which affects performance in environments where latency and bandwidth are more scarce.

2.4 Wireless Ad Hoc Networks

This section surveys research on wireless ad hoc networks. It describes solutions to consensus and other related problems specifically designed for wireless ad hoc networks. This includes mutual exclusion and leader election, broadcasting algorithms, and group communication primitives (e.g., group membership).

2.4.1 Consensus

Over the past decade, there have been some contributions to the solution of consensus in wireless ad hoc networks. However, practically none of them considered the presence of Byzantine nodes. Research on Byzantine fault-tolerant protocols for wireless environments has been basically restricted to broadcasting problems, which are discussed in the following section.

Badache *et al.* (1999) were the first to present a consensus protocol specifically for wireless environments. Their approach, however, is not applicable to wireless ad hoc networks *per se* because it relies on a preexisting infrastructure. Their system model assumes the presence of two types of entities: mobile hosts (MHs) and fixed hosts that act as mobile support stations (MSSs). Each MH is connected to the particular MSS responsible for its geographical area, and all MSSs are fully connected by a static network. The message propagation delay is assumed to be arbitrary but finite. Naturally, this accounts to an asynchronous system with reliable links. The set of MHs involved in a consensus execution is assumed to be known *a priori*. To solve consensus, each MH communicates its initial proposal value to the respective MSS. The MSSs execute amongst themselves the Chandra-Toueg consensus protocol using a $\diamond S$ failure detector (Chandra & Toueg, 1996), and then communicate the decision value to the associated MHs. The decision value is a vector V containing the initial values of α of mobile hosts, with α being a configurable parameter such that $\alpha \geq 1$. Both the MHs and MSSs can fail by crashing. In a system composed by n MSSs and m MHs the protocol tolerates $f < \frac{n}{2}$ MSS failures and $f' = m - \alpha$ MH failures. Later on, this work was extended by Seba *et al.* (2002) to take into consideration the dynamism in the set of MSSs executing consensus due to the handover of MHs.

Wu *et al.* (2007b) describe a hierarchical consensus protocol for mobile ad hoc networks, which can be seen as a variation of the protocol of Badache *et al.* (1999). Both protocols share essentially the same system model. The main difference is that Wu et al. do not assume the presence of infrastructure support. Instead, they assume that the system is composed by n mobile hosts distributed in clusters and that there is a static subset of k predefined nodes that act as *clusterheads*, which take essentially the same role of the MSSs in the protocol of Badache *et al.* (1999). The protocol requires connectivity amongst the clusterheads, and between a clusterhead and every node on its cluster. 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. The protocol tolerates the crashing of $f < min(k, \frac{n}{2})$ nodes.

Vollset & Ezhilchelvan (2005) implement the randomized consensus protocol of Ezhilchelvan *et al.* (2001) in a wireless context. The protocol is a simple generalization of the classic local coin protocol of Ben-Or (1983) to an arbitrary domain of decision values. It is designed for an asynchronous system with a static set of n nodes and tolerates up to $f < \frac{n}{2}$ node crashes. Their adaptation to a wireless context assumes arbitrary connectivity changes in the system, but requires a fairness condition such that every pair of correct nodes is eventually directly connected to each other, ensuring that messages exchanged between them are eventually delivered. This is essentially the same as modeling the system as being asynchronous with reliable links. The protocol was subject to a simulation in which a consensus instance required on average between 50 and 100 seconds, with the number of nodes ranging from 1 to 40.

The research discussed so far assumes, in one form or another, an asynchronous system with reliable links. Already described in Section 2.1.3.2, the work of Chockler *et al.* (2005) solves consensus in collision-prone wireless networks, where communication is synchronous but unreliable. Under their system model, nodes can fail by crashing and messages can be lost, but only due to collisions. To cope with the uncertainty in this model, they introduce the collision detector abstraction. Reminiscent of the failure detector, the collision detector is a device attached to each process that in every communication round provides information about message collisions. This allows consensus to be solved with anonymous processes in single-hop networks and

to tolerate any number of process crashes. The protocol was also adapted to a multi-hop scenario, where the network is divided into a series of grid squares. Here, every process must know its position in the grid and the number of grid squares in the network. A single-hop consensus is executed individually in each grid square and the results are gossiped throughout the network. Once a node has received a value for every grid square, it decides the final value by applying some deterministic function to these values.

Borran *et al.* (2008) approach to consensus in wireless ad hoc networks assumes a static and known group of n processes. The classic Paxos algorithm is expressed under the *heard-of* (HO) model (Charron-Bost & Schiper, 2007) and then it is extended with a communication layer for wireless networks. The HO model does not explicitly state timing or fault assumptions. Instead, it defines the system through a predicate expressing the patterns of message delivery that are allowed to occur. For Paxos, the conditions for liveness are: (1) eventually a majority of processes consider the same coordinator, (2) a majority of processes receive the messages from the coordinator, and (3) the coordinator receives messages from a majority of processes. Safety is always assured regardless of the pattern of message delivery. The communication layer provides a leader election service that is used as a primitive by Paxos (Lamport, 1998) and a message propagation algorithm for multi-hop communication. The model for the communication layer assumes an asynchronous system with periods of reliable message delivery bounded by a known constant value, which, in essence, represents a partially synchronous system. The leader election algorithm tolerates up to f process crashes, with $n \leq 2f + 1$.

2.4.2 Mutual Exclusion and Leader Election

Two other related problems of consensus - leader election and mutual exclusion - have been extensively studied in the context of wireless ad hoc networks. These are relevant problems because they regulate access to shared resources, a common problem in these networks. Nevertheless, these studies have been focused on the mobility of processes, and they typically assume that communication is reliable (at least among neighbors) and that processes do not fail in a malicious way. In fact, most of the approaches to these problems consider reliable processes, i.e., processes never crash.

Mutual Exclusion. The work of Walter *et al.* (2001b) solves mutual exclusion in mobile ad hoc networks by inducing a logical directed acyclic graph (DAG) on the network, dynamically changing the logical structure to adapt to the changing physical topology. The DAG is constructed such that it is token-oriented, i.e., directed edges lead to the node holding the token. Thus, requests to the token holder are forwarded through a path along the DAG, while the token is delivered to the requesting node in reserve order of the path. For this work, the authors assume that: node failures do not occur; the network graph is connected; incipient link failures are detectable; and each node is always aware of the set of neighboring nodes. This solution was later extended to the $k-$mutual exclusion problem (Walter *et al.*, 2001a). Building and maintaining the DAG incurs an overhead on message exchanges.

The proposal of Baldoni *et al.* (2002) follows essentially the same model and attempts to eliminate the overhead of maintaining the logical structure. Instead of a DAG, they employ a logical ring with a circulating token. Each time a node receives a token from the predecessor, it decides the successor on-the-fly according to some policy (e.g., hop distance). The algorithm proceeds in rounds, which have different coordinators. In each round, the coordinator circulates the token, allowing each process to enter the CS. After the token is returned to the coordinator, it enters an idle state waiting for a request another process to enter the CS. When this happens, this other process assumes the role of coordinator and initiates another round. This way, the algorithm remains idle until some process requires to enter the CS. Chen & Welch (2002) propose a self-stabilizing algorithm also based on dynamic logical rings. The algorithm requires the topology to be static while converging. After it has converged, it can guarantee safety under arbitrary mobility, and liveness with a restricted mobility pattern.

Wu *et al.* (2005) propose a permission-based approach to mutual exclusion in ad hoc networks. The rationale is that token-based algorithms are inherently fragile in mobile environments, due to token loss caused by mobility and frequent disconnections. This approach does not need to maintain a logical structure, and avoids message exchange if no node wants to enter the CS. In order to keep the number of exchanged messages low, the protocol resorts to the 'look-ahead' technique, requiring mutual exclusion only amongst the nodes competing for the CS (Singhal & Manivannan, 1997).

Another approach to tolerate token losses is proposed by Wu *et al.* (2007a). This solution is based on a dual-token algorithm, where two tokens monitor each other to detect token losses. A fundamental limitation of this approach is related to the simultaneous loss of both tokens, in which case liveness cannot be guaranteed. The paper suggests that more tokens can be used, but this only leads to the recurrent problem of estimating the right number of needed tokens.

Attiya *et al.* (2010) focus on the problem of mutual exclusion amongst neighboring nodes, i.e., no two nodes that can communicate directly can be in their critical sections simultaneously. The paper presents two algorithms. The first assigns colors to nodes and resolves conflicts based on the colors. As nodes move, they choose new colors to reflect their repositioning in the network. This algorithm has two variations. The first has a *response time* (i.e., delay between a node requesting to enter the CS and actually entering the CS) that is polynomial in the number of neighboring nodes, and has a failure locality of n, where n is the total number of nodes in the system (e.g., a failure locality of n determines that a process makes progress if no failures occurs within n hops). The second variation has also a polynomial response time in the number of neighboring nodes, and failure locality logarithmic with the number of nodes in the system. The second algorithm uses dynamic priorities to regulate access to the CS. The basic idea is that a node that enters the CS resets its priority, thus lowering its priority relative to the neighbors. This algorithm has optimal failure locality (i.e., 2), and a response time of $O(n^2)$ (but can be $O(n)$ if the network remains static).

Leader Election. Hatzis *et al.* (1999) were the first to study the problem of leader election in mobile networks. In particular, the paper focuses on how the motion of the nodes can affect the problem of leader election. To this end, the algorithms are classified in two classes. *Non-compulsory* algorithms do not affect the motion of the nodes and instead try to use the nodes natural movements to achieve leader election. *Compulsory* algorithms determine the motion of the nodes in order to ensure the correctness of the protocol. For both classes, it is assumed that nodes move in a three-dimensional space bounded by some regular polyhedron. The nodes must know in advance the dimensions of this polyhedron and must be able to measure the distance that they cover. The non-compulsory protocols have the additional limitation of not guaranteeing that a unique leader is elected.

Malpani *et al.* (2000) propose a leader election protocol based on a directed acyclic graph (DAG). The basic idea is that each node is assigned a height, links are logically directed from higher to lower heights, and, as the network topology changes, the heights are manipulated such that the network forms a DAG with exactly one sink, which is considered the leader. The proposal is based on TORA, a routing algorithm for mobile ad hoc networks (Park & Corson, 1997). The system is assumed to be synchronous, nodes are connected by reliable channels, and only one link is created or destroyed at a time. An extension to this algorithm was proposed by (Ingram *et al.*, 2009), which can tolerate multiple topology changes.

The algorithms of Vasudevan *et al.* (2004) and Boukerche & Abrougui (2006) focus on extrema-finding solutions to leader election in ad hoc networks. These are algorithms that elect as leader the node that ranks higher or lower in some category. It is an useful abstraction in wireless ad hoc networks because one might want to elect as leader the node, for example, with the most energy or computational resources.

2.4.3 Broadcasting

Research on fault-tolerant broadcasting protocols for wireless environments have received considerable attention from the scientific community. This section focuses on contributions that tolerate Byzantine failures because research on intrusion-tolerant protocols for wireless networks has been practically restricted to broadcasting problems.

The first contribution in this line of research is from Koo (2004). His work addresses the problem of reliable broadcast (Pease *et al.*, 1980) in multi-hop networks with Byzantine nodes. The model assumes that nodes are distributed uniformly in a square grid with each integral point representing a node. It also assumes that nodes adhere to a predetermined TDMA message transmission schedule, which implies a synchronous system, and that message spoofing and collisions do not occur. The contribution lies in defining upper and lower bounds on the number of Byzantine neighbors t that a process can have within its communication radius r. Koo proves that reliable broadcast is possible for $t < \frac{1}{2}r(r + \sqrt{\frac{r}{2}} + 1)$ and impossible for $t \geq \lceil \frac{1}{2}r(2r + 1) \rceil$. A year later, Bhandari & Vaidya (2005) showed the lower bound to be tight by providing a matching algorithm that tolerates $t < \lceil \frac{1}{2}r(2r + 1) \rceil$. This model was later extended

by Koo *et al.* (2006) to allow spoofed messages and collisions. They show that if these are bounded by a known value, then the previous threshold for t still holds.

Pelc & Peleg (2005) study the problem of broadcasting under essentially the same model of Koo (2004), but considering arbitrary graphs. Their paper presents upper and lower bounds for the feasibility of reliable broadcast based on graph-theoretic parameters. These bounds, however, are not tight, indicating that different parameters may need to be used in order to close the gap. They also show that for certain graphs, only algorithms that have knowledge of the topology can solve reliable broadcast.

Drabkin *et al.* (2005) present a broadcast protocol for wireless ad-hoc networks in asynchronous systems. The protocol employs an overlay on which messages are disseminated. In parallel, signatures of these messages are being gossiped by all nodes in the system in an unstructured manner. It is assumed that each device can obtain the public key of every other device. When a node learns about a message it is missing, it requests this missing message from another node. The model is extended with three types of failure detectors: verbose, mute, and trust. These, respectively, detect messages sent too often, messages not sent, and incorrect nodes. The broadcast protocol is then built by combining together these failure detectors along with the use of digital signatures and gossiping. The protocol tolerates an arbitrary number of Byzantine nodes as long as the correct nodes form a connected graph.

2.4.4 Group Communication

Most research on group communication protocols for wireless ad hoc networks has been focused on managing group membership in face of mobility. Many of the proposals described below do not tolerate process failures, and none are intrusion-tolerant, but instead they address the uncertainty caused by mobility. This is perhaps an indication of the difficulty of ensuring strong agreement primitives in these environments.

The first contribution of a group communication protocol for wireless ad hoc networks was proposed by Prakash & Baldoni (1998). Their proposal consists of a group communication architecture augmented with a synchronous *proximity layer*. They state that group membership is likely to be mostly determined by the location of the nodes. The proximity layer identifies the nodes within a certain distance of a node p. For this purpose, messages are time and location stamped, and a flood-based broadcast

protocol is employed such that, for a given node p, it determines the nodes within a certain distance D, even if several hops are necessary to identify all the nodes. On top of the proximity layer lies the *group membership layer*. Using this layer, a node p can create a group G composed by a subset of nodes within distance D, which are selected according to some desired criteria, using a three round commit protocol. This contribution is another example of the need for local synchronization in wireless ad hoc networks because its correctness is dependent on the execution of an (unspecified) algorithm that ensures mutual exclusion on communication channels after a group is established.

Liu *et al.* (2005) present a similar contribution. Their approach, however, generalizes the attributes that define group membership. Besides location, they also consider scale (i.e., number of hops), trustworthiness, and QoS attributes (e.g., CPU load, memory, battery, etc.) in order to restrict group membership. They present a group service that is generic with respect to membership constraints, and realizes three basic functions: discovering mobile nodes that are eligible for membership, group initialization, and group maintenance.

The proposals of Roman *et al.* (2001) and Murphy *et al.* (2006) focus on achieving and maintaining consistent group membership despite node mobility. They identify the fact that consensus is impossible in the presence of link failures as a major obstacle. Thus, to achieve consistent group membership, they focus on hiding mobility-induced link failures. The basic idea is to make link failures unobservable to the processes. This is accomplished through the concept of announced disconnection, which basically relies on location information to disconnect processes from the group. Their model assumes a safe distance r for which communication is reliable. As soon as a process is not within r, it is disconnected from the group before a link failure can happen. Additionally, they employ group discovery and group reconfiguration protocols. The former identifies groups with the purpose of merging, while the latter deals with the actual merging and splitting of groups.

Another contribution, by Killijian *et al.* (2001), follows the same principle of setting group membership based on location. Their main contribution, however, lies on describing an approach to coverage estimation (i.e., determining if a certain set of nodes covers a predetermined geographical area) and not on group communication *per se*.

Luo & Hubaux (2004) concentrate on the problem of small scale group communication, in which wireless nodes contend for resources dictated by proximity. Their proposal, called NASCENT, employs a local membership service that tracks nodes within k hops through periodic beacon broadcasting. This protocol is used to build a directed acyclic graph (DAG) with all the nodes in the system. A token circulation protocol passes a token around the DAG to periodically visit each node. This basic operation can is then employed to construct other distributed protocols. The authors provide examples of mutual exclusion, reliable broadcast, and leader election algorithms. Key assumptions for correctness are that the system is stable enough for the token circulation period to have a small variance, and that the token visits each member infinitely often.

PILOT is designed as a toolkit for reliable group communication in wireless ad hoc networks (Luo *et al.*, 2004). Its basic building block is a protocol for probabilistic multicast, called Route Drive Gossip (RDG). This protocol relies on an underlying generic on-demand routing protocol to help in membership and message dissemination. Upon RDG two other services are offered: Reliable RDG (R^2DG) and a probabilistic quorum system for ad hoc networks (PAN). The R^2DG protocol is tailored for multicasting a stream of packets from a source. It exploits the sequence numbers of these packets to provide information about packet loss and improving the reliability of multicasting. PAN is a reliable data sharing service based on (probabilistic) quorums.

Another line of research looks at group communication through a random walk perspective. Dolev *et al.* (2006) propose a self-stabilizing group membership service for mobile ad hoc networks. Their approach is based on the notion of an agent that traverses the network by means of a random walk. This agent is responsible for collecting and distributing information about group membership. Self-stabilization consists of ensuring that exists exactly one agent in the system. When a node does not receive an agent for a predefined period of time, it produces an agent. Then, whenever a node receives two or more agents, a single agent is generated at the node. This guarantees that eventually a agent is present in the system.

Bar-Yossef *et al.* (2008) propose a membership service in the same line of research, based on random walks, called RaWMS. The main differences are that (1) while in the work of Dolev *et al.* (2006) the agent constructs a full membership of the system,

RaWMS can be used to create partial membership views, and (2) RaWMS uses multiple agents that simultaneously cover the connectivity graph, thus improving time and communication complexities of the group membership algorithm.

Chapter 3

Assessment of Intrusion-Tolerant Protocols

In order to design efficient protocols for wireless ad hoc networks, we should first understand how the current solutions work. This chapter analyzes the performance of intrusion-tolerant agreement protocols based on randomization in different environments - both wired and wireless. The knowledge obtained from this work will: (1) bring insight into the various tradeoffs involved in the implementation of intrusion-tolerant protocols, which will later support the design of new protocols specifically tailored for wireless ad hoc networks; and (2) provide a considerable understanding about the performance of randomized protocols in practical settings, a knowledge that was mainly inexistent prior to the work of this thesis.

To carry this task, we chose to evaluate the protocols from two stacks discussed in Chapter 2: RITAS and SINTRA. These stacks of randomized protocols embody a series of characteristics that we believe to represent the best fit for wireless ad hoc networks. First, they are intrusion-tolerant, being optimal in this respect by assuming up to less than one third of Byzantine processes. Second, they are general enough, since both stacks provide a series of agreement protocols (e.g., reliable broadcast, binary consensus, multi-valued consensus, atomic broadcast, etc.) that can be used to implement various distributed services. Finally, very importantly, their execution is inherently decentralized - a characteristic that is aligned with the nature of wireless ad

hoc networks. Unlike other approaches, these protocols do not rely on a designated process (i.e., a leader) to make progress.

This chapter is divided in two main sections. Section 3.1 describes an experimental performance comparison of the two classes of randomized protocols: *local coin* and *shared coin*. RITAS implements a local coin algorithm, while SINTRA implements a shared coin algorithm. These two classes are fundamentally different. Local coin protocols are usually devoid of computationally intensive cryptographic operations, but exchange many messages. Shared coin protocols, on the other hand, exchange less messages, but rely on expensive asymmetric cryptography. Since the two classes result in fundamentally different algorithms, the objective is to obtain a foundational knowledge of the environmental settings favorable to each protocol class. Section 3.2 is concerned with the performance evaluation of the protocol stacks in wireless LANs. This task also evaluates the performance of agreement algorithms other than binary consensus. The objective is to understand how the particular characteristics of wireless networks affect the performance of the protocols.

3.1 Local vs. Shared Coin Randomized Consensus

Randomized consensus protocols are based on a random step (i.e., tossing a coin), which (normally) returns values 0 or 1 with equal probability. These protocols can be divided in two classes depending on how the tossing operation is performed. There are those that use a *local coin* mechanism in each process (started in Ben-Or (1983)), and those based on a *shared coin* that returns the same value to all processes (initiated in Rabin (1983)). Typically, local coin protocols are simpler but terminate in an expected exponential number of rounds (Bracha, 1984), while shared coin protocols require a sophisticated cryptographic scheme for coin sharing but can terminate in a constant number of rounds (Cachin *et al.*, 2000; Canetti & Rabin, 1993).

To conduct the investigation, one protocol from each class was selected, implemented, and evaluated in a LAN setting under different conditions, such as distinct network characteristics and types of faults. During the experiments, the bandwidth was progressively restricted in order to understand how each protocol behaves with constrained resources, closer to a wireless environment.

The shared coin class is represented by the ABBA protocol (Cachin *et al.*, 2000), part of the SINTRA stack, which is theoretically the most efficient protocol currently found in the literature, terminating in two rounds with high probability. This protocol uses an interesting combination of cryptographic primitives, like threshold signatures and a threshold coin-tossing scheme, which makes it have a very good time complexity - it reaches decision in one or two rounds with negligible inverse probability. The local coin class is represented by Bracha's consensus protocol (Bracha, 1984), which is implemented by the RITAS stack. This protocol resorts to almost no cryptographic operations, but potentially terminates in an exponential number of rounds. Nevertheless, it has been shown to be efficient in practice, when used as part of the RITAS stack (Moniz *et al.*, 2006b).

Both protocols solve binary consensus, i.e., the values proposed and decided are binary digits, 0 and 1. This is a natural choice since most randomized consensus protocols are binary. Multi-valued consensus and other variants of consensus can be implemented on top of binary consensus protocols (see, e.g., Cachin & Poritz (2002); Correia *et al.* (2006)).

The experimental evaluation lead to several interesting conclusions. The local coin protocol was the fastest in all experiments. However, the shared coin protocol is apparently more scalable, since its performance degraded less when the number of processes was increased. The local coin protocol was also more sensitive to a network bandwidth decrease than the other, which indicates that it should perform worse in environments where the bandwidth is limited (e.g., a wireless network). The average number of rounds executed in practice was close to 1 in both protocols, although theoretically the local coin protocol runs in an exponential number of rounds.

The remainder of this section is organized as follows. Section 3.1.1 formally describes the binary consensus problem. Section 3.1.2 describes the system model and provides an overview of the protocols. Finally, Section 3.1.3 presents a comparative performance evaluation of the two protocols and discusses the results.

3.1.1 The Binary Consensus Problem

In the binary consensus problem, a set of processes p_i proposes some initial value $v_i \in \{0, 1\}$ and then they must decide on a common value. Since we are considering the

randomized model, the termination of the protocol is only guaranteed in a probabilistic way. More formally, the binary consensus problem is specified with the following properties:

Agreement If all correct processes propose the same value v, then any correct process that decides, decides v.

Validity No two correct processes decide differently.

Termination All correct processes eventually decide with probability 1.

3.1.2 System Model and Protocols

The system is composed by a set of n processes $P = \{p_0, p_1, ...p_{n-1}\}$. The processes are said to be *correct* if they follow the protocol until termination (i.e., they do not fail). Processes that fail are said to be *corrupt*. A maximum of $f = \lfloor \frac{n-1}{3} \rfloor$ processes can be corrupt during the lifetime of the system. There are no constrains on the actions taken by corrupt processes – they can, for instance, stop executing, omit messages, or send invalid messages, either alone or in collusion with other corrupt processes. This class of unconstrained faults is usually called *arbitrary* or *Byzantine*. The system is asynchronous meaning that no assumptions are made about the bounds on processing times or communication delays. It is assumed that the communication channels are unreliable, which indicates, for instance, that messages can be lost or (maliciously) modified while in transit in the network. We assume the adversary cannot break the cryptography employed in the protocols.

Bracha's Local Coin Protocol (LCP). Bracha's binary consensus protocol exchanges $O(n^3)$ point-to-point messages per round and the expected number of rounds until termination is 2^{n-f} under the *strong adversary* model[1]. The algorithm itself does not use any kind of cryptographic operations, albeit its dependence on a reliable communication channel implies the use of a relatively inexpensive cryptographic hash function.

[1] In the strong adversary model it is assumed that the adversary completely controls the network scheduling, having the power to decide the timing and the order in which the messages are delivered to the processes.

The protocol requires an extension to the basic system model to provide the binary consensus properties. Specifically, it requires a *reliable channel* abstraction, and on top of this abstraction a *reliable broadcast* primitive.

The reliable channel abstraction provides point-to-point communication between any pair of correct processes with the *reliability* and *integrity* properties. Reliability means that messages are eventually received, and integrity says that messages are delivered without modifications (i.e., messages with changes are detected and removed). In practical terms, these properties can be enforced using retransmissions and Message Authentication Codes (MACs). A MAC is a cryptographic checksum which can be calculated with a hash function and a shared key (Menezes *et al.*, 1997). Therefore, it is assumed that every pair of processes (p_i, p_j) share a secret key k_{ij}. The way these keys are given to the processes is out of the scope of the protocol, but it may require some kind of trusted dealer or key distribution protocol. In any case, since this task is performed during initialization, it does not affect performance during the execution of the protocol. Standard Internet protocols can be employed to implement the reliable channels. In the experiments, reliability was achieved with TCP and integrity with the IPSec Authentication Header protocol (Kent & Atkinson, 1998).

The reliable broadcast primitive keeps corrupt processes from broadcasting conflicting values to different processes. It ensures that: (1) all correct processes deliver the same messages, and (2) if the sender is correct then the message is delivered. Section 3.2.1 describes the protocol used to implement this primitive.

The execution of Bracha's Local Coin Protocol proceeds in 3-step rounds, running as many rounds as necessary for a decision to be reached. Each process p_i executes a round as follows:

Step 1 Reliable broadcast the initial proposal value. Wait for $n - f$ *valid* messages for this step (the meaning of *valid* is explained below). Set the new proposal value to reflect the majority of the received values. If all the $n - f$ messages have the same value v, then decide, but continue the execution of the protocol to allow the other processes also to finish.

Step 2 Reliable broadcast the proposal value. Wait for $n - f$ *valid* messages for this step. The new proposal value is set to $v \in \{0, 1\}$ if more than $n/2$ of the received

messages have the same value v. Otherwise, the new proposal value is set to a default value $\perp$.

Step 3 Reliable broadcast the proposal value. Wait for $n - f$ *valid* messages for this step. If at least $2f + 1$ messages have the same value $v \neq \perp$, then the process decides v (if it had not decided previously). Otherwise, if at least $f + 1$ have the same value $v \neq \perp$, then the process sets the new proposal value to v and a new round is initiated. If none of the previous conditions apply, then the process sets the new proposal value to a random bit with value 1 or 0, each with probability $\frac{1}{2}$, and a new round is initiated.

A message received in the first step of the first round is always considered *valid*. A message received in any other step k, for $k > 1$, is *valid* if its value is congruent with any subset of $n - f$ values accepted at step $k - 1$. Suppose that process p_i receives $n - f$ messages at step 1, where the majority has value 1. Then at step 2, it receives a message with value 0 from process p_j. Remember that the message a process p_j broadcasts at step 2 is the majority value of the messages it received at step 1. That message cannot be considered *valid* by p_i since value 0 could never be derived by a correct process p_j that received the same $n - f$ messages at step 1 as process p_i. If process p_j is correct, then p_i will eventually receive the necessary messages for step 1, which will enable it to form a subset of $n - f$ messages that validate the message with value 0.

This protocol will be referred to as simply the LCP (local coin protocol).

The ABBA Shared Coin Protocol (SCP). The ABBA binary consensus protocol exchanges $O(n^2)$ point-to-point 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.

The protocol needs the following extensions to the basic system model: reliable channels for point-to-point communication, and two cryptographic primitives – dual threshold signatures and a threshold coin-tossing scheme.

The reliable channels in the ABBA protocol only need to provide the reliability property (i.e., that messages are eventually received) between every pair of correct

processes[1]. The integrity of the messages is guaranteed by the use of public-key signatures inside the protocol itself. Therefore, the TCP protocol can be employed in the implementation of these channels.

An (n, k, f) *dual-threshold signature scheme* is a technique where n processes, from which up to f can be corrupt, hold *shares* of a private key. The processes can generate *shares of signatures* on particular messages, and k of such shares are both necessary and sufficient to assemble a valid signature. Every process has the ability to individually verify every generated share and the assembled signature. In practice, this scheme can be (and was) implemented using a vector of RSA signatures.

An (n, k, f) *dual-threshold coin-tossing scheme* is also a technique where there are n processes and at most f of them may be corrupt. Processes hold shares of an unpredictable function F that maps the coin name C to a binary value $F(C) \in \{0, 1\}$. The processes can generate shares of the coin and k of those shares are both necessary and sufficient to assemble the function F. The implemented threshold coin-tossing scheme is the Diffie-Hellman based solution of Cachin *et al.* (2000).

The execution of the ABBA Shared Coin Protocol proceeds in rounds of three steps each, except for the first round where there is an additional communication step at the beginning of the round. Let cid be a unique identifier for each execution of the protocol. Every process p_i executes the protocol as follows:

Step 0 (first round only) Broadcast a *pre-process* message containing the initial proposal value v_i along with an $(n, f + 1, f)$-signature share on the message *(cid, pre-process,* v_i*)*. Wait for $2f + 1$ *valid* pre-process messages (see the meaning of *valid* below).

