Doctoral Dissertation

Dealing with Dynamic, Intermittent, and Propagating Faults in Distributed Systems

Thanh Dang Nguyen

Supervisor: Professor Xavier Défago

School of Information Science
Japan Advanced Institute of Science and Technology (JAIST)

March, 2015

To those I love . . .
You know who You are.

Abstract

Today, when many computing services are relying on large scale distributed systems, the system reliability becomes one of the most challenging research topics. A distributed system is defined as the set of computational processes that collaboratively work to solve the same problem. In distributed systems, fault, which usually occurs everywhere, is the most critical issue of reliability. In some system configurations, a single simple fault can easily corrupt the correctness of the system. Moreover, the recovering time from failed state is usually much larger than the execution time of a computation in the systems. Thus, fault-tolerance, which ensures the system still works correctly in the presence of faults, becomes a fundamental property of distributed systems.

Fault-tolerant distributed systems have been widely studied in literature for both process and communication channel failure. However, most of existing works focus on tolerating static faults occurring in processes of a stationary network whereas computational machines are more dynamic today due to the development of communication and mobile computing technology. Therefore, dynamic fault-tolerance problem is gaining more research interest. Unlike a static one, a dynamic fault can randomly move between the processes. Consequently, dynamic faults are more difficult to tolerate due to the change of their location and the increasing of the number of faults.

In this research, we aim to tolerate the dynamic faults, which are modeled as the movement of malicious mobile agents between processes in a stationary network. We study to deal with different levels of dynamicity of faults including the *intermittent* to *propagating* malicious fault. In particular

- For intermittent malicious fault, we propose a model that balances the power between the malicious agent and the correct process, which is not justified in the previous models. Under the proposed model, we prove the tight bound on the total number of processes to tolerate a given number of faults as well as the optimal algorithm.

- For problems with propagating fault, we study the possibility to limit the number of faults in a bounded value. This is a two-side problem of the spread of fault (which is also called as infection) and the containment of such spread. This problem is studied under a stochastic model, in which, three parameters play an important role: (1) the probability of successful infection, (2) the probability of successful detection and (3) the countermeasure against the infection.

By both simulation and mathematical analysis, we found that *long-edge*[1], which connects different clusters, is an important factor favoring the propagation. The propagation can be contained in graph that does not include long-edges, while it is impossible to isolate in graph having long-edges.

Keywords: distributed systems, fault-tolerant, dynamic, transient, propagation

[1]The definitions of *long-edge* and *local-edge* are given in Section 4.5.1

Acknowledgements

First of all, I would like to express my deep appreciation to Professor Xavier Défago for guiding me in more than four years of my academic research. During this time, he has taught me so many fundamentals of a scientist. Among those, some of the most important principles are: the methodology to develop the scientific problem from precisely defining problem to clearly evaluating the result; and the strict way of doing scientific research with a strong ethic and correct motivation. These are essential building blocks of my scientific path. Besides, I am very grateful to his trust during the time when I went to do the research in Laboratoire d'informatique de Paris 6 (LIP6) at University Pierre and Marie Curie (UPMC).

Secondly, I especially appreciate François Bonnet to his invaluable collaboration and insight comments through my research. Especially, his solid knowledge in distributed computing has helped me to easily understand various difficult problems in this research area. Working with him has given me a chance to develop the skills set of formalizing the problem and proving it mathematically.

I would also like to thank Professor Maria Potop-Butucaru in LIP6-UPMC for her indispensable comments and suggestions in the collaborative work that is presented in chapter 3. During a short time working with her, I have learnt a wider picture about current trends of different classical research problems in distributed computing.

My deep thanks also go to Quentin Bramas for his suggestion and introduction to the world of infinite, which has given me a hint to develop the mathematical analysis for a problem that is much more complex to be computed in the finite world. The result of this collaboration comes out as the content of chapter 6 of this dissertation.

Reviewing and evaluating a dissertation always cost a lot of time and efforts. For this reason, I would like to thank so much the members of the committee, Professor Koichi Wada, Professor Tatsuhiro Tsuchiya, Professor Naohiro Hayashibara, Professor Kazamasa Omote, especially for their insight comments that have helped me a lot in improving the quality of this dissertation.

I also want to thank Professor Ho Tu Bao and Professor Le Hoai Bac for creating the collaborative program between JAIST and the University of Science in Ho Chi Minh City that gave me an opportunity of studying at JAIST.

Japan is a great country with admirable people in many different aspects of life. I deeply thank my generous Japanese friends, who have helped me to keep a balanced life by discovering amazing culture and the true beauty of Japan.

I would like to express my gratitute to my parents, my sister for their unconditional love; especially, to my parents who have taught me about the value of knowledge since I was a small boy. This is the key step leading me to the path of discovering knowledge. Until now, they always constantly support me by many ways.

The last words are dedicated to my wife, Huynh Thi My Phung, for being my side in the final years of my Ph.D., the most critical period; encouraging me by showing different positive aspects of life; and together with me completing long journey to become a scientist as well as a father of our expecting child.

Science is organized knowledge. Wisdom is organized life.

— Immanuel Kant (1724–1804)

Contents

List of Figures

List of Tables

Part I

Context

Chapter 1

Introduction

A distributed system is one in which the failure of a computer you did not even know existed can render your own computer unusable.
— Leslie Lamport (b.1941)

Nowadays, the computer world has migrated from single and small-scale networks of stationary machines located in a small place to very large-scale (even planetary scale) systems with heterogeneous devices from stationary to mobile, hand-held and wearable devices. The motivation of this migration originates from the demand of sharing computation power and collaborative working. According to the time, more and more instances of distributed system appear such as cloud-based systems, mobile cloud, smart-home, planetary scale sensor network, social network, network of commodity and super-parallel computers. Such real systems are based on a well-developed theory called distributed systems which established in the last decades the fundamental concepts for replicated state machine.

The current situation is that the large-scale distributed systems become more loosely connected, mobile and heterogeneous. It is the consequence when data centers are increasingly deployed and more and more the mobile devices that can connect to the systems. The mobility property is not anymore an exceptional case but becomes normal situation. This circumstance motivates the research on dealing with dynamic faults.

1.1 Distributed Systems

Distributed system is a set of computational processes (called *processes* in this thesis) that tries to work together to solve a common problem based on the partial information input from every process. Since only a part of input is provided to each process, processes must exchange the information to each other in order to complete the computation. From a methodological point of view, there are two ways of communication: directly talking

to each other and indirectly communicating via an intermediate resource. These two ways are abstracted by two models in distributed systems; which are "message passing" corresponding to direct communication, and "shared memory" corresponding to indirect communication.

In message passing model, any two processes exchange their information to each other by sending message via a communication channel connecting them. In reality, multiple examples of message passing systems are LAN, WAN or Internet, replicated data center, cloud-based systems, and sensor network. On the other hand, in shared memory model, every process maintains a local shared memory. All processes remotely access the shared memory of the other ones via a communication protocol. There are also different models of shared memory such as single-bus multiprocessor, or switched multiprocessor where processors access to a shared physical memory, or Non-uniform memory access (NUMA) [9,49] where multiple computers access to a shared file systems on disk or Midway [8] that allows a concurrent software program to use a shared variable. In this work, we concern on the message passing model.

The fundamental characteristic of distributed systems is that all processes have to collaborate to solve a common problem with a partial input. Without the collaboration, system is only the set of independent sequential processes working in parallel. Thus, no failure of single process or communication channel affects to the correctness of systems. In contrast, because of the dependency, failure that occurs in any component of a distributed system strongly affects the collaboration result.

1.2 Faults

In distributed systems, fault is the most critical issue. Fault is originated from the fact that, during the execution time, many kinds of faults can occur everywhere from process (in both hardware and software) to communication channel. For instance, any process may randomly crash or be switched off at arbitrary time, and any communication attempt can be failed or delayed arbitrarily. It is critical because a single crash failure, in an asynchronous distributed system, can lead the system to undecidable or indistinguishable situation [33].

In spite of the improvements of computer reliability, the growth of the number of processes and communication channel leads to increase the chance of failure in the system. In addition, the time to discover and recover from local faults is prolonged and significantly larger than the time to execute an algorithm.

Therefore, in a distributed system, *fault-tolerance*, the attribute enabling a system to continue working correctly in the presence of failure in some components, is one of the most important problems focused by many research in decades. Most of fault-tolerant mechanisms are based on the redundancy techniques, which use redundant correct processes to tolerate the failed one. The complexity of redundancy depends on the types and serious levels of faults.

1.2.1 Fault classifications

There are various types of faults that originate from different reasons during the execution of system. There are also multiple approaches from different points of view in classifying these types of faults.

- From dependability and secure point of view, Avizienis et al. [5], provide an exhaustive classification of various faults and failures according to eight basic aspects including *Phase of occurrence, System boundaries, Phenomenological cause, Dimension, Objective, Intent, Capacity*, and *Persistence*. *Dimension* aspect in Avizienis' classification is used to distinguish between *software* and *hardware* component. In distributed system, the interest of dimension does not come from software or hardware but comes from two main entities of distributed systems that are *process* and *communication channel*.

  Other aspects such as the *Phase of occurrence, System boundaries*, and the *Phenomenological cause* are not important because most of research and also this work concern on **operational** fault which is usually initiated **inside** the system. It also does not matter whether the fault is natural or human-made.

- On the other hand, from the distributed system point of view, fault is classified by the originality (from accidental crash, omission failure or malicious behaviors) and the component in which fault occurs (in processes or communication channels).

This dissertation classifies different types of faults according to the viewpoint of distributed systems. We concern on the behavior of faults in the system in not only the intention or originality but also in the mobility of failure. In addition to the existing classifications, we put the effort in formalizing the change of the set of failure components in different levels.

1.2.2 Dynamic faults

In distributed system, faults are in either process or communication channel. While the fault in process is usually considered as fixed set of failed processes, the fault in communication channel is more dynamic and it is considered according to the success of message exchanging.

In the failure of communication channel, the most significant model of faults is proposed by Santoro and Widmayer for complete graph [63, 65] and arbitrary graph [64] in order to study agreement problem. These models are later used to study different problem such as reliable broadcast in multiple families of graphs [24, 25, 27, 35].

Consider failure of process, most of research study different types of fault (*i.e.* Crash, Byzantine, Omission) with a static set of failed processes. Besides, a few works model the transient fault where the faulty set is changed such as Crash-Recovery [1, 56], Self-stabilizing [26] or Mobile Byzantine [36, 62], however, in these works, the size of faulty set is bounded or finally converge to a bounded number and the property of dynamicity is not formalized and considered exhaustively.

1.3 Contributions

This research aims to study the dynamicity of fault that is formally defined in Chapter 2. The formulation covers all possible changes of the set of failed components including *Permanent, Intermittent*, and *Propagating* fault. The two later types of **Intermittent** and **Propagating** fault are modeled by refining and generalizing the existing model of *malicious mobile agent* proposed by Garay [36]. Under defined models, the main contributions of this research are dealing with these dynamic faults by two mechanisms including:

Tolerating intermittent malicious fault When the size of the set of failed processes is bounded by a number t smaller than the size of system, fault can be handled by a fault-tolerant mechanism. It is usually a masking technique such as replication.

Isolating propagating fault Reversely, if there is no assumption on the size of the set of failed processes and it can increase to the size of the whole system, the fault-tolerant mechanisms cannot guarantee the availability of the system anymore. However, if we have a chance to detect the fault when it tries to propagate, in this case, instead of using a

masking tolerant technique, it is important to consider the possibility to limit the growth of failed components in a bounded size and save some remaining correct processes by a specific fault-isolation (*i.e.* a *countermeasure* of propagation). Interestingly, studying fault isolation makes a bridge connecting to the research on the defense against the propagation of an infection or information.

The more details of two mechanisms are presented more specifically in the following sections.

1.3.1 Tight bound on Mobile Byzantine Agreement

We refine and generalize the model of malicious mobile agent to simulate the Intermittent malicious faults. The refined model is more realistic than the previous ones because it ensures the balanced power between the malicious agent and the correct algorithm. Besides, it is more general because it considers more options for malicious agents to move.

In the presence of Intermittent malicious fault, we study fault-tolerant algorithms for Byzantine Agreement, a building block problem of distributed systems. We prove the tight bound on the number of processes to tolerate a bounded number of malicious mobile agents. This tight bound can be achieved by a rotating coordinator algorithm.

1.3.2 Propagating fault

When the size of failed process is not limited and can grow to the size of system, no fault-tolerant mechanism can guarantee the correctness of the system anymore. Then, in order to maintain the correctness of the system, the tolerance mechanism must limit the number of failed processes to the bounded value by a *countermeasure*. Then, we can apply a replication technique to tolerate those failed processes. This circumstance leads to a two-side probabilistic game between the growth of fault and the countermeasures based on imperfect fault detection. The effect of countermeasures in containing the infection propagation is varied in different graph topologies.

Locality graphs In the graphs where any two nodes are connected by a local-edges[1] (*e.g.* grid/torus and unit-disk graphs), the countermeasure based on killing strategy can contain the infection. When the infection probability grows, increasing the number of

[1]The definition of *local-edge* is given in Section 4.5.1

killing processes helps to contains the infection. This claim is discovered by simulation and proved for the case of infinite grid.

Graphs with long-edges Reversely, in the graphs where there is a moderated number of long-edges[2] (such as small-world, scale-free), the infection cannot be contained due to the effect of long-edges and the small diameter of graph.

1.4 Outline

The rest of this dissertation is organized as follows. Chapter 2 defines the model used in the entire of this dissertation. This model includes the definitions of two main entities (process and communication channel) of a distributed system and the fault with the dynamicity property. Chapter 3 gives the definition of the Mobile Byzantine Agreement problem, proves both possibility and impossibility results for Mobile Byzantine Agreement in complete graph. Chapter 4 introduces the two-side game between the propagation and the countermeasure with imperfect malicious detector. The game is extensively studied by simulation for some basic networks in Chapter 5. The arising questions from the simulation are formalized in Chapter 6 for infinite graphs. Then, an application of the countermeasure is introduced in Chapter 7 with the Internet topology. Finally, Chapter 8 summarizes the contributions of this research and discusses the open questions that can lead to potential research.

[2]The definition of *long-edge* is given in Section 4.5.1

Chapter 2

Model

Simplicity is prerequisite for reliability.
— Edsger Dijkstra (1930–2002)

Science is beautiful when it makes simple explanations of phenomena or connections between different observations.
— Stephen Hawking (b.1942)

A system model, on one hand, must precisely and correctly describes all entities, parameters and necessary assumptions of a system. On the other hand, it should be as general as possible to be able to open for further problems. In this chapter, we define the system model as well as the fault in the system and necessary definition used in this dissertation.

In literature, at the abstract level, a distributed system is usually represented by concurrent executions of the set of *computational processes*. During the executions, those processes exchange their information via a method of *inter-process communication*. The two dominant models of inter-process communication are indirect communication via shared memories (*shared memory* model) or direct communication via message passing (*message passing* model). Another fundamental property of a distributed system is the *timing model* that determines the relative model of time for events occurring in the system. For example, the synchronization of time in different processes, or the upper bound of delay for a message exchange. In this work, distributed system is abstracted by *synchronous message passing* model, which has the two main entities *process* and *communication channel* under a *synchronous timing model*.

Synchronous message passing distributed systems Formally, a distributed system is modeled as a set of processes $\Pi = \{p_1, p_2, \ldots, p_n\}$, in which every two process p_i, p_j communicate by exchanging message via a bidirectional communication channel c_{ij}. We call Γ the set of communication channels. The whole system evolves in a round-based synchronous time model where every numbered synchronous round is called *round*.

2.1 Processes

In this dissertation, every process p_i is a deterministic automaton with an internal state $s_i \in S$ (where S is the set of all states) and the transition function $f : S \times M \rightarrow S$, where M is the set of all available messages exchanging.

- Transition function f depends on two parameters: *the internal state* of process and *the information* received from all other processes.
- The internal state at time round r, denoted by $s_i(r)$, evolves according the execution time and transition function f.

Process p_i has a unique identity $i \in \mathbb{N}$. p_i can access a global discrete time clock according to the round number. Every synchronous round consists of three steps as follow:

- SEND: process creates messages based on its internal local state and sends those messages to all its neighbors.
- RECEIVE: process receives messages sent by its neighbors.
- COMPUTE: process computes a new state by transition function f with two parameters including the internal state in until current step and received messages from other process.

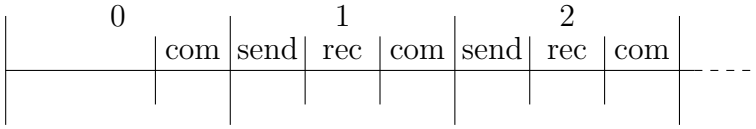


Fig. 2.1 Illustration of the three first rounds in *send-receive-compute* order.

The first round always starts with compute step as in Figure 2.1.

2.2 Communication channel

Throughout this work, all pairs of processes p_i, p_j are linked through a communication channel c_{ij} which is a reliable and timely channel with following properties:

Reliable channel: Channel c_{ij} is reliable if it satisfies the following conditions: If both process p_i and p_j are correct, then every message sent by p_i is eventually received at p_j only once and there is no message received at p_j without p_i as sender.

- NO LOSS: every message sent by p_i is eventually received at p_j.
- NO DUPLICATION: every message sent by p_i is received at p_j only once.
- NO ADDITION: No message is received at p_j without sending by p_i
- NO ALTERATION: content of a sent message is not changed by the communication channel.

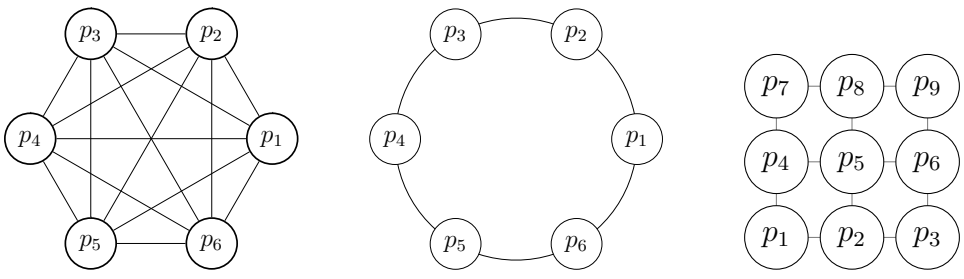
Timely channel: Timely channel is a reliable channel which satisfies the following property:

- NO DELAY: every message sent by process p_i to process p_j in round r is received in process p_j in round r.

The four properties NO LOSS, NO DUPLICATION, NO ADDITION, NO ALTERATION are the fundamental properties of any *reliable channel*. While NO DELAY property originated from the assumption of round synchronous system.

2.3 Topology

A graph $G(\Pi, \Gamma)$ represents the topology of a distributed systems where Π is the set of processes and Γ is the set of communication channel as given in previous section. In this dissertation, we consider the following basic communication topologies:



(a) complete (b) ring (c) grid

Fig. 2.2 Basic topologies

- **Complete graph** is the graph where every pair of processes is connected by a communication channel.
- **Ring** is the graph where processes are connected in a single sequence through each process in which each process is connected with exactly two neighbors.
- **Grid** is the graph where nodes are located on the square lattice. Each node is connected to the nodes at distance one in each of the four cardinal directions.

The more complex and practical topologies including some random graphs are introduced in more detail later, in the particular related chapter.

2.4 Fault

Fault is a deviation of the system execution from the expected behavior. Fault divides the system into the set of *failed components* and *correct components*.

- **Failed component** is the one whose behavior deviates from its specification in a round.
- **Correct component** is the one that works according to the specification (i.e. is not failed) in a round.

There are various kinds of faults that can occurs (*i.e.* different fault models) in a distributed system, many of them are not equivalent. In this dissertation, we classify faults under two properties: the Dimension property, corresponding to where the fault is; and the Dynamicity property, corresponding to how the set of failed components evolves according to the execution time.

2.4.1 Dimension of fault

Dimension is the location where the fault occurs. In a distributed system, there are two main entities including the process and the communication channel. Thus, fault can be located and corrupt the correctness of either process or communication channel.

There are various failure models that simulate different types of fault. In this work, faults are categorized by different levels of seriousness. These levels are adapted from definitions by Cristian [22] and Schneider [71]. However, the authors did not classify these types according to the main components of a system, which is later briefly mentioned by Timo Warns [76] and continuously developed in this work.

Process failure

A process fails when it does not work according to the specification. It could be caused by the corruption of the transition function (algorithm), internal computation state or just simply crash accidentally. All faults in process fall into one of four types as below:

- **Failstop:** Any process can fail by halting. Any other process is eventually notified of that fail [70].

  Failstop model assumes that other processors can detect and continue the task of halted processors. It is simplified to consider the fault tolerance property in small system.

- **Crash:** Any process can fail by halting. Other processes may not detect that fail [19, 33, 34].

  Crash model is studied in many research. However, in asynchronous system, a famous result from Fischer et al. proved that it is impossible to reach the agreement, a building block problem of distributed systems, even with one crash failure. Later, research on agreement of asynchronous concern on the weakest level of synchrony enabling the agreement. It will be discussed in more detail in Section 3.1.

- **Omission:** Any process can skip sending message (in sender omission model) or skip receiving message (in general omission model).

  The behavior of skipping sending message of a process can be considered and treated as similarly as those of not sending message of crash failure. Hence, in this work, we just consider the Omission failure originating from the failed attempt of communication channel in exchanging messages.

- **Byzantine Failure:** Any process can fail by acting arbitrarily without any constraint [43].

  A process is called Byzantine if it exhibits an arbitrary faulty behavior. It may crash or omit messages, but also construct fake messages that are not part of the protocol. In the extreme case, Byzantine processes can maliciously attempt to disrupt the rest of the system by participating to the protocols and sending syntactically valid messages carrying invalid information or sending inconsistent messages to different processes. An arguably weaker variant considers accidental

arbitrary faults. Although its worst case corresponds to the malicious case, accidental faults can be dealt with through simpler mechanisms. Some other research try to categorize arbitrary faults in Byzantine models into different types (Byzantine, Altruistic, and Rational) in order to reduce the cost in tolerating weaker fault [2].

Communication channel failure

A *failed communication channel* is the one that does not transmit the messages correctly according to the defined properties of a communication channel. In particular, in this dissertation, that is timely reliable channel with four properties defined in Section 2.2.

Communication channel failure affects to the result of every message transmission via that channel even when the process is working correctly. This classification adapted from the failure model in "Time is not a healer" of Santoro and Widmayer [63] which is later used in [50, 68]. Consider a transmission of message m via communication channel c_{ij}, let (α, β) be a transmission; where α is the sent message at sender and β is the received message at receiver, and δ denotes an absence of communication by process. All failure modes are classified into the following main classes:

- **Omission**: Channel c_{ij} fails then any message m_{ij} sent by process p_i is not received at p_j. A transmission fails by unable delivery to destination.

$$\alpha \neq \delta = \beta$$

  An Omission fault can occur in a communication channel for different reasons such as the collision of different messages or the noise that make message not sent and received correctly.

- **Spurious Creation**: Channel c_{ij} fails then any message m_{ij} is received at process p_j without sending event occurring at process p_i. A transmission is created in the communication channel, not a sender nodes.

$$\alpha = \delta \neq \beta$$

- **Tampering**: Channel c_{ij} fails then the content of transmitted message m_{ij} is corrupted.

$$\delta \neq \alpha \neq \beta \neq \delta$$

The content of message can be corrupted by the noise of communication channel or by the third party factor that intentionally alternate the content of message to create the misbehavior in the system.

- **Arbitrary** includes all types of Omission, Spurious Creation and Tampering. It is some time called Byzantine Channel [68].

  Similar to the Byzantine fault of process, an arbitrary fault in communication channel cannot be predicted. In different execution time, it exists as different shape of Omission, Spurious Creation and Tampering fault.

2.4.2 Dynamicity of fault

Avizienis et al. [5] mention to PERSISTENCE aspect which is defined by the presence time of fault in the host. So, the fault is either permanent fault (where the presence of fault is assumed unbounded) or transient fault (where the presence of fault is in a bounded time). Consider the set of failed entities in the system, this set is not static when fault is either a transient one or a permanent fault with the self-replication ability.

DYNAMICITY is the change of the set of failed entities in distributed system. The different levels of dynamicity include the *permanent fault*, *intermittent fault* or *propagating fault*, which are defined as follows:

Let $\mathcal{F}_r$ be the set of failed processes at round r,

- **Permanent**: When a fault occurs in a component, it will retain the component in an unbounded time.

$$\forall r, r' \quad r < r', \mathcal{F}_r \subset \mathcal{F}_{r'} \subset \Pi$$

  A permanent fault does not necessarily occurs at the starting time of the system however when it occurs, the failed state of failed component is sustained forever.

- **Intermittent**: When the members of the set of failed components change according to the execution round but the size of faulty set is bounded by a constant t.

$$\forall r, \mathcal{F}_r \subset \Pi \wedge |\mathcal{F}_r| \leq t$$

  In contradiction with the permanent fault, the intermittent fault can appear in a component then disappear in that component and appear in another component in different rounds. When the number of failed components reaches the bound,

a new intermittent failed component can appear provided that an existing one is removed. Consequently, intermittent fault does not retain the component forever.

- **Propagating**: When the members of the set of failed components change according to the execution time and the size of faulty set is not bounded by any $t < |\Pi|$.

$$\forall r, \mathcal{F}_r \subset \Pi$$

In propagating fault, the number of failed components is not bounded by any $t < |\Pi|$. Therefore, in this model, a failed component does not necessarily disappear in order to have a new failed component. We call this property is Propagating.

2.5 State of the art

2.5.1 Classifying faults

Table 2.1 shows the classification of failure models of process and communication channel according to different Dynamicity levels of failed set. There is a small different between different levels:

- In the Permanent and Intermittent fault, the goal is to guarantee the correctness of system in the presence of fault by some fault-tolerant mechanism. Normally, to guarantee the correctness, the fault-tolerant mechanism requires a bound of the number of failed components.
- On the contrary, in Propagating fault, the goal is to isolate or contain the failed components such that the number of failed components is limited in a targeted number then we can use a fault-tolerant mechanism to guarantee the correctness.

With traditional fault models that simulate the failure in process, most of them illustrate the permanent faults except two models of Crash-recovery [1, 56] and Self-stabilizing [26]. While, the Propagating fault has not been illustrated yet in both process and communication channel.

- *Crash-recovery* illustrates only the Intermittent fault for crash failure. In Crash-recovery, any process can fail by halting, but may subsequently recover after some periods of time. During the time a process is failed, other processes may not detect that fail [1, 56]. Crash-recovery model is considered in an asynchronous

	Dynamicity		
	Permanent	Intermittent	Propagating
Process	Failstop [70], Crash [33, 34]	Crash-recovery [1, 56]	this thesis [implied result]
	Byzantine [43]	Self-stabilizing Byzantine [23]; this thesis [11]	this thesis
Channel	Time is not a healer [63], Heard-Of [20]		

Table 2.1 Failure models with dynamicity property

distributed system. In this model, the process is categorized into different types such as Always-up, Eventually-up, Eventually-down, and Unstable [1]. Among which, Unstable is the most serious type.

- On the contrary, *Self-stabilizing*, defined by Dijkstra [26], simulates the Intermittent fault. A recent significant work on the Self-stabilizing Byzantine, proposed by Daliot and Dolev [23], studies the self-stabilizing property for a fundamental agreement problem of distributed systems.

  However, unlike the Crash-recovery model and all Permanent fault models, the correctness of systems is not always guaranteed. A self-stabilizing system can start arbitrarily from illegitimate states but converges to legitimate state after a finite time. In Self-stabilizing Byzantine [23], authors propose a subtle modification where in the legitimate state the number of faults in the system converges to t as the number of faults that can be tolerated by the system. Tel [73] uses two terminologies "Robust algorithm" and "Self-stabilizing algorithm" to distinguish the two types of systems in which the algorithm is permanently and intermittently correct.

On the other hand, the major models of communication channel failure ("Time is not a healer" [63] and "Heard-Of" [20]) naturally illustrate the *Intermittent* fault because the correctness of communication channel is considered according to every message exchange.

Consider now, a malicious failure, in realistic environment, even when a process is a software thread of the operating system, malicious faults are usually separated logical units that are installed into the system from the opponent besides the correct units

of process. This malicious can corrupt the correct local state of process as well as the algorithm. Moreover, it can move between processes via message exchange and self-replication ability. Consequently, the set of faulty process is dynamically changed. This change could be either Intermittent or Propagating. This is the motivation of our dynamic malicious fault models.

2.5.2 Dynamic malicious fault

The dynamic malicious fault is modeled by malicious factor (*e.g.* malware, intruder, worms, and virus) as a separate entity on the top of distributed system, it is called "worms" or "mobile agents" depending on whether it can replicate or cannot replicate itself inside the system.

Malicious mobile agent Faults are represented by the set of malicious mobile agents. In every round, there are at most t malicious agents in the system ($t < n$). An agent can occupy any process. A process is said to be failed in a given round if it is occupied by a malicious agent. Malicious agent can change local state of the faulty process and control the operation (send/receive/compute) of process. However, it cannot corrupt the correct algorithm which is assumed to be stored in a tamper-proof memory.

An important ability is that Malicious mobile agent can move from one process to another process. There are two parameters related to the movement of malicious agent:

- **The time of movement** is the time in a synchronous round that agent moves from one process to another one.

  Agent can move from one process to another one between two steps of a same round or between two steps of two continuous rounds. The detailed definition is given in Chapter 3.

- **Self-replication** is the ability of malicious agent to make a copy of itself and attach into the sending message.

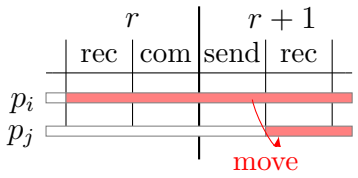


Fig. 2.3 Virus propagating via send/receive step

Consider malicious agent can replicate itself into the sending message then it can still occupy the process and copied versions of malicious agent will try to move to another process via the sent messages (see Figure 2.3), the number of malicious agents will be increased. In this case, the failed process becomes a permanent fault and the set of failed processes also expands. Considering fault as an infectious factor, the development of the set of failed processes is infection propagation.

In this dissertation, we consider two models of dynamic malicious fault caused by mobile malicious agent.

Intermittent malicious fault

The first model, called *Intermittent Malicious Fault*, considers dynamic malicious fault caused by malicious agent without the *self-replication* ability.

Under this model, like the classical works, we investigate the condition to solve Byzantine Agreement, a fundamental problem of distributed systems. The movement of malicious agent in this model causes more difficulties for system to maintain the correctness because the agent can always break the correct state of any healthy process. Moreover, by continuously moving in rounds, a malicious agent can corrupt the correct state of more than one process. The formal definition and the state of the art in Byzantine Agreement problem is presented in Chapter 3.

Propagating malicious fault

The second model, called *Propagating Malicious Fault*, considers dynamic malicious fault cause by malicious agent with the *self-replication* ability.

The detail of this model is formally defined in Chapter 4 by the two-side game between the infectious factors and its countermeasure. The two-side game is a probabilistic one with tunable parameters of the infection probability p; and the detection probability q for each infection attempt. Under this model, we investigate how the system changes according to the power of the infection and the countermeasure of the infection.

Part II

Intermittent Malicious Faults

Chapter 3

Agreement in the Presence of Intermittent Malicious Faults

An algorithm must be seen to be believed, and the best way to learn what an algorithm is all about is to try it.
— Donald Knuth (b.1938),
The Art of Computer Programming, vol.1, p.4

In this chapter, under model of *Intermittent Malicious Fault*, we investigate the case where transient state corruptions, which can be abstracted as malicious "agents", can move through the network and corrupt the nodes they occupy. This models the situation where, as soon as a faulty node is repaired (e.g., by software rejuvenation), another one becomes compromised. For more than two decades, the main case study problem in this context is Byzantine Agreement. Briefly stated, it requires processes, some of which are malicious, that start the computation with an initial value to decide on the same value. When faults are mobile the problem is known as Mobile Byzantine Agreement and requires special attention for preserving agreement once it has been reached.

3.1 Byzantine Agreement problem

The agreement problem is a building block of any distributed system. It is the heart of other activities and applications of a system (*e.g.* atomic broadcast, election, renaming) and the topic of many research in fault-tolerant property. In Byzantine Failure model, the problem is called Byzantine Agreement, which is formalized by Lamport *et al.* [43,60] and originally studied in the complete graph of stationary processes.