Step 1 If in round $r = 1$, the new proposal value v_i is the majority value of the received *pre-process* messages. If in round $r > 1$, wait for $n - f$ *coin* messages and let $v_i = b$ if there was a *main-vote* in round $r - 1$ for $b \in \{0, 1\}$. Otherwise, let $v_i = F(C)$, where $C = (cid, r)$. Broadcast a *pre-vote* with value v_i along with an $(n, n - f, f)$-signature share on the message *(cid, pre-vote, r,* v_i*)*.

[1]Even tough "Bracha's reliable channels" have one more property than the "ABBA reliable channels", we decided to call them with the same name in order to avoid an extra channel qualifier, and therefore to keep the presentation as simple as possible.

Step 2 Wait for $n - f$ *valid* pre-votes. If there were $n - f$ pre-votes for $b \in \{0, 1\}$, then set $v_i = b$. Otherwise, set $v_i = abstain$. Broadcast a *main-vote* with value v_i along with an $(n, n - f, f)$-signature share on the message *(cid, main-vote, r, v_i)*.

Step 3 Wait for $n - f$ *valid* main-votes. If there were $n - f$ main-votes for b, then decide b and continue for one more round up to step 2. Otherwise, generate a share of the coin with name $C = (cid, r)$. Broadcast the *coin share* in a coin message, and proceed for round $r = r + 1$.

In round $r = 1$, a pre-vote for value b is valid when accompanied by an $(n, f+1, f)$-threshold signature on the message *(cid, pre-process, b)*. In round $r > 1$, a pre-vote for value b is valid when accompanied by either an $(n, n - f, f)$-threshold signature on the message *(cid, pre-vote, r-1, b)* (hard pre-vote), or an $(n, n - f, f)$-threshold signature on the message *(cid, main-vote, r-1, abstain)* (soft pre-vote). A hard pre-vote is cast when there was a main-vote for either 0 or 1 in round $r - 1$. A soft pre-vote is cast when all the main-votes in round $r - 1$ were $abstain$. The soft pre-vote value is $F(cid, r - 1)$.

A main-vote for value $abstain$ in round r is valid when it is accompanied by either the validations of two conflicting round r pre-votes (i.e., one pre-vote for value 0, and another for 1). A main-vote for value $b \in \{0, 1\}$ in round r is valid when it is accompanied by an $(n, n - f, f)$-threshold signature on the message $(cid, pre - vote, r, b)$.

Besides these validations, all received signature shares also need to be verified to accept the corresponding messages. This also includes the coin shares generated in step 3. They need to be verified before being assembled into a coin function $F(C)$.

The ABBA Shared Coin Protocol will be referred to as simply the SCP (shared coin protocol).

3.1.3 Testbed and Implementation

The experiments were conducted in a testbed consisting of 11 Dell PowerEdge 850 computers. The characteristics of the machines are the same: a single Pentium 4 CPU with 2.8 GHz of clock speed, and 2Gb of RAM. The machines were connected by a

Dell PowerConnect 2724 network switch with 10/100/1000 Mbps bandwidth capacity. The operating system was Linux, with kernel version 2.6.11. The protocols were implemented as part of the RITAS suite using the C language. The ABBA protocol, while not part of the original RITAS suite, was independently implemented and integrated into the RITAS framework.

3.1.4 Performance Metrics and System Parameters

The metrics are the set of criteria used to compare the performance of the protocols. The system parameters are the configurable variables of the system that define specific execution environments.

The two main performance metrics utilized in most experiments were the *latency* (L) and the *maximum throughput* (T_{max}). Latency is always relative to a particular process 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 consensus value. The *average latency* is obtained by taking the mean value of the sample of latency values of all processes. We also evaluate the protocols in terms of *burst latency* (L_{burst}). Given a burst of k concurrent consensus executions, the burst latency is the interval of time between the moment p_i proposes the first value and the moment it decides the k^{th} value. The throughput T_{burst} is the number of decisions per second obtained for a burst of a given size k. It is calculated by dividing the burst size k by the burst latency L_{burst}. The maximum throughput T_{max} is the value at which the throughput stabilizes (i.e., does not change with increasing burst sizes).

The system parameters selected for the experiments were the faultload, distribution of process proposals, group size, network bandwidth, and cryptography.

The *faultload* defines the types of faults that are injected in the system during its execution. In the *failure-free* faultload, all processes behave correctly. The *fail-stop* faultload makes f processes crash before the measurements are taken. In the *Byzantine* faultload, f processes try to keep the correct processes from reaching a decision by attacking the protocol execution. This is accomplished as follows. In the LCP, a Byzantine process in steps 1 and 2 always proposes the opposite value that it would propose if it were behaving correctly, and in step 3 always proposes the default value $\perp$. In the SCP, since a Byzantine process has no possibility of proposing an invalid value

without detection (because of the employed asymmetric cryptography), it transmits messages with invalid signatures and justifications in order to force extra computation in the correct processes.

The *proposal distribution* defines the initial values to be proposed by the processes. The *uniform* proposal distribution makes all processes propose the same initial value 1. In the *corrosive* proposal distribution, processes with an odd process identifier propose 1 and the others propose 0. The *random* proposal distribution chooses for the initial proposal of each process a randomly selected value.

The *group size* defines the number of processes n in the system. In our case it can take three values: 4, 7, and 10.

The *network bandwidth* is the bandwidth of the network links defined in the network switch. It can take three values: 10 Mb/s, 100 Mb/s, and 1000 Mb/s. The default network bandwidth used in the experiments was 1000 Mb/s.

Cryptography defines the type of cryptography employed by the protocols. The LCP can only either use the IPsec AH protocol (with SHA-1) or no cryptography at all (in this case the protocol correctness is affected but this setting is utilized for the sake of comparative evaluation only). The SCP uses RSA with three possible key sizes: 512, 1024, and 2048 bits. The cryptographic default settings are set to IPSec for the LCP, and 1024-bit RSA keys for the SCP.

3.1.5 The Experiments

Three main experiments are presented that demonstrate how different system settings affect the performance of the protocols. The first experiment aims to understand the impact of the group size and proposal distribution. The second analyzes how the protocols perform under different types of faults. Finally, the third looks into how the cryptographic and bandwidth parameters influence performance.

Group Size and Proposal Distribution. The results presented in this subsection are a starting point for the rest of the performance analysis. They show the latency values for both consensus protocols with different group sizes and proposal distributions. The remaining parameters were set to the default values and no faults were injected (i.e., the failure-free faultload).

The measurements were taken the following way: a signaling machine, which does not participate in the protocols, is selected to control the benchmark execution. It broadcasts m 1-byte UDP messages to the n processes involved in the experiment, each one separated by a five second interval (in this case m was set to 100). Whenever one of these messages arrives to a specific process, it executes whatever protocol is relevant for the current experiment (LCP or SCP). The average latency is obtained from the sample of 100 executions.

	Local Coin (μs)			Shared Coin (μs)		
	$n = 4$	$n = 7$	$n = 10$	$n = 4$	$n = 7$	$n = 10$
uniform	824	2187	4132	21590	31315	43633
corrosive	2453	6172	12075	33834	38529	55169
random	2056	5812	11501	24320	36325	49206

Table 3.1: Average latency in microseconds (μs) for different group sizes and proposal distributions.

The results of this experiment are shown in Table 3.1. It can be observed that the LCP is very fast, reaching a decision in less than 1 ms with 4 processes and a uniform proposal distribution. The SCP is comparatively slower, despite having a lower communication complexity. In these environmental settings, the lower number of exchanged messages of the SCP clearly does not compensate for the computationally intensive asymmetric cryptography.

Nevertheless, even though the LCP is significantly faster than the SCP, its latency grows at a faster rate as the group size increases. While the latency of the LCP roughly doubles at successive larger group sizes, the latency for the SCP grows at roughly 50%. This indicates that SCP could potentially outperform LCP in groups with high numbers of processes.

Protocol Behavior under Different Faultloads. This section studies the behavior of the protocols when subject to different faultloads. Two system parameters were varied in this experiment: the faultload and the number of processes. The proposal distribution was fixed to random, and the rest of the parameters were set to the default values. The metrics used to assess the performance of the protocols were the latency and throughput.

The experiment was carried out by having the n processes run several concurrent consensus executions. A signaling machine, which does not participate in the execution of the protocols, sends a 2-byte UDP message to all n processes, containing a number k. When a process receives this message, it starts a burst of k simultaneous consensus executions. The burst latency L_{burst} and the burst throughput T_{burst} are the metrics used for this experiment. For every tested burst size k, the displayed result reflects the average value of 10 executions.

The results for each faultload are presented in three separate pairs of graphs. The first graph of each pair studies the latency and the second the throughput.

Failure-free Faultload. The results when there are no faults are shown in Figures 3.1 and 3.2, respectively for the LCP and the SCP protocols. Each curve represents a different group size n.

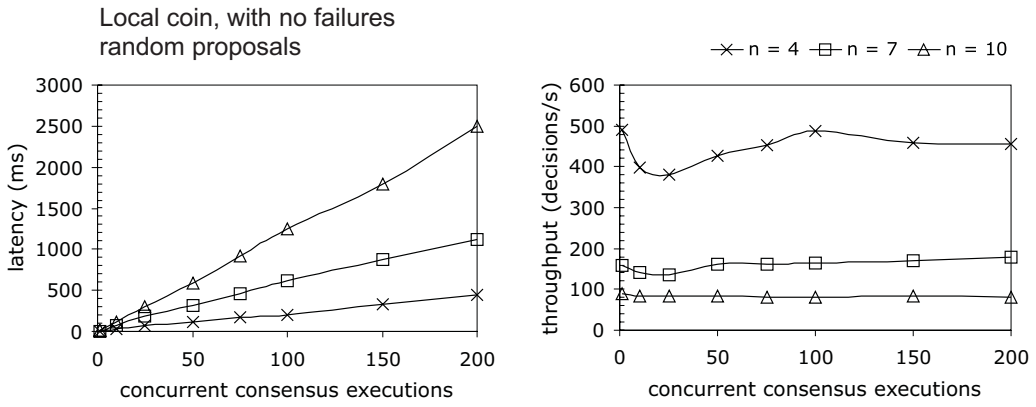

Figure 3.1: Burst latency and throughput for the LCP with no failures

From the graphs it is possible to observe that the burst latency L_{burst} is linear with the number of concurrent consensus executions. The stabilization points in the throughput curves indicate the maximum throughput T_{max} for each n. It can be seen that the LCP is much faster than the SCP, achieving much lower latency and higher throughput values, irrespective of the group size.

For the LCP, for 200 concurrent consensus executions, L_{burst} has a value of 439 ms with $n = 4$, 1126 ms with $n = 7$, and 2492 ms with $n = 10$. The maximum

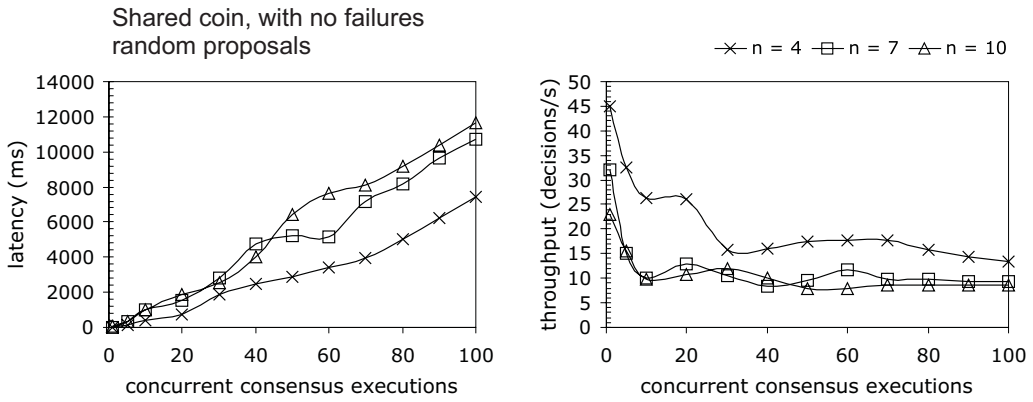

Figure 3.2: Burst latency and throughput for the SCP with no failures

throughput T_{max} is around 455 decisions/s with $n = 4$, 175 decisions/s with $n = 7$, and 81 decisions/s with $n = 10$.

As for the SCP, for 100 concurrent consensus executions, L_{burst} has a value of 7454 ms with $n = 4$, 10713 ms with $n = 7$, and 11625 ms with $n = 10$. The maximum throughput T_{max} is around 13 decisions/s with $n = 4$, 9 decisions/s when $n = 7$, and 8 decisions/s when $n = 10$.

Fail-stop Faultload. Figures 3.3 and 3.4 display the performance of the protocols when there are f crashed processes in the system. Each curve shows the latency and throughput for a different group size n.

For the LCP, the performance is noticeably better with f crashed processes than it is in the failure-free scenario. This happens because with fewer processes there is less contention on the network and more bandwidth is available to exchange the messages. This result gives a hint that the performance bottleneck of the LCP is indeed the communication (as opposed to the computation). In more detail, for 200 concurrent consensus executions, L_{burst} has a value of 329 ms with $n = 4$, 890 ms with $n = 7$, and 1820 ms with $n = 10$. The maximum throughput T_{max} is around 608 decisions/s with $n = 4$, 225 decisions/s with $n = 7$, and 110 decisions/s with $n = 10$.

As for the SCP, the performance results when there are f crashed processes also show a significant improvement over the failure-free scenario. In this case, however, the reduced network contention does not explain entirely what happens. In more detail,

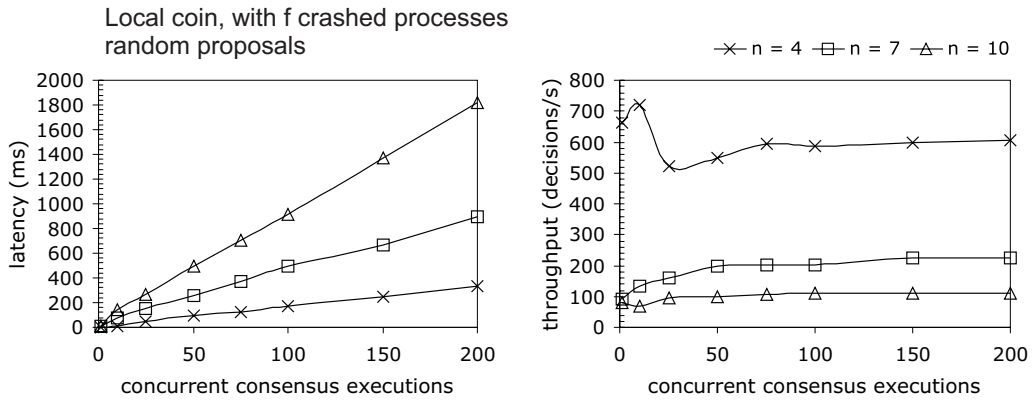

Figure 3.3: Burst latency and throughput for the LCP with f crashed processes

for 100 concurrent consensus executions, L_{burst} has a value of 3215 ms with $n = 4$, 3959 ms with $n = 7$, and 4981 ms with $n = 10$. The maximum throughput T_{max} is around 31 decisions/s with $n = 4$, 25 decisions/s when $n = 7$, and 20 decisions/s when $n = 10$.

While this can not be inferred from the graphs, what really speeds up the SCP is the fact that, with only $n - f$ processes proposing values, all the correct processes see exactly the same $n - f$ messages at every protocol step, resulting always in 1-round decisions for all protocol executions (this behavior also happens in the LCP, but its performance is affected to a lesser degree by it). In the failure-free scenario, even with an overwhelming majority of executions reaching decision in only one round, it only takes a single execution needing more than one round to considerably delay the burst latency, hence the performance improvement in the fail-stop case.

Byzantine Faultload. Figures 3.5 and 3.6 depict the protocols' performance when there are f processes trying to disrupt their execution. Each curve shows the latency and throughput for a different group size n.

For the LCP, the curves show that the performance is negatively affected by the Byzantine failures. For 200 concurrent consensus executions, L_{burst} has a value of 592 ms with $n = 4$, 2290 ms with $n = 7$, and 6772 ms with $n = 10$. The maximum

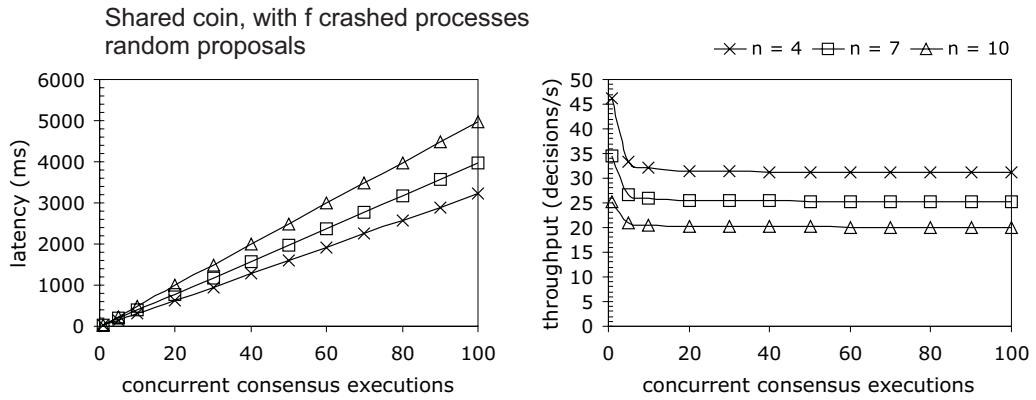

Figure 3.4: Burst latency and throughput for the SCP with f crashed processes

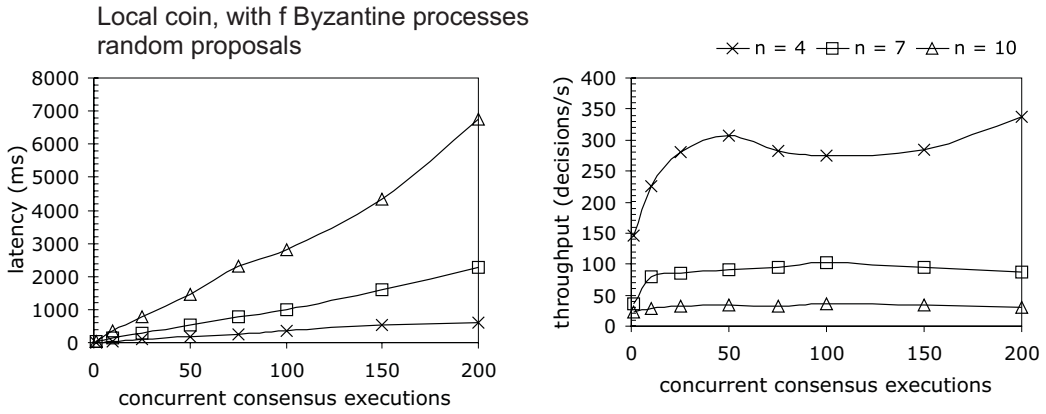

Figure 3.5: Burst latency and throughput for the LCP with f Byzantine processes

throughput T_{max} is around 337 decisions/s with $n = 4$, 87 decisions/s with $n = 7$, and 30 decisions/s with $n = 10$. While it is not possible for f malicious processes to prevent correct processes from reaching a decision, it is still possible for them to increase the number of rounds needed to reach a decision. This can be directly linked to the lack of cryptographic verifications on the received messages, which affects the robustness. In practice, this means that the processes usually have to wait for extra messages beyond the $n - f$ threshold because the malicious processes are proposing values that fail the validation. A good example can be found in step 3 of the the protocol. In alternative to $b \in \{0, 1\}$, the processes are allowed to propose an 'undecided' default value $\perp$ for which the verification step is somewhat relaxed (it only needs $n - f - \frac{n}{2}$ *valid*

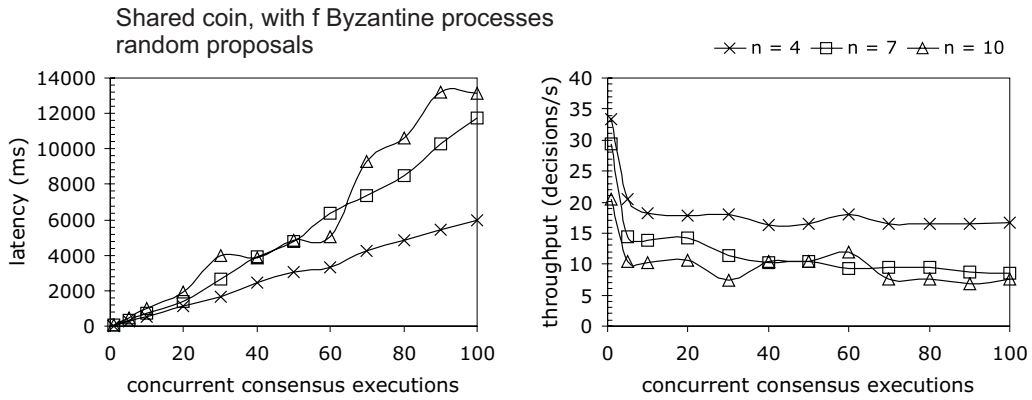

Figure 3.6: Burst latency and throughput for the SCP with f Byzantine processes

different proposals from the majority value in step 2 to be considered valid by a correct process), which Byzantine processes exploit in order to try to postpone a decision for one extra round.

For the SCP, it is clear from the curves that there is not a noticeable performance penalty in relation to the failure-free scenario. In more detail, for 100 concurrent consensus executions, L_{burst} has a value of 5987 ms with $n = 4$, 11712 ms with $n = 7$, and 13139 ms with $n = 10$. The maximum throughput T_{max} is around 16 decisions/s with $n = 4$, 9 decisions/s when $n = 7$, and 8 decisions/s when $n = 10$. These results are directly related to the extensive use of public-key cryptography which provides superior robustness to the protocol when compared to the LCP. In this case, a Byzantine process has no room to "lie" since it has to justify all of its proposals with a vector of signatures received from the other processes. The best a malicious process can do is to force correct processes to verify more signatures by sending proposals with invalid justifications. A correct process has to verify an extra vector of signatures beyond the $n-f$ threshold for each invalid justification it receives before gathering $n-f$ valid proposals. Nevertheless, this does not have a significant performance impact because the cost of verifying signatures is much smaller than the cost of constructing signatures.

Number of Rounds. Tables 3.2 and 3.3 show the number of rounds needed to reach a decision by the LCP and SCP, respectively. The presented results are the average number of rounds until decision of all the consensus executions of the above experiments

which sum up to approximately 6000 executions per each tested group size/faultload pair.

Local Coin			
	$n = 4$	$n = 7$	$n = 10$
failure-free	1.004 (0.42)	1.005 (0.14)	1.009 (0.19)
fail-stop	1 (0)	1 (0)	1 (0)
Byzantine	1.462 (1.52)	1.569 (1.69)	2.289 (2.79)

Table 3.2: Average number of rounds for the LCP. The standard deviation is shown in parenthesis.

Shared Coin			
	$n = 4$	$n = 7$	$n = 10$
failure-free	1.013 (0.23)	1.018 (0.27)	1.01 (0.2)
fail-stop	1 (0)	1 (0)	1 (0)
Byzantine	1.016 (0.25)	1.017 (0.26)	1.012 (0.22)

Table 3.3: Average number of rounds for the SCP. The standard deviation is shown in parenthesis.

The first noticeable result is that the average number of rounds is much lower than what is suggested by previous theoretical results even when considering Byzantine faults. For instance, the expected theoretical number of rounds for the LCP with a strong adversary is 2^{n-f}. The obtained average number of rounds with 10 processes (of which 3 of them are malicious) is a little bit above 2 rounds (2.289) which is very far from the theoretical result (128 rounds).

When comparing the two protocols, their performance is similar except for the Byzantine case where the SCP is much more robust. In fact, there is practically no difference in the number of rounds of the SCP between the failure-free, and the Byzantine fault loads. The SCP also shows no degradation with an increasing number of processes, while the LCP shows some degradation, but only in the Byzantine scenario.

The fact that both protocols always reach decision in one round with f crashed processes has a simple explanation. When f processes crash just before the execution of the protocols, and the system is left with $n - f$ processes it is guaranteed that at every step of the protocols, the correct processes always see the same set of expected $n - f$ messages. Since the proposed values at each step are based on this set of $n -$

f messages, they will always propose the same values every step of the way, thus achieving a decision by the first round.

Cryptography vs. Bandwidth. Although the previous results already give some idea of the impact of the cryptographic and bandwidth parameters on the performance of the protocols, this section provides a comprehensive analysis of the relative costs of communication and computation. In this experiment the only varying parameters are the bandwidth and the cryptographic settings. The tested bandwidth values are 1000, 100, and 10 Mbit/s. The measurements made for the LCP use IPSec activated and deactivated. For the SCP, the experiments are taken with three RSA key sizes: 512, 1024, and 2048 bits. No failures were injected during the experiments, the group size was 4 processes, and the proposal distribution was set to uniform.

Figure 3.7 depicts the performance of the LCP with each curve corresponding to a specific bandwidth/IPSec combination. It shows that cryptography (IPsec) has almost no impact on the performance of the protocol, and that reducing the available bandwidth results in a great performance cost. Every time the bandwidth is downgraded, the latency and throughput suffer a significant degradation. Considering 200 concurrent consensus executions, the throughput has a value of approximately 1300 decisions/s with a bandwidth of 1000 Mbit/s, 170 decisions/s with 100 Mbit/s, and 60 decisions/s for with 10 Mbit/s.

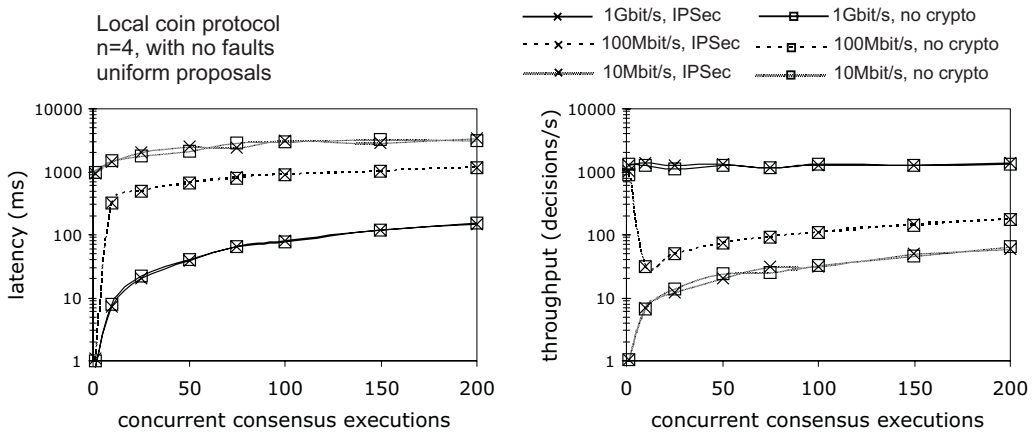

Figure 3.7: Burst latency and throughput for the LCP with different bandwidth and cryptographic parameters

The performance of the SCP is shown in Figure 3.8. This scenario is quite different from the LCP. While reducing the network bandwidth has a negative effect on performance, this cost is not as accentuated as the cost of increasing the key size. Regardless of the available bandwidth, the measurements with 2048-bit keys rank the bottommost among all the experiments, and the measurements with 512-bit keys rank among the first four spots. More closely, for 200 concurrent consensus executions, the measured throughput with a bandwidth of 1000 Mb/s was approximately 46, 30, and 10 decisions/s for 512, 1024, and 2048 bit keys, respectively. With 100 Mb/s it was 39, 23, and 8 decisions/s for 512, 1024, and 2048 bit keys, and with 10 Mb/s was 19, 16, and 6 decisions/s for 512, 1024, and 2048 bit keys.

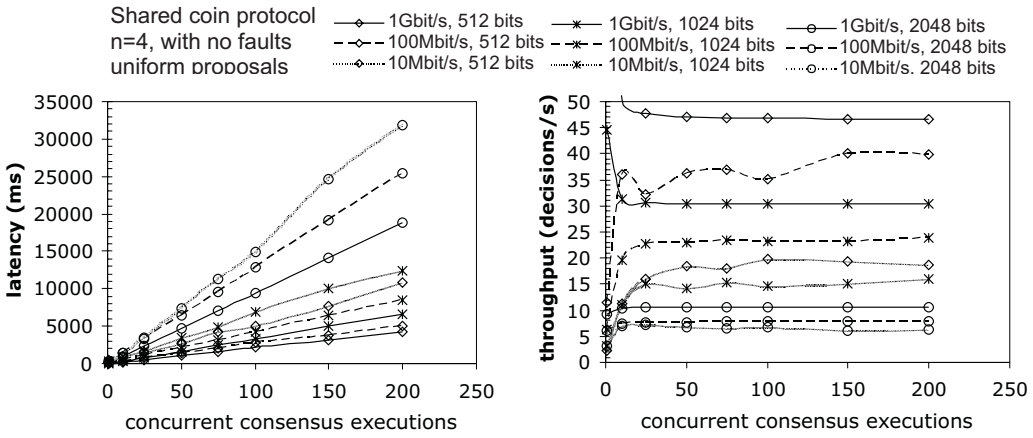

Figure 3.8: Burst latency and throughput for the SCP with different bandwidth and cryptographic parameters

The SCP, despite getting significantly closer to the LCP in terms of performance each time the network settings were downgraded, never matched the performance of the LCP with identical network bandwidth. Nevertheless, in a wireless setting where there is much less bandwidth available, and there is a significantly higher network delay, it seems plausible for the SCP to outperform the LCP. Besides benefiting from its relatively lower number of exchanged messages, the network latency would offset the cryptographic computation costs of the SCP.

3.1.6 Summary of Results

The most important conclusions from this experimental evaluation are summarized in the following points:

- The LCP is significantly faster than the SCP with similar system parameters for all the environmental settings tested.
- The SCP, while slower, proved to be more scalable since its performance degraded to a lesser degree than the LCP with an increasing number of processes.
- The SCP is more robust than the LCP since it was not affected in terms of performance when malicious faults were injected in the system. The LCP evidenced a little degradation with respect to the number of rounds, latency and throughput.
- The measured average number of rounds of both protocols was quite small, being close to one with no faults, and exactly one with f crashed processes. With respect to the situation with f malicious processes, the SCP scored similar to the failure-free scenario, and the LCP showed a small degradation which was accentuated when the number of processes was higher.
- The performance bottleneck for the LCP, when many consensuses were executed concurrently, was the network because of its high number of exchanged messages and use of cheap cryptography. For the SCP, the bottleneck was the CPU because it exchanged a small number of messages and utilized expensive asymmetric cryptography.

3.2 Evaluation of Intrusion-Tolerant Protocols in Wireless LANs

This section extends the performance analysis to wireless environments, in particular to 802.11 wireless LANs, and to a greater number of protocols. In addition to the binary consensus protocols described in the previous section, this section evaluates the protocols from the RITAS stack (Moniz *et al.*, 2010), discussed in Section 2.3.

The evaluated protocols are echo broadcast, reliable broadcast, multi-valued consensus, and vector consensus. They are arranged in a stack depicted in Figure 3.9. At the lowest level of the stack there are two broadcast primitives: reliable broadcast and echo broadcast. These ensure that processes do not receive contradictory messages. On top of these primitives, there is the most basic form of consensus, the binary consensus, which can be the LCP or the SCP. The rest of the protocols are simply built on the top of this one. Multi-valued consensus allows agreement on values with an arbitrary domain. Vector consensus lets processes decide on a vector with values proposed by a subset of the processes. Most protocols from the RITAS stack were originally designed by Correia et al. (Correia *et al.*, 2006). They were further improved, implemented, and evaluated under LAN, WAN and Wireless environments in three subsequent papers (Moniz *et al.*, 2006b, 2007, 2010).

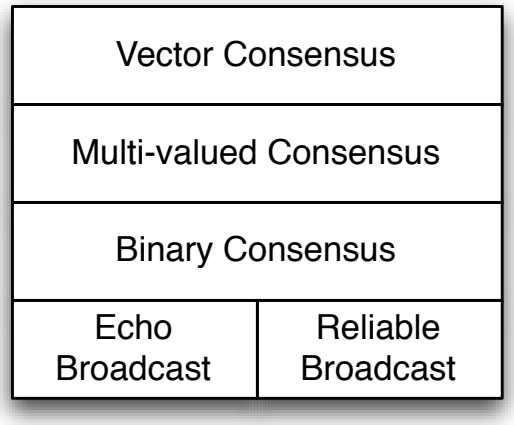

Figure 3.9: Evaluated protocol stack.

3.2.1 System Model and Protocols

The system model for the protocols is the same as the one described in Section 3.1.2 with the reliable channel extension. In a nutshell, the assumptions are that the system is asynchronous, it is composed by a set of n processes, $f \leq \lfloor \frac{n-1}{3} \rfloor$ may exhibit arbitrary behavior, and any pair of correct processes is connect by a reliable channel. The

reliable channel enforces the reliability and integrity properties. Reliability guarantees that messages are eventually received and integrity ensures that messages are not modified in the channel.

Reliable Broadcast and Echo Broadcast. The *reliable broadcast* primitive ensures two properties: (1) all correct processes deliver the same messages; (2) if the sender is correct then the message is delivered. The implemented protocol was originally proposed by Bracha (Bracha, 1984). The protocol starts with the sender broadcasting a message (INIT, m) to all processes. Upon receiving this message a process sends a (ECHO, m) message to all processes. It then waits for $\lfloor \frac{n+f}{2} \rfloor + 1$ (ECHO, m) messages or $f + 1$ (READY, m) messages, and then it transmits a (READY, m) message to all processes. Finally, a process waits for $2f + 1$ (READY, m) messages to deliver m.

The *echo broadcast* primitive is a weaker and more efficient version of the *reliable broadcast*. Its properties are somewhat similar, however, it does not guarantee that all correct processes deliver a broadcast message if the sender is corrupt (Toueg, 1984). In this case, the protocol only ensures that the subset of correct processes that deliver will do it for the same message. The protocol is essentially the described *reliable broadcast* algorithm with the last communication step omitted. An instance of the protocol is started with the sender broadcasting a message (INITIAL, m) to all processes. When a process receives this message, it broadcasts a (ECHO, m) message to all processes. It then waits for $\lfloor \frac{n+f}{2} \rfloor + 1$ (ECHO, m) messages to accept and deliver m.

Binary Consensus. The *binary consensus* protocol is the Bracha's Local Coin Protocol (Bracha, 1984). This protocol was described in detail in Section 3.1.2.

Multi-Valued Consensus. A *multi-valued consensus* allows processes to propose a value $v \in \mathcal{V}$ with arbitrary length. The decision is either one of the proposed values or a default value $\perp \notin \mathcal{V}$. The implemented protocol is based on the multi-valued consensus proposed by Correia *et al.* (2006). It uses the services of the underlying *reliable broadcast*, *echo broadcast*, and *binary consensus* layers. The main differences from the original protocol are the use of echo broadcast instead of reliable broadcast at a specific point, and a simplification of the validation of the vectors used to justify the proposed values.

The protocol starts when every process p_i announces its proposal value v_i by reliably broadcasting a (INIT, v_i) message. The processes then wait for the reception of $n - f$ INIT messages and store the received values in a vector V_i. If a process receives at least $n - 2f$ messages with the same value v, it echo-broadcasts a (VECT, v, V_i) message containing this value together with the vector V_i that justifies the value. Otherwise, it echo-broadcasts the default value $\perp$ that does not require justification. The next step is to wait for the reception of $n-f$ *valid* VECT messages. A VECT message, received from process p_j, and containing vector V_j, is considered *valid* if one of two conditions hold: (a) $v = \perp$; (b) there are at least $n - 2f$ elements $V_i[k] \in \mathcal{V}$ such that $V_i[k] = V_j[k] = v_j$. If a process does not receive two *valid* VECT messages with different values, and it received at least $n-2f$ *valid* VECT messages with the same value, it proposes 1 for an execution of the *binary consensus*, 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 already) with the same value v and decides on that value.

Vector Consensus. *Vector consensus* allows processes to agree on a vector with a subset of the proposed values. The protocol is the one described in Correia *et al.* (2006) and uses *reliable broadcast* and *multi-valued consensus* as underlying primitives. It ensures that every correct process decides on a same vector V of size n; if a process p_i is correct, then $V[i]$ is either the valued proposed by p_i or the default value $\perp$, and at least $f + 1$ elements of V were proposed by correct processes.

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 is carried out 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.

3.2.2 Testbeds and Implementation

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 (White *et al.*, 2002). 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 done 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.

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 reliable channels were implemented by the use of TCP for reliability, and the IPSec Authentication Header protocol for integrity (Kent & Atkinson, 1998).

RITAS was originally implemented in C and compiled in gcc in Linux. Although only wired LANs were assumed, it was not necessary to change it for the experiments in the wireless testbed with Linux PCs (testbed *tb-emulab*). However, to test it with PDAs (testbed *tb-pda*), most RITAS code had to be ported to the Windows Mobile platform. 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.

3.2.3 Performance Metrics and System Parameters

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, it is the interval of time between the moment the sender process, say p_i, initiates the execution of the protocol, and the moment p_i delivers the broadcasted value.

The system parameters are configurable parameters 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). Additionally, in the experiments described in this section, every process starts with the same proposal value. Hence, an unanimous proposal distribution is used.

3.2.4 The Experiments

Four different experiments are presented. Each one was tailored to evaluate the impact of specific system parameters in the latency of the algorithms. 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 with the goal of expanding the knowledge obtained from the evaluation described in Section 3.1 to wireless settings.

The experiments were all carried out the same 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 process 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.

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). The evaluated binary consensus protocol was always Bracha's (Bracha, 1984), except where an explicit comparison between the two binary consensus protocols is presented.

Wireless Standard and Group Size. This experiment 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 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 3.4 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 it is justified by the large messages that multi-valued consensus has to reliably broadcast to justify the proposed values (Correia *et al.*, 2006). The gaps from reliable broadcast to binary consensus, and from multi-valued consensus to vector consensus are smaller and directly related to the overhead incurred from the respective upper-layer protocols.

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

Table 3.4: Latency measurements for different wireless standards and group sizes in testbed *tb-emulab* in infrastructure mode.

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 latency cost is 2504.6 ms in 802.11g, and 1828.5 ms in 802.11a, which is a considerable difference.

Computational Capability. The second set of experiments measures how the computational capability of the individual nodes affects the performance of the protocols. 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

Group Size	Testbed	Latency (ms)			
		Reliable Broadcast	Binary Consensus	Multi-valued Consensus	Vector Consensus
$n = 4$	tb-emulab	12	35	160	210
	tb-pda	26	211	320	374
$n = 7$	tb-emulab	36	154	972	1474
	tb-pda	52	1626	2555	3221

Table 3.5: Average latency in 802.11b ad-hoc network for both testbeds.

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).

The obtained measurements are presented in Table 3.5. From the results it is clear that the computational characteristics of the individual nodes greatly affect 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.

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.

The measurements are presented in Table 3.6. 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

Group Size	Testbed	Latency (ms)			
		Reliable Broadcast	Binary Consensus	Multi-valued Consensus	Vector Consensus
$n = 4$	ad-hoc	26	211	320	374
	infrastructure	43	631	952	1190
$n = 7$	ad-hoc	52	1616	2555	3221
	infrastructure	138	7620	10062	12929

Table 3.6: Average latency for *tb-pda* with 4 and 7 processes.

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.

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 that were studied in Section 3.1: Bracha's protocol, which was the local coin protocol (LCP), and the ABBA protocol, which was the shared coin protocol (SCP). In particular, this section presents some considerations about the performability of the two strategies employed by the protocols – one depends on the heavy use of public-key cryptography (ABBA), and the other on abundant message exchanges (Bracha's).

Only testbed *tb-emulab* was used in the experiments. This is justified by the fact that the computational cost of the cryptographic operations of the SCP made its evaluation unfeasible in testbed *tb-pda*. In our tests, the generation of just a single signature took several seconds. The system parameters evaluated were: group size of 4, 7, and 10 processes; 802.11b and 802.11g wireless standards; ad hoc and infrastructure network topologies.

Given the features of the protocols it was expected for the LCP to outperform the SCP given more favorable network conditions, and to exist a certain point, as network

Group Size	Algorithm	Latency (ms)			
		802.11g		802.11b	
		ad-hoc	infra.	ad-hoc	infra.
$n = 4$	LCP	11	18	35	43
	SCP	147	148	146	160
$n = 7$	LCP	43	94	154	293
	SCP	210	211	211	270
$n = 10$	LCP	95	326	415	833
	SCP	290	311	301	717

Table 3.7: Average latency for binary consensus protocols in *tb-emulab*.

conditions degrade, where the SCP strategy would pay off to the point of being faster than the LCP. Table 3.7 presents the measurements observed for the various environmental settings tested.

The results confirm the expectations. The table shows the measurements for the best network configuration (802.11g and ad-hoc), where the LCP is clearly faster than the SCP, even for a group size of ten processes – 95 ms against 290 ms, respectively. In remaining scenarios 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 the SCP outperforms the LCP. For the 802.11b/adhoc and 802.11g/infrastructure scenarios, this happens when the group size is ten, and for the 802.11b/infrastructure, which is the worst network configuration, this happens at $n = 7$. The conclusion is that the LCP is much faster with few processes and "good" network conditions, but it quickly degrades with the network capacity up to a point where the SCP, being more resilient to the network limitations, becomes faster.

3.2.5 Summary of Results

The most important conclusions from this experimental evaluation are summarized in the following points:

- 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 (LCP) is faster than ABBA binary consensus (SCP) 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.

- In devices with limited computational capabilities the use of public-key cryptography must be kept to a minimum. The generation of a single signature took several seconds in testbed *tb-pda*, making the evaluation of the SCP infeasible.

3.3 Discussion of Results

The experimental evaluation of Section 3.1, which compares two classes of randomized consensus protocols in a wired LAN, lead to several important conclusions. The first is that the local coin protocol has a much better latency and throughput than the shared coin protocol in a LAN, both with and without crash or Byzantine failures. Another conclusion is that the higher number of messages sent by the local coin protocol tends to increase the latency and decrease the throughput when many consensuses are executed in parallel, especially when the bandwidth is reduced. Shared coin protocols are somewhat less sensitive to these factors, since much of their cost is in computing cryptographic operations in the machines.

These results give the idea that, for devices with similar computational power of those used in these experiments, shared coin protocols potentially perform better than local coin protocols in a WLAN or WAN. This is because the available bandwidth in these environments is usually not higher than the minimum we considered in the experiments (10Mbit/s). Moreover, since the communication delay is typically much higher, the extra communication steps of the LCP protocol will have an important cost. It it not clear, however, how devices with constrained computational power, usually found in wireless ad-hoc network, will perform with the SCP protocol given the significant amount of asymmetric cryptographic operations involved.

The set of experiments described in Section 3.2 provided a rich corpus of results regarding the performance of intrusion-tolerant protocols in wireless networks. The most relevant observations to extract are related to (1) how the stack of protocols performed as a whole and if their performance is acceptable for the deployment of distributed applications in wireless environments, and (2) how the two binary consensus protocols fared in relation of each other.

For this first point it can be said that the performance can be acceptable in some circumstances, but not in general. For example, the performance may be acceptable if the application environment involves a reduced number of processes, relatively powerful hardware, and a considerable amount of bandwidth (e.g., 4 processes, Pentium III processes, and 802.11g). Even still, the application must be comfortable with protocols delays of, roughly, up to 1 second. It is clear then that this current generation of intrusion-tolerant protocol cannot sustain the execution of distributed applications with more than relatively modest performance requisites. Of particular importance is that the protocols are very limited when it comes to scaling to a higher number of processes. For example, the results with 10 processes can only rarely be acceptable, specially when considering more resource-constrained environments, either in terms of computational power or network bandwidth.

For the second point, it was interesting to confirm the results that were suggested by the experiments described in Section 3.1. As the networking resources became increasingly restricted, either because less bandwidth was available or because more processes were present in the system, the shared coin protocol (ABBA) evolved from a position where it was clearly outperformed by the local coin protocol (Bracha's) to a

position where it fared better than the local coin protocol. Nevertheless, the high computational cost involved in the cryptographic operations of the shared coin protocol made the evaluation of that protocol impractical in testbed *tb-pda* where, for instance, the generation of a single valid signature took several seconds. These results indicate that none of these protocols are fit to resource-constrained environments. They react poorly with either restricted computational power (e.g., ABBA) or restricted communication capacity (e.g., Bracha's).

Chapter 4

Consensus with Faulty Transmissions of a Restricted Source

This chapter draws on the results obtained in the previous chapter and introduces the communication failure model of Santoro and Widmayer as a new system model suited for wireless ad hoc networks (Section 4.1). The introduction of the new model is completed with the presentation of a consensus algorithm designed for a simplified version of the desired model (Section 4.2).

4.1 A New System Model for Wireless Ad hoc Networks

As the experiments of the previous chapter demonstrated, protocols for wireless ad hoc networks should strive to keep communication complexity low and avoid expensive cryptography. These two characteristics represent the respective performance bottlenecks for each class of randomized protocols, since these networks are usually restricted in their communicational and computational resources. Apart from these two characteristics, which pertain to the design of the protocols, the system model usually employed by intrusion-tolerant systems also plays a significant role in their (poor) performance in wireless ad hoc networks.

The design of both RITAS (Moniz *et al.*, 2010) and SINTRA (Cachin & Poritz, 2002), and intrusion-tolerant protocols in general, follows a traditional modeling ap-

proach where it is assumed an asynchronous timing model, static process failures, and reliable point-to-point communication links. In such a model, as long as any two processes are correct, any message sent from one process to another is eventually delivered. If the environment does not provide such guarantee, then reliable channels have to be implemented in order to abstract the underlying unreliable communication from protocol design (e.g., TCP). In wireless environments, where the communication medium is shared, this approach significantly increases the medium access contention, impairing the overall performance.

Since wireless ad hoc networks provide a natural broadcasting medium, the cost of transmitting a message to multiple processes can be just the same of transmitting it to a single process, as long as they are within communication range. To take advantage of this feature, it becomes necessary to depart from the common modeling assumption of reliable point-to-point channels. The unreliability inherent to radio communications has to be dealt with in some other way. As such, models that assume unreliable communication links are more adjusted to wireless networking. Tolerance to message loss becomes integrated within the semantics of the algorithms, instead of being abstracted by typically inefficient implementations. To this end, the following section introduces the communication failure model, where links are assumed to be unreliable.

4.1.1 The Communication Failure Model

In the traditional models for distributed systems, faults are static and component-bound, i.e., a fault is associated to a particular component that is forever considered faulty. The faulty component can be a process or a communication link (e.g., (Pease *et al.*, 1980; Perry & Toueg, 1986)). These models are referred to as *component failure models*. For systems based on these models to operate correctly, a certain number of components must not exhibit failures during their entire operation time.

The *communication failure model* (Santoro & Widmayer, 2007; Santoro & Widmeyer, 1989) differs from the component failure models in this regard: failures are not static. Instead, they are *dynamic* and *transient* in the sense that they can occur anywhere in the system, and following a failure, normal functioning can be resumed after some unknown time. This pattern may repeat any number of times. Such an approach

implicitly allows every component of the system to eventually fail. The only restriction is placed on the number of faults that simultaneously manifest in the system.

This model is also more aligned with the inherent uncertainty of wireless ad hoc networks. Nodes are usually subject to momentary disconnection due to node mobility and other environmental phenomena such as electromagnetic interference, fading, collisions, etc. These events may result in message loss or corruption, but should not be sufficient to permanently assume a process or link as faulty, specially because they can possibly affect many processes during the lifetime of the system.

In the communication failure model, faults are assumed to occur only in message transmissions. A transmission amongst a source process p_i and a destination process p_j is faulty if one of the following occurs:

Omission. The message sent by p_i is not received by p_j.

Addition. A message is delivered to p_j when no such message was sent.

Corruption. The message sent by p_i is received by p_j with different content.

Under this model, processes are modeled as not to exhibit faulty behavior. The notion of a faulty process is instead captured by the assumption of faulty message transmissions. On message-passing systems, any component failure will ultimately manifest itself as transmission faults. For example, a process crash will manifest into a series of transmission omission faults with the crashed process as sender, and a process that is attacked and falls under the control of a maliciously adversary may manifest into a series of transmission corruption faults where the contents of the messages are modified relative to the original protocol. Message corruptions and additions are exclusively dedicated to capture the arbitrary actions of malicious processes, which will happen in practice. Of course, accidental corruptions will also occur, but these are assumed to be detected by integrity checking mechanisms (e.g., a checksum function). Consequently, these can be safely treated as message omissions (e.g., if a message fails an integrity check, then it is simply discarded). Since faults can occur non-uniformly at the receivers, a message broadcast is modeled as n separate transmissions. For example, a process p_j may receive a message broadcast by p_i, while some other process p_l may receive a corrupted version of that message or no message at all.

4. CONSENSUS WITH FAULTY TRANSMISSIONS OF A RESTRICTED SOURCE

The communication failure model is constrained by the Santoro-Widmeyer impossibility result (Santoro & Widmayer, 2007; Santoro & Widmeyer, 1989). This result states that no agreement protocol is possible in a synchronous system if just $n - 1$ omission faults may occur. This means that, in practice, the crashing of any one node (which causes $n - 1$ transmission omission failures) renders any form of agreement impossible. This result is parallel to the widely known FLP impossibility result for asynchronous systems (with reliable links) which states that consensus is impossible if at least one node can crash (Fischer *et al.*, 1985).

Given the usefulness of this model for wireless ad-hoc environments, it deserves a systematic study in order to circumvent its impossibility result. It is of particular importance, however, that this process does not involve extending the model with unreasonable assumptions regarding the system. In this regard, randomization seems to be the best approach because it just implies assuming that processes have access to some local source of random information. Other approaches, previously used to circumvent the FLP result, do not appear to be appropriate either for the communication failure model or our target environment. The Santoro-Widmayer impossibility applies for synchronous system, so further strengthening of timing assumptions is not possible. The failure detector approach is also elusive, mainly because failure detectors usually hide timing assumptions in their implementation. The collision detector approach of Chockler *et al.* (2005) does not use time, but instead makes simplistic assumptions about the environment. Namely, that message omissions are only due to collisions and that these can be detected. Finally, the wormhole approach may not be appropriate for wireless ad hoc networks. Assuming a hybrid model where some subpart of the system provides stronger reliability properties is very difficult to obtain in these environments.

With respect to randomization, the goal is to design protocols for the communication failure model that avoid the performance bottlenecks observed in Chapter 3. In order to achieve this goal, local coin protocols appear more appropriate for two reasons: (1) shared coin protocols involve cryptographic operations whose cost is usually prohibitive for mobile devices, and (2) the new model promises to cope much better with the communication costs involved because it reflects more closely the broadcasting nature of wireless environments.

The following material explores the problem of consensus under variations of this fault model, each one dealing with increasingly complex aspects of the model. The

next section presents a binary consensus algorithm that tolerates omission and corruption faults, however, it simplifies the problem by assuming a synchronous system and restricting the number of processes where faults may originate at any given step. Chapter 5 significantly extends this result by tolerating truly dynamic omission failures (i.e., no restrictions on the number of processes at the source of faulty transmissions), and presents the first randomized algorithm that achieves this. It does not handle, however, arbitrary failures. Chapter 6 introduces intrusion tolerance: it assumes an asynchronous model and presents an algorithm that tolerates a combination of dynamic omission failures and a static subset of Byzantine processes, thus achieving the desired goal of intrusion-tolerant consensus in wireless ad hoc networks.

4.2 Consensus Algorithm

This section represents a first step into the exploration of the communication failure model. It presents a randomized binary consensus algorithm that tolerates transmission faults of the omission and corruption types under a synchronous system.

In this initial approach, the problem is simplified by restricting the number of processes at any step in which transmission faults can originate. Since the set of processes which are the source of a transmission failure can be different from one broadcasting step to another, we say the protocol is resilient to dynamic process failures. Despite being a restricted version of the communication failure model, this model is still constrained by the Santoro-Widmayer impossibility result. The uncertainty captured by this model is similar to that of the FLP result: it is impossible to determine a process as permanently faulty. The fact that the system is synchronous does not bring any additional power in this respect. It is possible to detect failures, but since these are dynamic and transient in nature, they do not provide a reliable indication of a process being permanently faulty. In essence, it is impossible to distinguish a failed process from one whose transmissions are just temporarily faulty.

4.2.1 The Binary Consensus Problem

The binary consensus represents the most basic form of agreement (i.e., on a single bit of information) and it is used as a building block for other agreement and broadcast protocols (e.g., multi-valued consensus and atomic broadcast). In the protocol, each process p_i proposes a bit value $v_i \in \{0, 1\}$ and all processes decide on the same value $v \in \{0, 1\}$. Additionally, if all processes propose the same initial value v, then the decision has necessarily to be v. Since the algorithm is randomized, the termination property is formulated in a probabilistic way.

Formally, the properties of the protocol are defined as follows:

Validity. If all processes propose the same value v, then any process that decides, decides v.

Agreement. No two processes decide differently.

Termination. Every process eventually decides with probability 1.

4.2.2 System Model

The system is composed by a static set of n processes $\Pi = \{p_0, p_2, ..., p_{n-1}\}$ that exchange messages through two communication primitives: `broadcast` and `receive`. The primitive `broadcast(m)` transmits a message m to all processes in Π, including itself. The primitive `receive()` returns a set with the messages that arrived due to invocations of `broadcast` from the processes in Π.

The system is synchronous, meaning that both the processing times of processes and the communication delays are bound by known constants. Communication between processes occurs at synchronous steps. At each step, every process executes the following actions: (1) invokes `broadcast(m)`, (2) invokes `receive()`, and (3) performs a state transition based on its current state and the messages received so far. The messages are stored by each process p_i in a local set V_i. It is assumed that at most one message per step from each process p_j is stored at V_i.

A broadcast operation originates n transmissions of a message m, one for each process in the system. A transmission of a message amongst a source process p_i and a destination process p_j is faulty if one of the following occurs:

Omission. The message m broadcast by p_i is not received by p_j.

Corruption. The message m broadcast by p_i is received by p_j with different content $m' \neq m$.

Faults can occur non-uniformly at a message broadcast, i.e., of the n transmissions originated by a broadcast operation some may be faulty and others not. The system can exhibit transmission faults of the omission and corruption types with the restriction that faults cannot *affect* more than f source processes per step, with $n \geq 3f + 1$. A fault is said to affect a source process p_i if it occurs at a transmission made by p_i. Message addition failures (i.e., some message is received when it was not sent by the source process) are not considered because it is assumed that processes have access to some message authentication mechanism which prevents spoofing of messages. The implementation of such mechanism is not taken into account for now.

Processes are modeled as not to exhibit faulty behavior, i.e., they correctly follow the protocols until termination. The notion of a faulty process is instead captured by the assumption of faulty message transmissions. For example, a process p_i whose communication system is malfunctioning is captured by omission faults originating at p_i. Similarly, a malicious process p_j that sends messages with different content to every other process is captured by corruption faults originating at p_j. In order to properly capture the malicious behavior of processes into the model, it is also assumed that integrity-checking mechanisms cannot detect this kind of message corruption. Otherwise, corruption faults could be easily converted into omission faults.

Finally, each process p_i has access to a local random bit generator through a function $\texttt{coin}_i\texttt{()}$ that returns unbiased bits.

4.2.3 The Algorithm

The protocol is presented in Algorithm 1. It runs in rounds of two steps each and takes as input the proposal value for the consensus execution. Every process $p_i \in \Pi$ starts the protocol by initializing both the round number r_i and step number s_i to 1, the decision value $decision_i$ to an undefined value $\perp$ indicating that no decision has been made yet, and the variable $stop_i$ to an undefined value $\perp$. This variable keeps the round number

where the algorithm should halt the execution. Variable v_i keeps the current proposal value, and V_i is a set storing the messages received so far (initialized as empty).

Each process p_i then enters step 1 of the protocol (lines 8-14). It broadcasts a message of the form $\langle i, r_i, s_i, v_i \rangle$, containing the process identifier i, the round number r_i, the step number s_i, and the proposal value v_i. It then receives the messages broadcast in the current step and stores them in V_i (lines 8-9). At line 10, it checks if there are more than $\frac{n+f}{2}$ messages with the same value v in V_i that were broadcast in the current round and step (with the $*$ indicating a wild card for the process identifier). If true, then v_i is set to v to indicate the preference value of p_i. Otherwise, v_i is set to $\perp$, which indicates a lack of preference (lines 10-14).

Processes then proceed to step 2 (lines 15-24). Each process p_i broadcasts a new message containing the updated proposal value v_i and saves the arriving messages in V_i (lines 15-16). If there are more than $\frac{n+f}{2}$ messages with the same value v in V_i, then p_i sets the decision value $decision_i$ to v and sets the variable $stop_i$ to indicate the algorithm should stop the execution at the next round (lines 17-20). If there are at least $f + 1$ messages with v (note that entering the previous condition implies entering this one), then the proposal value v_i is updated to v (lines 21-22). Otherwise, if none of the previous thresholds are observed, then v_i is set to a random value 1 or 0, each with probability $\frac{1}{2}$ (lines 23-24). This random step ensures that eventually there will be a round where every process proposes the same value, in which case it is easy to see that the algorithm reaches a decision by the end of that round.

Finally, the algorithm checks if it should stop the execution by comparing the current round number r_i with the $stop_i$ variable (lines 25-26). If the numbers match, the algorithm halts, otherwise it increments the current round number, sets the step number to 1 (lines 27-28), and continues for another round. The algorithm completes the execution exactly one round after making a decision. This means that an efficient implementation must not wait until the end of the protocol execution to output a decision value. It should output the decision value immediately when one is available and then continue the execution for an extra round.