Byzantine Agreement: Every process tries to achieve the agreement in two steps by firstly *proposing* a value and finally *deciding* on a proposed value. The two steps must satisfy the three properties:

1. TERMINATION: Eventually, all non-faulty processes decided some value.
2. AGREEMENT: No two non-faulty processes decide different values.
3. VALIDITY: If all initially-correct processes propose the same value w, correct processes can decide only w.

Later, the problem has been studied for decades in static distributed systems under different aspects (e.g., *possibility*, *complexity*, and *cost*) in both process failure model and communication failure model. Table 3.1 gives a complete picture about impossibility and possibility bounds for Byzantine Agreement problem in different fault models. The details of these bounds are explained in the next sections.

3.1.1 Agreement in process failure models

Synchronous Initially, Byzantine Agreement problem is studied with synchronous model in "Byzantine Generals Problem" paper [43, 60] by Leslie Lamport with tight bound $n \leq 3t$ for impossibility and an algorithm with $n > 3t$. Other line of work concern early stopping condition of agreement protocol on number of rounds [16, 30]. The problem is also concerned in unauthenticated system model, with the same bounds of correct processes but cost more time to reach the agreement [29, 55]).

	Model	Impossibility result	Possibility result
Process	Synchronous [43]	$n \leq 3t$	$n > 3t$
	Asynchronous [33]	$t = 1$	$t = 0$
Channel	Time is not a healer [63]	$(n-1)$ transmission faults $\Leftrightarrow t = 1$	$< (n-1)$ transmission faults
Agent	Garay [7, 36]	open question	$n > 4t$
	Sasaki *et al.* [66]	$n \leq 6t$	$n > 6t$
	This result	$n \leq 5t$	$n > 5t$

Table 3.1 Lower and upper bounds for Byzantine Agreement with different failure models

Asynchronous For asynchronous distributed system, in a well-known result FLP [33], Fischer et al., based on the bivalent situation, proved that even with only one crash failure the agreement cannot be solved in asynchronous system. Due to this discouraging result, the later research try to circumvent with the impossibility by either using Failure

Detector (with some recent results [21,51,61]) or using probabilistic approaches [12,32]. In Failure Detector approach, authors assume an "oracle" that can provide to processes the information about faulty factor. The key point is what level of synchrony is necessary such that the agreement is achievable. In probabilistic approach, the processes use the characteristic of stochastic procedure to overcome the bivalent state because there is a chance (event very small) in which all correct processes can randomly propose and choose the same value.

Another line of research focuses on providing a practical Byzantine consensus protocol based on Quorum system such that a faster time of agreement is achievable such as Paxos [44,45], View Change [17].

3.1.2 Agreement in communication failure models

Santoro *et al.* [63,64], and later Schmid *et al.* [69], investigate the agreement problem in dynamic transmission failure models for both complete and arbitrary networks. These models assume that different communication links may randomly fail at different times. Santoro and Widmayer [63] study the k-agreement problem, where the system reaches a k-agreement if, in finite time, k processes choose the same value, either 0 or 1, with $k > \lceil n/2 \rceil$,[1] where n is the total number of processes.

Based on the bivalent argument of Fischer *et al.* [33], they state that $(\lceil n/2 + 1 \rceil)$-agreement is impossible in a synchronous system if at each time there are more than or equal to $(n - 1)$ transmission faults. It is equivalent to the whole message from one process may be corrupted. Although not explicitly stated, the impossibility applies to the mobile Byzantine model. Thus, works on Mobile Byzantine Agreement typically rely on the assumption that at least one process remains uncorrupted for $\Omega(n)$ rounds of communication.

3.1.3 Agreement in intermittent malicious fault model

In the *Intermittent Malicious Fault* model, because of the continuous movement of malicious agent, the classical Termination property cannot be satisfied. Therefore, the Agreement and Validity property must be maintained forever. The Mobile Byzantine Agreement, firstly mentioned by Reischuk [62], has regained much attention recently. Research on the problem, in synchronous systems, follows two main directions: constrained or unconstrained mobility.

[1] If $k \leq \lceil n/2 \rceil$ the k-agreement problem is trivial.

Constrained mobility. This direction, studied by Buhrman *et al.* [14], considers that malicious agents move from one node to another only when protocol messages are sent (similar to how viruses would propagate). In that model, they prove a tight bound for Mobile Byzantine Agreement ($n > 3t$, where t is the maximal number of simultaneously faulty processes) and propose an algorithm that matches this bound.

Unconstrained mobility. In this direction, which includes the work in this paper, the mobility of malicious agents is *not* constrained by message exchanges [7, 36, 57, 62, 66].

Reischuk [62] proposed a first sub-optimal solution under an additional hypothesis on the stability/stationarity of malicious agents for a given period of time. Later, Ostrovsky and Yung [57] introduced the notion of an adversary that can inject and distribute faults in the system at a constant rate in every round and proposed solutions (mixing randomization and self-stabilization) for tolerating the attacks of mobile viruses. Then, Garay [36] and, more recently, Banu *et al.* [7] and Sasaki *et al.* [66] consider, in their model, that processes execute synchronous rounds composed of three phases: *send*, *receive*, and *compute*. Between two consecutive rounds, malicious agents can move from one host to another, hence the set of faulty processes has a bounded size although its membership can change from one round to the next. Garay's model is particular in that, a process has a limited ability to detect its own infection after the fact. More precisely, during the first round following the leave of the malicious agent, a process enters a state, called *cured*, during which it can take preventive actions to avoid sending messages that are based on a corrupted state. Under this assumption, Garay [36] proposes an algorithm that solves Mobile Byzantine Agreement provided that $n > 6t$.

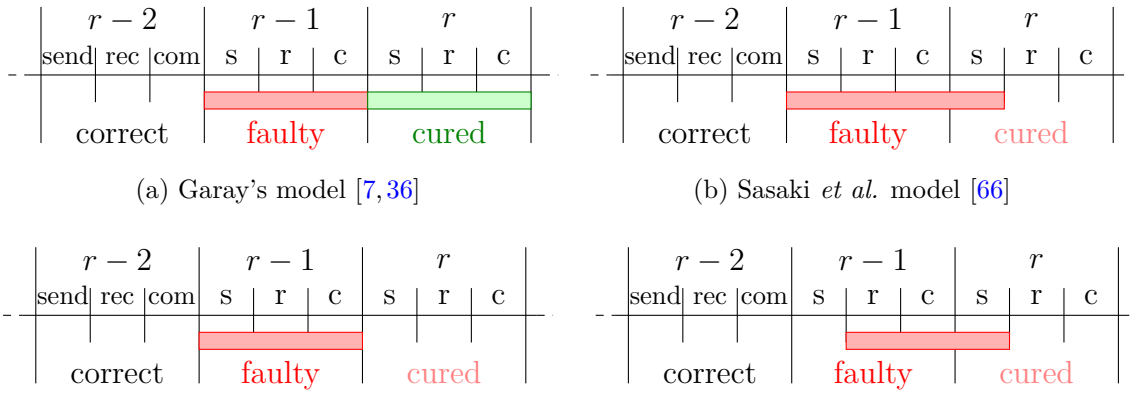
Notice that Garay's model advantages the cured processes since they have the possibility of miraculously detecting the leave of malicious agents. In the same model, Banu *et al.* [7] propose a Mobile Byzantine Agreement algorithm for $n > 4t$. However, to the best of our knowledge, the tightness of the bound remains an open question.

Sasaki *et al.* [66] investigate the problem in a different model where processes do not have this ability to detect when malicious agents move. This is similar to our model with the subtle difference that cured processes have *no control* on the messages they send. That is, messages are computed in the previous round (i.e., when the process was still faulty) and the cured process cannot control the buffer where these messages are stored, even though the process is no longer faulty. It follows that a cured process may behave as a malicious one for one additional round. They propose tight bounds for Mobile Byzantine Agreement in arbitrary networks if $n > 6t$ and the degree of the network is $d > 4t$. This work extends the tight bounds ($n > 3t$ and $d > 2t$) for Byzantine

Agreement of Dolev [28] in arbitrary networks with static faults.

3.2 Motivation

Analyzing the results proposed in [7, 36, 66], it is clear that there is a gap between how these models capture the power of malicious agents or cured processes. Garay's model [36] is biased toward the cured processes, whereas the model of Sasaki *et al.* [66] favors the malicious agent, as it can control the send buffer of a cured process even though it is no longer hosted by the process. Our research fills the gap by avoiding these biases; similarly to Sasaki's model [66], a cured process may send corrupted messages, but only computed based on the corrupted state left by a malicious agent. In particular, a malicious agent can corrupt neither the code nor the identity of the process it occupies, and a cured process always executes a correct code which ensures, for instance, that it will send the same message to all of its neighbors.



(a) Garay's model [7, 36]

(b) Sasaki *et al.* model [66]

(c) Our model[2]

(d) Buhrman *et al.* model [14]

Fig. 3.1 Graphical representation of the various fault models

The difference between the three models are subtle (see Figure 3.1) but they have important consequences (Table 3.2). Figure 3.1 depicts the effects of a malicious agent on a process. Red areas correspond to the steps controlled by the malicious agent. In Sasaki's model [66] (Figure 3.1b), a single malicious agent can corrupt a process for two rounds even though it occupies the process only for a single round. In Garay's model [36]

[2]In the related models [7, 36, 66], the movement of the malicious agent is defined between compute step of current round and send step of the next round. In our research, all possible patterns of movement are investigated. However, to analyze the subtle different of existing models, we use the common pattern among the models.

(Figure 3.1a) a cured process is aware of its current state (cured), which is represented in green. In our model (Figure 3.1c; defined in Section 3.3) malicious nodes have the same power as in Garay's model, but the cured processes may send messages with corrupted content as in Sasaki's model.

Model	Impossibility	Possibility	Byzantine vs Cured Game
Garay	open question	$n > 6t$ [36]	Advantaged Cured
	open question	$n > 4t$ [7]	Advantaged Cured
Sasaki *et al.* [66]	$n \leq 6t$	$n > 6t$	Advantaged Byzantine Agent
This work	$n \leq 5t$	$n > 5t$	No one advantaged
Buhrman *et al.* [14]	$n \leq 3t$	$n > 3t$	Virus like propagation

Table 3.2 Lower and upper bounds for Mobile Byzantine Agreement

Contribution. In this model we prove a tight bound for the agreement problem. We prove in Section 3.5 that the problem has no solution if the size of the network is $n < 5t$ (where t is an upper bound on the number of faulty agents) and propose an algorithm that matches this bound in Section 3.6. We also formalize the Mobile Byzantine Agreement problem in Section 3.4. Following the results proved in [36], our solution is also asymptotically time optimal.

3.3 Intermittent Malicious Fault Model

We consider a synchronous message-passing system as defined in Chapter 2 including additional definition of malicious mobile agent and its movement time. Let $G(\Pi, \Gamma)$ represent to the system where Π is the set of processes and Γ is the set of communication channel. In this chapter, $G(\Pi, \Gamma)$ is a complete graph.

Malicious mobile agents

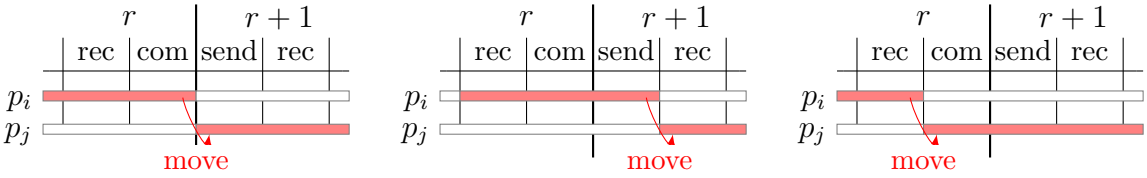
Faults are represented by malicious mobile agents that can move from process to process between two steps of the same round or between two continuous rounds. There are at most t malicious agents, with $t < n$, and any process can be occupied by an agent. A process is said to be *faulty* in a given round if it is occupied by an agent in that round. A process, which is not occupied by a malicious agent but was occupied in the previous

round, is called a *cured* process. A process, which is neither faulty nor cured, is called a *correct* process. $\mathcal{F}_r$, $\mathcal{C}o_r$, and $\mathcal{C}u_r$ denote respectively the set of faulty, correct, and cured processes at round r. For ease of writing, we also consider the combined sets of correct/cured processes as the set of non-faulty processes $\mathcal{C}_r = \mathcal{C}o_r \cup \mathcal{C}u_r = \Pi \setminus \mathcal{F}_r$.

Moving time of malicious agents The time when malicious agents can move from one process to another process is a parameter of the model. We consider three patterns of the movement of malicious agents as below:

1. Between **compute step** of the current round and **send step** of the next round (Figure 3.2a).
2. Between **send step** and **receive step** of the same round (Figure 3.2b).
3. Between **receive step** and **compute step** of the same round (Figure 3.2c).

Once the moving time is chosen, it is a parameter of model and is applied for all malicious agents in the whole execution time of the system. It means that the malicious agent cannot move with two different patterns and any two malicious agents move with the same pattern.



(a) compute/send (b) send/receive (c) receive/compute

Fig. 3.2 Time where malicious agent moves from p_i to p_j

The behavior of a faulty process is controlled by the malicious agent. In particular, the agent can corrupt the local state of its host process, and force it to send arbitrary messages (potentially different messages to different processes). However, a malicious agent cannot corrupt the identity of that process (*i.e.*, it cannot send messages using another identity), and is unable to modify the code of the algorithm (*i.e.*, the process resumes executing the correct algorithm after the malicious agent moves away). So, as suggested in [14], we assume a secure, tamper-proof read-only memory where the identity and the code are stored.

While it is possible for each non-faulty process to rejuvenate its code at the beginning of each round, local variables may still be corrupted (and of course cannot be recovered).

Therefore, in the case of cured processes the computation may be performed using a corrupted state.

Comparison with previous models. Our proposed model studies the movement of malicious agent in all possible cases. In the second case, when malicious agent moves between *send* and *receive* steps of the same round, the malicious agent will move according to one of the sent messages. Hence, it is exactly the model proposed by Buhrman *et al.* [14].

As explained in Section 3.1 and graphically depicted in Figure 3.1, the above model differs from Garay's [36] and Sasaki's [66] as follows. In Sasaki's model [66], a single malicious agent can corrupt a process for more than a round although occupying this process only for a round. In our model, once the malicious agent leaves a process, that process will execute the correct code even though the computation will be performed on a corrupted state. Differently from the Garay's model [36], where a cured process has the knowledge of its cured state and exploits it in the algorithm, in our model processes cannot access and exploit this knowledge.

Notation. In the formal definitions and proofs, var_i^r denotes the value of variable var in process p_i at the end of round r. We also use the notation $\#_w(\mathcal{W})$ to refer to the number of occurrences of w in tuple $\mathcal{W}$.

3.4 Mobile Byzantine Agreement problem

We now formally define the Mobile Byzantine Agreement problem introduced first by Garay *et al.* [36] and refined most recently by Sasaki *et al.* [66]. The definition presented here is stronger than the definition proposed by Sasaki [66] (see discussion below).

Each initially-correct process p_i has an initial value w_i. All processes must *decide*[3] a value *dec* such that the following properties hold:

1. BA-Deciding: Eventually, all non-faulty processes during a round terminate the round with a non-bottom decided value.

$$\exists r, \quad \forall r' > r \quad \forall i \in \mathcal{C}_{r'} \quad dec_i^{r'} \neq \perp$$

[3]We use a terminology consistent with the classical definition of Byzantine agreement. However, the action "decide" does not in itself guarantee a permanent decision. Indeed, due to the mobility of the malicious agents, non-faulty processes must *re-decide* the decision at the end of each round.

2. BA-Agreement: No two non-faulty processes decide different values:

$$\forall r, r' \quad \forall i \in \mathcal{C}_r \quad \forall j \in \mathcal{C}_{r'} \quad \left(dec_i^r \neq \perp \ \wedge \ dec_j^{r'} \neq \perp\right) \Rightarrow \left(dec_i^r = dec_j^{r'}\right)$$

3. BA-Validity: If all initially-correct processes propose the same value w, correct processes can decide only w.

$$\forall w \quad (\forall i \in \mathcal{C}o_0 \quad w_i = w) \Rightarrow (\forall r \quad \forall i \in \mathcal{C}_r \quad dec_i^r \in \{\perp, w\})$$

Note that specification of Mobile Byzantine Agreement given in this section is actually stronger than the definition proposed by Sasaki *et al.* [66]. They differ in two important aspects. Firstly, where we require that, after some time, all non-faulty processes decide a value at every round, their definition requires a decision only from processes that are not faulty infinitely often. Secondly, where we allow non-faulty processes to decide only on a unique non-bottom value, Sasaki's algorithm [66] allows the variable storing the decision to take arbitrary values for a finite number of rounds. In other words, our specification requires *perpetual* consistency whereas Sasaki's algorithm ensures *eventually* consistency.