Algorithm 1: Binary Consensus Algorithm

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

$r_i \leftarrow 1$; `// round number;`
$s_i \leftarrow 1$; `// step number;`
$stop_i \leftarrow \perp$; `// round number where execution halts;`
$decision_i \leftarrow \perp$;
$v_i \leftarrow proposal_i$;
$V_i \leftarrow \emptyset$;
while do
  `broadcast(`$\langle i, r_i, s_i, v_i \rangle$`);` `// step 1;`
  $V_i \leftarrow$ `receive();`
  if $\exists v \in \{0, 1\} : |\{\langle *, r, s, v \rangle \in V_i\}| > \frac{n+f}{2}$ **then**
    $v_i \leftarrow v$;
  else
    $v_i \leftarrow \perp$;
  $s_i \leftarrow 2$;
  `broadcast(`$\langle i, r_i, s_i, v_i \rangle$`);` `// step 2;`
  $V_i \leftarrow$ `receive();`
  if $\exists v \in \{0, 1\} : |\{\langle *, r, s, v \rangle \in V_i\}| > \frac{n+f}{2}$ **then**
    if $decision_i = \perp$ **then**
      $decision_i \leftarrow v$;
      $stop_i \leftarrow r_i + 1$;
  if $\exists v \in \{0, 1\} : |\{\langle *, r, s, v \rangle \in V_i\}| \geq f + 1$ **then**
    $v_i \leftarrow v$;
  else
    $v_i \leftarrow$ `coin`$_i$`();`
  if $r_i = stop_i$ **then**
    `halt();` `// stop execution;`
  $r_i \leftarrow r_i + 1$;
  $s_i \leftarrow 1$;

4.2.4 Correctness Proof

This section shows that the above protocol ensures the three properties from Section 4.2.1. The proof is supported by Lemmas 3 and 5, which essentially demonstrate that the messages processes receive at each step can differ from one another in a limited way. The agreement property (Theorem 9) is further supported by Lemma 8. Lemmas 10 and 11 specifically support the termination property (Theorem 12).

Definition 1. Let Ω be the set of messages broadcast by all processes in Π at some round r and step s, with $|\Omega| = n$.

Definition 2. Let V_i be the set of messages received by a process p_i at some round r and step s, with $n - f \leq |V_i| \leq n$.

Lemma 3. *For any process p_i at any round r and step s, $|\Omega \cap V_i| \geq n - f$ is true.*

Proof. If no transmission failures occur, then $\Omega = V_i$, i.e., the set of broadcasted messages is equal to the set of received messages. Since, according to the system model, transmission failures can originate, at most, from f processes per step, then at most f messages in V_i are subject to transmission failures (i.e., not received or received with different content). Therefore, the sets Ω and V_i have to contain at least $n - f$ messages in common. Hence, $|\Omega \cap V_i| \geq n - f$. □

Corollary 4. *For any process p_i at some round r and step s, set V_i has at most f elements not contained in set Ω. Hence, $|V_i \backslash \Omega| \leq f$.*

Lemma 5. *Let V_i and V_j be the sets of messages received at some round r and step s by processes p_i and p_j, respectively, with $i \neq j$. Then, $|V_i \cap V_j| \geq n - f$.*

Proof. According to the system model, transmission failures can originate, at most, from the same f processes per step. Therefore, any two sets V_i and V_j have to include at least the same $n - f$ elements, i.e., the messages not subject to transmission failures. It follows that $|V_i \cap V_j| \geq n - f$ must be true. □

Corollary 6. *For any two processes p_j and p_i at some round r and step s, set V_j has at most f elements not contained in set V_i. Hence, $|V_j \backslash V_i| \leq f$.*

Theorem 7. *If all processes propose the same value v, then any process that decides, decides v.*

Proof. Let V_i be the set of messages received by a process p_i at round 1, step 1. Since all processes propose the same value v, then, by definition, all messages in Ω have the same value v. According to Lemma 3, at least $n - f > \frac{n+f}{2}$ messages in Ω are also in V_i. Therefore, V_i includes at least $n - f$ messages with value v. This means that all processes execute line 11 of the algorithm, setting $v_i = v$. A trivial inspection of the protocol shows that a similar reasoning applies to step 2. Here, since a process receives at least $n - f > \frac{n+f}{2}$ messages with the same value v, then it must decide v. □

Lemma 8. *If some process p_i decides v at round r, then any process either broadcasts v or $\perp$ at step 2 of round r.*

Proof. For a process p_i to decide on a value v at round r, it means that it must have received more than $\frac{n+f}{2}$ messages with value $v \neq \perp$ at step 2 of round r (lines 17-23). This implies that some processes broadcasted value $v \neq \perp$ at step 2 (line 15), and, hence, they must have set their proposal value to v at step 1 of round r (line 11). For a process p_i to set its proposal value to $v \neq \perp$ at step 1, it must receive more than $\frac{n+f}{2}$ messages with v (lines 9-11). Consequently, no other process can receive more than $\frac{n+f}{2}$ messages with value $v' \neq v$ at step 1. It follows that, at step 1, a process can only set its proposal value to v or $\perp$. Thus, any process either broadcasts v or $\perp$ at step 2 of round r. □

Theorem 9. *No two processes decide differently.*

Proof. The proof assumes that some process p_i decides on a value v at some round r, and proceeds to show that no other process can decide on a value $v' \neq v$. A process p_i

decides on a value v if it receives more than $\frac{n+f}{2}$ messages with value $v \neq \perp$ at step 2 of round r (lines 17-20). Additionally, by Corollary 6, every other process receives more than $\frac{n+f}{2} - f = \frac{n-f}{2} > f$ messages with value v at step 2 of the same round. By Lemma 8, since p_i decides at round r (line 19), then any process either broadcasts v or $\perp$ at step 2 of round r (line 15). This implies, by Corollary 4, that no process receives more than f messages with value $v' \neq v$, where $v' \in \{0, 1\}$, at step 2 of round r (line 17). Consequently, the only value (not $\perp$) for which every process receives $f + 1$ or more messages is v. Therefore, every process at step 2 of round r sets its proposal value to v (line 22), and propose the same value v at the beginning of step 1 of round $r + 1$. According to Theorem 7, any remaining process that needs to decide will also chose v. Thus, no two processes decide differently. □

Lemma 10. *At step 2 of any round r, if a process p_i sets its proposal value to v without resorting to a coin flip, then no other process sets its proposal value to $v' \neq v$ unless it resorts to a coin flip.*

Proof. At step 1, a process p_i sets its proposal value to $v \neq \perp$ if it receives more than $\frac{n+f}{2}$ messages with v (lines 10-11). Consequently, no other process can receive more than $\frac{n+f}{2}$ messages with $v' \neq v$, where $v' \in \{0, 1\}$. Therefore, no other process sets its proposal value to v'. This implies that for step 2 every process broadcasts the same value v or $\perp$ (line 15) and, based on Corollary 4, no process receives more than f messages with value v'. Thus, no process sets its proposal value at step 2 to $v' \neq v$ unless it resorts to a coin flip (line 24) because, otherwise, at least $f + 1$ messages with a value v' had to be received at step 2, which is a contradiction. □

Lemma 11. *If every process proposes the same value v at the beginning of some round r, then every process decides v by the end of round r.*

Proof. Since all processes propose the same value v at the beginning of round r, then, by Lemma 3, all processes receive at least $n - f > \frac{n+f}{2}$ messages with v at step 1.

Therefore, all processes propose v for step 2 because they all executed line 11. This implies that, again by Lemma 3, all processes receive at least $n - f > \frac{n+f}{2}$ messages with v at step 2. Consequently, all processes decide v at line 19. □

Theorem 12. *Every process eventually decides with probability 1.*

Proof. According to Lemma 11, if all processes propose the same value v at the beginning of a round r, then all processes decide v by the end of that round. The proof consists of showing that eventually all processes propose the same value v.

At step 2 of any round r, processes set their proposal values to be broadcasted at the beginning of round $r + 1$. This value can be set deterministically (lines 21-22) or randomly (lines 23-24). At any round r, let $D(r)$ be the set of processes that set their value deterministically and $R(r)$ the set of processes that set their value randomly. By Lemma 10, all processes in $D(r)$ set their proposal to the same value v. Therefore, with probability $2^{-|R(r)|}$, all processes in $R(r)$ set their proposal value to v at the beginning of round $r + 1$. In a worst-case scenario, where $|R(r)| = n$, the probability of the processes not proposing the same value at the beginning of a round is $1 - 2^{-n}$. Thus, the probability P that all processes set their proposal to the same value v within r rounds is at worst $P = 1 - (1 - 2^{-n})^r$. Thus, $\lim_{r\to\infty} P = 1$, meaning that eventually all processes propose the same value v at the beginning of some round with probability 1. □

Chapter 5

Consensus with Dynamic Omission Failures

This chapter presents a randomized binary k-*consensus* algorithm that tolerates dynamic omission transmission faults in both a practical and efficient way. The algorithm allows at least k processes to decide on a common binary value in a system with n processes such that $k > \frac{n}{2}$. It is the first algorithm to circumvent the Santoro-Widmayer impossibility result without restricting the pattern of faults (unlike other approaches, e.g., Section 4.2.1; Biely *et al.* (2007); Schmid *et al.* (2009)). The safety properties of consensus (i.e., validity and agreement) are ensured even with an unrestricted number of faults, while the liveness property (i.e., termination) is ensured if the number of faults per round is $\sigma \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$. Termination is achieved with probability 1 when communication becomes stable, i.e., when the threshold above is satisfied.

As stated before, the considered model focuses on the effects of faults (e.g., message omissions), rather than their source. As a consequence, it does not take into account the explicit failure of processes. The actual crashing of a process results simply in a certain pattern of message omissions. It is the latter that is captured by our model. The notion of faulty process is, however, implicitly present in the parameter k. Since k processes have to decide, $n-k$ processes can, in practice, be crashed. Naturally, the algorithm tolerates this behavior and makes progress as long as the number of message omissions does not exceed σ.

From a theoretical point of view, the main contribution of the algorithm is a new upper bound for the number of omission transmission faults. It has, however, additional characteristics that make it very interesting for practical deployment in wireless ad hoc networks. It allows an efficient utilization of the broadcasting medium and, at the same time, ensures safety under arbitrary message loss, regardless of its cause. Furthermore, the algorithm is efficient in the sense that it is *fast-learning* (Lamport, 2006), i.e., it terminates in 2 communication steps under favorable conditions (i.e., with no message losses, benign patterns of message losses, and/or all processes having the same initial value).

The remainder of the chapter is organized as follows: Section 5.1 formalizes the k-*consensus* problem, and the next section presents the system model. Section 5.3 describes the algorithm, and the correctness proofs are provided in the following section. Finally, Section 5.5 discusses some extensions to the algorithm.

5.1 The k-Consensus Problem

The k-*consensus* problem considers a set of n processes where each process p_i proposes a binary value $v_i \in \{0, 1\}$, and at least $k > \frac{n}{2}$ of them have to decide on a common value proposed by one of the processes. The remaining $n - k$ processes 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 processes propose the same value v, then any process that decides, decides v.

Agreement. No two processes decide differently.

Termination. At least k processes eventually decide with probability 1.

5.2 System Model

The system is composed by a fixed set of n processes $\Pi = \{p_0, p_1, ..., p_{n-1}\}$, where each process p_i has an unique identifier i. The timing model is assumed to be syn-

chronous. As previously explained, this implies that (1) there is a known upper bound on time required by a process to execute a step, (2) there is a known upper bound on message transmission delays, and (3) every process has a local clock with a known bounded rate of drift with respect to real-time.

The communication between processes proceeds in synchronous rounds. At each round, every process $p_i \in \Pi$ executes the following actions: (1) transmits a message m to every process $p_j \in \Pi$, including itself, by invoking `broadcast(`m`)`, (2) receives the messages broadcast in the current round by invoking `receive()`, and (3) performs a local computation based on its current state and the set of messages received so far. It is assumed that a broadcast operation generates n transmissions, one for each process in Π. This arises from the necessity of capturing the possibility of non-uniform message delivery by the processes. Of course, in practice, this operation can still be implemented efficiently by transmitting only a single message.

Processes do not to exhibit faulty behavior, i.e., they correctly follow the protocol until termination. The notion of a faulty process is instead captured by the assumption of faulty message transmissions. For example, a crashed process can be expressed by the loss of every message transmitted by it. The model considers only omission transmission failures. A transmission between two processes p_i and p_j is subject to an omission failure if the message sent by p_i is not received by p_j.

In rounds where omission faults are bounded by $\sigma \leq \lceil \frac{n}{2} \rceil (n - k) + k - 2$ out of the n^2 transmissions that occur (where k is the number of processes required to decide), the protocol necessarily makes some progress that eventually leads to a decision. Therefore, if enough of these rounds occur, then the protocol ensures termination with probability 1. Nevertheless, to simplify the correctness proofs we will assume that there is some unknown time after which at most σ faulty transmissions occur at each round. The number of faults per round prior to this is unrestricted and can for instance match the total number of transmissions.

Finally, every process $p_i \in \Pi$ has access to a local random bit generator acessible via a function $\texttt{coin}_i\texttt{()}$ that returns unbiased bits observable only by p_i.

5.3 The Algorithm

This section presents a k-consensus algorithm (Algorithm 2). The algorithm is tolerant to omission faults and relies on each process p_i having access to a *local coin*mechanism that returns random bits observable only by p_i (e.g., Ben-Or (1983); Bracha (1984)). Safety (i.e., the *validity* and *agreement* properties of consensus) is ensured by the algorithm regardless of the number of omission faults that occur per round, while liveness (i.e., the *termination* property) is ensured if, after some arbitrary number of rounds, the number of omission faults per round does not exceed the threshold $\sigma \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$.

The internal state of a process p_i is 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 process starts the execution with $\phi_i = 1$, $status_i = undecided$, while v_i is set to the initial proposal value indicated by the input register $proposal_i$.

A round of the algorithm is executed as follows. Upon every clock tick (line 5), each process p_i broadcasts a message of the form $\langle i, \phi_i, v_i, status_i \rangle$ containing its identifier and the variables that comprise the internal state, and receives the messages broadcast by all processes (lines 6-7). Some of the messages that a process is supposed to receive may be lost. The messages that a process p_i receives at any round are accumulated in a set V_i (line 8). Based on its current internal state and the messages accumulated so far in set V_i, each process p_i performs a state transition (i.e., modifies ϕ_i, v_i or $status_i$).

Before continuing the explanation of the state transition, it is important to note the distinction between round and phase. The term *round* pertains to a periodic execution of the protocol activated by a synchronous event, a clock tick in this case. The term *phase* pertains to a monotonic variable ϕ_i that is part of the internal state of a process p_i, and whose value increases as p_i accumulates messages of a certain form in set V_i. How exactly ϕ_i is updated is explained below. For now, it is beneficial to retain that for any given round, any two processes p_i and p_j can have different phase values $\phi_i \neq \phi_j$.

A process p_i performs a state transition when one of two conditions occur:

1. The set V_i holds one message from some process p_j whose phase ϕ_j is *higher* than the phase ϕ_i of p_i.

Algorithm 2: k-consensus algorithm

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

$\phi_i \leftarrow 1$;
$v_i \leftarrow proposal_i$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;
for each *clock tick* **do**
  broadcast($\langle i, \phi_i, v_i, status_i \rangle$);
  $M_i \leftarrow$ receive();
  $V_i \leftarrow V_i \cup M_i$;
  while $\exists_{\langle *, \phi, v, status \rangle \in V_i} : \phi > \phi_i$ **do**
    $\phi_i \leftarrow \phi$;
    $v_i \leftarrow v$;
    $status_i \leftarrow status$;
  end
  if $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
    if $\phi_i \bmod 2 = 1$ **then** `/* odd phase */`
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow \perp$;
      end
    else `/* even phase */`
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
        $status_i \leftarrow decided$;
      end
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| \geq 1$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow$ `coin`$_i$`()`;
      end
    end
    $\phi_i \leftarrow \phi_i + 1$;
  end
  if $status_i = decided$ **then**
    $decision_i \leftarrow v_i$;
  end
end

2. The set V_i holds more than $\frac{n}{2}$ messages whose phase is *equal* to the phase ϕ_i of p_i.

The first case is straightforward (lines 9-13). As long as there is some message in V_i with a higher phase than that of p_i, the loop iterates until the state of p_i is matched with the state of the message with the highest phase value.

The second case involves the execution of more steps (lines 14-32). The way a process p_i updates its state depends on whether the current number of its phase ϕ_i is odd (i.e., $\phi_i \bmod 2 = 1$) or even (i.e., $\phi_i \bmod 2 = 0$). The odd phase essentially guarantees that if two processes set their proposal to a value 1 or 0, they do it for the same value. The even phase is where a process decides if it learns that a majority of processes have the same proposal value.

If $\phi_i \bmod 2 = 1$ (lines 15-20), then the proposal value v_i is updated in the following way: if there are more than $\frac{n}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with $\phi = \phi_i$ and the same value $v \in \{0, 1\}$, then v_i is set to v (lines 16-17). Otherwise, it is set to a special value $\perp \notin \{0, 1\}$ indicating a lack of preference (lines 18-19).

If $\phi_i \bmod 2 = 0$ (lines 21-30), then the process sets $status_i$ to $decided$ if there are more than $\frac{n}{2}$ messages of the form $\langle *, \phi, v, * \rangle$ in V_i with the same value $v \neq \perp$ and $\phi = \phi_i$ (lines 22-24). The proposal value v_i is updated to v if there is at least one message of the form $\langle \phi, v, * \rangle$ in V_i with a value $v \neq \perp$ and $\phi = \phi_i$. Otherwise, v_i is set to the value of function $\texttt{coin}_i\texttt{()}$, which returns a random number 0 or 1, each with a probability $\frac{1}{2}$ (lines 25-29). Regardless of whether the phase ϕ_i is odd or even, its value is always incremented by one unit (line 31).

At the end of each round, a process 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 33-35). Any further accesses to this variable do not alter its value. Hence, they have no impact on the correctness of the algorithm.

5.4 Correctness Proof

The correctness proof of the algorithm is divided in two subsections. Section 5.4.1 is concerned with the safety properties of the algorithm: *validity* and *agreement*. Section 5.4.2 proves the liveness property: *termination*.

5.4.1 Safety

The *validity* and *agreement* properties are proven on the assumption that the system might be subject to an unbounded number of transmission omission faults per round.

The first lemma shows that if there is some phase where every process has the same proposal value, then at any subsequent phase they all have that same proposal value. This supports Lemma 14, which essentially states that whenever there is an odd phase ϕ where every process has the same proposal value, every process that reaches a phase higher than $\phi + 1$ decides on that value. With these two lemmas, the validity property is easily proven (Theorem 15).

Lemma 13. *If every process p_i with phase value $\phi_i = \phi$ has the same proposal value $v_i = v \neq \perp$, then every process p_j that sets $\phi_j = \phi + 1$ also sets $v_j = v$.*

Proof. The lemma is going to be proven by induction on the number of processes that reach phase $\phi + 1$. *Basis:* Without loss of generality, let p_1 be the first process that sets $\phi_1 = \phi + 1$. In this case, process p_1 must have received more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ (Line 14). Since every process p_i with $\phi_i = \phi$ has the same value $v_i = v \neq \perp$, every broadcast message of the form $\langle *, \phi, *, * \rangle$ carries the same proposal value v (Line 6). This implies that the more than $\frac{n}{2}$ messages received by process p_1 have the form $\langle *, \phi, *, * \rangle$ with the same value v. Therefore, p_1 must set its proposal value to v (either on Line 17 or 26). *Inductive step:* Assume that every process p_u with $1 \leq u \leq j - 1$ has $\phi_u = \phi + 1$ and $v_u = v$. Now we want to demonstrate that when p_j sets $\phi_j = \phi + 1$ it will also set $v_j = v$. In order for process p_j to set $\phi_j = \phi + 1$ it must have in set V_j (1) more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ (Line 14) or (2) at least a message of the form $\langle *, \phi + 1, *, * \rangle$ (Line 9). Condition (1) corresponds to the basis case, and therefore it has already been shown that p_j sets $v_j = v$. Condition (2) also results in the same outcome, since by hypothesis message $\langle *, \phi + 1, *, * \rangle$ must have been transmitted by one of the p_u processes, and therefore p_j also sets $\phi_j = \phi + 1$ and $v_j = v$ (Lines 10-11). □

Lemma 14. *Let ϕ be some odd phase (i.e., $\phi \bmod 2 = 1$). If every process with phase value ϕ has the same proposal value v, then every process that sets its phase to any value $\phi' > \phi + 1$ decides v.*

Proof. Since every process with odd phase value ϕ has the same proposal value v, by Lemma 13, every process that reaches even phase $\phi + 1$ also has proposal value v (either on Lines 10-11 or Lines 17 and 31). Let p_i be the first process to set phase value $\phi_i = \phi + 2$. Since there is no other process p_j with phase value $\phi_j > \phi + 1$, the only way for p_i to go from phase $\phi + 1$ to $\phi + 2$ is to receive more than $\frac{n}{2}$ messages of the form $\langle *, \phi + 1, *, * \rangle$ (Line 14). Since $\phi + 1$ is even and all these messages carry the same proposal value v, this implies that p_i sets $status_i = decided$, $v_i = v$ and $\phi_i = \phi + 2$ (Lines 23, 26, 31). Consequently, process p_i can now decide v (Line 34).

The next process that sets its phase value to $\phi + 2$ also decides v because it either accumulates more than $\frac{n}{2}$ messages with phase value $\phi + 1$ and same proposal value v (Lines 23, 26, 31 and 34), or receives a message from p_i of the form $\langle *, \phi + 2, v, decided \rangle$ (Lines 10-12 and 34). This reasoning can be applied recursively to any other process that sets its phase value to $\phi + 2$. It follows that any process that sets its phase value to $\phi' \geq \phi + 2$ must either had been at phase $\phi + 2$, and hence decided, or it must have received some message from a process that went through phase $\phi + 2$, and thus also decided. Therefore, every process that sets its phase to any value $\phi' > \phi + 1$ decides v. □

Theorem 15. *If all processes propose the same value v, then every process that decides, decides v.*

Proof. If every process has the same initial proposal value v, then they all and set proposal value to v in odd phase 1 (Lines 1-2). Therefore, by Lemma 13, every process p_j that sets phase $\phi_j = 2$ also has proposal value $v_j = v$. Moreover, by Lemma 14, every process p_i that sets its phase to $\phi_i > 2$, decides v. □

The following lemma proves that any two processes in the same even phase can not have different proposal values 0 and 1. In other words, if some process has proposal value $v \in \{0, 1\}$, then any other process can only have a proposal value of v or $\perp$. This lemma is essential to the *agreement* property in Theorem 17, which shows that when the first process decides, every process proposes the same value at the following phase. By lemma 14, this inevitably leads to a decision on the same value.

Lemma 16. *In some even phase ϕ, there are no two process p_i and p_j that receive messages of the form $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$, respectively.*

Proof. Suppose otherwise. Then p_i and p_j are two processes with phase value ϕ that, respectively, receive a message $\langle *, \phi, 0, * \rangle$ from p_u and a message $\langle *, \phi, 1, * \rangle$ from p_w. This implies that process p_u has set $v_u = 0$ either because on odd phase $\phi - 1$ it accumulated more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, 0, * \rangle$ (Lines 16-17, 31), or because it received a message $\langle *, \phi, 0, * \rangle$ (Lines 10-11) from a process that had accumulated that majority of $\langle *, \phi - 1, 0, * \rangle$ messages. Using a similar reasoning, in order for process p_w to have set $v_w = 1$, some process must have received on odd phase $\phi - 1$ more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, 1, * \rangle$. But this is a contradiction because only one of the proposal values 0 and 1 can be in a majority of the messages broadcast for any particular phase number. □

Theorem 17. *No two processes decide differently.*

Proof. Let p_i be the first process to decide, and do so when phase $\phi_i = \phi$ (Line 34). Without loss of generality, let the decision value be 1. Then, set V_i must contain more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, 1, undecided \rangle$, and $\phi - 1$ must be even (to allow the execution of Lines 23, 26, and 31). By Lemma 16, no other process p_j can receive a message of the form $\langle *, \phi - 1, 0, * \rangle$. Therefore, every other process p_j with phase $\phi_j = \phi$ has proposal value $v_j = 1$ either because it accumulates more than $\frac{n}{2}$ messages with at least one being of the form $\langle *, \phi - 1, 1, * \rangle$ (Line 26), or because it receives a message $\langle *, \phi, 1, * \rangle$ (Line 11) transmitted by process p_i (or another process that sets

its proposal value to 1). Additionally, since all processes with phase ϕ have proposal value 1, then by Lemmas 13 and 14, every process that decides in phase $\phi' > \phi$ will do it for value 1. □

5.4.2 Liveness

The remainder of the proof handles the *termination* property of consensus (Theorem 23). For this part we work on the assumption that the message scheduling falls under the control of an *adversary* that can cause no more than σ faults per round, with $\sigma \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$.

The rationale for the *termination* property is based on the idea that as long as processes keep increasing their phase values, a decision is eventually reached. As we have seen from the safety proofs, if there is unanimity at some odd phase ϕ, then every process that reaches any phase higher than $\phi + 1$ decides. The idea is to show that k processes can reach any arbitrarily high phase value, and that unanimity eventually happens.

The following two lemmas dictate how processes increment their phase values in tandem. Lemma 19, in particular, states that for any process with phase value ϕ, eventually k processes must have a phase value equal or higher than $\phi - 1$.

Lemma 18. *If some process p_i has some phase value $\phi_i > 1$, then there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi_i - 1$ and $|S| > \frac{n}{2}$.*

Proof. Given a phase number $\phi > 1$, then there must be some process p_i that is the first to set its phase to $\phi_i = \phi$. In order to do this, p_i must have more than $\frac{n}{2}$ messages of the form $\langle *, \phi - 1, *, * \rangle$ in set V_i (Line 14). It follows that there are more than $\frac{n}{2}$ processes that were at some point in time in phase $\phi - 1$. □

Lemma 19. *If some process p_i has phase value $\phi_i = \phi$, then eventually there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi - 1$ and $|S| \geq k$.*

Proof. Suppose otherwise. By Lemma 18, if some process p_i has $\phi_i = \phi > 1$, then there is a set of processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi - 1$ and $|S| > \frac{n}{2}$. Let $R^+ = S$

where $\frac{n}{2} < |R^+| < k$, and R^- be the set of remaining processes, i.e., $\forall_{p_u \in R^-} : \phi_u < \phi - 1$ where $n - k < |R^-| < \frac{n}{2}$.

By assumption, the adversary can create at most $\sigma = \sigma_1 + \sigma_2$ message omissions per round, where $\sigma_1 = \lceil \frac{n}{2} \rceil (n-k)$ and $\sigma_2 = k-2$. In order to prevent processes in R^- from reaching $\phi_u \geq \phi - 1$, the adversary must omit every message from processes of R^+ to R^- (due to Lines 9-13). This implies the elimination of more than $\frac{n}{2}$ messages in more than $n-k$ processes because $|R^+| > \frac{n}{2}$ and $|R^-| > n-k$. It is clear that after consuming σ_1 faults, there are at most $n-k$ processes in R^- that do not receive any message from R^+.

Since by definition $|R^-| - (n-k) = k - |R^+| > 0$, there must be $k - |R^+|$ processes in R^- that could still receive messages from every process in R^+. Let R^-_* denote the set of processes in this situation. To prevent every process p_u in R^-_* from reaching $\phi_u \geq \phi - 1$, the adversary must create $|R^+||R^-_*|$ omissions, where $|R^+| + |R^-_*| = k$. However, the adversary only has $\sigma_2 = k - 2 = |R^+| + |R^-_*| - 2$ faults available. This creates a contradiction because $|R^+||R^-_*| > |R^+| + |R^-_*| - 2$, for all $|R^+| \geq 1$ and $|R^-_*| \geq 1$. This implies that some process in $|R^-|$ always increases its phase value when $\frac{n}{2} < |R^+| < k$. □

The following lemma is central to this part of the proof. It shows that in any communication round where the number of omission faults is not higher than $\lceil \frac{n}{2} \rceil (n - k) + k - 2$, some process increases its phase value.

Lemma 20. *Let R^+ represent a set of processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi$, with $|R^+| = k + \alpha$ and $0 \leq \alpha \leq n - k$. Let α or more processes in R^+ have phase ϕ and the remaining processes of R^+ have phase $\phi + 1$. Let R^- be the set of process such that $\forall_{p_j \in R^-} : \phi_j < \phi$, with $|R^-| = n - k - \alpha$. Whenever a round has such configuration, some process increases its phase value.*

Proof. Suppose otherwise. Then, under the lemma conditions, there must be a message schedule where at some round no process increases its phase value.

In order to prevent every process in R^- from increasing its phase value, the adversary must omit every message from R^+ to R^- (due to Lines 9-13). This requires that $|R^+||R^-|$ faults must be spent. Since $|R^+||R^-| = (k+\alpha)(n-k-\alpha)$ and the total number of omissions per round is $\sigma = \lceil \frac{n}{2} \rceil (n-k) + k - 2$, then the adversary is left with no more than $\sigma - |R^+||R^-| \leq (\alpha + \lceil \frac{n}{2} \rceil + k - n)\alpha + k - 2$ faults.

In order to block each of the α processes in R^+ with phase ϕ, the adversary must omit all messages from processes in R^+ with phase $\phi + 1$ (Line 9) and it must prevent the reception of more than $\frac{n}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ also from processes in R^+ (Line 14). This implies that each of the α processes with phase ϕ can receive the $n - k - \alpha$ messages from processes in R^- and at most $\lfloor \frac{n}{2} \rfloor$ messages from processes in R^+. Therefore, the adversary must create at least $\left[n - (\lfloor \frac{n}{2} \rfloor + n - k - \alpha)\right] \alpha$ faults to stop the progression of the α processes. Since $\left[n - (\lfloor \frac{n}{2} \rfloor + n - k - \alpha)\right] \alpha = (\alpha + \lceil \frac{n}{2} \rceil + k - n)\alpha$, the adversary is left with no more than $k - 2$ faults.

For the remaining k processes in R^+, there are two possible cases:

1. First consider the two extreme situations, where all k processes either have phase value ϕ or $\phi + 1$. Since the adversary only has $k - 2$ faults left, some process has to receive more than $\frac{n}{2}$ messages with the same phase ϕ or $\phi + 1$. Therefore, some process increases its phase value (Line 14).

2. Second consider that some of the k processes have phase value $\phi + 1$ and the others have phase value ϕ. Let H be the set of processes with $\phi + 1$ and L the set of processes with ϕ, such that $|H| + |L| = k$. To block the processes in L, the adversary has to omit $|H||L|$ messages (due to Line 9). Since the adversary only has $k - 2 = |H| + |L| - 2$ faults left, it can not prevent some process from increasing its phase because $|H||L| > |H| + |L| - 2$ for all $|H| \geq 1$ and $|L| \geq 1$.

□

Lemma 21. *Let $\phi_{init} = 1$ be the initial phase value for all processes. Some process p_i eventually sets $\phi_i > \phi_{init}$.*

Proof. If every process has the same phase value ϕ_{init}, then according to the conditions of Lemma 20, this is equivalent to having every process in set R^+ with phase ϕ_{init}, such that $|R^+| = n$. Therefore, by Lemma 20, some process has to increase its phase value and set $\phi_i > \phi_{init}$. □

The following lemma concludes the progress of phase values by stating that some process can reach any arbitrarily high phase value.

Lemma 22. *If some process has phase value ϕ, then eventually some process must have phase value $\phi + 1$.*

Proof. If some process has phase value ϕ, then by Lemma 19, eventually there is a set R^+ of k or more processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi - 1$. This implies that the system must reach a configuration where there are two sets of processes R^+ and R^- according to the conditions of Lemma 20. When this happens, by the same lemma, some process will increase its phase. This process can be in one of three possible cases: (1) a process of R^-; (2) a process with phase number $\phi - 1$ of R^+; or (3) a process with phase number ϕ of R^+. The system configuration resulting from cases (1) and (2) falls under the conditions of Lemma 20, and therefore more processes will continue to increase their phase. Consequently, in the most extreme scenario, the system will evolve to a configuration where all process are in phase number ϕ, and case (3) will necessarily have to occur, and some process p_i will set its phase number to $\phi_i = \phi + 1$. □

Finally, the *termination* property is proven by showing that as long as processes keep increasing their phase value, then unanimity eventually happens and, consequently, a decision by k processes.

Theorem 23. *At least k processes eventually decide with probability 1.*

Proof. The proof is organized in two parts. First, we show that as messages are received, processes make progress on the protocol execution and continue to increase their phase number. Second, we demonstrate that due to this progression, eventually the system will reach to a configuration where at least k processes decide with probability 1.

First part: By Lemma 21, some process p_i eventually increases its phase number from the initial phase number, i.e., $\phi_i = \phi > \phi_{init}$. Then, by Lemma 22, some process will eventually set its phase number to $\phi + 1$. Moreover, by Lemma 19, k or more processes set their phase value to at least ϕ. Since these lemmas can be applied repeatedly, this ensures that for any arbitrary phase value ϕ', there is some subset S of at least k processes such that every process $p_j \in S$ eventually has phase value $\phi_j \geq \phi'$.

Second part: By Lemma 16, no two processes with the same even phase value ϕ can receive messages $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$. Therefore, any process p_i that enters the **if** condition of Line 14, and sets $\phi_i = \phi + 1$ (Line 31), must set its proposal value v_i either to a common value v (Line 26) or to a random value 1 or 0 (Lines 28). This implies that the probability of every process with odd phase value $\phi + 1$ having the same proposal value v is $p \geq 2^{-\gamma}$, where γ is the number of processes with phase $\phi + 1$.

As the protocol progresses, and the phase number of processes increases, the probability of not existing an odd phase where every process proposes the same value v is $\lim_{\phi \to \infty}(1-p)^{\phi} = 0$. Thus, eventually there will be an odd phase ϕ_t where every process p_i with phase $\phi_i = \phi_t$ has the same proposal value v. According to Lemma 14, every process p_i that sets its phase value to $\phi_i > \phi_t + 1$ decides v. By the discussion in the first part of the proof there is a subset S of k processes such that every process $p_j \in S$ eventually has phase value $\phi_j > \phi_t + 1$. Consequently, at least k processes eventually decide. □

5.5 Extensions to the Algorithm

The section discusses important extensions to the algorithm. It is divided in two subsections. The first discusses how the algorithm can be changed so the processes stop sending messages, and the third discusses practical performance considerations of the algorithm.

5.5.1 Quiescence

In the presented algorithm, processes do not voluntarily stop sending messages. The fact that the system stabilization time is unknown, combined with the assumed fault model, means that processes have no way of knowing when other processes have decided.

One possible solution to this limitation is by having the processes execute for an additional round after deciding, where the broadcast operation is performed through a reliable channel. Raynal & Roy (2005) showed that it is possible to implement reliable and asynchronous communication on top of an unreliable and synchronous model, and vice-versa. One can assume the presence of an asynchronous reliable channel that is judiciously used in such situations.

There is another practical solution, which is not guaranteed to achieve quiescence, but, in practice, is likely to work amongst processes that are not permanently disconnected from the system. The idea is to add a new status called $quiescent$. If a process process receives k messages of the form $\langle j, \phi, v, decided \rangle$, then it sets its status to $quiescent$ and stops periodically broadcasting messages, changing to message-triggered broadcasting rounds (i.e., it only starts a new round whenever some message whose status is not $quiescent$ is received). If a process receives a message of the form $\langle j, \phi, v, quiescent \rangle$, it sets its status to $quiescent$ (if it has not done so), and changes to message-triggered broadcasting rounds. The effect of this extension is that processes that are still periodically broadcasting (i.e., that have not yet achieved quiescence) trigger communication on processes that have already stopped sending messages on their own. Eventually, a process that has not yet achieved quiescence might receive a message with a $quiescent$ status, and achieve quiescence itself.

5.5.2 Performance

The algorithm guarantees the termination property of consensus in a probabilistic fashion. Since the execution of the algorithm may need to be extended for any number of rounds and any process may reach an arbitrarily high phase, eventually there will be a phase where all processes flip the same coin value v and decide (Theorem 23). The expected number of rounds for this to happen is $O(2^n)$ after the system stabilizes at the upper bound of f faults per round. There are other randomized protocols based on a shared coin constructed with cryptographic mechanisms that can achieve termination in polynomial time (Cachin *et al.*, 2000; Canetti & Rabin, 1993; Rabin, 1983).

Although the expected worst-case running time is exponential, a simple inspection of the algorithm suffices to observe that it is fast-learning, i.e., it can decide within two communication rounds in runs with no faults or with certain fault patterns. This is true even if processes have different initial proposal values. As long as k processes see the majority of one value during the first phase, they propose the same value for the second phase and then decide.

There are two optimizations, however, that can make the algorithm perform even better in practical scenarios. The first allows termination in one communication round, and the second improves the performance against certain fault patterns. These are discussed below and the modified algorithm is presented in Algorithm 3.

One-round Decision

The algorithm can be modified such that processes decide if they receive, for their current phase ϕ_i, n messages with the same value v. One-round decision, in particular, can be achieved if every process proposes the same initial value and every message is delivered to at least k processes in the first communication round, then k processes decide by the end of the round. It can be easily seen by Lemma 14 why this does not affect the correctness of the algorithm.

Three-step Variant

When the initial proposal values of the processes are not unanimous, there are certain fault patterns that can make Algorithm 2 terminate in the worst case expected number

of phases. The precise conditions for this to happen are: (1) the fault pattern is such that in every round there is no communication from one fixed subset of k processes to a fixed subset of the other $n - k$ processes, and (2) the subset of k processes does not have unanimous proposal values. Under this scenario the algorithm is forced into the worst expected number of phases until a decision is reached, which is $O(2^n)$.

This issue can be overcome by introducing an extra phase in the algorithm, following an approach similar to the local coin protocol of Bracha (1984). This way, phase $\phi \bmod 3 = 1$ is our new extra phase, phase $\phi \bmod 3 = 2$ is equal to the previous odd phase, and phase $\phi \bmod 3 = 0$ is equal to the previous even phase. In the new phase $\phi \bmod 3 = 1$, processes simply set their proposal value to the value present in the majority of messages that were received for ϕ. The majority can be biased for a predefined proposal value in order to overcome potential ties. This way, the set of k processes converges to the same proposal value in phase $\phi \bmod 3 = 1$, deciding by the end of phase $\phi + 2$ under the considered scenario. It has been demonstrated that the presence of an adversary that enforces a worst-case scheduling against this algorithmic construction is very unlikely to happen in practice (Moniz *et al.*, 2006a, 2010).

Algorithm 3: Optimized k-consensus algorithm

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

$\phi_i \leftarrow 1$;
$v_i \leftarrow proposal_i$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;
for each *clock tick* **do**
  broadcast($\langle i, \phi_i, v_i, status_i \rangle$);
  $M_i \leftarrow$ receive();
  $V_i \leftarrow V_i \cup M_i$;
  if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| = n$ **then** `/* early decision */`
    $status_i \leftarrow decided$;
  end
  while $\exists_{\langle \phi, v, status \rangle \in V_i} : \phi > \phi_i$ **do**
    $\phi_i \leftarrow \phi$;
    $v_i \leftarrow v$;
    $status_i \leftarrow status$;
  end
  if $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
    if $\phi_i \pmod 3 = 1$ **then** `/* phase` $\phi_i \pmod 3 = 1$ `*/`
      $v_i \leftarrow$ majority value v in messages with phase $\phi = \phi_i$;
    else if $\phi_i \pmod 3 = 2$ **then** `/* phase` $\phi_i \pmod 3 = 2$ `*/`
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow \perp$;
      end
    else `/* phase` $\phi_i \pmod 3 = 0$ `*/`
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n}{2}$ **then**
        $status_i \leftarrow decided$;
      end
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| \geq 1$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow$ `coin`$_i$`()`;
      end
    $\phi_i \leftarrow \phi_i + 1$;
  end
  if $status_i = decided$ **then**
    $decision_i \leftarrow v_i$;
  end
end

Chapter 6

Consensus with Byzantine Processes and Dynamic Omission Failures

This chapter conciliates intrusion tolerance with the unreliable resource-constrained nature of ad hoc networks. As in previous chapters, it focuses on the problem of binary consensus for single-hop wireless ad hoc networks, assuming that nodes are subject to transitory disconnection, but also assumes that nodes may suffer from permanent corruption by a malicious entity. Furthermore, it assumes an asynchronous system, which is a fundamental assumption in order to resist attacks in the domain of time.

In more detail, the model is an asynchronous system composed of n ad hoc nodes where a subset t of them may be compromised by a malicious adversary (with $t < \frac{n}{3}$). Compromised nodes can fail in an arbitrary (or Byzantine) manner, namely by sending messages with erroneous content or by simply becoming silent. Therefore, it will be considered that potentially all transmissions from these nodes may be lost (or discarded), either due to network omission faults or bad behavior. Additionally, it will be assumed the existence of dynamic omission transmission faults that might affect the communications between correct nodes.

The consensus algorithm described in this chapter, named Turquois[1], is, to the best of our knowledge, the first consensus protocol to tolerate a combination of Byzan-

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

tine nodes and dynamic omission transmission faults. Furthermore, since the system is asynchronous, consensus is bound by the FLP and the Santoro-Widmayer impossibility results (Fischer *et al.*, 1985; Santoro & Widmeyer, 1989). Turquois is able to circumvent both impossibility results. Like the approaches of previous chapters, this is accomplished by resorting to randomization, ensuring termination with probability 1. The protocol ensures progress towards a decision in rounds where omission faults are $\sigma \leq \lceil \frac{n-f}{2} \rceil (n-k-f)+k-2$ (where k is the number of nodes required to decide and $f \leq t$ is the number of processes that are actually faulty). If a higher number of faults occurs, then the protocol always guarantees safety, but progress might be stopped until the network starts to behave better. Turquois employs a novel mechanism for broadcast message authentication that resorts to an inexpensive hashing operation instead of typical public-key cryptography. Therefore, through this mechanism, Turquois is capable of preserving the computational restrictions usually associated with mobile nodes and increasing efficiency.

The remainder of the chapter is organized in the following way. Section 6.1 formalizes the k-*consensus* problem with Byzantine processes, and the next section presents the system model. Section 6.3 describes the algorithm. Section 6.3.1 presents the message authentication and validation mechanism. Finally, the correctness proofs of the algorithm are provided in Section 6.4.

6.1 The k-Consensus Problem with Byzantine processes

The k-*consensus* with Byzantine processes considers a system composed by a set of n processes where $f \geq t$ can be Byzantine. Each process p_i proposes a binary value $v_i \in \{0, 1\}$, and at least k correct processes have to decide on a common value proposed by some of the correct processes (with $\frac{n+t}{2} < k \leq n - t$). The remaining non-Byzantine processes (at most $n - k - t$) 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 defined by the properties:

Validity. If all correct processes propose the same value v, then any correct process that decides, decides v.

Agreement. No two correct processes decide differently.

Termination. At least k correct processes eventually decide with probability 1.

6.2 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 process $p_i \in \Pi$ transmits a message m to every process $p_j \in \Pi$, including itself, by invoking `broadcast(`m`)`. A round r is defined as the r^{th} time that processes invoke the `broadcast()` primitive and is triggered by a clock tick local to each process.

The fault model assumes that up to $t < \frac{n}{3}$ processes can be Byzantine, and that these processes may fail in an arbitrary way. For example, a Byzantine process can become silent, send messages with wrong values, or collude with other Byzantine processes to disrupt the correct operation of the system. Such processes are said to be *faulty*, while processes that follow the algorithm are called *correct*. The actual number of faulty processes in the system is represented by f, where $f \leq t$.

The fault model also accommodates dynamic omission failures in message transmissions amongst correct processes. A transmission between two correct processes 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 processes is bounded by $\sigma \leq \lceil \frac{n-f}{2} \rceil (n-k-f)+k-2$. If a message m transmitted by process p_i to process p_j is not subject to a dynamic omission failure and both processes 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 process $p_i \in \Pi$ can call a local random bit generator to obtain unbiased bits observable only by p_i.

6.3 The Algorithm

The algorithm Turquois allows k processes out of n to reach consensus on a binary value $v \in \{0, 1\}$ (see Algorithm 4). Correctness in maintained as long as the number of faulty (Byzantine) processes f is bounded by $t < \frac{n}{3}$. 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 $\sigma \leq \lceil \frac{n-f}{2} \rceil (n - k - f) + k - 2$.

In the algorithm, each processes 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 process 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).

The algorithm is run in parallel by two exclusive tasks (only one of the tasks is allowed to advance from the respective **when** condition (lines 5 and 8)). Task T1 defines a broadcasting round and is activated periodically upon a local clock tick (lines 5-7). A process p_i broadcast 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 process is supposed to receive may be lost, or may carry invalid content if transmitted by a Byzantine process. Therefore, all arriving messages are subject to a validation procedure that constrains the wrongful actions of Byzantine processes. Essentially, a message is considered *valid* if it could have been sent by a process that followed the algorithm (details in Section 6.3.1). Valid messages are accumulated in a set V_i (line 9), and the others are discarded.

Based on its current internal state and the messages accumulated so far in set V_i, each process 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 *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.

Algorithm 4: Turquois: a Byzantine k-consensus algorithm

Input: Initial binary proposal value $proposal_i \in \{0, 1\}$
Output: Binary decision value $decision_i \in \{0, 1\}$

$\phi_i \leftarrow 1$;
$v_i \leftarrow proposal_i$;
$status_i \leftarrow undecided$;
$V_i \leftarrow \emptyset$;

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

TASK T2:
when $m = \langle j, \phi_j, v_j, status_j \rangle$ *is received* **do**
  $V_i \leftarrow V_i \cup \{m : m \text{ is } valid\}$;
  if $\exists_{\langle *, \phi, v, status \rangle \in V_i} : \phi > \phi_i$ **then**
    $\phi_i \leftarrow \phi$;
    if $\phi \pmod 3 = 1$ ***and*** v *is the result of a coin flip* **then**
      $v_i \leftarrow \texttt{coin}_i\texttt{()}$;
    else
      $v_i \leftarrow v$;
    end
    $status_i \leftarrow status$;
  end
  if $|\{\langle *, \phi, *, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ **then**
    if $\phi_i \pmod 3 = 1$ **then** /* phase $\phi_i \pmod 3 = 1$ */
      $v_i \leftarrow$ majority value v in messages with phase $\phi = \phi_i$;
    else if $\phi_i \pmod 3 = 2$ **then** /* phase $\phi_i \pmod 3 = 2$ */
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow \perp$;
      end
    else /* phase $\phi_i \pmod 3 = 0$ */
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| > \frac{n+f}{2}$ **then**
        $status_i \leftarrow decided$;
      end
      if $\exists_{v \in \{0,1\}} : |\{\langle *, \phi, v, * \rangle \in V_i : \phi = \phi_i\}| \geq 1$ **then**
        $v_i \leftarrow v$;
      else
        $v_i \leftarrow \texttt{coin}_i\texttt{()}$;
      end
    end
    $\phi_i \leftarrow \phi_i + 1$;
  end
  if $status_i = decided$ **then**
    $decision_i \leftarrow v_i$;
  end
end

The first case is simpler (lines 10-18). When the condition is met (line 10), process 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 6.3.1), then p_i executes a local coin flip to determine v_i (lines 12-13). Since it is not possible to force Byzantine processes into a fair coin flip, this step becomes necessary to guarantee that correct processes assume a random value.

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

If $\phi_i \pmod 2 = 2$ (lines 22-27), then the proposal value v_i is updated the following way: 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 process either proposes the same value $v \in \{0, 1\}$ or $\perp$. Furthermore, if there was unanimity amongst correct processes at the previous phase $\phi_i - 1$, then every process 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 process receives the same value $v \in \{0, 1\}$ in all valid messages and decides.

If $\phi_i \pmod 2 = 0$ (lines 28-37), then the process 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 updated 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 a 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 process 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.

6.3.1 Validation of Messages

A process 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 process can accomplish. There are two types of validation that a message must pass: authenticity validation and semantic validation. The first guarantees that of some of the fields of a message were actually generated by a process 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.

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 process p_j assurance that the values of ϕ and v originated at the alleged source process 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 impact the correctness of the protocol because our semantic validation mechanism (see next section) requires processes 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-shot 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 employment 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) (Menezes *et al.*, 1997). The key exchange procedure resorts to a more computationally expensive trapdoor one-way function F (e.g., RSA (Rivest *et al.*, 1978)) that

is used to sign an array of verification keys. It is assumed that each process 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 process knows the public key of all other processes.

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 keys are valid for m phases of the k-consensus protocol. If m is equal 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 process p_i, the key exchange $e \geq 1$ consists of the following steps. Process 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 $v \in \{0, 1, \perp\}$ [1]. 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 processes using an out-of-band reliable channel.

When VK_i arrives to a process, 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.

Message Authentication. For any phase ϕ, a message $\langle i, \phi, v, status \rangle$ broadcast by process p_i is authenticated by attaching $SK_i[\phi][v]$. When a process 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.

[1] 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.

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 process broadcasts the same value 0, then it is not possible for a process 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 process, and it can be discarded without impacting the protocol. In practice, this validation mechanism restricts the way that Byzantine processes 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 process receives a message, it sees if enough messages have arrived to justify the values carried by the message just received. For example, if a process 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 process 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 process 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. Next, we explain in more detail how to perform the validations.

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.

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.
- *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 $\perp$, 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 $\langle *, \phi - 2, 1, * \rangle$.
- *Messages with phase* $\phi \pmod 3 = 1$: The validity of proposal value v in these messages depends if it was obtained deterministically (lines 32-33) or randomly (lines 34-35). 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 $\langle *, \phi - 1, \perp, * \rangle$.

Status value. For the $status$ variable, any message with phase $\phi \leq 3$ must necessarily carry value $undecided$ because no process can decide prior to phase 3. 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.

6.4 Correctness Proof

The correctness proof of the algorithm is divided in two parts. Up to Theorem 29 we are concerned with the safety properties of the algorithm: *validity* and *agreement*. From Lemma 30 on, we are concerned with the liveness property: *termination*.

The *validity* and *agreement* properties are proven on the assumption that the system might be subject to an unbounded number of transmission omission faults per round.

The first two lemmas show that if there is some phase $\phi \pmod 3 = 1$ where every process has the same proposal value, then at any subsequent phase they all have that same proposal value. With these two lemmas, the validity property is easily proven (Theorem 26).

Lemma 24. *Let ϕ^u be some phase such that $\phi^u \pmod 3 = 1$ and any correct process p_i with $\phi_i = \phi^u$ has the same proposal value $v_i = v$. No process p_j can produce a valid message of the form $\langle j, \phi^u + 1, v', * \rangle$, $\langle j, \phi^u + 2, v', * \rangle$, or $\langle j, \phi^u + 3, v', * \rangle$, where $v' \neq v$.*

Proof. The proof for all forms of message is obtained by contradiction. Suppose that some process p_j can produce a valid message of the form $\langle j, \phi^u + 1, v', * \rangle$. If so, according to the semantic validation, the message requires more than $\frac{n+f}{2}/2$ messages of the form $\langle *, \phi^u, v', * \rangle$. This implies that at least one correct processes must have broadcast a message of the form $\langle *, \phi^u, v', * \rangle$. This is a contradiction because it is assumed that every correct process with phase ϕ^u has proposal value v. Thus, any valid message of the form $\langle *, \phi^u + 1, *, * \rangle$ must have proposal value $v \neq v'$.

Now suppose that some process p_j can produce a valid message of the form $\langle j, \phi^u + 2, v', * \rangle$. There are two cases to consider: $v' = \perp$ and $v' \in \{0, 1\}$. First, if $v' = \perp$, then, according to the semantic validation, the message requires more than $\frac{n+f}{2}/2$ messages of the form $\langle *, \phi^u, 0, * \rangle$ and more than $\frac{n+f}{2}/2$ messages of the form $\langle *, \phi^u, 1, * \rangle$ from different processes. By assumption, however, only one of the values (0 or 1) is the proposal value of every correct process with phase ϕ^u. This implies that the other value can be in at most in f messages. This is a contradiction because $f < \frac{n+f}{2}/2$. Second, if $v' \in \{0, 1\}$, then, according to the semantic validation, the message requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi^u + 1, v', * \rangle$. As demonstrated in the previous paragraph, every correct process broadcasts the same value v at phase $\phi^u + 1$. This means that an adversary can create at most f messages of the form $\langle *, \phi^u + 1, v', * \rangle$. This is a contradiction because $f < \frac{n+f}{2}$.

Finally, suppose that some process p_j can produce a valid message of the form $\langle j, \phi^u+3, v', * \rangle$. According to the semantic validation, such message either requires (1) more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi^u + 1, v', * \rangle$, or (2) more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi^u + 2, \perp, * \rangle$. This is a contradiction because it has already been shown that any valid message of the form $\langle *, \phi^u + 1, *, * \rangle$ or $\langle *, \phi^u + 2, *, * \rangle$ must carry proposal value v. Thus, an adversary can create, at most, $f < \frac{n+f}{2}$ messages of the form $\langle *, \phi^u + 1, v', * \rangle$ or $\langle *, \phi^u + 2, \perp, * \rangle$. □

Lemma 25. *If there is some phase ϕ, where $\phi \pmod 3 = 1$, such that every correct process p_j that sets $\phi_j = \phi$ also sets $v_j = v$, then no process p_j can produce a valid message of the form $\langle j, \phi', v', * \rangle$, for any $\phi' > \phi$ and $v' \neq v$.*

Proof. The proof is obtained by complete induction on the phase number ϕ'. Thus, it can be assumed that there exists a number $a > 0$ such that no process p_j can produce a valid message of the form $\langle j, \phi + m, v', * \rangle$, for all $0 < m \leq a$. We must prove that no process p_j can produce a valid message of the form $\langle j, \phi + a + 1, v', * \rangle$, with $v' \neq v$. There are three cases to consider, depending on the value of $(\phi+a) \pmod 3$. If $(\phi+a) \pmod 3 = 1$, then, by Lemma 24, no process p_j can produce a valid message of the form $\langle j, \phi+a+1, v', * \rangle$. If $(\phi+a) \pmod 3 = 2$, then, by assumption, it must be true that no process p_j can produce a valid message of the form $\langle j, \phi+a-1, v', * \rangle$. It follows at phase $(\phi + a - 1) \pmod 3 = 1$ every correct process must have the same proposal value v. Thus, by Lemma 24, no process p_j can produce a valid message of the form $\langle j, \phi + a + 1, v', * \rangle$. Likewise, if $(\phi + m) \pmod 3 = 0$, then, by assumption, it must be true that no process p_j can produce a valid message of the form $\langle j, \phi + a - 2, v', * \rangle$. It follows at phase $(\phi + a - 2) \pmod 3 = 1$ every correct process must have the same proposal value v. Thus, by Lemma 24, no process p_j can produce a valid message of the form $\langle j, \phi + a + 1, v', * \rangle$. □

Theorem 26. *If all correct processes propose the same value v, then every correct process that decides, decides v.*

Proof. If every correct process p_i proposes the same initial value v, then every p_i has phase $\phi_i \pmod 3 = 1$ and $v_i = v$. Therefore, by Lemma 25, every correct process p_i with any phase value $\phi_i > 1$ must have proposal value $v_i = v$. Since any correct process p_i that decides must do it on its proposal value v_i (Line 41), it follows that p_i cannot decide on any value other than v. □

Corollary 27. *If there is some phase ϕ, where $\phi \pmod 3 = 1$, such that every correct process p_j that sets $\phi_j = \phi$ also sets $v_j = v$, then every correct process that decides, decides v.*

The following lemma proves that any two processes in the same phase $\phi \pmod 3 = 0$ can not have different proposal values 0 and 1. In other words, if some correct process has proposal value $v \in \{0, 1\}$, then any other process can only propose a value of v or $\perp$. This lemma is essential to the *agreement* property in Theorem 29, which shows that when the first correct process decides, every process has to propose the same value at the following phase. By Lemma 25, this can never lead to a decision on different values.

Lemma 28. *Let ϕ be any phase such that $\phi \pmod 3 = 0$. There are no two correct processes p_i and p_j that receive valid messages of the form $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$, respectively.*

Proof. Suppose otherwise. Then p_i and p_j are two processes with phase value ϕ where p_i receives a valid message $\langle u, \phi, 0, * \rangle$ from p_u and p_j receives a valid message $\langle w, \phi, 1, * \rangle$ from p_w. According to the semantic validation, message $\langle u, \phi, 0, * \rangle$ requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, 0, * \rangle$ and message $\langle w, \phi, 1, * \rangle$ requires than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, 1, * \rangle$. This implies there are at least $f + 1$ faulty processes that must have produced messages with contradictory values for phase $\phi - 1$. This is a contradiction because, by definition, there are at most f faulty processes. □

Theorem 29. *No two correct processes decide differently.*

Proof. Let p_i be the first correct process to decide. Without loss of generality, let the decision value be 1. Then, p_i must have received either more than $\frac{n+f}{2}$ valid messages of the form $\langle *, \phi, 1, undecided \rangle$ with $\phi \pmod 3 = 0$ (Lines 29-31), or some valid message of the form $\langle *, \phi', 1, decided \rangle$ with $\phi' > \phi$ (Lines 10-18). Either way, more than $\frac{n-f}{2}$ correct processes must have broadcast a message of the form $\langle *, \phi, 1, undecided \rangle$ because, according to the semantic validation, a message with $status = decided$ requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, 1, undecided \rangle$ with $\phi \pmod 3 = 0$. Additionally, for these correct processes to have a proposal value of 1 at phase ϕ, more than $\frac{n-f}{2}$ correct processes must have broadcast a message of the form $\langle *, \phi - 1, 1, * \rangle$ (Lines 23-24). Hence, more than $\frac{n-f}{2}$ correct processes have a proposal value 1 at both phases $\phi - 1$ and ϕ.