We now state two lemmas, proved in earlier models [36, 63], which also apply to our model. The first lemma states a necessary condition. That condition is however not sufficient; as explained previously, a bound on the number of faults is also required.

Lemma 1 (stated in [36]; formal proof derivable from [63]) *Mobile Byzantine Agreement requires that at least one process remains uncorrupted for $\Omega(n)$ rounds of communication.*

Lemma 2 (from [36]) *Every Mobile Byzantine Agreement protocol requires $\Omega(n)$ rounds in its worst case execution.*

3.5 Upper bound on the number of faulty processes

The tight bound of Mobile Byzantine Agreement in the presence of t malicious mobile agents that can move from one process to another process between send and receive steps of the same round is proved in [14]. Interestingly, the upper bound and lower bound on the number of faulty processes given in this chapter is common for two patterns of moving time of malicious agent where malicious mobile agents can move either between

compute step of current round and *send* step of the next round or between *receive* and *compute* step of the same round.

In this section, we prove that, in the presence of t malicious mobile agents that can move according to the two patterns (*compute/send* and *receive/compute*) above, Mobile Byzantine Agreement cannot be solved with $5t$ processes or less, even if some process remains uncorrupted forever.

Sasaki *et al.* [66] proved a similar result by reduction from a well-known existing bound. From the classical bound ($n \leq 3t$) on synchronous Byzantine agreement, they could obtain their bound ($n \leq 6t$) by considering both faulty and cured processes as Byzantine.

However, we cannot use the same approach because, in sharp contrast with Sasaki's model [66] and as explained in Section 3.3, in our model, the adversary cannot entirely control cured processes.

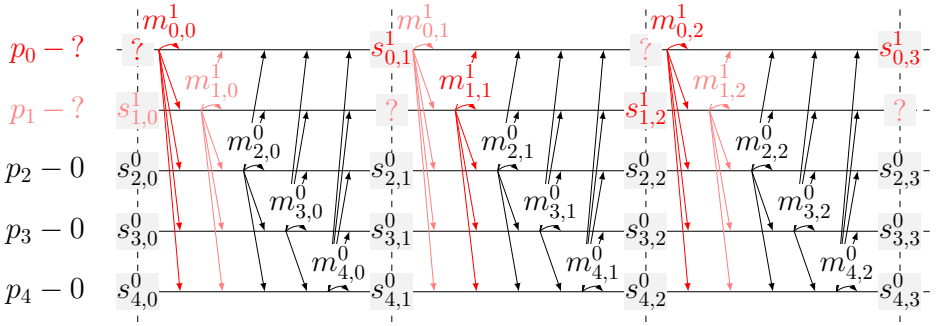
Theorem 1 *There is no deterministic algorithm that solves Mobile Byzantine Agreement in a synchronous five-process system in the presence of a single mobile Byzantine agent (even with a permanently correct process).*

Proof: The proof is by contradiction. Given a system consisting of five processes $\{p_0, \ldots, p_4\}$, where at least one is permanently correct, let us suppose that there exists an algorithm that can solve the BA problem in the presence of a single malicious mobile agent. Suppose that, in this algorithm, processes send the same message to all processes.[4] Note that, during an execution, nothing prevents a faulty processes from sending different messages to other processes.

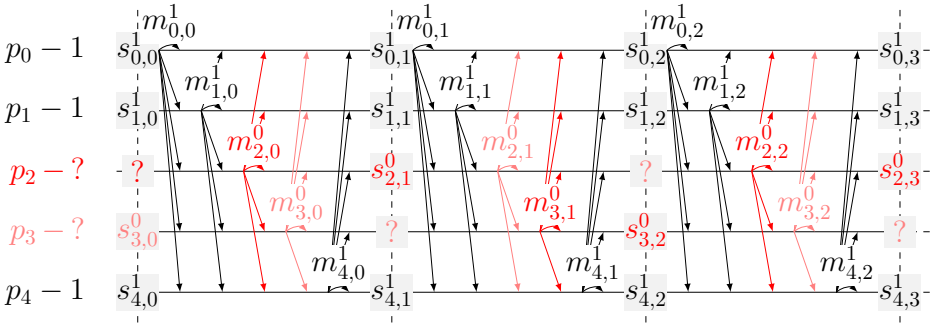
General idea. We consider three executions of this algorithm. In executions E^0 and E^1, all correct processes propose the same value; 0 and 1 respectively. The BA properties imply that, eventually, non-faulty processes respectively decide 0 and 1 in these two executions. The third execution, called E^{01}, brings a contradiction: some processes decide 0 while others decide 1.

The three executions are represented on Figure 3.3. Red (resp. light red) arrows correspond to corrupt messages sent by faulty (resp. cured) processes. The values proposed by correct processes appear on the left. Non-correct processes do not have proposed values since they may have been corrupted by the malicious agent. Vertical dashed lines separate successive rounds.

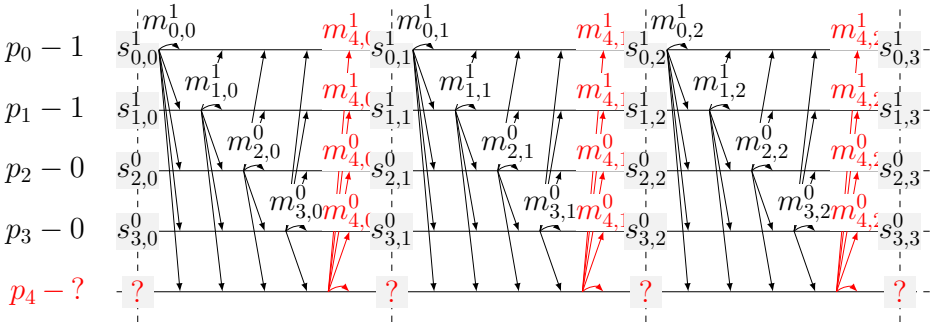
[4] If not the case, we can trivially define an algorithm that satisfies this property by combining the set of sent messages into a single message.



(a) Execution E^0 where initially-correct processes p_2, p_3, and p_4 propose value 0.



(b) Execution E^1 where initially-correct processes p_0, p_1, and p_4 propose value 1.



(c) Execution E^{01} where initially-correct processes p_0 and p_1 propose value 1 while initially-correct processes p_2 and p_3 propose value 0. Process p_4 is faulty and sends different messages to each process.

Fig. 3.3 Three executions leading to a contradiction of the existence of a BA protocol in a 5-process system with one mobile malicious agent.

(Legend: Arrows correspond to messages exchanged between processes. Gray boxes contain the new local state computed by each process at the end of each round, which is then used to send message in the following round. Red indicates actions taken by the faulty processes while light red refers to actions taken by cured processes. Vertical dashed line separate successive rounds.)

For each execution, we choose the process occupied by the single malicious agent. As required, there is at least one process which is permanently non-faulty in each execution.

Executions E^0 and E^1. In execution E^0, the malicious agent alternates between processes p_0 and p_1. In execution E^1, it alternates between processes p_2 and p_3. Processes p_2, p_3, and p_4 are initially correct and propose 0 in E^0, while processes p_0, p_1, and p_4 are initially correct and propose 1 in E^1.

For non-faulty processes, the messages sent during these executions are computed by the algorithm based on the local states of processes. For correct processes (*i.e.*, excluding cured ones), let us denote by $s^0_{i,r}$ (resp. $s^1_{i,r}$) the local state of process p_i at the beginning of the round r in execution E^0 (resp. E^1). Based on this local state, let $m^0_{i,r}$ (resp., $m^1_{i,r}$) denote the message computed and sent by a correct process p_i at round r in execution E^0 (resp., E^1).

We now define the behavior of the malicious agent. For the faulty process p_i (either p_0 or p_1) at round r of execution E^0:

- We choose that p_i sends the message $m^1_{i,r}$ (*i.e.*, the message it would have sent at the same round in E^1).
- We choose that p_i updates of receives message such that its local state is updated to $s^1_{i,r+1}$ at the end of the round (*i.e.*, the same state it would have computed in E^1).
    - In case of agent moves between compute step of current round and send step of the next round, p_i updates local state to $s^1_{i,r+1}$.
    - In case of agent moves between receive and compute steps of the same round, agent in p_i changes the received messages to $m^1_{i,r+1}$, so in the compute step the correct algorithm will compute its local state to $s^1_{i,r+1}$.

Similarly we choose that the faulty process p_i (either p_2 or p_3) at round r of execution E^1 sends the message $m^0_{i,r}$ and its state is computed to $s^0_{i,r+1}$.

Execution E^{01}. In execution E^{01}, the malicious agent always occupies process p_4. The four other processes are initially (and forever) correct. As in E^0, processes p_2 and p_3 propose 0. As in E^1, processes p_0 and p_1 propose 1. In this execution, the faulty process p_4 does not send the same message to all processes. At any round, p_4 sends the message $m^1_{4,r}$ to p_0 and p_1, but sends $m^0_{4,r}$ to p_3 and p_4.

Indistinguishability. In the sequel, we prove the following claim: *E^0 and E^{01} are indistinguishable for p_2 and p_3, and similarly E^1 and E^{01} for p_0 and p_1.* This can be proven by induction on the round number, using the following predicate $\mathcal{P}(r)$ for $r \geq 0$:

$$\mathcal{P}(r) = \begin{cases} p_0 \text{ starts round } r \text{ in } E^1 \text{ and } E^{01} \text{ with the same local state} \\ p_1 \text{ starts round } r \text{ in } E^1 \text{ and } E^{01} \text{ with the same local state} \\ p_2 \text{ starts round } r \text{ in } E^0 \text{ and } E^{01} \text{ with the same local state} \\ p_3 \text{ starts round } r \text{ in } E^0 \text{ and } E^{01} \text{ with the same local state} \end{cases}$$

The proof is only for p_0. The proofs for p_1, p_2, and p_3 are identical.

Case $r = 0$. p_0 proposes the same value in E^1 and E^{01} and therefore starts round 0 with the same initial local state, namely $s^0_{0,0}$.

Case $r \geq 0$. Let us suppose that predicate $\mathcal{P}(r)$ is true.

- p_0 is correct in E^1 and E^{01} and, by induction hypothesis, starts round r with the same local state. Therefore p_0 necessarily sends the same message, namely $m^1_{0,r}$, to all processes in round r of both E^1 and E^{01}.

  Similarly, p_1 sends the same message $m^1_{1,r}$ to all processes in round r of both E^1 and E^{01}.

- p_2 is correct in E^0 and E^{01} and, by induction hypothesis, starts round r with the same local state. Therefore p_2 necessarily sends the same message, namely $m^0_{2,r}$, to all processes in round r of both E^0 and E^{01}. Considering execution E^1, there are two cases to consider; (1) p_2 is faulty during round r and then, by construction, the malicious agent forces p_2 to send the message $m^0_{2,r}$; (2) p_2 is cured during round r, which means that it was faulty in the previous round and the malicious agent forced p_2 to start round r in the local state $s^0_{2,r}$ which implies that p_2 still sends the message $m^0_{2,r}$. In all cases, p_2 sends the same message in round r of both E^1 and E^{01}.

  Similarly, p_3 sends the same message $m^0_{3,r}$ to all processes in round r of both E^1 and E^{01}.

- p_4 is faulty in E^{01}. By construction, in each round, it sends to p_0 the same message as in E^1. It means that p_4 sends the same message, namely $m^0_{4,r}$, to p_0 in round r of both E^1 and E^{01}.

Process p_0 receives the same messages from all processes in round r of E^1 and E^{01}. Since

p_0 is correct in both executions, it computes the same new local state and starts round $r+1$, which prove $\mathcal{P}(r+1)$.

Thus by induction, the predicate $\mathcal{P}(r)$ is true for all rounds and therefore the claim holds. Since p_0 and p_1 eventually decide 1 in E^1, they also decide 1 in E^{01}. Similarly, since p_2 and p_3 eventually decide 0 in E^0, they also decide 0 in E^{01}. Contradiction. □

When $n \leq 5t$, the proof of Theorem 1 can be generalized by replacing any process appearing in the proof by a group of processes of size at most t.

Corollary 1 *There is no deterministic algorithm that solves the Mobile Byzantine Agreement problem in a synchronous n-process system in the presence of t mobile byzantine agent if $n \leq 5t$ (even with a permanently correct process).*

3.6 Algorithm for Mobile Byzantine Agreement

Given a system with t malicious mobile agents, we introduce an algorithm that solves Mobile Byzantine Agreement under the following two conditions: (1) there are at least $5t+1$ processes in total, and (2) at least one process remains uncorrupted for $3n$ consecutive rounds (see Lemma 1).

3.6.1 Description of the algorithm

The algorithm builds upon earlier ones [7,36,66] but contains some important improvements; (i) a clear separation between the deciding and the maintaining parts, (ii) a simplification of the code of the algorithm, and (iii) additional code in order to satisfy our stricter BA-Agreement property. The algorithm (lines $1-23$) consists of two main parts:

1. Deciding part: processes execute $3n$ rounds to agree on a value.
2. Maintaining part: processes execute the same round forever to keep the decided value.

Maintaining part (lines $18-23$) This part is simple and repeats forever from round $3n$. The goal is to allow cured processes to recover the decided value from correct ones, since that value may have been corrupted by the malicious agent. All processes exchange their current decided values *dec* and update their variable *dec* to the value that has been

Algorithm 1: BA algorithm (code for p_i with proposed value w_i)

```
1  Function MBA(w_i):
2      v_i ← w_i;
3      for s = 0 to n − 1 do
4          // proposing round r = 3s
5          v_i ← propose(v_i);
6          dec_i ← ⊥;
7          // collecting round r = 3s + 1
8          SV_i ← collect(v_i);
9          dec_i ← ⊥;
10         // deciding round r = 3s + 2
11         v_i ← decide(s, SV_i);
12         dec_i ← ⊥;
13     end for
14     dec_i ← v_i;
15     for r = 3n to ∞ do
16         // maintaining round
17         send dec_i to all processes;
18         dec_i ← the value received at least n − 2t times;
19     end for
20 Function propose(v):
21     PV[1..n] ← [⊥, ..., ⊥];
22     send v to all processes;
23     foreach j ∈ Π do
24         if v_j received from j then PV[j] ← v_j;
25     if ∃w ≠ ⊥, #_w(PV) ≥ n − 2t then return w;
26     return ⊥;
27 Function collect(v):
28     SV[1..n] ← [⊥, ..., ⊥];
29     send v to all processes;
30     foreach j ∈ Π do
31         if v_j received from j then SV[j] ← v_j;
32     return SV;
33 Function decide(s, SV):
34     EV[1..n][1..n] ← [[⊥, ..., ⊥], ..., [⊥, ..., ⊥]];
35     send SV to all processes;
36     foreach j ∈ Π do
37         if SV_j received from j then EV[j] ← SV_j;
38     RV[1..n] ← [⊥, ..., ⊥];
39     foreach j ∈ Π do
40         if ∃w ≠ ⊥, #_w(EV[·][j]) > 2t then RV[j] ← w;
41     if ∃w ≠ ⊥, #_w(RV) > 3t then return w;
42     else
43         c ← s mod n;
44         if ∃w ≠ ⊥, #_w(EV[c][·]) > 2t then return w;
45         return 0;
```

received at least $n-2t$ times. During each of these rounds, there must be at least $n-2t$ correct processes according to the model. If all of them send the same value (which is guaranteed by the algorithm), all non-faulty processes receive $n-2t$ messages containing this same value and thus decide accordingly.