The following paragraph shows that any correct process that reaches phase $\phi + 1$ also sets proposal value 1, and by Lemma 26 can not decide on a value different than 1. A correct process p_j that sets phase $\phi_j = \phi + 1$ must have received either (1) more than $\frac{n+f}{2}$ messages with phase ϕ (Line 19), or (2) some message with $\phi + 1$ (Line 10). In case (1), at least one of the messages must be of the form $\langle *, \phi, 1, * \rangle$ because, as we have demonstrated, more than $\frac{n-f}{2}$ correct processes must have broadcast a message of the form $\langle *, \phi, 1, undecided \rangle$. Additionally, by Lemma 28, no process can receive valid a message of the form $\langle *, \phi, 0, * \rangle$. This means that, in this case, p_j sets $v_j = 1$ (Line 33). In case (2), any message with phase $\phi + 1$ requires either (a) more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, v, * \rangle$, or (b) more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, \perp, * \rangle$. Scenario (a) is impossible with $v \neq 1$ and scenario (b) is impossible altogether because, as we have demonstrated, more than $\frac{n-f}{2}$ correct processes have a proposal value 1 at both phase $\phi - 1$ and ϕ. Hence, in this case, no message with phase $\phi + 1$ can be considered valid if it carries a proposal value $v \neq 1$. Consequently, any process that sets its phase value to $\phi + 1$ also sets its proposal value to 1. Thus, by Corollary 27 no two correct processes can decide different values. □

The remainder of the proof handles the *termination* property of consensus (Theorem 36). For this part, we work on the assumption that the message scheduling falls under the control of an *adversary* that can cause no more than σ dynamic omission faults per round, with $\sigma \leq \lceil \frac{n-f}{2} \rceil (n-k-f) + k - 2$

The rationale for the *termination* property is based on the idea that as long as correct processes keep increasing their phase values, a decision is eventually reached. As we have seen from the safety proofs, if there is unanimity amongst the correct processes at some $\phi \pmod 3 = 1$, then every correct process that reaches any phase higher than $\phi + 1$ decides. The idea is to show that k correct processes can reach any arbitrarily high phase value, and that unanimity eventually happens.

The following two lemmas dictate how correct processes increment their phase values in tandem. Lemma 31, in particular, states that for any process with phase value ϕ, eventually k correct processes must have a phase value equal or higher than $\phi - 1$.

Lemma 30. *If some process broadcasts a valid message of the form $\langle *, \phi, *, * \rangle$ with $\phi > 1$, then more than $\frac{n-f}{2}$ correct processes must have broadcast some valid message of the form $\langle *, \phi - 1, *, * \rangle$.*

Proof. According to the semantic validation, a message of the form $\langle *, \phi, *, * \rangle$ requires more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi - 1, *, * \rangle$. Since at most f messages come from Byzantine processes, this implies that, for a process to broadcast a valid message of the form $\langle *, \phi, *, * \rangle$, more than $\frac{n+f}{2} - f = \frac{n-f}{2}$ correct processes must have broadcast a valid message of the form $\langle *, \phi - 1, *, * \rangle$. □

Lemma 31. *If some correct process p_i has phase value $\phi_i = \phi$, then eventually there is a set of correct processes S such that $\forall_{p_j \in S} : \phi_j \geq \phi - 1$ and $|S| \geq k$.*

Proof. Suppose otherwise. By Lemma 30, if some process p_i has $\phi_i = \phi > 1$, then there is a set of correct processes S' such that $\forall_{p_j \in S'} : \phi_j \geq \phi - 1$ and $|S'| > \frac{n-f}{2}$. Let $R^+ = S'$ where $\frac{n-f}{2} < |R^+| < k$, and R^- be the set of remaining correct processes, i.e., $\forall_{p_u \in R^-} : \phi_u < \phi - 1$ where $n - k - f < |R^-| < \frac{n-f}{2}$.

By assumption, the adversary can create at most $\sigma = \sigma_1 + \sigma_2$ message omissions per round, where $\sigma_1 = \lceil \frac{n-f}{2} \rceil (n-k-f)$ and $\sigma_2 = k-2$. In order to prevent processes in R^- from reaching $\phi_u \geq \phi - 1$, the adversary must omit $|R^+|$ messages from processes of R^+ to R^- (due to Lines 10-18). This implies the elimination of more than $\frac{n-f}{2}$ messages in more than $n-k-f$ processes because $|R^+| > \frac{n-f}{2}$ and $|R^-| > n-k-f$. It is clear that after consuming σ_1 faults, there are at most $n-k-f$ processes in R^- that do not receive any message from R^+.

Since by definition $|R^-| - (n-k-f) = k - |R^+| > 0$, there must be $k - |R^+|$ processes in R^- that could still receive messages from every process in R^+. Let R^-_* denote the set of processes in this situation. To prevent every process p_u in R^-_* from reaching $\phi_u \geq \phi - 1$, the adversary must create $|R^+||R^-_*|$ omissions, where $|R^+| + |R^-_*| = k$. However, the adversary only has $\sigma_2 = k - 2 = |R^+| + |R^-_*| - 2$ faults available. This creates a contradiction because $|R^+||R^-_*| > |R^+| + |R^-_*| - 2$, for all $|R^+| \geq 1$ and $|R^-_*| \geq 1$. This implies that some correct process in $|R^-|$ always increases its phase value when $\frac{n-f}{2} < |R^+| < k$. □

The following lemma is central to the liveness part of the proof. It shows that in any communication round where the number of dynamic omission faults is not higher than $\lceil \frac{n-f}{2} \rceil (n-k-f) + k - 2$, some correct process increases its phase value.

Lemma 32. *Let R^+ be a set of correct processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi$, with $|R^+| = k + \alpha$ and $0 \leq \alpha \leq n-k-f$. Let α or more process in R^+ have phase ϕ and the remaining processes of R^+ have phase $\phi + 1$. Let R^- be the set of correct process such that $\forall_{p_j \in R^-} : \phi_j < \phi$, with $|R^-| = n-k-f-\alpha$. Whenever a round has such configuration, some correct process increases its phase value.*

Proof. Suppose otherwise. Then, under the Lemma conditions, there must be a message schedule where at some round no process increases its phase value.

In order to prevent every process in R^- from increasing its phase value, the adversary must omit every message from R^+ to R^- (due to Lines 10-18). This requires that

$|R^+||R^-|$ omission faults must be spent. Since $|R^+||R^-| = (k+\alpha)(n-k-f-\alpha)$ and the admissible number of omissions per round is $\sigma = \lceil \frac{n-f}{2} \rceil (n-k-f)+k-2$, then the adversary is left with no more than $\sigma - |R^+||R^-| \leq (\alpha + \lceil \frac{n-f}{2} \rceil + k + f - n)\alpha + k - 2$ omission faults.

In order to block each of the α processes in R^+ with phase ϕ, the adversary must omit all messages from processes in R^+ with phase $\phi + 1$ (Line 10) and it must prevent the reception of more than $\frac{n+f}{2}$ messages of the form $\langle *, \phi, *, * \rangle$ also from processes in R^+ (Line 19). This implies that each of the α processes with phase ϕ can receive the $n - k - f - \alpha$ messages from processes in R^- and at most $\lfloor \frac{n+f}{2} \rfloor$ messages from processes in R^+. Therefore, the adversary must create at least $\left[n - (\lfloor \frac{n+f}{2} \rfloor + n - k - f - \alpha)\right] \alpha$ omission faults to stop the progression of the α processes. Since $\left[n - (\lfloor \frac{n+f}{2} \rfloor + n - k - f - \alpha)\right] \alpha = (\alpha + \lceil \frac{n-f}{2} \rceil + k + f - n)\alpha$, the adversary is left with no more than $k - 2$ omission faults.

For the remaining k processes in R^+, there are two possible cases:

1. First consider the two extreme situations, where all k processes either have phase value ϕ or $\phi + 1$. Since the adversary only has $k - 2$ omission faults left, some process has to receive more than $\frac{n+f}{2}$ messages with the same phase ϕ or $\phi + 1$. Therefore, some process increases its phase value (Line 19).

2. Second consider that some of the k processes have phase value $\phi + 1$ and the others have phase value ϕ. Let H be the set of processes with $\phi + 1$ and L the set of processes with ϕ, such that $|H| + |L| = k$. To block the processes in L, the adversary has to omit $|H||L|$ messages (due to Line 10). Since the adversary only has $k - 2 = |H| + |L| - 2$ omission faults left, it can not prevent some process from increasing its phase because $|H||L| > |H| + |L| - 2$ for all $|H| \geq 1$ and $|L| \geq 1$.

□

Lemma 33. *Let $\phi_{init} = 1$ be the initial phase value for all correct processes. Some correct process p_i eventually sets $\phi_i > \phi_{init}$.*

Proof. If every correct process has the same phase value ϕ_{init}, then according to the conditions of Lemma 32, this is equivalent of having every correct process in set R^+ with phase ϕ_{init}, such that $|R^+| = n - f$. Therefore, by Lemma 32, some correct process has to increase its phase value and set $\phi_i > \phi_{init}$. □

The following lemma wraps up the progress of phase values by stating that some correct process can reach any arbitrarily high phase value.

Lemma 34. *If some correct process has phase value ϕ, then eventually some correct process must have phase value $\phi + 1$.*

Proof. If some correct process has phase value ϕ, then by Lemma 31, eventually there is a set R^+ of k or more correct processes such that $\forall_{p_i \in R^+} : \phi_i \geq \phi - 1$. This implies that the system must reach a configuration where there are two sets of correct processes R^+ and R^- according to the conditions of Lemma 32. When this happens, by the same Lemma, some correct process will increase its phase. This process can be in one of three possible cases: (1) a process of R^-; (2) a process with phase number $\phi - 1$ of R^+; or (3) a process with phase number ϕ of R^+. The system configuration resulting from cases (1) and (2) falls under the conditions of Lemma 32, and therefore more correct processes will continue to increase their phase. Consequently, in the most extreme scenario, the system will evolve to a configuration where all correct processes are in phase number ϕ, and case (3) will necessarily have to occur, and some correct process p_i will set its phase number to $\phi_i = \phi + 1$. □

This lemma supports Theorem 36 by restricting Byzantine processes from broadcasting $undecided$ messages after a certain point if there is unanimity amongst correct processes.

Lemma 35. *If there is some phase ϕ^u, where $\phi^u \pmod 3 = 1$, such that every correct process p_i that sets $\phi_i = \phi^u$ also sets $v_i = v$, then no process can broadcast a valid message of the form $\langle *, \phi^u + 3 + c, *, undecided \rangle$, for any $c \geq 0$.*

Proof. Suppose otherwise, that some process can broadcast a valid message of the form $\langle *, \phi^u + 3 + c, *, undecided \rangle$. According to the semantic validation, such message 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 $\phi^u + 3 + c$. This means that $\phi' \geq \phi^u + 1$. However, it is easy to see that, by Lemma 25, no process can broadcast a message of the form $\langle *, \phi, v', * \rangle$ with $v' \neq v$, for any $\phi \geq \phi^u + 1$. This results in a contradiction. Hence, no process can broadcast a valid message of the form $\langle *, \phi^u + 3, *, undecided \rangle$. □

Finally, the *termination* property is proven by showing that as long as correct processes keep increasing their phase value, then unanimity eventually happens and, consequently, a decision by k correct processes.

Theorem 36. *At least k correct processes eventually decide with probability 1.*

Proof. The proof is organized in three parts. First, we show that as messages are received, correct processes make progress on the protocol execution and continue to increase their phase number. Second, we demonstrate that due to this progression, with probability 1, there will be some phase where every process that reaches it proposes the same value. Third, we prove that when this happens at least k correct processes decide.

First part: By Lemma 33, some correct process p_i eventually increases its phase number from the initial phase number, i.e., $\phi_i = \phi > \phi_{init}$. Then, by Lemma 34, some correct process will eventually set its phase number to $\phi + 1$. Moreover, by Lemma 31, k or more correct processes set their phase value to at least ϕ. Since these Lemmas can be applied repeatedly, this ensures that, for any arbitrary phase value ϕ', there is some

subset S of at least k correct processes, such that every process $p_j \in S$ eventually has phase value $\phi_j \geq \phi'$.

Second part: By Lemma 28, no two processes with the same phase $\phi \pmod 3 = 0$ can receive messages $\langle *, \phi, 0, * \rangle$ and $\langle *, \phi, 1, * \rangle$. Therefore, any process p_i that enters the **if** condition of Line 19, and sets $\phi_i = \phi + 1$ (Line 38), must set its proposal value v_i either to a common value v (Line 33) or to a random value 1 or 0 (Line 35). This implies that the probability of every correct process with phase value $\phi + 1$ (where $\phi + 1 \pmod 3 = 1$) having the same proposal value v is $p \geq 2^{-\gamma}$, where γ is the number of correct processes with phase $\phi + 1$.

As the protocol progresses, and the phase number of processes increases, the probability of not existing a phase $\phi^u \pmod 3 = 0$ where every correct process has the same value v is $\lim_{\phi \to \infty} (1-p)^{\phi} = 0$. Thus, eventually there will be a phase $\phi \pmod 3 = 1$ where every correct process p_i that sets $\phi_i = \phi^u$ also sets v_i to the same proposal value v.

Third part: Since every correct process with phase $\phi^u \pmod 3 = 1$ has the same proposal value, by Lemma 35 no process with phase value $\phi^u + 3$ or higher can broadcast a message with the status set to $undecided$. This implies that any correct process with phase $\phi^u + 3$ or higher must have status set to $decided$. The first part of this lemma shows that k correct processes can reach any arbitrarily high phase value. As such, k correct processes must eventually have phase value $\phi^u + 3$ or higher, which implies that those processes decide. □

Chapter 7

Performance Evaluation of Turquois

This chapter studies the performance of Turquois in 802.11b wireless ad hoc networks under both a real-world network testbed and a simulation environment. In the former, Turquois is compared with the binary consensus protocols of the RITAS and SINTRA protocol stacks (Cachin & Poritz, 2002; Moniz *et al.*, 2006b), whose performance was previously assessed in Chapter 3. These are, respectively, Bracha's binary consensus (Bracha, 1984), and the ABBA binary consensus (Cachin *et al.*, 2000).

Like Turquois, Bracha's and ABBA are leader-free randomized protocols that attain optimal resilience in terms of Byzantine processes. Unlike Turquois, they were not designed with a wireless environment in mind, and employ the typical intrusion-tolerant asynchronous model with reliable point-to-point links. 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 usually terminates in a constant number of steps (at most two rounds of three steps each), but relies heavily on expensive public-key cryptography.

The chapter is composed by two main sections. Section 7.1 compares the performance of Turquois against the Bracha's and ABBA protocols. The protocols are executed in the Emulab platform (White *et al.*, 2002), on a testbed composed of up to 16 rack-mounted and Wi-Fi enabled hosts. Section 7.2 analyzes the performance

of Turquois in the ns-3 network simulator (Henderson *et al.*, 2006) and evaluates the impact of several additional parameters - not possible to evaluate in the experimental setting of the previous section - such as a significantly higher number of nodes.

7.1 Protocol Comparison in Emulab

This section compares Turquois with the Bracha's and ABBA binary consensus protocols in 802.11b wireless ad hoc networks. It evaluates the latency of the protocols under several parameters such as the number of processes, types of faults in the system, and distribution of the initial proposal values. Other aspects such as the clock tick mechanism of Turquois and optimizations to the protocol are also evaluated.

7.1.1 Testbed and Implementation

Testbed. The experiments were carried out on the Emulab testbed (White *et al.*, 2002). A total of 16 nodes were used, each one with the following hardware characteristics: Pentium III processor, 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.

Implementation. All the protocols were implemented in C. In Turquois, processes communicate using UDP broadcast. A local clock tick is triggered if one of the following conditions is true: (1) x ms have passed since the last broadcast (where $x = 10$ for $n = 4$ and $n = 7$, and $x = 20$ for $n = 10$, $n = 13$ and $n = 16$), or (2) the phase value was changed. In both Bracha's and ABBA, the processes use TCP to communicate because of their requirement of reliable point-to-point links. 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.

Non-optimized vs. optimized protocols. To obtain a deeper knowledge about the performance of these protocols, each protocol was implemented and tested in two variants: a non-optimized and an optimized one. The optimized variants consist exclusively of the inclusion an early decision step, similar to the one employed by Bracha's binary consensus as described in Section 3.1.2, where processes decide in step 1 if they receive unanimous proposals.

This implies that the non-optimized implementation of Bracha's differs slightly from the version described in Section 3.1.2 such that it does not include the early decision at step 1. The non-optimized versions of ABBA and Turquois correspond exactly to the protocols described, respectively, in Section 3.1.2 and Section 6.3.

In the case of the optimized implementations, Bracha's protocol is exactly the one described in Section 3.1.2, with the early decision at step 1. ABBA and Turquois optimized versions are described below in Section 7.1.4, where we present the performance results of the optimized protocols. Their optimizations consist of an early decision step similar to Bracha's.

7.1.2 Methodology

The performance metric utilized in the experiments is the *latency*. This metric is always relative to a particular process 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 processes 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 processes involved in the experiment. When a process receives such a message, it starts a consensus execution. 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 repeating this procedure 50 times, and then by averaging the latencies collected by all processes. 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 processes in the system. In our experiments, the values are 4, 7, 10, 13, and 16 processes. The proposal distribution defines the initial values to be proposed by the processes. In the *unanimous* proposal distribution all processes propose the same initial value 1. In the *divergent* distribution processes with an odd process 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 processes behave correctly. The *fail-stop* fault load makes $f = \lfloor \frac{n-1}{3} \rfloor$ processes crash before the measurements are initiated. In the *Byzantine* fault load, $f = \lfloor \frac{n-1}{3} \rfloor$ processes try to keep the correct processes from reaching a decision by attacking the execution of the protocol. This is accomplished as follows. In both Bracha's and Turquois, a Byzantine process 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, since at least it causes a delay due to message validations. In ABBA, since the protocol terminates in a constant number of steps, a Byzantine process 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 processes. Finally, the value of the parameter k in Turquois is set to $k = n - f$ in all fault loads, with $f = \lfloor \frac{n-1}{3} \rfloor$.

7.1.3 Results for the Non-Optimized Implementations

Failure-free fault load. Table 7.1 and Figure 7.1 present the average latency for every tested combination of group size and proposal distribution, in executions without process 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 processes 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 sig-

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

Table 7.1: Average latency and confidence interval in a 802.11b network with no process failures (latency in milliseconds and confidence level of 95%).

nature 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, 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 the heavy cryptography proves to be too much of a burden.

At a first glance, these results may seem contradictory to the ones described in Chapter 3, where Bracha's protocol evidenced better performance than ABBA in most settings. There are two reasons for this. The first is that the version of Bracha's evalu-

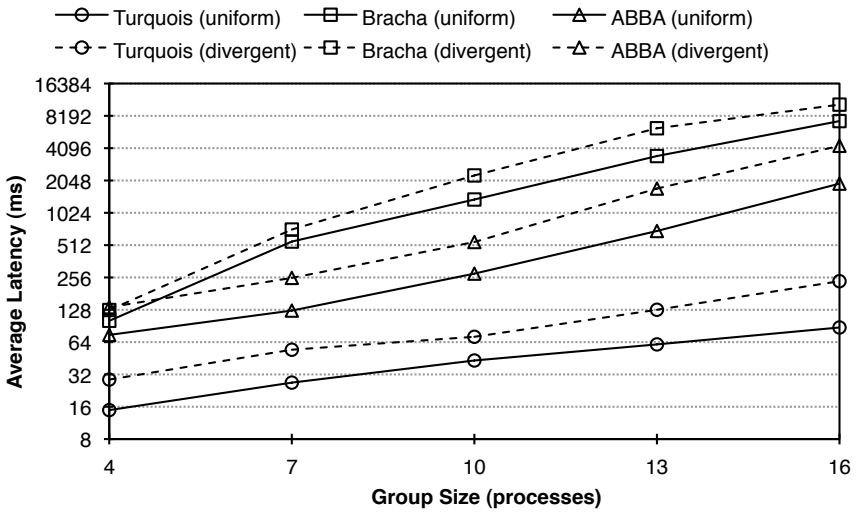

Figure 7.1: Average latency and confidence interval in a 802.11b network with no process failures (logarithmic scale of base 2).

ated here is removed of the optimization that allowed it to decide on the first step if the proposals were unanimous. Hence, the protocol is forced to execute for at least three steps until it decides. The second reason is related to the fact that the ABBA protocol itself was significantly faster in this experiment. The explanation for this was due to the different operating system images used in the experiments of Section 3.2 and the ones described here. While both sets of experiments were made on the same exact hardware, at the time when the experiments described in this section were taken, the Emulab testbed employed a more recent Linux kernel version (2.6.18 against 2.4.34 from Section 3.2) along with a newer openssl package (0.9.7f against 0.9.7a from Section 3.2). Under this combination, the speed of public-key cryptographic operations is significantly improved. Table 7.2 shows a direct comparison of the time required to generate and verify 1024-bit RSA signatures on both OS images. It is worth to note, however, that these differences in the results do not represent any inconsistencies whatsoever. This experiment simply shows that the superior scalability of ABBA in relation to Bracha's (a trend that was already seen in Chapter 3) is now observed earlier, due to the reasons stated above.

Another observation in this experiment is that the relative difference between proposal distributions was approximately the same across all protocols, with the latency

	Time (ms)			
	Kernel 2.6.18		**Kernel 2.4.34**	
N	Sign	Verify	Sign	Verify
10	132.13	6.77	300.31	15.62
100	1276.73	64.05	2807.66	156.17
1000	12765.49	642.27	28087.20	1562.70
10000	127681.27	6428.06	280871.99	15631.49

Table 7.2: Time in milliseconds for N sign and verify 1024-bit RSA operations (160-bit hash size) under Linux kernel 2.6.18 (used in this section) and 2.4.34 (used in Section 3.2).

roughly doubling from an unanimous to a divergent proposal distribution. The reason for this is that when processes propose different values, the protocols usually need to execute for an additional cycle of steps. For example, in Turquois, processes 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 processes 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.

Fail-stop fault load. Table 7.3 and Figure 7.2 show the performance of the protocols when $f = \lfloor \frac{n-1}{3} \rfloor$ processes 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 processes crash, for every group size tested, exactly $n - f = \lfloor \frac{n+f}{2} \rfloor + 1$ processes are left in the system. This means that, as the processes make progress, they necessarily have to receive the same set of messages. Thus, never diverging in their proposal values after the first phase.

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 processes 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 $n - f$ processes are present in the system. More retransmis-

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

Table 7.3: Average latency and confidence interval in a 802.11b network with fail-stop process failures (latency in milliseconds and confidence level of 95%).

sions are needed to ensure that processes 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 $n - 1$ processes not receiving a message, while in the protocols that employ TCP one collision results in just one process not receiving the message; (2) although the chosen timeout value for the experiments turned out to be well-adjusted regarding the number of processes (see below the paragraph discussing the timeout mechanism of Turquois), the fact that it assumes a static value means that, unlike TCP, it is not adaptable to the network conditions, and therefore it does not self-adjust to dynamic factors like collisions, external interference, etc. This also explains its proportionally wider confidence interval. An optimization of the retransmission mechanism could potentially improve the performance of Turquois in these scenarios. Nevertheless, Turquois still performs

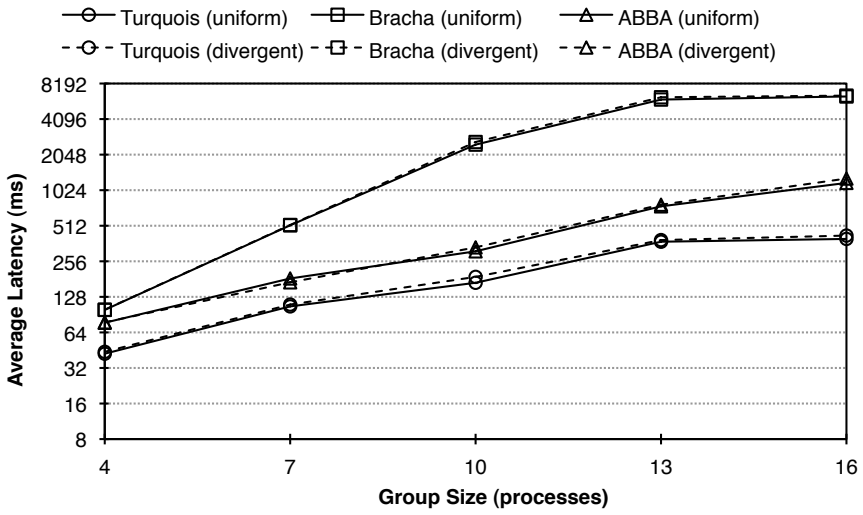

Figure 7.2: Average latency in a 802.11b network with fail-stop process failures (logarithmic scale of base 2).

significantly better than the other two protocols with this fault load.

In the Bracha's and ABBA protocols with 16 nodes, results seem to start to contradict the idea that protocols perform better in the failure-free fault load when compared with the fail-stop fault load. 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 processes would be necessary to confirm this.

Byzantine fault load. Table 7.4 and Figure 7.3 show the performance of the protocols when $f = \lfloor \frac{n-1}{3} \rfloor$ processes 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 process failures, with the latency very roughly doubling in the divergent distribution. Like in the failure-free scenario, this is because divergent proposal values force processes to execute for extra rounds to reach a decision.

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 is that many messages

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

Table 7.4: Average latency in a 802.11b network with Byzantine process failures (latency in milliseconds and confidence level of 95%).

being broadcasted by Byzantine processes 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 processes broadcasting messages). As for Turquois, despite its non-optimized timeout mechanism making it more sensitive to this issue, it still manages to be the faster protocol. The question of the performance impact of the local clock tick value is further explored in the following section.

Timeout Mechanism of Turquois. Under the experiments carried out so far, Turquois assumed a constant local clock tick value of 10 ms or 20 ms. This section analyzes how an optimized local clock tick can impact the performance of the protocol. To this end, Turquois was executed with a variable number of processes - 4 to 16 - and local clock tick value - 2, 5, 10, 20, 50, and 100 ms. Every other parameter was fixed. The pro-

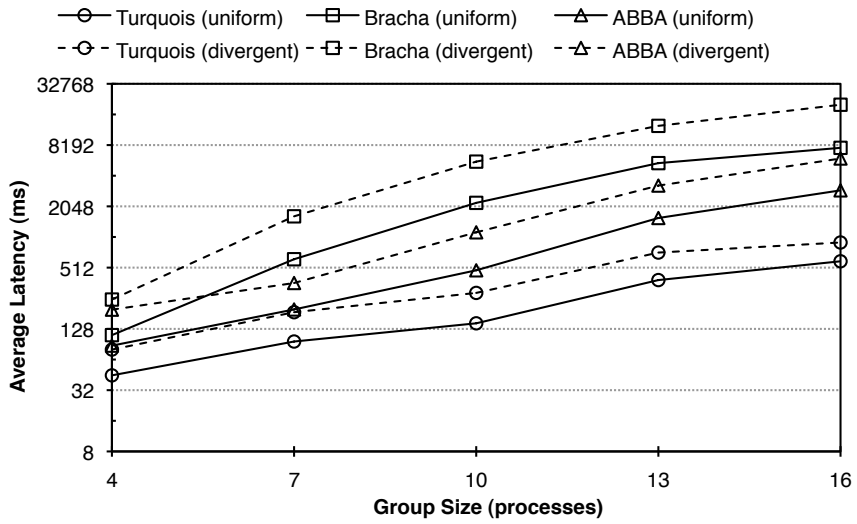

Figure 7.3: Average latency in a 802.11b network with Byzantine process failures (logarithmic scale of base 2).

tocol was run in a 802.11b network with the failure-free fault load and a unanimous proposal distribution.

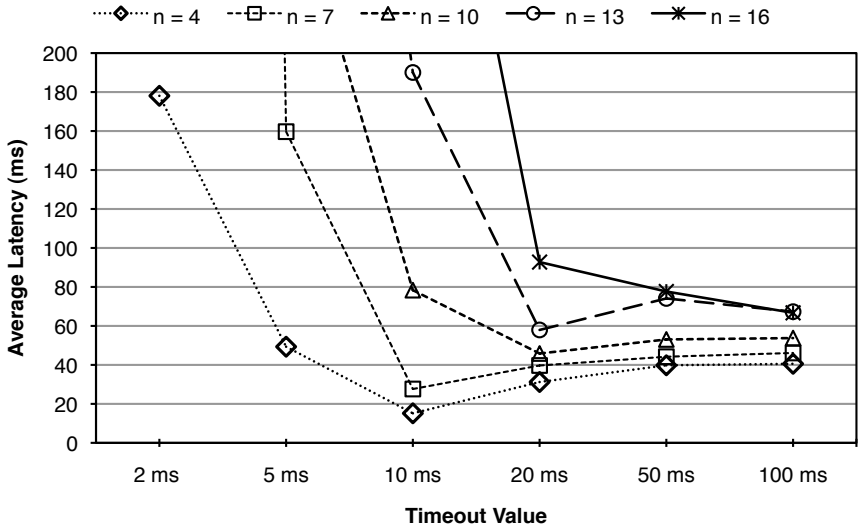

Figure 7.4: Average latency in a 802.11b network with varying timeout value.