Deciding part (lines $3-16$) This part is complex and consists of n phases of 3 rounds each. The goal is to guarantee that, at the end of round $3n-1$, all non-faulty processes have the same value v and therefore decide it (line 17). During the first $3n$ rounds, v may take different non-bottom values, which is why processes cannot decide in earlier rounds.[5]

This part uses the rotating coordinator paradigm. Recall that, in each round, there are at least $n-t$ non-faulty processes, and at least $n-2t$ correct ones. Each of the n phases are divided into 3 rounds:

- Proposing round; all non-faulty processes (at least $n-t$) end the round with at most one non-bottom value v. Consequently, it guarantees that the (at least $n-2t$) correct processes of the next round start with at most one non-bottom value v.
- Collecting round; processes exchange the values computed in the previous round and store them in array SV (the set of received values).
- Deciding round; processes try to agree on the same value v using the rotating coordinator paradigm. If the coordinator of the current round is correct during the entire phase, non-faulty processes are guaranteed to terminate the phase with the same value. Such a coordinating round exists since, by assumption, there is one process which is correct for at least $3n$ rounds.

  In the deciding round, processes exchange the array SV computed during the previous round. Based on the arrays they received, each process computes a new[6] array RV (the vector of reconstructed values). For each non-faulty process, both SV and RV contain "almost" the same values ($SV = RV$ if all processes are correct), but, as it appears in the proof, these two arrays are necessary to guarantee the correctness of our algorithm.

After the phase corresponding to a correct coordinator, all non-faulty processes have the same value v. This property will continue during all subsequent phases even if the

[5] This is different from previous papers as already mentioned in Section 3.3.

[6] Technically, as in [66], it is possible to use the same variable for both SV and RV. We choose to use two different names for the clarity of the proof.

corresponding coordinators are faulty (in fact lines $46-49$ will not be executed anymore as shown in the proof).

Additional code (lines 6, 10, 14) Usually, the variable *dec* is initialized to $\perp$ at the beginning of an algorithm. However, this value may be corrupted for any process that becomes faulty during the execution. To satisfy the BA-Agreement property, it is therefore necessary for each non-faulty process to re-initialize its variable *dec* to $\perp$ at the end of each of the first $3n$ round.

3.6.2 Proof of the algorithm

In the formal definitions and proofs, var_i^r denotes the value of variable var in process p_i at the end of round r. We also use the notation $\#_w(\mathcal{W})$ to refer to the number of occurrences of w in tuple $\mathcal{W}$. $d_H(\cdot,\cdot)$ denotes the Hamming distance which corresponds to the number of different elements between two tuples and $\mathcal{X}$ can be any arbitrary set. In all following lemmas, we suppose that $t \geq 0$ and $n \geq 5t+1$. As stated previously, we also suppose that there exists some process which remains correct for at least $3n$ rounds. Moreover, for each round, the proposed algorithm makes each process to send the same message to all processes. Consequently, in each round, only faulty processes may send different message to different processes.

We prove first a simple preliminary lemma that will be used in the main proof of the algorithm.

Lemma 3 *Let $t \geq 0$, $n \geq 5t+1$, and two n-tuples that differ on at most t values. If two values appear $n-2t$ times respectively in each tuple, then they are the same. Formally:*

$$\forall t \geq 0 \quad \forall n \geq 5t+1 \quad \forall T, T' \in \mathcal{X}^n \quad \forall x, x' \in \mathcal{X}$$
$$\Big((d_H(T,T') \leq t) \wedge (\#_x(T) \geq n-2t) \wedge (\#_{x'}(T') \geq n-2t)\Big) \Rightarrow (x = x')$$

Proof: The proof is by contradiction. Let us assume that for some $t \geq 0$ and some $n \geq 5t+1$ there exist two n-tuples T and T' such that

$$(d_H(T,T') \leq t) \wedge (\#_x(T) \geq n-2t) \wedge (\#_{x'}(T') \geq n-2t) \wedge (x \neq x')$$

Since $\#_x(T) \geq n-2t$ and $x \neq x'$, it implies that $\#_{x'}(T) \leq 2t$. Then from $d_H(T,T') \leq t$, we deduce that $\#_{x'}(T') \leq \#_{x'}(T) + t \leq 2t + t = 3t$. Finally $n \geq 5t+1$ implies that $\#_{x'}(T') \leq n-2t-1$, which is a contradiction. □

Lemma 4 *There exists a phase where all non-faulty processes terminate with the same (non-bottom) value v. Formally:*

$$\exists s \leq n-1, \quad \exists w \neq \perp, \quad \forall i \in \mathcal{C}_{3s+2} \quad v_i^{3s+2} = w$$

Proof: By assumption, there is a process which remains non-faulty for at least $3n$ rounds. Let p_c be this process and consider the c^{th} phase where p_c is the coordinator (line 47). We consider sequentially the three rounds of this phase. (To simplify the explanations; when it is not specified, faulty/non-faulty/correct processes are defined with respect to the current round.)

- Round $3c$. Each non-faulty process updates its variable v at line 5 from the value returned by the function `propose`. Since there are at most t faulty processes, the tuples PV (computed at line 28) of non-faulty processes differ on at most t values. Function `propose` returns a non-bottom value (line 29) only if this value appears at least $n-2t$ times in the tuple PV. From Lemma 3, it implies that the $n-t$ non-faulty processes can have only one non-bottom value v at the end of the round. Formally:

  $$\forall i,j \in \mathcal{C}_{3c} \quad \Big((v_i^{3c} \neq \perp) \wedge (v_j^{3c} \neq \perp)\Big) \Rightarrow (v_i^{3c} = v_j^{3c}) \tag{3.1}$$

  If any non-faulty process has a non-bottom value v_i^{3c}, let w denote this specific value.

- Round $3c+1$. All correct processes in this round were non-faulty in the previous round: $\mathcal{C}o_{3c+1} \subseteq \mathcal{C}_{3c}$. Observation (3.1) implies that all correct processes send either a bottom value or the value w (if it exists) at line 33. Only cured and faulty processes may send other values. Therefore, all non-faulty processes receive at most $2t$ values which are different from w and $\perp$.

  It also means that the arrays SV computed at line 35 by non-faulty processes contain at most $2t$ elements different from w and $\perp$ and these elements are all located at the same indexes. Formally:

  $$\forall j,j' \in \mathcal{C}_{3c+1} \quad \forall k \in \mathcal{C}o_{3c+1} \quad SV_j^{3c+1}[k] = SV_{j'}^{3c+1}[k] \in \{w, \perp\} \tag{3.2}$$

- Round $3c+2$. All correct processes in this round were non-faulty in the previous round: $\mathcal{C}o_{3c+2} \subseteq \mathcal{C}_{3c+1}$. Observation (3.2) implies that all correct processes send an array SV (line 39) satisfying the previous conditions and therefore all non-faulty

processes update their matrix EV such that:

$$\forall i \in \mathcal{C}_{3c+2} \quad \forall j, j' \in \mathcal{C}o_{3c+2} \quad \forall k \in \mathcal{C}o_{3c+1} \quad EV_i^{3c+2}[j][k] = EV_i^{3c+2}[j'][k] \in \{w, \perp\}$$

Since $|\Pi \setminus \mathcal{C}o_{3c+2}| \leq 2t$, for all such indexes k, the test of line 44 can be true only for the value w:

$$\forall i \in \mathcal{C}_{3c+2} \quad \forall k \in \mathcal{C}o_{3c+1} \quad RV_i^{3c+2}[k] \in \{w, \perp\}$$

Since $|\Pi \setminus \mathcal{C}o_{3c+1}| \leq 2t$, it implies that the test of line 45 can be true only for the value w. (Note that the test of line 45 uses the threshold $3t$ instead of $2t$ for another reason, as explained later in the proof.)

Non-faulty processes update their variable v from the value returned by the function `decide`. This value may be returned at lines 45, 48, or 49. There are two cases to consider:

1. No (non-faulty) process receives the value from line 45. All non-faulty processes update their variable v according to the test of line 48. Since, by hypothesis, the current coordinator p_c is correct, it sends the same array SV_c^{3c+1} to all processes and thus all non-faulty processes have the same line $EV[c][\cdot]$, which means that all non-faulty processes terminate the phase with the same non-bottom value v (returned at line 48 or 49).
2. At least one non-faulty process updates its variable v from a value returned at line 45. Let p_m be such a process. As stated above, p_m necessarily updates v_m to the value w and any other non-faulty process that returns from `decide` at line 45 also updates its variable v to w. It remains to prove that the remaining non-faulty processes (if any) that update their variable v from a value returned at line 48 or 49 also update to the same value w. Since p_m returns from `decide` at line 45, it means that RV_m contains more than $3t$ times the value w. Let us called J the set of indexes j corresponding to the value w:

   $$J = \{j,\ RV_m^{3c+2}[j] = w\}$$

   For all indexes of J, p_m has executed line 44 which means that the column $EV_m[\cdot][j]$ contains more than $2t$ times the value w. Since there are at most $2t$ non-correct (faulty and cured) processes; it means that at least one correct

process sends an array SV containing the value w at the j^{th} position for all indexes j of J. Formally:

$$\forall j \in J \quad \exists i \in \mathcal{C}o_{3c+2}, \quad SV_i^{3c+2}[j] = w \tag{3.3}$$

Let us consider the subset J' of J that contain all processes of J that are non-faulty in the round $3c+1$. Since there are at most t faulty processes per round, $|J'| \geq |J| - t \geq 2t$. According to Observation (3.3), for each element of J', there is at least one correct process whose corresponding entry in its array SV contains the value w. This value comes from the execution of line 35 of round $3c+1$. Since, by definition, all processes of J' are non-faulty during the round $3c+1$, it implies that all processes of J' sends the value w at line 33 of round $3c+1$. Formally:

$$\forall j' \in J' \quad v_{j'}^{3c+1} = w$$

Therefore, all non-faulty processes of round $3c+1$ have received these values w from processes of J'. It includes the process p_c which is, by assumption, always correct. It means that p_c sends at line 39 of round $3c+2$ an array SV_c that contains at least $|J'| \geq 2t$ values w. Consequently, during round $3c+2$, all non-faulty processes that has not returned from `decide` at line 45 returns at line 45 (since the test of line 45 is true). All non-faulty processes update their variable v to w, which concludes the proof of the lemma.

□

Lemma 5 *If all correct processes start a phase with the same variable v, then all non-faulty processes terminate the phase with this same value. Formally:*

$$\forall s \leq n-1, \quad \forall w \neq \perp, \quad \Big(\forall i \in \mathcal{C}o_{3s} \quad v_i^{3s-1} = w\Big) \Rightarrow \Big(\forall i \in \mathcal{C}_{3s+2} \quad v_i^{3s+2} = w\Big)$$

Proof: We consider sequentially the three rounds of phase s:

- Round $3s$. All correct processes start the round with the same value $v = w$. They all send this value at line 26. Since there are at least $n - 2t$ correct processes, all non-faulty processes receive the value w at least $n - 2t$ times and therefore all

non-faulty processes update their variable v to w. Formally:

$$\forall i \in C_{3s} \quad v_i^{3s} = w \tag{3.4}$$

- Round $3s+1$. All correct processes in this round were non-faulty in the previous round: $\mathcal{C}o_{3s+1} \subseteq \mathcal{C}_{3s}$. Observation (3.4) implies that all correct processes send the value w at line 33. Therefore all non-faulty processes obtain an array SV which contains at least $n-2t$ times the value w at the indexes corresponding to correct processes. Formally:

$$\forall j \in C_{3s+1} \quad \forall k \in Co_{3s+1} \quad SV_j^{3s+1}[k] = w \tag{3.5}$$

- Round $3s+2$. All correct processes in this round were non-faulty in the previous round: $\mathcal{C}o_{3s+2} \subseteq \mathcal{C}_{3s+1}$. Observation (3.5) implies that all correct processes send an array SV at line 39 that contains at least $n-2t$ times the value w. Therefore all non-faulty processes obtain a variable EV that contains "many" w. Formally:

$$\forall i \in C_{3s+2} \quad \forall j \in Co_{3s+2} \quad \forall k \in Co_{3s+1} \quad EV_i[j][k] = w$$

Since $|Co_{3s+2}| > 2t$, all non-faulty processes will execute line 44 for all indexes corresponding to correct processes of the previous round:

$$\forall i \in C_{3s+2} \quad \forall k \in Co_{3s+1} \quad RV_i^{3s+2}[k] = w$$

Consequently, for all non-faulty processes, the array RV contains at least $|Co_{3s+1}|$ times the value w. Since $|Co_{3s+1}| \geq n-2t > 3t$, all non-faulty processes evaluate positively the test of line 45 and return w, which means that all non-faulty processes terminate the round $3s+2$ with their variable v to value w. It concludes the proof.

□

Theorem 2 *Algorithm 1 solves the Mobile Byzantine Agreement problem in a synchronous n-process system in the presence of t mobile Byzantine agents provided that $n \geq 5t+1$ and that there is at least one process which remains uncorrupted.*

Proof: Lemmas 4 and 5 guarantee that all non-faulty processes of round $3n-1$ terminate the round with the same non-bottom value w in variable v. At line 17, all these non-faulty processes decide the value w. Starting from round $3n$, a simple induction shows

that all non-faulty processes will always decide this same value w since there are at most $2t$ non-correct processes during each round. It proves the BA-Termination and BA-Agreement properties.

To prove the BA-Validity, it is sufficient to apply Lemma 5 directly from the first round. If all initially-correct processes propose the same value, this value will be propagated until round $3n-1$ where it will be decided. This concludes the proof. □

3.7 Coping with corruption of the round counter

To the best of our knowledge, all existing algorithms on Mobile Byzantine Agreement [7, 14, 36, 66] rely on the assumption that the round counter is stored in a tamper-proof register. It is possible to remove this assumption and we propose such an improvement. However, we still need to assume that, at the beginning, all correct processes start with the same round counter (initialized at zero). The new algorithm is given in Algorithm 2.

Explanations of the algorithm The algorithm is very similar to Algorithm 1. We only add a mechanism to recover the round counter from its potential corruptions in the compute step of each round. Thus, for all rounds, each process includes its round counter into its sent messages. Furthermore, at every round, since the round counter might be corrupted until compute step, each process does not know the correct current round type (proposing, collecting, deciding, or maintaining round), and consequently which message it is supposed to send. Therefore, each process includes all possible messages, which consists in variables v, SV, and dec (Line 5 of Algorithm 2).

In the compute step, each non-faulty process deduces the correct value for round counter from the majority (at least $n-2t$) of the received value (Line 12 of Algorithm 2). According to this correct round counter, each non-faulty process is able to decide what is the current step of the Algorithm and therefore executes the same operations as it would do in Algorithm 1 (Lines 13-23 of Algorithm 2).

Proof of maintaining correct round counter Remember that during each round, there are at least $n-2t$ correct processes. By assumption, all correct processes start the first round with the same round counter. This property will be easily maintained for subsequent rounds since $n-2t$ corresponds to a strict majority of the set consisting of at least $5t+1$ processes and therefore all non-faulty processes (at least $n-t$) always obtain the same value at Line 12.

Algorithm 2: BA algorithm without tamper-proof on round counter (code for p_i)

```
1  Function MBA2(w_i):
2      v_i ← w_i;
3      r_i ← 0;
4      while true do
5          send ⟨r_i, v_i, SV_i, dec_i⟩ to all processes;
6          foreach j ∈ Π do
7              if ⟨r_j, v_j, SV_j, dec_j⟩ received from j then
8                  RC_i[j] ← r_j;
9                  PV_i[j] ← v_j;
10                 EV_i[j] ← SV_j;
11                 DV_i[j] ← dec_j;
12         r_i ← getMajority(RC_i, n − 2t);
13         if (r_i < 3n) then
14             if (r_i mod 3 ≡ 0) then
15                 v_i ← propose(PV_i);
16             else if (r_i mod 3 ≡ 1) then
17                 SV_i ← PV_i;
18             else
19                 v_i ← decide(EV_i, r_i);
20             if r_i = 3n − 1 then dec_i ← v_i;
21             else dec_i ← ⊥;
22         else
23             dec_i ← getMajority(DV_i, n − 2t);
24         r_i ← r_i + 1;
25     end

26 Function propose(PV):
27     if ∃w ≠ ⊥, #_w(PV) ≥ n − 2t then return w;
28     return ⊥;

29 Function decide(EV, r):
30     RV[1..n] ← [⊥, ..., ⊥];
31     foreach j ∈ Π do
32         if ∃w ≠ ⊥, #_w(EV[·][j]) > 2t then RV[j] ← w;
33     if ∃w ≠ ⊥, #_w(RV) > 3t then return w;
34     else
35         c ← (r/3) mod n;
36         if ∃w ≠ ⊥, #_w(EV[c][·]) > 2t then
37             return w;
38         return 0;

39 Function getMajority(V, m):
40     return value appearing at least m times in V;
```

3.8 Summary of Mobile Byzantine Agreement

We proposed a new model for Mobile Byzantine Agreement, that balances the power of correct and malicious agents. In our model, a process cannot detect its own infection and cannot instantly recover its state after the malicious agent moves away. Hence, our model gives less power to correct processes than Garay's model [36]. Recall that, in this model, a cured process can magically detect the leave of the malicious agent. In contrast, in our model, a cured process (a process that has been infected by a malicious agent) will not behave maliciously after the agent left it. That is, a cured process may send corrupted messages (computed based on a corrupted state) but it will send the same corrupted message to all neighbors. In this respect, our model gives less power to the Byzantine agents than Sasaki's model [66] where a Byzantine agent can prepare messages and control the sending of these messages even after it left that process. In our model, we prove that there is no protocol for Mobile Byzantine Agreement in synchronous networks with $n \leq 5t$. We propose then a tight algorithm which can tolerate t mobile Byzantine agents with at least $5t + 1$ processes.

In the following, we list several open questions and non trivial research directions in this area. The next step in our research is the study on the feasibility of Mobile Byzantine Agreement on arbitrary topologies. Another interesting direction would be to decrease, via randomization, the time complexity of the algorithm.

Notice that, even though our model has a self-stabilization flavor, our work is different in several aspects from the self-stabilizing Byzantine agreement of [23]. Note that in the case of self-stabilizing Byzantine agreement the studied model assumes that the Byzantine set is fixed. That is, it does not change during the execution. Also it is assumed, as in all self-stabilizing algorithms, that the system eventually becomes coherent (i.e. the communication network and a sufficient fraction of nodes is not faulty for sufficient long time period for the pre-conditions for convergence of the protocol to hold). More specifically, in self-stabilization it is assumed that during the convergence period the system does not suffer additional perturbations. In our case the system is permanently stressed due to the mobility of the Byzantine nodes. Note also that the problem solved in [23] is different since it allows the output of inconsistent decision values during transient periods.

In our model, a malicious agent can move anywhere in the network, and likely most work on the subject, we considered a fully connected topology. Sasaki *et al.* [66] have considered the case of different topologies. An interesting line of work is to generalize

to arbitrary topologies, and also to consider when the mobility of the malicious agents is constrained by a, possibly different, topology.

Finally, to the best of our knowledge, so far no investigation of Mobile Byzantine Agreement has been done in anonymous settings or networks where node identities are not unique. In these contexts, algorithms based on a coordinator are not applicable.

Part III

Propagating Faults

Chapter 4

Modeling Propagation and Countermeasures

Model building is the art of selecting those aspects of a process that are relevant to the question being asked.
— John Henry Holland (b.1929)

A theory has only the alternative of being right or wrong. A model has a third possibility: it may be right, but irrelevant.
— Manfred Eigen (b.1927)

In previous chapter, we study the *Intermittent Malicious Fault* abstracted as a mobile malicious agent in a stationary network. The size of set of faulty agents is bounded, hence, the number of faulty processes is bounded. However, in a real distributed system like networks, an important ability of the malicious factor (*e.g.* malware, virus or worm) is *self-replication*. By this ability, the growth of the set of faulty processes can increase from a single node to the whole network. The left parts of this dissertation discuss the *Propagating Fault model* abstracted by a two-side probabilistic game between the fault propagation and the countermeasure of this propagation.

4.1 A two-side game

Consider the propagation of a worm in a large distributed system (such as the Internet, a sensor network, or a social network), an epidemic starts with an arbitrary node being initially infected, then continues with every newly infected node attempting to infect its neighbors. Obviously, if nothing is done, all connected nodes are eventually infected, and the time it takes only depends on the infection rate (*i.e.*, the probability of success for each infection attempt), the topology of the network, and the location of the initial node.

Consider now that, upon an unsuccessful attempt at infection, the targeted node has a chance to detect the attempt and react to it. One simple strategy consists in propagating the detection information of virus (virus signature) such that all other nodes can detect virus correctly and the system is immune. However, this mechanism only works well when a node can always use the receiving information to detect the infection correctly. Consider, if after receiving detection information, a node cannot always detect the infection correctly (*i.e. an imperfect detection*). In this case, another simple approach is letting targeted node kill itself or cutting the communication channel with the infected node and, before this, inform some of its neighbors to do the same, so that the infection can no longer spread through them. The hope is that, given enough nodes doing the same, they could actually isolate infected nodes from sane ones, thus containing the spread. The chance of this happening depend mainly on the following factors: (1) the infection rate, (2) the detection probability, (3) the extent of the countermeasure upon detection, and (4) the topology of the network.

Such systems are notoriously known to exhibit a *critical threshold*, that is, the spread is almost surely contained if the infection rate is below the threshold, but very unlikely so if it is above that threshold. This is reminiscent of another problem, known as percolation [10]. Even after half a century of research on the topic, the value of the percolation threshold is known exactly for only very few classes of graphs. An exact characterization has turned out to be elusive even for the simplest classes of graphs, such as the grid (square lattice), for which no exact value is known for the threshold. The currently best-known approximation is 0.59274598(4) [46] and was obtained through Monte-Carlo simulations.

In this work, we look at the following question: When detection may possibly be inaccurate, is it is possible to contain the spread and, if it is, then under what conditions? Even with a topology as simple and regular as the grid and under the simplest strategy, it is unlikely that an analytic calculation for exact threshold exists. For this reason, the study has been originally studied under the impact of several containment strategies in various topologies, by relying on extensive simulations. We found that containment strategies do have an impact on the critical threshold of the system, and so does the topology. Simulation results told us that killing more nodes upon detection could help increase the critical threshold, and that preventing the spread in some topologies, such as small-world graphs, was nearly helpless. An important question is whether or not these observations can be formalized and analyzed mathematically. Interestingly, by enlarging the network and considering the problem in infinite graphs we can formalize

some observations achieved in the simulation.

Therefore, in the following part, after giving an overview of the state-of-the-art on propagation research, we define the model of propagation versus the countermeasures with an imperfect detection. Under this model, we study the extensively via simulation the propagation versus countermeasures in different basic graph topologies including torus grid, and some realistic topologies such as unit-disk graph representing to sensor network, small-world representing for social network. The arising questions from simulation are analyzed mathematically in the context of infinite graphs in Chapter 6. Finally, an application of proposed countermeasures in the Internet is also presented in Chapter 7.

4.2 Existing propagation models

The propagation has been widely studied from human and biology (epidemiology) to computer virus (virus spread), and information propagation. In this section, we review the related work of propagation according to two parts:

- The first part reviews the model includes only **propagation** in a network. In these works, researchers try to propose a model that can reflect exactly the evolving of individuals in a disease or computer in a spread of virus.
- The second part discusses different **defense mechanisms** based on a perfect detection ability against the propagation. The defense mechanisms are classified into two classes including *proactive* and *reactive*.

4.2.1 Propagation

Epidemiology

Starting from the epidemiology in human community, much research has been conducted on modeling an epidemic. The first mathematical models appeared in the 18th century, but modern models were essentially developed in the middle of the 20th century by different scientists including Kermack et al. [41], Anderson et al. [3, 52]. While original models did not study the spread according to the geographic distributions, more recent epidemic models consider it in geographic topologies, such as an infinite grid [37]).

In each model, individuals are assumed to be in various states of the propagating disease. Letters $\mathcal{S}, \mathcal{I}, \mathcal{R}$, and $\mathcal{E}$ are usually used to denote these states. As being used

in existing models, in this section, the letters are used to denote:

- $\mathcal{S}$: Susceptible
- $\mathcal{I}$: Infected
- $\mathcal{R}$: Removed from the disease with immunity or death
- $\mathcal{E}$: Exposed disease, in latent period

The original model proposed by Kermack and McKendrick is SIR model in which an individual during the spread of disease could be consequently in three states:

$$\mathcal{S} \rightarrow \mathcal{I} \rightarrow \mathcal{R}$$

It means that an individual firstly is susceptible ($\mathcal{S}$), when it is infected by disease, it changes to infected ($\mathcal{I}$), and finally is removed ($\mathcal{R}$) from the considered group by either immunization or death. Later, the model is extended in two different directions in which researchers consider the spread with:

1. *The recovery without immunization ($SIRS, SIS$)*: The model is extended with state S at the end representing for the transition from infected ($\mathcal{I}$) or immune ($\mathcal{R}$) state to susceptible ($\mathcal{S}$). In those models, after recovering from the infection, the recovered individual finally changes to susceptible state, from which the disease could infect him again.
2. *An incubation period in which individual is invisibly infected by the disease (with two models of $SEIS, SEIR$)*: In these models, the transition $\mathcal{S} \rightarrow \mathcal{I}$ is replaced by $\mathcal{S} \rightarrow \mathcal{E} \rightarrow \mathcal{I}$. It means that individual after getting the disease will be in a latent period before the infection is exposed.

Computer viruses

In computer virus research, most of works aim to study either the epidemic threshold based on existing model (SIS) or the exact behavior of virus and worm in the real network.

Epidemic threshold: Kephart and White [39] propose a birth-death model, which is equivalent to SIS model, to study the spread of computer viruses in homogeneous sparse graphs and conclude that a pandemic occurs only when the infection rate exceeds a finite threshold that depends on the connectivity of the network (phase transition). They also extend their model to allow doing a virus scan [40].

Later, many works improve the results on the birth-death model and compute new epidemic thresholds. Pastor-Satorras and Vespignani [58, 59] look at the dynamics of epidemics in power-law scale free networks for which they find the critical threshold. Chakrabarti et al. [18] study an epidemic model in multiple of real computer networks (network of email accounts, network of Autonomous Systems (AS)) and find that the propagation threshold related to the eigenvalues of the adjacency matrix of the network. Lately, Van Mieghem et al. use mean field approximation to transform from individual random infection rates into an average infection rate [74, 75]. Their model is called N-intertwined Markov chain.

Modeling actual virus/worms: In another direction, after the incident of Code Red in 2001, a lot of research look for the most accurate model that reflects the spread of different kinds of viruses in the Internet. They propose different models from the *scanning worms* [72, 77, 80] to the *event-based* worms [78, 81]. Here, the question is to predict, as accurately as possible, the evolution of the expected number of infected entities in the network after the virus starts propagating.

Event-based worms: By the development of the Internet, many large-scale networks of users (such as social network (*e.g.* Facebook, Twitter) or public email networks (*e.g.* Yahoo! Mail, Google Mail)) are created. Such kinds of networks are exactly the environment supporting the propagation of event-based worms. Therefore, the event-based worm is gaining more and more interest of research [78, 81]. The stochastic model proposed in this work helps us studying the possibility to defense the propagation of an event-based worm.

4.2.2 The defense against a propagation

In virus defense area, there are many works studying how to contain or quarantine the virus or worms in different network environments [54, 79, 81]. The containment strategies can be classified into two main classes; *proactive* or *reactive*. In the first one, some nodes are initially immune to the virus and only other nodes can be infected. In the latter,

all nodes are initially susceptible, but eventually any node may become immune if it detects the virus (or receive some informations from other nodes).

Moore et al. [54] proposed a model for scanning worms in complete graph topology and give a comparison between two reactive strategies; (1) *blacklisting*, upon detection of an infection, a node adds the attacker into a blacklist; and (2) *filtering-content*, upon detection of a virus, a node transmits its signature to all other nodes. They assume that when a node detects an infection, the information (blacklisted IP address, or virus' signature) will be available to all other nodes after some time. They study the efficiency of both strategies when this delay varies. Under this model, filtering-content strategies perform better than blacklisting strategies.

Later, Zhou et al. [79] studied the containment of worms in peer-to-peer network when an infected node randomly selects some of its neighbors to attack. Among the peers, some are proactively immune (called guardian nodes) and can detect any attack; the others are always infected (no chance of detection). With this model, by simulation of some classical peer-to-peer networks (Gnutella and KaZaA), they study the relationship between the final fraction of infected nodes and the fraction of guardian nodes. Not surprisingly, they found that choosing for guardians the nodes with a large number of neighbors helps to contain the infection.

Zou et al. [81] analyze the existing models of computer viruses spread and propose an event-based model to study the defense of email worms spread in three topologies power-law, small-world, random graph. They first show that usual mathematical models largely overestimate epidemic spreading speed, justifying the need of simulations. Moreover they also introduce and study two proactive strategies: (1) *random immunization* in which immune nodes are randomly chosen and (2) *selective immunization* where a given percentage of the most connected nodes are immune.

In all these work, immune nodes can always detect virus attack successfully.

Flooding and Percolation This containment problem is equivalent to probabilistic broadcast (or information flooding) albeit with an opposite objective. Sasson et al. [67] and later Hu et al. [38] both study the question and show the relationship with the theory of percolation [10].

4.2.3 A picture of existing propagation models

In summary, the propagation and countermeasure of propagation are studied in both human and computer community. The researchers try to solve different problems and

Works		State transition				Mechanism		
		Model	Infection	Recovery	Incubation	Propagation	Countermeasure	Detection
Human	Kermack et al. [41]	SIR	✓			✓		
	Bailey [6], Anderson et al. [4]	SIS	✓	✓		✓		
	Anderson et al. [3, 52]	$SEIS$	✓		✓	✓		
Computer	Kephart & White [39, 40], Pastor-Satorras & Vespignani [58, 59], Chakrabarti et al. [18], Van Mieghem et at. [74, 75]	SIS	✓	✓		✓		
	Moore et al. [54], Zhou et al. [79], Zou et al. [81]	SI	✓			✓	✓	Pf
	This work	SI	✓			✓	✓	Im

Table 4.1 Propagation models

Legend: Pf denotes a perfect virus detection. Im denotes an imperfect virus detection

questions related to the epidemic threshold, modeling real worms or viruses, and some countermeasure against the spread of virus. Table 4.1 provides the overview of this line of research according to two points of view:

- **State transition** is the evolution of the state of a member of network during a spread. In this point of view, the evolution is classified into (1) immediately infection, (2) infection and recovery, (3) incubation time before really infection.
- **Mechanism** is the different factors of the problem that are studied. The mechanism aspect consists of *propagation*, *detection* and *countermeasure* based on the detection. In the table, Pf denotes a *perfect detection* in which after some period, all nodes or some nodes of network have ability that always detect virus correctly. While Im denotes an *imperfect detection*, where all nodes can detect virus but the detection is not always successful.

All existing models are widely used in many research of infection propagation in a long time. Each model is suitable for some specific diseases and problems. The choice of model is decided by several questions that are summarized as follow:

Immune or not immune (SIR vs SIS and $SIRS$) Firstly, in human epidemiology, Kermack and Mc.Kendrick propose SIR as a compartmental differential-equation model that exactly reflects the state transition of individuals in the spread of many diseases such as *measles, mumps, rubella*; where patient's state changes from $\mathcal{S} \rightarrow \mathcal{I} \rightarrow \mathcal{R}$. In such kind of infectious disease, the patients either die or become immune with the infection by a vaccine.

However, SIR does not reflect exactly some other disease such as *flu, common cold* in which the patient after recovering from the infected state is not immune with the disease. This problem is studied under SIS model by Bailey [6] and Anderson et al. [4]. In this model the patient after recover can be immediately affected by the same disease ($\mathcal{S} \rightarrow \mathcal{I} \rightarrow \mathcal{S}$). Besides, there is another modification called $SIRS$ where an individual will have a temporary immune period because of the medicine before being susceptible again ($\mathcal{S} \rightarrow \mathcal{I} \rightarrow \mathcal{R} \rightarrow \mathcal{S}$).

Incubation with $SEIS$, $SEIR$: Anderson and May [3,52] found that some parasites need an incubation period in the hosts before really infected them. In this period, the individual already has (exposes) the virus but has not yet been infected. The length of incubation period is varied depending on the ecology of interactions between parasites and hosts. Therefore, it is necessary to study separately the rate of which virus can locate in a healthy individual (*i.e.* transition rate from susceptible to exposed ($\mathcal{S} \rightarrow \mathcal{E}$)), and then the rate of which virus can really infect a patient (*i.e.* the transition rate from exposed to infected ($\mathcal{E} \rightarrow \mathcal{I}$)).