The graph from Figure 7.4 shows how the latency of the protocol can be affected

by the local clock tick value. The values shown are the average latency for 50 executions of the protocol obtained through the same experimental methodology described in Section 7.1.2. 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 impairing performance. It can also be observed that as the number of processes increases, the optimal timeout value tends to be higher in order to alleviate the contention created by having extra processes in the system.

7.1.4 Results for Optimized Protocols

This section describes the results from the evaluation of the optimized versions of the protocols. For this set of experiments, it was added an early decision step to each protocol, such that if they receive an unanimity of proposals they can decide immediately. To this end, each protocol was adapted as described below.

Bracha's. In the Bracha's binary consensus, the protocol was exactly as it is described in Section 3.1.2, where, in each round, a processes can decide v at the end of step 1 if it receives $n - f$ messages with the same proposal value v.

ABBA. Since ABBA does not rely on reliable broadcast as an underlying communication primitive, unlike Bracha's, it cannot decide on the first communication step even if it receives $n - f$ unanimous proposals. This is because there is no way of knowing (without additional communication steps) if the received messages by a process are consistent with the messages that arrive at the other correct processes. Nevertheless, an early decision step was still obtained by having the processes keep track of the *pre-process* messages they receive for step 0 (see Section 3.1.2) even if they already have progressed to a higher step. So, in the case a process accumulates n pre-process messages with the same proposal value v, it can decide v. Like in Bracha's algorithm, even if a process decides early in this fashion, it continues to execute the algorithm because other processes might require its help to make progress. This optimization does not violate safety. A trivial inspection of the algorithm shows that if $n - f$ correct

processes propose the same initial value v (which must be the case if a process receives n pre-process messages with v), then the algorithm must decide v.

Turquois. Turquois suffers from the same limitation of ABBA: it cannot decide based solely on $n - f$ messages with the same value v because it does not rely on a reliable broadcast primitive. For this reason, in Turquois, a process decides early if it eventually accumulates n messages for some phase $\phi \pmod 3 = 1$ with the same proposal value v. A simple analysis of the protocol shows that if $n - f$ correct processes propose the same value v at some phase $\phi \pmod 3 = 1$, then every process must decide v by phase $\phi + 2$ (Corollary 27). Additionaly, since in Turquois messages can be arbitrarily lost without the possibility of recovery, the protocol was changed so that the proposal value of the last $\phi \pmod 3 = 1$ phase reached by a process is piggybacked in the messages for the subsequent two phases. This improves the changes of an early decision in case a process does not receive some $\phi \pmod 3 = 1$ messages. Like the other two protocols, a process that decides early has to continue its execution to guarantee the progress of the other processes.

Experiments. A a similar set of experiments with varying values of group size and proposal distributions was run to evaluate how each protocol performed. The results are presented in Table 7.5. This table is directly comparable to Table 7.1.

In general, the results show that every protocol benefited with the early decision step optimization. Bracha's protocol was the one that had the most dramatic improvement, in which the decision time was cut down to approximately 1/3 in the unanimous proposal distribution, but also to about 1/2 with higher numbers of processes and divergent proposals. These results are very consistent with the employed optimization because with unanimous proposals it is guaranteed that only the first of the 3 communication steps are required to decide. With divergent proposals, the performance gains are justified by the fact that, even though the protocol cannot decide early in the first round, it might still decide on the first step of a subsequent round.

ABBA also evidenced a noteworthy performance improvement with unanimous proposals, deciding in about 1/2 the time. The fact that the decision time was cut to around 1/2 and not 1/3 like in Bracha's (despite ABBA being also a 3-step round algorithm) is justified by the early decision requiring n messages instead of the $n - f$

	Average Latency ± Confidence Interval (ms)	
Group Size	**Turquois**	
	unanimous	divergent
$n = 4$	7.29 ± 1.00	18.54 ± 2.56
$n = 7$	18.35 ± 2.70	30.18 ± 3.21
$n = 10$	29.14 ± 4.00	55.09 ± 9.23
$n = 13$	38.39 ± 5.35	103.83 ± 40.63
$n = 16$	72.34 ± 19.21	221.50 ± 87.84
Group Size	**Bracha**	
	unanimous	divergent
$n = 4$	32.27 ± 4.32	121.37 ± 12.15
$n = 7$	158.39 ± 24.35	528.74 ± 53.35
$n = 10$	349.93 ± 14.18	1482.28 ± 172.87
$n = 13$	849.90 ± 141.87	3335.86 ± 393.82
$n = 16$	1914.84 ± 340.07	5647.30 ± 142.66
Group Size	**ABBA**	
	unanimous	divergent
$n = 4$	35.32 ± 0.37	136.40 ± 38.26
$n = 7$	63.08 ± 6.80	245.37 ± 45.99
$n = 10$	130.04 ± 17.08	572.54 ± 72.09
$n = 13$	386.95 ± 61.03	1673.56 ± 265.64
$n = 16$	703.62 ± 205.73	4039.64 ± 499.61

Table 7.5: *Optimized Protocols.* Average latency and confidence interval in a 802.11b network with no process failures (latency in milliseconds and confidence level of 95%).

in Bracha's. The overhead incurred by waiting for the last f messages accounts for the decreased efficiency. A pattern observed in the experiments was that the last message of some step s was commonly received only after the first messages from step $s + 1$. The performance in ABBA with divergent proposals was similar to the non-optimized version. Since ABBA guarantees termination by the end of the second round with high probability and only the first round includes step 0 where the early decision is tried (see Section 3.1.2), this implies that without unanimity of proposals the algorithm executes just like in the non-optimized version.

Turquois also benefited from its optimization, although to a lesser degree. With $n = 4$ and unanimous proposals it executes in about 1/2 the time, but the differences to the non-optimized version get progressively narrower as the number of processes increases. A similar pattern occurs with the divergent proposals, but with a smaller

margin. These results are justified because Turquois has to wait for n messages to decide early. Additionally, the progressively narrower differences are related to the fact that as the number of processes increases, it becomes increasingly harder for a process to receive the n proposal values for any given phase $\phi \pmod 3 = 1$, even with piggybacking. With higher process numbers, it was usual for some process p to be unable to receive any message from another process q during a particular execution of the algorithm. Despite not benefiting so much from the early decision optimization as the other protocols, Turquois still performed significantly better than the others, in many cases by more than an order of magnitude.

7.2 Simulation

This section analyzes the performance of Turquois in the ns-3 network simulator (Henderson *et al.*, 2006). It tests Turquois under some additional interesting 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.

7.2.1 Timeout Value

Figures 7.5 to 7.8 plot the average latency and average number of rounds to termination of Turquois as a function of the timeout value. Each curve has an associated number of processes. Figures 7.5 and 7.7 show the curves for 4, 10, and 25 processes, while Figures 7.6 and 7.8 show the curves for 50, 75, and 100 processes.

The results obtained via simulation are congruent with the observed trend of the experimental evaluation of Section 7.1. 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

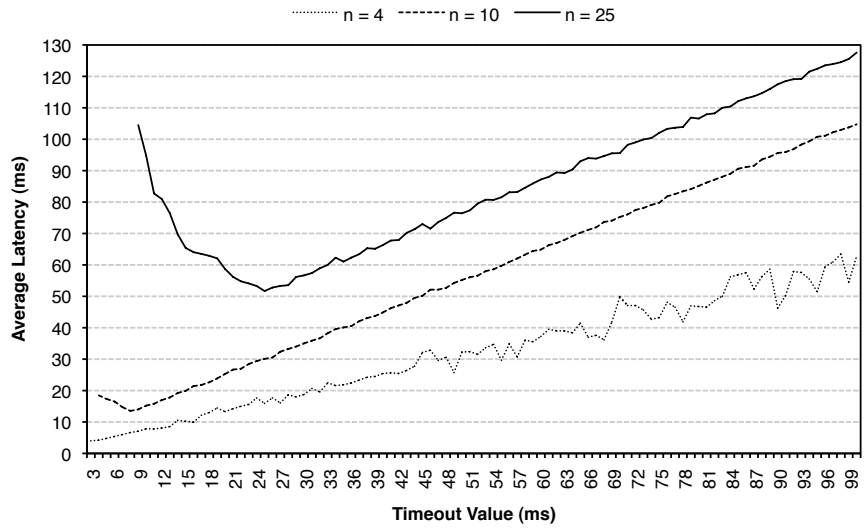

Figure 7.5: Average latency of Turquois with varying timeout value for 4, 10, and 25 processes.

Figure 7.6: Average latency of Turquois with varying timeout value for 50, 75, and 100 processes.

value. This pattern is clearly observable with n=25 and, to a lesser extent, with $n = 10$. With $n = 4$, the number of processes 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 start to be drawn from a progressively higher timeout value as n becomes larger. 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 generated contention is so high that the algorithm does not terminate in a reasonable number of steps (i.e., within 1000 rounds).

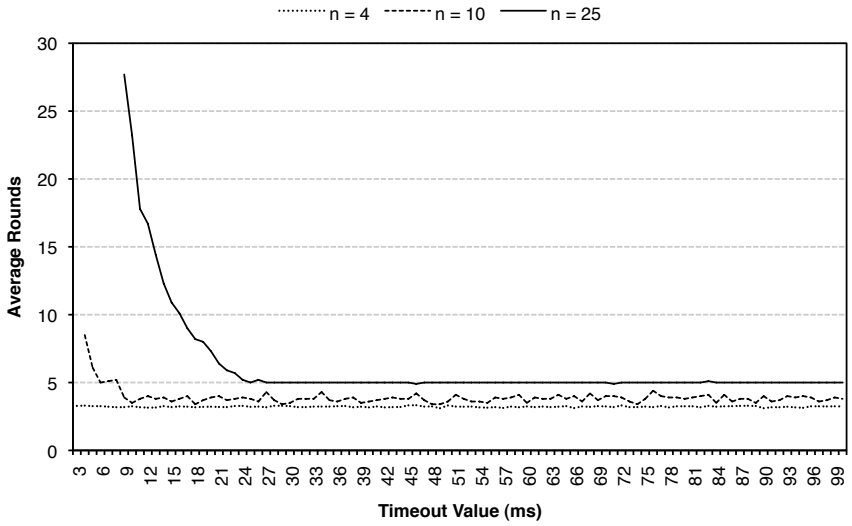

Figure 7.7: Average number of rounds of Turquois with varying timeout value for 4, 10, and 25 processes.

As for the average number of rounds, depicted in Figures 7.7 and 7.8, they show a strong correlation with the latency graphs. Like with the latency, the number of rounds is higher when the timeout value is too low. A very low timeout generates many concurrent broadcasts, resulting in message loss due to transmission collisions, which implies more broadcasting rounds. As the timeout increases, the contention decreases (less concurrent broadcasts) and, as it can be clearly observed in the graphs, the number of rounds stabilizes. This stabilization point occurs at about the same timeout value as the latency reaches its sweet spot (roughly around n ms).

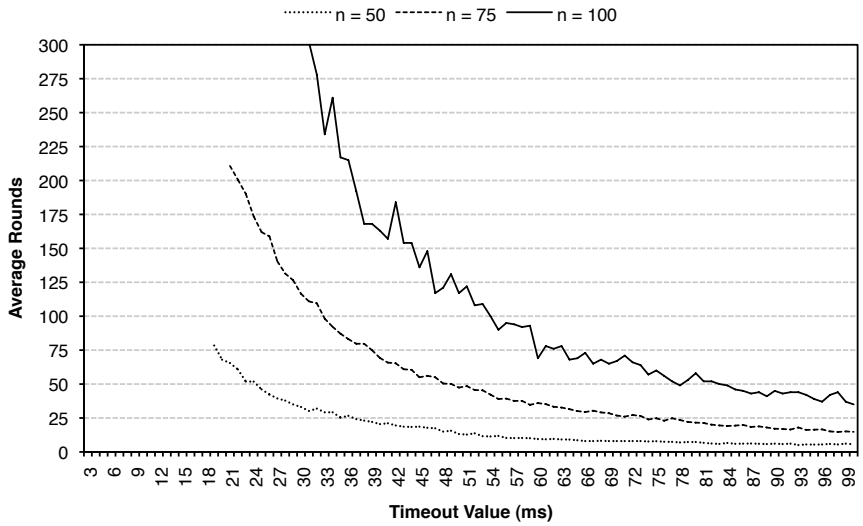

Figure 7.8: Average number of rounds of Turquois with varying timeout value for 50, 75, and 100 processes.

7.2.2 Physical 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 140 meters. Figures 7.9 and 7.10 plot, respectively, the average latency and average number of rounds until termination. The same pattern is observed for both figures. The latency increases linearly with the number of processes and the disc radius. As the disc radius grows, some processes fall off broadcasting range of each other. The degree of connectivity is still sufficient for the information to propagate within the system (i.e., for processes to increase they phase numbers), but the number of needed rounds becomes higher. Since some nodes fall off broadcasting range, the algorithm starts to be more sensitive to message loss, as the level of redundancy is lower.

7.3 Summary of Results

The chapter presented the performance evaluation of Turquois - an intrusion-tolerant binary consensus protocol specifically designed for wireless ad-hoc networks. The

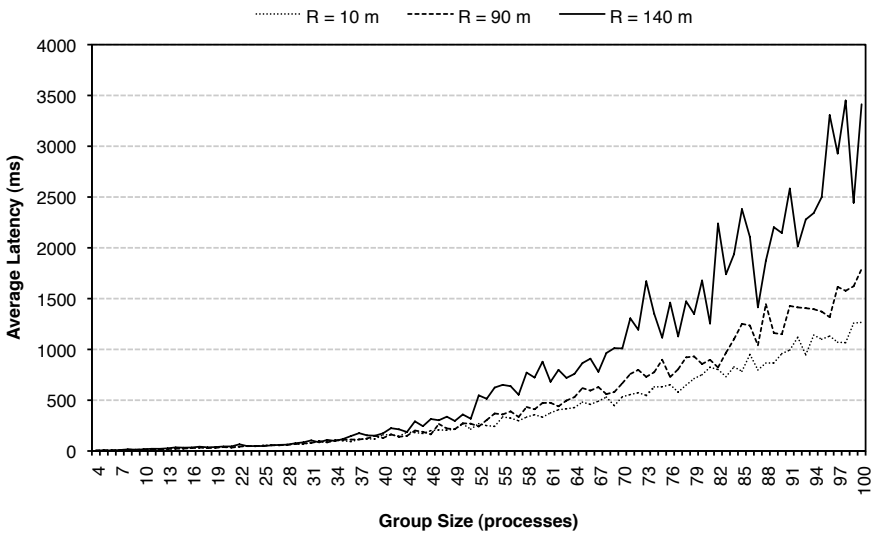

Figure 7.9: Average Latency of Turquois with disc radius of 10, 90, and 140 meters.

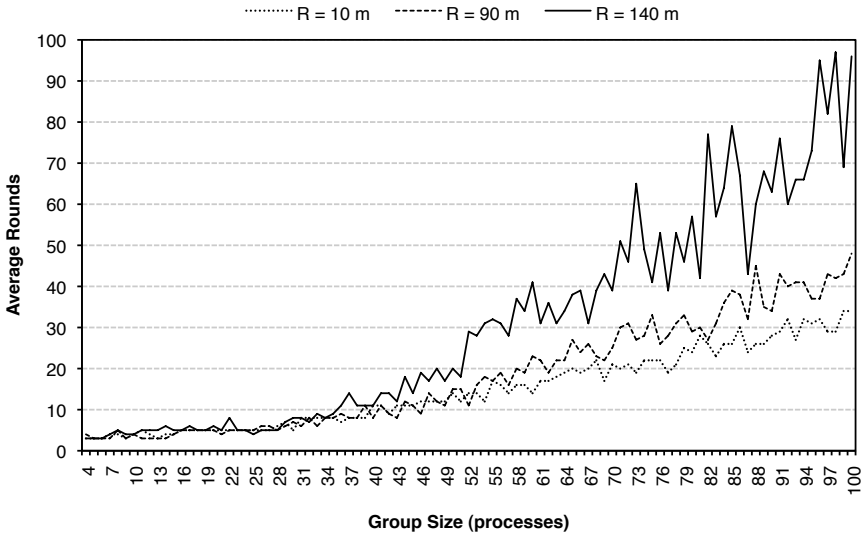

Figure 7.10: Average Rounds of Turquois with disc radius of 10, 90, and 140 meters.

protocol was subject to (1) a comparative performance evaluation with two well-known intrusion-tolerant consensus protocols in a real-world network testbed, and to (2) a simulation under the ns-3 network simulator, which studied a few additional relevant parameters.

The main results of these experiments are summarized in the following points:

- Turquois performs significantly better than the other two protocols - Bracha's and ABBA - in wireless ad hoc networks. This difference becomes wider as the number of processes increases, exceeding an order of magnitude in several cases.

- Bracha's protocol is the worst contender, showing serious performance degradation due to the $O(n^3)$ message complexity.

- The latency roughly doubles across all protocols from the unanimous to divergent proposal distribution. When processes propose different values, the protocols usually need to execute for an additional cycle of steps.

- The exception to the above observation happens with the fail-stop fault load where the latency results are similar for both proposal distributions. This is because when f processes crash, the remaining processes necessarily update their values based on the same set of messages.

- A counterintuitive observation is that, for the unanimous proposal distribution, the protocols perform worse in the fail-stop fault load than in the failure-free case. The reason for this is that with less nodes in the system, the protocols become more sensitive to message loss and more retransmissions are necessary to ensure progress.

- With the Byzantine fault load, the protocols suffer a performance degradation that becomes increasingly noticeable with a higher group size, specially with a divergent proposal distribution.

- The latency of Turquois can be affected by the local clock tick value. In particular, a too small timeout value generates too much contention in the network, which causes considerable message loss, severely impairing performance. Additionally, as the number of processes increases, the optimal timeout value tends to be higher in order to accommodate the extra contention.

- Every protocol benefited significantly from an optimization that allowed for early decision in good executions. Bracha's protocol was the one that benefited the most, deciding in about 1/3 of the time with unanimous proposals. ABBA decided in about 1/2 of the time. Turquois decided in 1/2 of the time with $n = 4$, but this margin got narrower as the number of processes increased. This is justified because processes have to wait for n messages with the same value, combined with the lack of reliable communication channels.

- A higher node dispersal over a spatial area requires more rounds to reach consensus. Thos occurs because processes fall off the broadcasting range of each other, and consequently the latency becomes higher.

Chapter 8

Conclusions and Future Research Directions

8.1 Conclusions

The operation of wireless ad hoc networks is intrinsically tied to the ability of nodes to coordinate their actions in a dependable and efficient manner. 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 thesis investigates the problem of agreement in wireless ad hoc networks. Its overarching goal is the design of consensus algorithms adapted to wireless ad hoc networks that are able to both (1) cope with the unreliability and potential hostility of wireless environments, and (2) operate efficiently. The ideal protocol would be able to withstand a portion of its nodes falling under the control of a malicious adversary and behaving in an arbitrary manner.

The first step towards this goal started with a thorough performance evaluation of existing intrusion-tolerant protocol stacks in both wired and wireless environments. The choice over which protocol stacks to evaluate was based on their potential attractiveness for wireless ad hoc networks - mainly, the fact that their protocols execute in a completely decentralized manner (i.e., they are leader-free). From the rich corpus of results obtained with these experiments, several important observations were made

that add to our knowledge about the practical behavior of intrusion-tolerant protocols, in particular of randomized algorithms, which are used by both protocol stacks.

The main conclusion to extract was that current intrusion-tolerant agreement protocols do not fare well in wireless ad hoc networks. This is attributed to several reasons: the high message complexity (in the case of local coin protocols), the computationally expensive cryptography (in the case of shared coin), but perhaps more importantly, to their underlying point-to-point reliable communication model, which forces the implementation of inefficient end-to-end message delivery mechanisms.

These results lead to the identification of the *communication failure model* as a system model more adjusted to wireless ad hoc networks. Under the communication failure model, message transmissions are assumed to be affected by dynamic and transient faults, i.e., communication between two nodes can be faulty at one time and be correct at another; past failures are not an indicator of future behavior. Furthermore, faults can strike anywhere in the system during its lifetime. This highly dynamic vision of faults is aligned with the nature of wireless ad hoc networks. It captures momentary node disconnection - pervasive in wireless environments - due to mobility and other environmental phenomena such as electromagnetic interference, fading, collisions, etc. The main benefit from this approach is that it makes no assumptions regarding the reliability of any particular communication between two processes. This effectively frees the system architect from implementing end-to-end delivery mechanisms and allows the protocols to directly tap into the natural broadcasting medium of wireless ad hoc networks, where the cost of transmitting a message to multiple processes can be just the same of transmitting it to a single process, as long as they are within communication range.

Despite its usefulness to represent wireless ad-hoc communication environments, research on the communication failure model has been limited. This is related to an associated impossibility result. This result, dubbed the *Santoro-Widmayer impossibility*, applies to the k-*agreement* problem among n processes, in which k out of n processes must agree on a binary value $v \in \{0, 1\}$. The Santoro-Widmayer impossibility result applies to non-trivial agreement, i.e., for $k > \lceil n/2 \rceil$. It states that there is no finite time deterministic algorithm that allows n processes to reach k-*agreement* if more than $n-2$ transmission failures occur in a communication step. This is a very discouraging result since the crashing of a single process necessarily results in $n - 1$ transmission failures,

rendering this form of agreement impossible. Moreover, this result is produced under the strongest timing assumptions where both the processes' relative processing times and communication delays are bound by known constants (i.e., a synchronous system).

A significant theoretical contribution of the thesis was to show that this impossibility result can be circumvented. This was achieved by employing randomization, which has never been applied before in the context of the communication failure model. The Santoro-Widmayer impossibility result rules out deterministic solutions to agreement in this model. Randomization takes a probabilistic approach to the problem. It overcomes previous limitations by supplying processes with access to random information and combining this with a refinement of the problem statement where a decision is ensured with probability 1.

To this end, it was presented a randomized k-consensus algorithm and the respective correctness proof. The algorithm allows at least k processes to decide on a common binary value in a system with n processes such that $k > \frac{n}{2}$. The safety properties of consensus are ensured even with an unrestricted number of faults, while the progress is ensured in communication rounds where the number of omissions is $\sigma \leq \lceil \frac{n}{2} \rceil (n-k) + k - 2$. The algorithm is adequate for wireless ad-hoc networks because it allows one to take advantage of the broadcasting medium in an efficient way and, at the same time, ensures safety under severe communication problems that lead to many message losses. The termination is achieved with probability 1 when communication becomes stable, i.e., when the above threshold is satisfied. Furthermore, the algorithm is efficient in the sense that it terminates in 2 communication rounds under favorable conditions.

The last step to attain the goal of efficient intrusion-tolerant consensus in wireless ad hoc networks was achieved with the Turquois protocol - designed to tolerate a combination of Byzantine nodes and dynamic omission transmission faults. To the best of our knowledge, this is the first consensus protocol that exhibits these characteristics. Besides tolerating Byzantine nodes, Turquois improves the previous protocol on a very important aspect: it assumes an asynchronous model - a fundamental assumption since it is usually hard to force wireless networks into a timely behavior. Furthermore, it tolerates $t < \frac{n}{3}$ Byzantine processes, being optimal in this aspect. Safety is maintained despite unrestricted message omissions, and liveness is ensured in broadcasting steps where the number of omissions is $\sigma \leq \lceil \frac{n-f}{2} \rceil (n-k-f) + k - 2$, with $f \leq t$ being

the number of processes in the system that are actually faulty. The protocol relies on a novel message authentication and validation mechanism that minimizes the use of computationally expensive public-key cryptography, which allows computational resources to be spared and keeps the authentication payload within a fixed length. The protocol evaluated together with two well-known intrusion-tolerant consensus protocols. The results confirmed the expectations regarding the usefulness of the communication failure model for wireless ad hoc networks and, in particular, for the design of intrusion-tolerant agreement protocols. Regardless of the type of faults present in the system, Turquois significantly outperformed the other protocols, sometimes by more than an order of magnitude, specially when the number of processes in the system increased.

8.2 Future Research Directions

There are several ways in which this work can be extended in order to provide further functionality to wireless ad hoc networks. Extending the model to consider multi-hop communication is certainly one of the most obvious ways of augmenting the usefulness of the approaches described in this thesis. Another one is to take into account a dynamic group of processes - an assumption very close to environments with mobile nodes. Finally, on a more theoretical perspective it would be interesting to close the gap on the upper bounds obtained for the number of omission faults and prove their tightness - both for the omissions-only model of Chapter 5 and the hybrid model - dynamic omission faults and Byzantine nodes - employed by Turquois.

References

ABD-EL-MALEK, M., GANGER, G.R., GOODSON, G.R., REITER, M.K. & WYLIE, J.J. (2005). Fault-scalable Byzantine fault-tolerant services. *ACM SIGOPS Operating Systems Review*, 39(5):59–74. 36, 37

ABRAHAM, I., DOLEV, D. & MALKHI, D. (2004). LLS: A locality aware location service for mobile ad hoc networks. In *Proceedings of the DIALM-POMC Joint Workshop on Foundations of Mobile Computing*, 75–84. 2

AKKOYUNLU, E.A., EKANADHAM, K. & HUBER, R.V. (1975). Some constraints and tradeoffs in the design of network communications. In *Proceedings of the 5th ACM Symposium on Operating Systems Principles*. 31

AMIR, Y., DOLEV, D., KRAMER, S. & MALKHI, D. (1992). Transis: A communication subsystem for high availability. In *Proceedings of the 22nd IEEE Fault-Tolerant Computing Symposium*. 39

AMIR, Y., DOLEV, D., MELLIAR-SMITH, P.M. & MOSER, L.E. (1994). Robust and efficient replication using group communication. Tech. rep., Institute of Computer Science, Hebrew University. 38

AMIR, Y., ATENIESE, G., HASSE, D., KIM, Y., NITA-ROTARU, C., SCHLOSSNAGLE, T., SCHULTZ, J., STANTON, J. & TSUDIK, G. (2000). Secure group communication in asynchronous networks with failures: Integration and experiments. In *Proceedings of the 20th IEEE International Conference on Distributed Computing Systems*, 330–343. 39

AMIR, Y., KIM, Y., NITA-ROTARU, C., SCHULTZ, J., STANTON, J. & TSUDIK, G. (2001). Exploring robustness in group key agreement. In *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems*, 399–408. 39

AMIR, Y., COAN, B., KIRSCH, J. & LANE, J. (2008). Byzantine replication under attack. In *Proceedings of the 38th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, 197–206. 36

AMIR, Y., DANILOV, C., DOLEV, D., KIRSCH, J., LANE, J., NITA-ROTARU, C., OLSEN, J. & ZAGE, D. (2010). Steward: Scaling Byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing*, 7(1):80–93. 36

ASPNES, J., EREN, T., GOLDENBERG, D.K., MORSE, A.S., WHITELEY, W., YANG, Y.R., ANDERSON, B.D.O. & BELHUMEUR, P.N. (2006). A theory of network localization. *IEEE Transactions on Mobile Computing*, 5(12):1663–1678. 2

ATTIYA, H., KOGAN, A. & WELCH, J.L. (2010). Efficient and robust local mutual exclusion in mobile ad hoc networks. *IEEE Transactions on Mobile Computing*, 9(3):361–375. 46

AWERBUCH, B. & PELEG, D. (1995). Online tracking of mobile users. *Journal of the ACM*, 42(5):1021–1058. 2

BADACHE, N., HURFIN, M. & MACEDO, R. (1999). Solving the consensus problem in a mobile environment. In *Proceedings of the 18th IEEE International Performance, Computing, and Communications Conference*, 29–35. 42, 43

BALDONI, R., VIRGILLITO, A. & PETRASSI, R. (2002). A distributed mutual exclusion algorithm for mobile ad-hoc networks. In *Proceedings of the 7th International Symposium on Computers and Communications*, 539–545. 45

BALDONI, R., HÉLARY, J.M. & PIERGIOVANNI, S.T. (2008). A methodology to design arbitrary failure detectors for distributed protocols. *Journal of Systems Architecture*, 54(7):619–637. 27

BAR-YOSSEF, Z., FRIEDMAN, R. & KLIOT, G. (2008). Rawms - random walk based lightweight membership service for wireless ad hoc networks. *ACM Transactions on Computer Systems*, 26(2):1–66. 50

BARBARA, D. & GARCIA-MOLINA, H. (1986). The vulnerability of vote assignments. *ACM Transactions on Computer Systems*, 4(3):187–213. 36

BEN-OR, M. (1983). Another advantage of free choice: Completely asynchronous agreement protocols. In *Proceedings of the 2nd ACM Symposium on Principles of Distributed Computing*, 27–30. 7, 10, 29, 43, 54, 104

BHANDARI, V. & VAIDYA, N. (2005). On reliable broadcast in a radio network. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 138–147. 47

BIELY, M., WIDDER, J., CHARRON-BOST, B., GAILLARD, A., HUTLE, M. & SCHIPER, A. (2007). Tolerating corrupted communication. In *Proceedings of the 26th ACM Symposium on Principles of Distributed Computing*, 244–253. 32, 101

BIRMAN, K. (1993). The process group approach to reliable distributed computing. *Communications of the ACM*, 36(12):37–53. 37