However, unlike a biology virus, a computer virus does not require an incubation period to infect the hosts, computer is immediately infected after a successful attack. Therefore, models related to incubation ($SEIS, SEIR$) are not used in the research area of computer viruses.

Recovery or non-recovery (SIS or SI): The main difference between different models of computer virus spread is that if it includes recovery or not. Most of studies in computer virus that include the recovery use the SIS model. It comes from the reason that a computer is never completely immune with virus. Thus, SIR is not commonly

used. On the other hand, with the research that do not study recovery, SI is used as a simplified model of SIS and SIR. Normally, such research only study how far and how fast a virus can propagate especially in the presence of virus detection [54, 79, 81].

Perfect or Imperfect detection: As mentioned in Section 4.2.2, most of existing works that consider a chance of detecting the infection also study the countermeasure against the propagation based on a perfect detection [54, 79, 81]. However, with a polymorphic (such as Sality [31]) or metamorphic virus, it may not be always possible to detect the virus correctly. In this context, several detectors are introduced in [13, 42, 47]. We call *imperfect detection* the ability to detect a virus but not always successfully.

If we only have an imperfect detection, what should the network do to mitigate a spread? We aims to study this question by varying the probability of detection. Therefore, the proposed model in this work consists of (1) the spread of fault according to SI model (without recovery mechanism) and (2) possible countermeasure based on (3) an imperfect detection. One of the open directions of this research is to consider the problem with the recovery mechanism (*i.e.* SIS model).

4.3 Model

As a normal distributed system, the network consists of a set of processes and a set of communication channels. The computation procedure of each process, the properties of communication channel and the execution of the whole network restrict the system model defined in chapter 2.

Let $G = (\Pi, \Gamma)$ be a (potentially infinite) connected undirected graph, where V is the set of vertices (also called nodes), and Γ the set of edges. The graph G models a network, in which node i represents a process and edge $e_{ij} \in \Gamma$ represents a communication link between the two processes $i, j \in \Pi$. For any node $i \in \Gamma$, let $N[i]$ be its closed set of neighbors (*i.e.* including i itself). By an abuse of notation, for any set S of nodes, let $N[S]$ be defined as $N[S] = \bigcup_{i \in S} N[i]$.

Starting from round $r = 0$ the system evolves for an infinite number of rounds. Initially, one node is infected by the virus. Then, in each round, the virus can propagate from infected nodes to their neighbors by sending message in *send/receive* steps. After receiving message, in *compute* step, node will change its state according to the two probabilistic parameters. The detail of infection propagation is described in section 4.3.3.

4.3.1 States of nodes

At anytime, a node is in one of the following states; *susceptible*, *infected*, *killed*, or *sane*. The state of node i at the end of timeslot r is denoted by $s_i(r)$ with $s_i(r) \in \{sus, inf, killed, sane\}$. The meaning of these states is explained below:

- *Infected*: a node which is infected with the virus.
- *Killed*: a deactivated node; it does not participate in the network anymore. Deactivation can be decided by itself or may be triggered by one of its neighbors. Intuitively, a node decides to deactivate when it detects the infection in order to mitigate the propagation.
- *Susceptible*: a node which is neither infected nor killed, but can still be infected in the future. A susceptible node is either a neighbor of an infected node, or there exists a path of susceptible nodes toward an infected node.
- *Sane*: a node which is neither infected nor killed, and has no risk of being infected in the future. A sane node is isolated from all infected nodes.

The state $s_i(r)$ of node i at the end of round r also corresponds to the state of i at the beginning of round $r+1$. Initially, only one node is infected and all other nodes are susceptible.

The model does not consider recovery mechanisms; hence when a node is infected, killed, or sane; it is a permanent state. At the end of an execution, there is no more susceptible nodes. Either the entire system is infected, or it is partitioned by killed nodes into infected and sane components. As stated in the introduction, the goal is to contain the spread and possibly maximize the number of sane nodes.

4.3.2 States of edges

At any time, an edge is in one of the following two states; *active* or *cut*. The state of edge e at the end of timeslot r is denoted by $s_e(r)$ with $s_e(r) \in \{active, cut\}$.

An edge is considered *cut*, when one (or both) of its extremities decides to stop using the corresponding communication link. Such unilateral decision (to cut a link) can be taken when a node detects or suspects one of its neighbors to be contaminated by the virus.

The state $s_e(r)$ of edge e at the end of timeslot r also corresponds to the state of e at the beginning of timeslot $r+1$. Initially, all edges are active.

4.3.3 Infection propagation

In the network, the worm can propagate from infected nodes to susceptible neighbors by sending the message via active communication links. The contamination is probabilistic and relies on two parameters p and q, respectively the *infection probability* and the *detection probability*. Both parameters are uniform for all edges. The risk of contamination of a susceptible node depends on its number of infected neighbors.

Single attack

When, at the beginning of a round r, a susceptible node j has only one infected neighbor i connected via an active edge, the worm may propagate from i to j. Formally, there can be a contamination from i to j if:

$$s_i(r-1) = inf \quad \wedge \quad s_j(r-1) = sus \quad \wedge \quad s_{e_{ij}}(r-1) = active$$

When these conditions are satisfied, there is an attack from i to j which leads to an infection, a detection, or a *status quo*:

- The attack is successful with probability p (where $0 \leq p \leq 1$). In this case, the node j becomes infected and $s_j(r) = inf$.
- The attack is detected with probability q (where $0 \leq q \leq 1 - p$). In this case, the node j detects the attack; it does not become infected. Depending on the *containment strategy*, the reaction will change. The node may decide (1) to sacrifice itself ($s_j(r) = killed$), (2) to cut the infectious link ($s_{e_{ij}}(r) = cut$), or (3) to follow a more complex strategy, as described later.
- The attack is neither successful nor detected with probability $1 - p - q$.

Multiple attacks

Multiple attacks can occur when a susceptible node j has more than a single infected neighbors connected with active edges. In our model, the multiple attacks are considered sequentially. Since the probabilities p and q are uniform for all edges, the order of attacks is not relevant.

- If a given attack is successful, the subsequent attacks are not considered. The targeted node becomes infected.

- If a given attack is neither successful nor detected, this attack is ignored and the subsequent attacks have to be considered.
- If an attack is detected, the consideration of subsequent attacks depends on the containment strategy (countermeasure) followed by the targeted node:
  - If it sacrifices itself, this sacrifice is prioritized compared to other attacks: the subsequent attacks are not considered. The targeted node changes to the killed state.
  - If it does not sacrifice itself (for example, it only cut the link carrying the attack): the subsequent attacks are considered. The targeted node may detect other attacks and/or may become infected from another attack.

4.3.4 Countermeasures

This paper studies the possible actions that a susceptible node can take when it detects an infectious attack. In the following, we call a *detector* a node that detects an attack. We propose two families of strategies a detector can follow to mitigate the spread.

- *Killing strategies (K-strategies):* The detector decides to sacrifice itself by its deactivation (*i.e.* it updates to the killed state). In some strategies, it may also trigger the deactivation of some of its neighbors. The success of these additional deactivations depends on the state of these neighbors; infected nodes ignore the deactivation messages and remain in the infected state.
- *Cutting strategies:* The detector decides to cut the communication link carrying the attack. In some strategies, it also tries to cut some other links by sending link-removal messages to its neighbors. Similarly, these additional cuts depend on the state of the neighbors.

4.3.5 Modeling event-based worms

The stochastic model proposed in this work can be used to model the event-based worms propagated in the network of user accounts. This paragraph describes the propagating procedure of email worms as an example. An email worm is sent from a malicious account to all addresses in its mailing list as a message. Consider this message with a recipient, this scam message can go across the scam filter, then is checked by a recipient with a particular infection probability (p), whereas it is detected by the filter and a user

with a detection probability (q). Whenever successfully attack an account, the email worm retrieves all addresses in the mailing list of infected account and try to attack all those addresses. The propagation is continuously occurred until either every account in network is attacked at least once or a countermeasure is applied for all susceptible accounts.

4.4 Containment strategies

This section describes the containment strategies that have been analyzed in this dissertation.

4.4.1 Killing strategies

We propose a family of killing strategies that differ in the number of nodes that are deactivated/killed. For all strategies, the detector deactivates itself, and in addition to its own sacrifice, it tries to deactivate its neighbors up to a given hop count. The three main strategies are $K0$-Hop, $K1$-Hop, and $K2$-Hop but it can easily be generalized up to Kx-Hop as defined in Table 4.2.

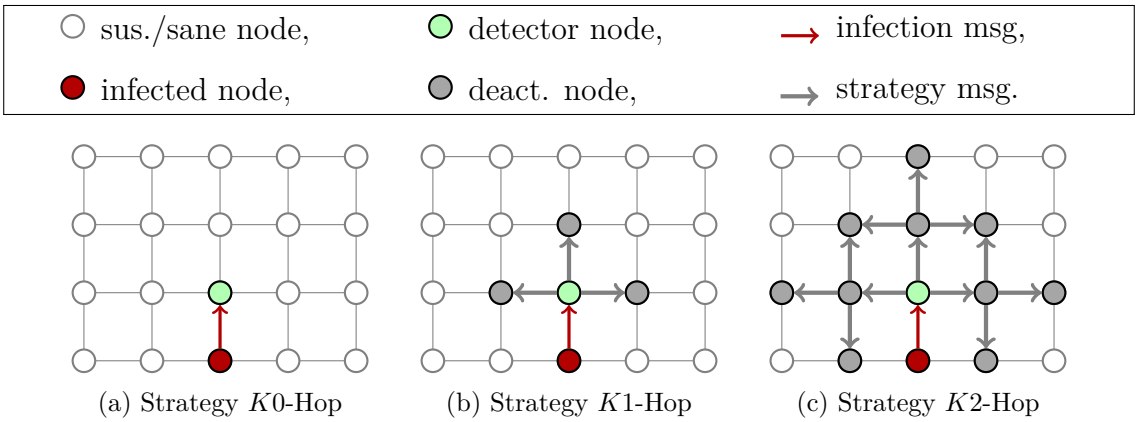
Name	List of deactivated vertices
$K0$-Hop	detector only
$K1$-Hop	detector and its 1-hop neighbors
$K2$-Hop	detector, its 1-hop and 2-hop neighbors
Kx-Hop	detector and all its neighbors up to x-hops

Table 4.2 Description of the killing strategies Kx-Hop

In order to deactivate its neighbors, the detector sends deactivation messages. The behavior of a node receiving such a message depends on its current state:

- A susceptible node sacrifices itself as requested by the message, after having potentially forwarded the message as requested by the strategy.
- An infected nodes ignores the message; it does not forward the message and does not sacrifice itself.
- A deactivated node does not receive the message; it does not forward the message and is already deactivated.

The Figure 4.1 represents the three main killing strategies on a grid topology (for simplicity). The green node detects the infection from the red node and tries to deactivate all gray nodes. One should note that these deactivations are not always successful (see example of Figure 4.2).



(a) Strategy $K0$-Hop (b) Strategy $K1$-Hop (c) Strategy $K2$-Hop

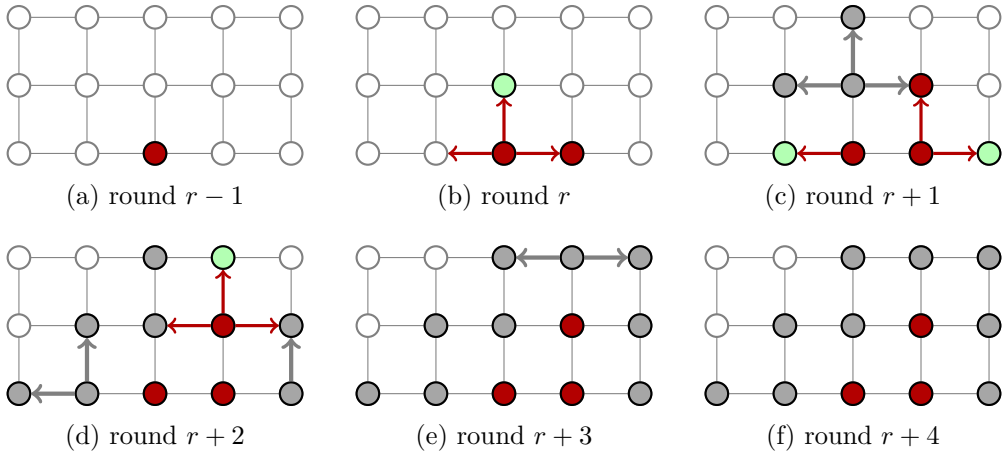
Fig. 4.1 Representation (in a grid) of the three killing strategies

Figure 4.2 depicts a step-by-step execution of strategy $K1$-Hop in a 3×5 grid. An infected node attacks its neighbors at time round r; the top neighbor detects the attack, the right one becomes infected, and the left one is unaffected. Following strategy $K1$-Hop, the top neighbor (detector) sends deactivation messages to its own neighbors and deactivates itself; two of its neighbors (top and left) react accordingly, but the right neighbor ignores it because it was infected in the meantime. In contrast, during time round $r+2$, a node deactivates itself after receiving both a deactivation and an infected message. The execution continues until timeslot $r+4$ when the network is partitioned and the spread is contained.

We propose a family of cutting strategies that differ in the number of links that are cut. For all strategies, the detector cuts the link carrying the infection. It also sends messages to *some* of its neighbors to ask them to cut *some* of their links. The set of links to be cut depends on the strategy and are the ones located *near* the infected node. The main strategies are $C0$, $C1$, $C2$, and C_{loc} and are defined in Table 4.3.

4.4.2 Cutting strategies

To help understanding the (complex) formal definitions, we represent the main strategies in Figure 4.3. The node I corresponds to the infected node and D to the detector. In $C0$, the detector cuts only the link between I and D. In $C1$, the detector sends a message



(a) round $r-1$ (b) round r (c) round $r+1$

(d) round $r+2$ (e) round $r+3$ (f) round $r+4$

Fig. 4.2 Example of an execution of the strategy $K1$-Hop

Table 4.3 Description of the cutting strategies

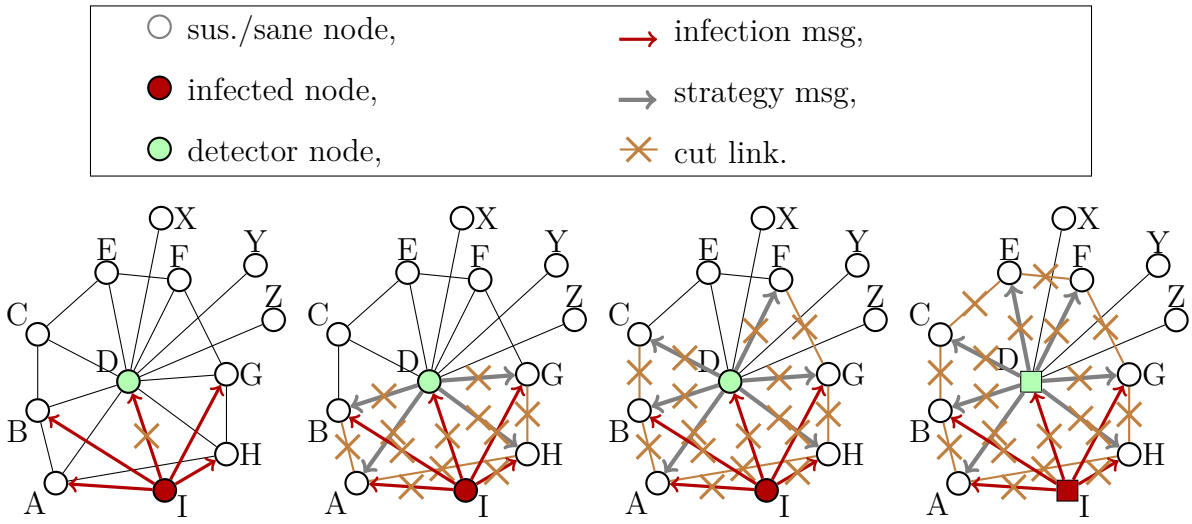
Name	Reaction of detector (node D) after infection (from node I)
$C0$	Cut the link carrying the infection[a]
$C1$	Ask nodes from $S_1[I] = N[I] \cap N[D]$ to cut all links between them
$C2$	Ask nodes from $S_2[I] = N[S_1[I]] \cap N[D]$ to cut all links between them
Cx	Ask nodes from $S_x[I] = N[S_{x-1}[I]] \cap N[D]$ to cut links between them
C_{loc}	Ask all local neighbors[b] of D to cut links between them

[a] Defining $S_0 = \{I\}$ allow to describe $C0$ and $C1$ similarly to CX.
[b] *i.e.* the neighbors coming from the underlying unit-disk graph

to nodes A, B, G, and H since they are the common neighbors of I and D. In $C2$, the detector sends additional messages to C and F since they are respectively neighbor of B and G. In all strategies, including C_{loc}, the detector does not send a message to nodes X, Y, or Z since they are not *close* to I.