BIRMAN, K. & JOSEPH, T. (1987a). Exploiting virtual synchrony in distributed systems. In *Proceedings of the 11th ACM Symposium on Operating Systems Principles*, 123–138. 37

BIRMAN, K. & JOSEPH, T. (1987b). Reliable communication in the presence of failures. *ACM Transactions on Computer Systems*, 5(1):46–76. 39

BORRAN, F. & SCHIPER, A. (2010). A leader-free Byzantine consensus algorithm. In *Proceedings of the 11th International Conference on Distributed Computing and Networking*, 67–78. 7

BORRAN, F., PRAKASH, R. & SCHIPER, A. (2008). 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*, 227–236. 44

BOUKERCHE, A. & ABROUGUI, K. (2006). An efficient leader election protocol for mobile networks. In *Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing*, 1129–1134. 47

BRACHA, G. (1984). An asynchronous $\lfloor (n-1)/3 \rfloor$-resilient consensus protocol. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, 154–162. 8, 13, 30, 54, 55, 74, 78, 104, 117, 139

BROCH, J., MALTZ, D., JOHNSON, D., CHU, Y. & JETCHEVA, J. (1998). A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, 85–97. 2

BROWN, M.D. (2007). *Air traffic control using virtual stationary automata*. Master's thesis, Massachusetts Institute of Technology. v, 2, 3

BUX, W., CLOSS, F.H., KUEMMERLE, K., KELLER, H.J. & MUELLER, H.R. (1983). Architecture and design of a reliable token-ring network. *IEEE Journal on Selected Areas in Communications*, 1(5):756–765. 33

CACHIN, C. & PORITZ, J.A. (2002). Secure intrusion-tolerant replication on the Internet. In *Proceedings of the International Conference on Dependable Systems and Networks*, 167–176. 39, 41, 55, 87, 139

CACHIN, C., KURSAWE, K. & SHOUP, V. (2000). Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *Proceedings of the 19th ACM Symposium on Principles of Distributed Computing*, 123–132. 13, 30, 54, 55, 59, 116, 139

CANETTI, R. & RABIN, T. (1993). Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 42–51. 30, 54, 116

CASTRO, M. & LISKOV, B. (1999). Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, 173–186. 9, 36

CASTRO, M., RODRIGUES, R. & LISKOV, B. (2003). Base: Using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems*, 21(3):236–269. 36

CHANDRA, T. & TOUEG, S. (1996). Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267. vii, 7, 10, 26, 35, 42

CHANDRA, T., HADZILACOS, V. & TOUEG, S. (1996a). On the impossibility of group membership. In *Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing*, 322–330. 38

CHANDRA, T., HADZILACOS, V. & TOUEG, S. (1996b). The weakest failure detector for solving consensus. *Journal of the ACM*, 43(4):685–722. 34

CHANG, E. & ROBERTS, R. (1980). Decentalized extrema-finding in circular configurations of processes. *Communications of the ACM*, 23(11):627–628. 34

CHARRON-BOST, B. & SCHIPER, A. (2007). The heard-of model: Computing in distributed systems with benign failures. Tech. Rep. LSR-REPORT-2007-001, EPFL. 32, 44

CHEN, Y. & WELCH, J.L. (2002). Self-stabilizing mutual exclusion using tokens in mobile ad hoc networks. In *Proceedings of the 6th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 34–42. 45

CHOCKLER, G., DEMIRBAS, M., GILBERT, S., NEWPORT, C. & NOLTE, T. (2005). Consensus and collision detectors in wireless ad hoc networks. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*. 43, 90

CHOCKLER, G., DEMIRBAS, M., GILBERT, S., LYNCH, N., NEWPORT, C. & NOLTE, T. (2008). Consensus and collision detectors in radio networks. *Distributed Computing*, 21(1):55–84. 32

CHOR, B. & DWORK, C. (1989). Randomization in Byzantine agreement. In *Advances in Computing Research 5: Randomness and Computation*, 443–497, JAI Press. 30

CLAYPOOL, M. (2005). On the 802.11 turbulence of Nintendo DS and Sony PSP handheld network games. In *Proceedings of 4th ACM SIGCOMM Workshop on Network and System Support for Games*, 1–9. 2

CLEMENT, A., WONG, E.L., ALVISI, L., DAHLIN, M. & MARCHETTI, M. (2009). Making Byzantine fault tolerant systems tolerate Byzantine faults. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation*, 153–168. 36

CLEMENTI, A., MONTI, A. & SILVESTRI, R. (2001). Selective families, superimposed codes, and broadcasting on unknown radio networks. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms*, 709–718. v, 3, 4

COHEN, H. (1993). *Quorum Systems: Dominatoin, Fault-Tolerance, Balancing*. Master's thesis, The Weizmann Institute of Science. 36

CORREIA, M., NEVES, N.F., LUNG, L.C. & VERISSIMO, P. (2005). Low complexity Byzantine-resilient consensus. *Distributed Computing*, 17(3):237–249. 28

CORREIA, M., NEVES, N.F. & VERÍSSIMO, P. (2006). From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *Computer Journal*, 41(1):82–96. vi, 4, 11, 20, 35, 39, 41, 55, 73, 74, 75, 78

CORREIA, M., NEVES, N.F., LUNG, L.C. & VERÍSSIMO, P. (2007). Worm-IT – a wormhole-based intrusion-tolerant group communication system. *Journal of Systems and Software*, 80(2):178–197. 39, 40

COWLING, J.A., MYERS, D.S., LISKOV, B. & RODRIGUES, R. (2006). HQ replication: a hybrid quorum protocol for Byzantine fault tolerance. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, 177–190. 36, 37

DELPORTE-GALLET, C., FAUCONNIER, H., GUERRAOUI, R. & KOUZNETSOV, P. (2005). Mutual exclusion in asynchronous systems with failure detectors. *Journal of Parallel and Distributed Computing*, 65(4):492–505. 33

DOLEV, D., DWORK, C. & STOCKMEYER, L. (1987). On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):77–97. vii, 7, 24

DOLEV, S., SCHILLER, E. & WELCH, J.L. (2006). Random walk for self-stabilizing group communication in ad hoc networks. *IEEE Transactions on Mobile Computing*, 5(7):893–905. 50

DONG, Q. & LIU, D. (2009). Resilient cluster leader election for wireless sensor networks. In *Proceedings of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad hoc Communications and Networks*, 108–116. v, 3

DOUDOU, A., GARBINATO, B. & GUERRAOUI, R. (2002). Encapsulating failure detection: From crash-stop to Byzantine failures. In *International Conference on Reliable Software Technologies*, 24–50. 27

DRABKIN, V., FRIEDMAN, R. & SEGAL, M. (2005). Efficient Byzantine broadcast in wireless ad-hoc networks. In *Proceedings of the International Conference on Dependable Systems and Networks*, 160–169. 48

DRAVES, R., PADHYE, J. & ZILL, B. (2004). Routing in multi-radio, multi-hop wireless mesh networks. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking*, 114–128. 2

DWORK, C., LYNCH, N. & STOCKMEYER, L. (1988). Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323. vii, 7, 10, 24, 25

ELSON, J. & ROMER, K. (2003). Wireless sensor networks: a new regime for time synchronization. *ACM SIGCOMM Computer Communication Review*, 33(1):149–154. 2

EZHILCHELVAN, P., MOSTEFAOUI, A. & RAYNAL, M. (2001). Randomized multivalued consensus. In *Proceedings of the 4th IEEE International Symposium on Object-Oriented Real-Time Computing*, 195–200. 43

FISCHER, M.J., LYNCH, N.A. & PATERSON, M.S. (1985). Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382. vi, 5, 15, 23, 24, 31, 90, 120

FRAGA, J.S. & POWELL, D. (1985). A fault- and intrusion-tolerant file system. In *Proceedings of the 3rd International Conference on Computer Security*, 203–218. vii, 5

FU, W. (1990). *Enhancing Concurrency and Availability for Database Systems*. Ph.D. thesis, Simon Fraser University. 36

GARCIA-MOLINA, H. (1982). Elections in distributed computer systems. *IEEE Transactions on Computers*, 31(1):48–59. 34

GARCIA-MOLINA, H. & BARBARA, D. (1985). How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860. 36

GRAY, J. (1978). Notes on data base operating systems. In R. Bayer, R.M. Graham & G. Seegmüller, eds., *Operating Systems: An Advanced Course*, vol. 60 of *Lecture Notes in Computer Science*, Springer-Verlag. 31

GROSSGLAUSER, M. & TSE, D.N.C. (2002). Mobility increases the capacity of ad hoc wireless networks. *IEEE/ACM Transactions on Networking*, 10(4):477–486. 2

GUERRAOUI, R. & SCHIPER, A. (2001). The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41. 38

GUPTA, P. & KUMAR, P.R. (2000). The capacity of wireless networks. *IEEE Transactions on Information Theory*, 46(2):388–404. 2

HADZILACOS, V. & TOUEG, S. (1993). Fault-tolerant broadcasts and related problems. In S. Mullender, ed., *Distributed Systems*, chap. 5, 97–146, ACM Press / Addison-Wesley. 20, 21, 34

HATZIS, K.P., PENTARIS, G.P., SPIRAKIS, P.G., TAMPAKAS, V.T. & TAN, R.B. (1999). Fundamental control algorithms in mobile networks. In *Proceedings of the 11th Annual ACM Symposium on Parallel Algorithms and Architectures*, 251–260. 46

HEIDE, F.M., SCHINDELHAUER, C., VOLBERT, K. & GRUNEWALD, M. (2004). Congestion, dilation, and energy in radio networks. *Theory of Computing Systems*, 37(3):343–370. 2

HENDERSON, T.R., ROY, S., FLOYD, S. & RILEY, G.F. (2006). ns-3 project goals. In *Proceedings from the 2006 Workshop on ns-2: the IP Network Simulator*, 13. 140, 153

HERLIHY, M. (1986). A quorum-consensus replication method for abstract data types. *ACM Transactions on Computer Systems*, 4(1):32–53. 36

HULL, B., BYCHKOVSKY, V., ZHANG, Y., CHEN, K., GORACZKO, M., MIU, A., SHIH, E., BALAKRISHNAN, H. & MADDEN, S. (2006). CarTel: a distributed mobile sensor computing system. In *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, 125–138. 2

INGRAM, R., SHIELDS, P., WALTER, J.E. & WELCH, J.L. (2009). An asynchronous leader election algorithm for dynamic networks. In *Proceedings of the 23rd IEEE International Symposium on Parallel and Distributed Processing*, 1–12. 47

JOHNSON, D. & MALTZ, D. (1996). Dynamic source routing in ad hoc wireless networks. In T. Imielinski & H. Korth, eds., *Mobile Computing*, chap. 5, Kluwer Academic Publishers. 2

KARP, B. & KUNG, H.T. (2000). GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, 243–254. 2

KENT, S. & ATKINSON, R. (1998). Security architecture for the internet protocol. IETF Request for Comments: RFC 2093. 57, 76

KIHLSTROM, K.P., MOSER, L.E. & MELLIAR-SMITH, P.M. (1997). Solving consensus in a Byzantine environment using an unreliable fault detector. In *Proceedings of the International Conference on Principles of Distributed Systems*, 61–75. 27

KIHLSTROM, K.P., MOSER, L.E. & MELLIAR-SMITH, P.M. (2001). The SecureRing group communication system. *ACM Transactions on Information and System Security*, 4(4):371–406. 39, 40

KIHLSTROM, K.P., MOSER, L.E. & MELLIAR-SMITH, P.M. (2003). Byzantine fault detectors for solving consensus. *The Computer Journal*, 46(1):16–35. vii, 7

KILLIJIAN, M.O., CUNNINGHAM, R., MEIER, R. & MAZARE, L. (2001). Towards group communication for mobile participants. In *Proceedings of the International Workshop on Principles of Mobile Computing*, 75–82. 49

KOO, C. (2004). Broadcast in radio networks tolerating Byzantine adversarial behavior. In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, 275–282. 47, 48

KOO, C.Y., BHANDARI, V., KATZ, J. & VAIDYA, N.H. (2006). Reliable broadcast in radio networks: the bounded collision case. In *Proceedings of the 25th Annual ACM Symposium on Principles of Distributed Computing*, 258–264, ACM. 48

KOTLA, R., ALVISI, L., DAHLIN, M., CLEMENT, A. & WONG, E. (2008). Zyzzyva: speculative Byzantine fault tolerance. *Communications of the ACM*, 51(11):86–95. 36, 37

KOWALSKI, D.R. (2005). On selection problem in radio networks. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 158–166. v, 3, 4

KUBIATOWICZ, J., BINDEL, D., CHEN, Y., CZERWINSKI, S., EATON, P., GEELS, D., GUMMADI, R., RHEA, S., WEATHERSPOON, H., WEELS, C. & ZHAO, B. (2000). Oceanstore: an architecture for global-scale persistent storage. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems*, 190–201. 38

KUHN, F., MOSCIBRODA, T. & WATTENHOFER, R. (2004). What cannot be computed locally! In *Proceedings of the 23rd Annual ACM Symposium on Principles of Distributed Computing*, 300–309. 2

KUHN, F., MOSCIBRODA, T. & WATTENHOFER, R. (2006). Fault-tolerant clustering in ad hoc and sensor networks. In *Proceedings of the 26th IEEE International Conference on Distributed Computing Systems*, 68–77. vi, 3

LAMPORT, L. (1978). Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565. 35

LAMPORT, L. (1984). Using time instead of timeout for fault-tolerant distributed systems. *ACM Transactions on Programming Languages and Systems*, 6(2):254–280. 35

LAMPORT, L. (1998). The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169. 9, 34, 44

LAMPORT, L. (2006). Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125. 102

LI, J., JANNOTTI, J., COUTO, D.S.J.D., KARGER, D.R. & MORRIS, R. (2000). A scalable location service for geographic ad hoc routing. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, 120–130. 2

LI, L., HALPERN, J.Y., BAHL, P., WANG, Y. & WATTENHOFER, R. (2005). A cone-based distributed topology-control algorithm for wireless multi-hop networks. *IEEE/ACM Transactions on Networking*, 13(1):147–159. 2

LIU, J., SACCHETTI, D., SAILHAN, F. & ISSAMY, V. (2005). Group management for mobile ad hoc networks: Design, implementation and experiment. In *Proceedings of the 6th International Conference on Mobile Data Management*, 192–199. 49

LOU, W. & WU, J. (2002). On reducing broadcast redundancy in ad hoc wireless networks. *IEEE Transactions on Mobile Computing*, 1(2):111–123. 2

LUBY, M. (1985). A simple parallel algorithm for the maximal independent set problem. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, 1–10. 2

LUO, J. & HUBAUX, J.P. (2004). Nascent: Network layer service for vicinity ad-hoc groups. In *Proceedings of the 1st IEEE Conference on Sensor, Mesh, and Ad hoc Communications and Networks*. 49

LUO, J., EUGSTER, P.T. & HUBAUX, J.P. (2004). Probabilistic lightweight group communication system for ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(2):164–179. 50

LYNCH, N.A. (1997). *Distributed Algorithms*. Morgan Kaufmann. 31

MALKHI, D. & REITER, M. (1997). Unreliable intrusion detection in distributed computations. In *Proceedings of the 10th Computer Security Foundations Workshop*, 116–124. 27

MALKHI, D. & REITER, M. (1998). Byzantine quorum systems. *Distributed Computing*, 11(4):203–213. 36, 37

MALKHI, D., OPREA, F. & ZHOU, L. (2005). ω meets Paxos: Leader election and stability without eventual timely links. In *Proceedings of the 19th International Symposium on Distributed Computing*, 199–213. 4

MALPANI, N., WELCH, J.L. & VAIDYA, N. (2000). Leader election algorithms for mobile ad hoc networks. In *Proceedings of the 4th International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 96–103. 46

MARTIN, J.P. & ALVISI, L. (2006). Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing*, 3(3):202–215. 37

MARTIN, J.P., ALVISI, L. & DAHLIN, M. (2002a). Minimal Byzantine storage. In *Proceedings of the 16th International Symposium on Distributed Computing*, 311–325. 37

MARTIN, J.P., ALVISI, L. & DAHLIN, M. (2002b). Small Byzantine quorum systems. In *Proceedings of the 32nd IEEE/IFIP International Conference on Dependable Systems and Networks*, 374–388. 37

MENEZES, A.J., OORSCHOT, P.C.V. & VANSTONE, S.A. (1997). *Handbook of Applied Cryptography*. CRC Press. 57, 125

MISENER, J.A., SENGUPTA, R. & KRISHNAN, H. (2005). Cooperative collision warning: Enabling crash avoidance with wireless technology. In *12th World Congress on ITS*. v, 3

MONIZ, H., NEVES, N.F., CORREIA, M. & VERÍSSIMO, P. (2006a). Experimental comparison of local and shared coin randomized consensus protocols. In *Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, 235–244. 117

MONIZ, H., NEVES, N.F., CORREIA, M. & VERÍSSIMO, P. (2006b). Randomized intrusion-tolerant asynchronous services. In *Proceedings of the International Conference on Dependable Systems and Networks*, 568–577. 39, 41, 55, 73, 139

MONIZ, H., NEVES, N.F., CORREIA, M., CASIMIRO, A. & VERISSIMO, P. (2007). Intrusion tolerance in wireless environments: An experimental evaluation. In *Proceedings of the 13th Pacific Rim International Symposium on Dependable Computing*, 357–364, IEEE Computer Society. 73

MONIZ, H., TEDESCHI, A., NEVES, N.F. & CORREIA, M. (2009). A distributed systems approach to airborne self-separation. In L. Weigang, A. Barros & I. Oliveira, eds., *Computational Models, Software Engineering and Advanced Technologies in Air Transportation*, IGI Global. v, 2, 3

MONIZ, H., NEVES, N.F., CORREIA, M. & VERÍSSIMO, P. (2010). RITAS: Services for randomized intrusion tolerance. *IEEE Transactions on Dependable and Secure Computing*, to appear. 39, 41, 72, 73, 87, 117

MOSCIBRODA, T. & WATTENHOFER, R. (2005). Maximal independent sets in radio networks. In *Proceedings of the 24th ACM Symposium on Principles of Distributed Computing*, 148–157. 2

MOSCIBRODA, T. & WATTENHOFER, R. (2006). The complexity of connectivity in wireless networks. In *Proceedings of the 25th IEEE International Conference on Computer Communications*, 1–13. 2

MOSER, L.E., MELLIAR-SMITH, P.M., AGARWAL, D.A., BUDHIA, R.K. & LINGLEY-PAPADOPOULOS, C.A. (1996). Totem: A fault-tolerant multicast group communication system. *Communications of the ACM*, 39(4):54–63. 39

MOSTEFAOUI, A. & RAYNAL, M. (2001). Leader-based consensus. *Parallel Processing Letters*, 11(1):95–108. 4

MOTANI, M., SRINIVASAN, V. & NUGGEHALLI, P.S. (2005). PeopleNet: engineering a wireless virtual social network. In *Proceedings of the 11th Annual International Conference on Mobile Computing and Networking*, 243–257. 2

MURPHY, A., PICCO, G. & ROMAN, G. (2006). LIME: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328. 49

NADEEM, T., DASHTINEZHAD, S., LIAO, C. & IFTODE, L. (2004). TrafficView: traffic data dissemination using car-to-car communication. *ACM Sigmobile Mobile Computing and Communications Review*, 8(3):6–19. 2

NAOR, M. & STOCKMEYER, L. (1993). What can be computed locally? In *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*, 184–193. 2

NEVES, N.F., CORREIA, M. & VERÍSSIMO, P. (2005). Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131. 7, 10, 28

OKI, B. & LISKOV, B. (1988). Viewstamped replication: A general primary copy. In *Proceedings of the 7th ACM Symposium on Principles of Distributed Computing*, 8–17. 35

PARK, V.D. & CORSON, M.S. (1997). A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of the 16th IEEE International Conference on Computer Communications*, vol. 3, 1405–1413. 47

PEASE, M., SHOSTAK, R. & LAMPORT, L. (1980). Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234. 47, 88

PELC, A. & PELEG, D. (2005). Broadcasting with locally bounded Byzantine faults. *Information Processing Letters*, 93(3):109–115. 48

PERKINS, C.E. & BELDING-ROYER, E.M. (1999). Ad-hoc on-demand distance vector routing. In *Proceedings of the 2nd Workshop on Mobile Computing Systems and Applications*, 90–100. 2

PERRY, K.J. & TOUEG, S. (1986). Distributed agreement in the presence of processor and communication faults. *IEEE Transactions on Software Engineering*, 12(3):477–482. 88

POLASTRE, J., HILL, J. & CULLER, D. (2004). Versatile low power media access for wireless sensor networks. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, 95–107, ACM. 4

PRAKASH, R. & BALDONI, R. (1998). Architecture for group communication in mobile systems. In *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems*, 235–244. 48

PRIYANTHA, N.B., CHAKRABORTY, A. & BALAKRISHNAN, H. (2000). The Cricket location-support system. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, 32–43. 2

PRIYANTHA, N.B., BALAKRISHNAN, H., DEMAINE, E.D. & TELLER, S. (2005). Mobile-assisted localization in wireless sensor networks. In *Proceedings of the 24th IEEE International Conference on Computer Communications*, vol. 1, 172–183. 2

RABIN, M.O. (1983). Randomized Byzantine generals. In *Proceedings of the 24th Annual IEEE Symposium on Foundations of Computer Science*, 403–409. 7, 11, 29, 30, 54, 116

RAJENDRAN, V., OBRACZKA, K. & GARCIA-LUNA-ACEVES, J.J. (2006). Energy-efficient, collision-free medium access control for wireless sensor networks. *Wireless Networks*, 12(1):63–78. 2

RAMASAMY, H., PANDEY, P., LYONS, J., CUKIER, M. & SANDERS, W.H. (2002). Quantifying the cost of providing intrusion tolerance in group communication systems. In *Proceedings of the International Conference on Dependable Systems and Networks*, 229–238. 40

RAYNAL, M. & ROY, M. (2005). A note on a simple equivalence between round-based synchronous and asynchronous models. In *Proceedings of the 11th IEEE Pacific Rim International Symposium on Dependable Computing*, 387–392. 115

REITER, M. (1994). Secure agreement protocols: Reliable and atomic group multicast in Rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, 68–80. 40

REITER, M., BIRMAN, K. & GONG, L. (1992). Integrating security in a group oriented distributed system. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, 18–32. 39

REITER, M.K. (1995). The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, vol. 938, 99–110, Springer-Verlag. 39, 40

REITER, M.K., BIRMAN, K.P. & VAN RENNESSE, R. (1994). A security architecture for fault-tolerant systems. *ACM Transactions on Computer Systems*, 12(4):340–371. 39

RICART, G. & AGRAWALA, A.K. (1981). An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17. 33

RIVEST, R., SHAMIR, A. & ADLEMAN, L. (1978). A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126. 125

RODEH, O., BIRMAN, K. & DOLEV, D. (2001a). The architecture and performance of security protocols in the Ensemble group communication system. *ACM Transactions on Information and System Security*, 4(3):289–319. 39

RODEH, O., BIRMAN, K. & DOLEV, D. (2001b). Using AVL trees for fault tolerant group key management. *International Journal on Information Security*, 1(2):84–99. 39

RODRIGUES, L. & VERISSIMO, P. (1992). xamp: a multi-primitive group communications service. In *Proceedings of the 11th IEEE Symposium on Reliable Distributed Systems*. 39

ROMAN, G., HUANG, Q. & HAZEMI, A. (2001). Consistent group membership in ad hoc networks. In *Proceedings of the 23rd International Conference on Software Engineering*, 381–388. 49

SABEL, L.S. & MARZULLO, K. (1995). Election vs. consensus in asynchronous systems. Tech. Rep. TR95-1488, Cornell University. 4, 34

SANTORO, N. & WIDMAYER, P. (2007). Agreement in synchronous networks with ubiquitous faults. *Theoretical Computer Science*, 384(2-3):232–249. 30, 88, 90

SANTORO, N. & WIDMEYER, P. (1989). Time is not a healer. In *Proceedings of the 6th Symposium on Theoretical Aspects of Computer Science*, 304–313. vi, viii, 5, 10, 15, 23, 30, 88, 90, 120

SASSON, Y., CAVIN, D. & SCHIPER, A. (2003). Probabilistic broadcast for flooding in wireless mobile ad hoc networks. *Wireless Communications and Networking*, 2:1124–1130. 2

SCHMID, U., WEISS, B. & KEIDAR, I. (2009). Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38(5):1912–1951. 32, 101

SCHNEIDER, F.B. (1990). Implementing faul-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319. vi, 4, 20, 35

SEBA, H., BADACHE, N. & BOUABDALLAH, A. (2002). 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*, 327–332. 42

SINGHAL, M. & MANIVANNAN, D. (1997). A distributed mutual exclusion algorithm for mobile computing environments. In *Proceedings of the IASTED International Conference on Intelligent Information Systems*, 557–561. 45

STOJMENOVIC, I., SEDDIGH, S. & ZUNIC, J. (2002). Dominating sets and neighbor elimination based broadcasting algorithms in wireless networks. *IEEE Transactions on Parallel and Distributed Systems*, 13(1):14–25. 2

SUNDARARAMAN, B., BUY, U. & KSHEMKALYANI, D. (2005). Clock synchronization for wireless sensor networks: a survey. *Ad Hoc Networks*, 3(3):281–323. 2

TOUEG, S. (1984). Randomized Byzantine agreements. In *Proceedings of the 3rd ACM Symposium on Principles of Distributed Computing*, 163–178. 30, 74

TUREK, J. & SHASHA, D. (1992). The many faces of consensus in distributed systems. *Computer Journal*, 25(6):8–17. 20

VAHDAT, A. & BECKER, D. (2000). Epidemic routing for partially-connected ad hoc networks. Tech. Rep. CS-2000-06, Duke University. 5

VAN RENESSE, R., BIRMAN, K.P. & MAFFEIS, S. (1996). Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83. 39

VARGHESE, G. & LYNCH, N.A. (1996). A tradeoff between safety and liveness for randomized coordinated attack. *Information and Computation*, 128(1):57–71. 31

VASUDEVAN, S., KUROSE, J. & TOWSLEY, D. (2004). Design and analysis of a leader election algorithm for mobile ad hoc networks. In *Proceedings of the 12th IEEE International Conference on Network Protocols*, 350–360. 47

VERÍSSIMO, P. (2002). Traveling through wormholes: Meeting the grand challenge of distributed systems. In *Proceedings of the International Workshop on Future Directions in Distributed Computing*, 144–151. 28

VERÍSSIMO, P., NEVES, N.F. & CORREIA, M. (2003). Intrusion-tolerant architectures: Concepts and design. In R. Lemos, C. Gacek & A. Romanovsky, eds., *Architecting Dependable Systems*, vol. 2677, 3–36, Springer-Verlag. vii, 5, 28

VERONESE, G.S., CORREIA, M., BESSANI, A.N. & LUNG, L.C. (2009). Spin one's wheels? Byzantine fault tolerance with a spinning primary. In *Proceedings of the 28th IEEE International Symposium on Reliable Distributed Systems*, 135–144. 36

VOLLSET, E. & EZHILCHELVAN, P.D. (2005). 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*, 166–175. 43

WALTER, J., CAO, G. & MOHANTY, M. (2001a). A k-mutual exclusion algorithm for wireless ad hoc networks. In *Proceedings of the ACM Workshop on Principles of Mobile Computing*. 45

WALTER, J.E., WELCH, J.L. & VAIDYA, N.H. (2001b). A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7(6):585–600. 45

WAN, P., ALZOUBI, K.M. & FRIEDER, O. (2004). Distributed construction of connected dominating set in wireless ad hoc networks. *Mobile Networks and Applications*, 9(2):141–149. 2

WHITE, B., LEPREAU, J., STOLLER, L., RICCI, R., GURUPRASAD, S., NEWBOLD, M., HIBLER, M., BARB, C. & JOGLEKAR, A. (2002). An integrated experimental environment for distributed systems and networks. In *Proceedings of the Fifth Symposium on Operating Systems Design and Implementation*, 255–270. 76, 139, 140

WHITEHOUSE, K., WOO, A., JIANG, F., POLASTRE, J. & CULLER, D. (2005). Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE workshop on Embedded Networked Sensors*, 45–52. 4

WU, J. & LI, H. (1999). On calculating connected dominating set for efficient routing in ad hoc wireless networks. In *Proceedings of the 3rd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications*, 7–14. 2

WU, W., CAO, J. & YANG, J. (2005). A scalable mutual exclusion algorithm for mobile ad hoc networks. In *Proceedings of the International Conference on Computer Communications and Networks*, 165–170. 45

WU, W., CAO, J. & RAYNAL, M. (2007a). A dual-token-based fault tolerant mutual exclusion algorithm for manets. In *Proceedings of 3rd International Conference on Mobile Ad-Hoc and Sensor Networks*, 572–583. 46

WU, W., CAO, J., YANG, J. & RAYNAL, M. (2007b). Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. *IEEE Transactions on Computers*, 56(8):1055–1070. 42

YE, W. & HEIDEMANN, J. (2004). Medium access control in wireless sensor networks. In C.S. Raghavendra, K.M. Sivalingam & T. Znati, eds., *Wireless Sensor Networks*, Springer. 2