There are two important notes about these strategies:

- Except $C0$, other strategies are meaningful only for the unit-disk and small-world topologies. Indeed, in the grid, two adjacent nodes do not share any common neighbor; therefore no additional edge will be cut and all strategies are equivalent to $C0$
- We consider only cutting strategies where nodes receiving a cutting-request also



(a) Strategy $C0$ (b) Strategy $C1$ (c) Strategy $C2$ (d) Strategy C_{loc}

Fig. 4.3 Representation of the main cutting strategies

cuts the link carrying the request. At first glance, this is not a useful property; in Figure 4.3d, it seems (and it is) not relevant to cut links between D and E or between D and F. However it is necessary to consider only strategies with this restriction since otherwise the infected nodes could also send cutting-requests in order to weaken the network.

4.4.3 Restriction of containment strategies

Containment strategies weaken the connectivity of a network by sacrificing the node and cutting the communication links. Without any restriction, infected node can exploit to weaken the healthy part of the network. Therefore, all strategies in this study are defined with a restriction that upon receiving a killing/cutting message the receiver must disconnect with the sender by killing itself or cutting its communication link to the sender. By this restriction, the adversary cannot exploit the strategies because it has nothing to gain from this because it only contributes in isolating the infected node from the sane ones. This behavior is equivalent to allow the sane neighbor to detect an attack with probability 1.

4.5 Topologies

This dissertation considers to isolate the propagating fault in various theoretical graphs including square torus, unit-disk graph, small-world, and scale-free graph. Among which, square torus is a regular graph that allows to understand more easily behaviors of different strategies. Whereas, other three graphs are random topologies which are more interesting from practical point of view. In particular, unit-disk graph represent to wireless sensor networks, while small-world and scale-free graphs are used to model different types of social networks. We define below more precisely each of these graphs.

Regular square torus: We consider two-dimensional tori of size $m \times m$, with m^2 nodes. A torus corresponds to an undirected square grid where the left side is connected to the right side and, similarly, the top and bottom sides are also connected. Contrary to a grid, the torus exhibits a uniform structure "without sides."

Unit-disk graph: We consider two-dimensional random unit-disk graphs $UDG(n, x \times y)$. A set of n nodes are deployed uniformly randomly on an $x \times y$ rectangle area. There is a communication link between two nodes if they are at Euclidean distance at most 1. Let Δ be the average degree of a $UDG(n, x \times y)$, we use the notation $G_{UDG}(n, \Delta)$ to refer to a unit disk graph having average degree Δ. Parameters are chosen such that the generated graphs are connected with high probability.

Small-world graph: We consider small-world graphs that are generated from unit-disk graphs. Given a unit-disk graph $G_{UDG}(n, \Delta)$, we add ℓ random edges for each node. These additional edges are chosen uniformly randomly. We use the notation $SW(n, \Delta, \ell)$ to represent such generated small-world.

Beside normal properties of a graph (*e.g.* degree, diameter), in this research, we also concern on another characteristic of graphs that is discussed in the following part.

4.5.1 Locality of edges

Given graph $G(\Pi, \Gamma)$:

For any node $i, j \in \Pi$, let $N[i]_j$ be the set of neighbors of i excluding node j, and $d_G(i, j)$ be the hop distance between i and j in graph G, where $d(i, j) = \infty$ if i and j are not in the same connected component.

For any subset S of Π and any node $i \in \Pi$, let $d_G(i, S)$ be the smallest hop distance between i and an arbitrary node in set S in graph G, $d_G(i, S) = \infty$ if S is empty. We call $d_G(i, S)$ distance from i to set of nodes S in graph G.

$$\forall i \in \Pi, \forall S \subset \Pi, d_G(i, S) = \begin{cases} \min\limits_{v \in S}(d_G(i, v)) & \text{if } S \neq \emptyset \\ \infty & \text{if } S = \emptyset \end{cases}$$

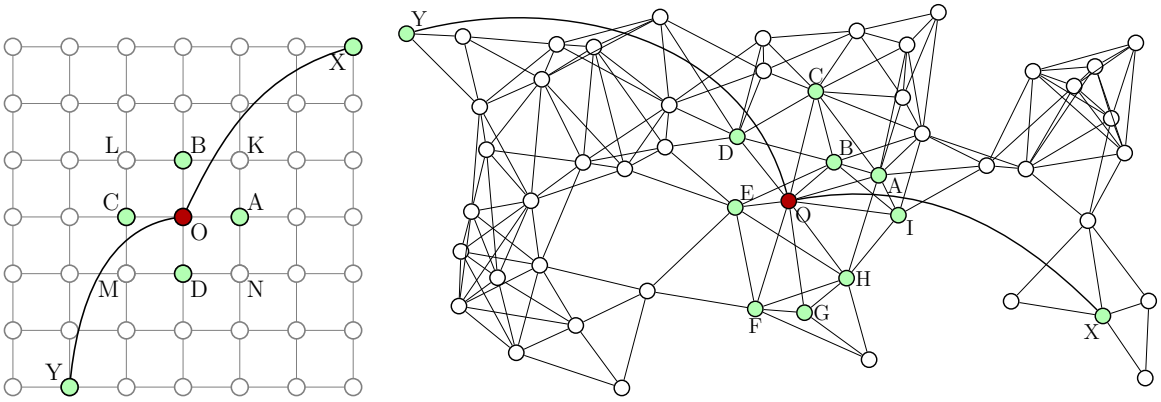
Locality The locality property of a particular edge is defined by its effect to the distances between a node and the set of neighbors of the other one. Let $G \setminus e_{ij}$ denote the graph created from G by removing the edge between i and j (if it exists). Consider an arbitrary edge $e_{ij} \in \Gamma$ that connects two nodes i and j.

- e_{ij} is a **local edge** when the existence of e_{ij} does not affect to the smallest distance between i (respectively j) and an arbitrary other neighbor of j (respectively i).

$$(d_G(j, N[i]_j) = d_{G \setminus e_{ij}}(j, N[i]_j) \wedge d_G(i, N[j]_i) = d_{G \setminus e_{ij}}(i, N[j]_i))$$

- e_{ij} is a **long edge** when:

$$(d_G(j, N[i]_j) < d_{G \setminus e_{ij}}(j, N[i]_j) \vee d_G(i, N[j]_i) < d_{G \setminus e_{ij}}(i, N[j]_i))$$



(a) Long edges added into grid

(b) Long edges added into unit disk graph

Fig. 4.4 Local edge and long edge

Figure 4.4 gives two examples of long-edges and local-edges in two basic graphs including grid and unit-disk graph. Let us consider neighbors of vertex O in figure 4.4a,

e_{OX}, e_{OY} are long-edges because removing e_{OX} increases the shortest distance between X and a neighbor of O from 2 to 5. Similarly, removing e_{OY} increases the shortest distance between X and a neighbor of O from 2 to 4. Other edges, including $e_{OA}, e_{OB}, e_{OC}, e_{OD}$, are local edges. In the same way of explanation, in figure 4.4b, e_{OX}, e_{OY} are long-edges; while e_{OA}, e_{OB}, ..., e_{OI} are local edges.

Table 4.4 Degree of long-edges of theoretical graphs

Graph	Degree of long-edges (Δ_ℓ)
Square torus	0
Unit-disk graph $UDG(n, \Delta)$	0
Small-world $SW(n, \Delta, \ell)$	2ℓ

Degree of long-edges: Given a graph $G(\Pi, \Gamma)$, let Δ_ℓ be the degree of long-edges in a graph. Δ_ℓ is calculated by the average number of long-edge neighbors of a node. A given graph G is a **locality graph** if $\Delta_\ell \simeq 0$. Whereas, it is called a **long-edges graph** if $\Delta_\ell \geq 1$.

Table 4.4 gives the degree of long-edges of some basic theoretical graph defined in this section (Section 6.1). According to the definition, square torus and unit-disk graph are locality topologies, and small-world is a long-edge topology.

The degree of long-edges is maybe a good criterion that classifies the graphs into either *possibility* or *impossibility* situation to isolate the propagation and it is worth to investigate the question.

Chapter 5

Mitigating Propagating Fault in Basic Networks

The purpose of simulation experiments is to understand the behavior of the system or evaluate strategies for the operation of the system.
— Encyclopedia of Computer Science,
Simulation Article

This chapter provides the extensive simulation result and analysis on three classical graph topologies; square torus, unit-disk graph, and one kind of small-worlds.

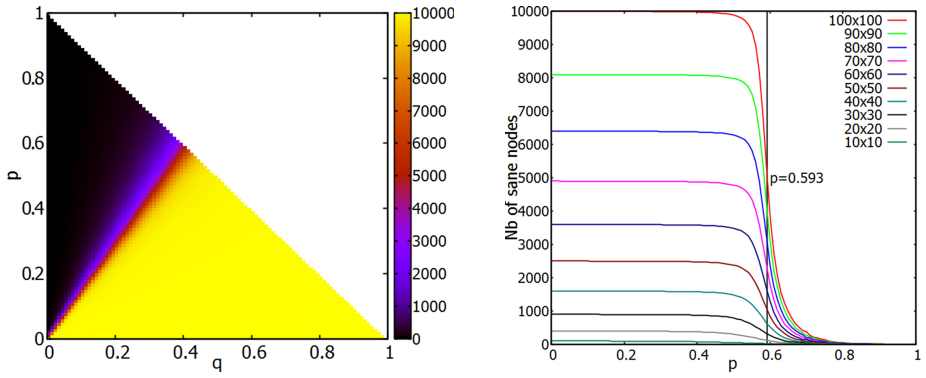
We wrote a discrete simulation in C++ to study our containment strategies. We measure the number of infected, sacrificed, and sane nodes at the end of the spread. The number of nodes varies from 100 to $10,000$. For the two random topologies, other parameters (density, number of long links) also vary. The probability of infection p and the probability of detection q are changed from 0 to 1 by 0.01 steps. The results are stated at the 95% confidence interval level for absolute value. In the critical area of the phase transition, we repeat the simulation until the confidence interval is smaller than 2% of the value.

5.1 In the torus

We study first the simplest killing strategy $K0$-Hop and then compare the performance of other strategies.

5.1.1 Strategy $K0$-Hop

The main results about $K0$-Hop are summarized on Figure 5.1. Both pictures represent the number of sane nodes at the end of the propagation.



(a) Torus 100×100

(b) Different torus size (with $q = 1 - p$)

Fig. 5.1 Strategy $K0$-Hop – Number of sane nodes

Existence of a phase transition

Figure 5.1a shows the evolution of the number of sane nodes in the 100×100 torus when the probabilities p and q of infection and detection vary.[1] The lighter the color is, the more sane nodes remain at the end of the simulation. As one could expect, the number of sane nodes increases when the probability of detection increases, and it decreases when the probability of infection increases.

Less expectedly, there are two clearly delimited regions; (1) the yellow region where almost all nodes are sane (*i.e.* very few infected or killed nodes) and (2) the black one where very few nodes are sane (*i.e.* mostly infected or killed nodes). The clear border indicates the existence of a phase transition. This border between the two regions consists in a line passing through the origin; it means that the transition occurs at a fixed ratio $\frac{p}{q}$. Let ρ_0 be this ratio. In order to analyze more precisely this phase transition, we study the case when $q = p - 1$ (corresponding to the hypotenuse of the triangle where the transition is larger).

Effect of the size of the torus on the phase transition

Figure 5.1b shows the evolution of the number of sane nodes for different sizes of toruses when the probability of infection vary. For all sizes of the system, the transition from a sane network to an infected network occurs at the same probability of infection, which is between 0.5 and 0.6. We conclude that the size of the torus has very little effect on

[1] The unusual triangular shape comes from the condition $p + q \leq 1$.

the phase transition.

Relation to the percolation theory

The value of the critical probability is not surprising since it corresponds to the site percolation threshold of the square lattice (grid). Indeed, there is an intuitive mapping from our strategy $K0$-Hop to the percolation theory. In our model, when $q = 1 - p$, p corresponds to the probability that a node with at least one infected neighbor becomes infected while $1 - p$ corresponds to the probability that this same node sacrifices itself. Using percolation terminology, this means that the node has probability p of being open and $1-p$ of being closed. Percolation theory analyzes the probability that there exists a "large" connected component of open nodes which, in our model, corresponds to a large connected component of infected nodes.

Therefore, it is consistent that the phase transition occurs just below $p = 0.6$ since the site percolation threshold for the grid has been experimentally calculated to be $\theta \simeq 0.593$. As mentioned in the introduction, the same behavior appears in probabilistic flooding protocol where a node decides to forward a message with a fixed probability.

Conclusion 1 *For the strategy $K0$-Hop (and it remains true for other strategies), the efficiency depends mostly on the ratio $\frac{p}{q}$. The exact values of probabilities p and q are not relevant.*

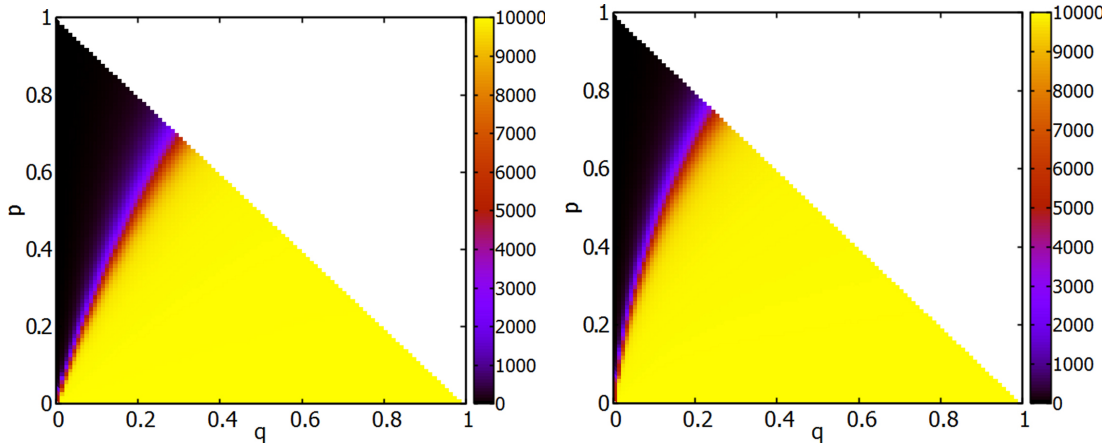
Conclusion 2 *The strategy $K0$-hop can contain efficiently the propagation of a virus in a square torus provided that the probability of successful infection is less than the percolation threshold $\rho_0 \simeq 0.59$. This threshold does not depend on the size of the network.*

5.1.2 Other killing strategies

Figure 5.2a and 5.2b show the evolution of the number of sane nodes in the 100×100 torus for $K1$-Hop and $K2$-Hop when the probabilities of infection and detection vary. As for $K0$-Hop, we observe a phase transition which roughly occurs at a fixed ratio $\frac{p}{q}$. Let ρ_1 and ρ_2 be these ratios.

Shift of the phase transition

Compared to $K0$-Hop, the surface of the yellow region (few of infected nodes) increases with $K1$-Hop and $K2$-Hop. Said differently, ratios are ordered such that $\rho_0 < \rho_1 < \rho_2$. It



(a) Strategy $K1$-Hop (b) Strategy $K2$-Hop

Fig. 5.2 Number of sane nodes in the 100×100 torus

means that killing more nodes mitigate the spread more efficiently when the probability of infection increases.

Figure 5.3 shows the comparison of $K0$-Hop, $K1$-Hop, and $K2$-Hop in the 100×100 torus when $q = 1 - p$. It confirms the previous observation: there is clearly a phase transition for each strategy, but they do not occur at the same probabilities. Sacrificing more nodes results in a shift of the critical probability of the phase transition to around 0.7 for $K1$-Hop and 0.75 for $K2$-Hop, when compared to 0.59 for the strategy $K0$-Hop.

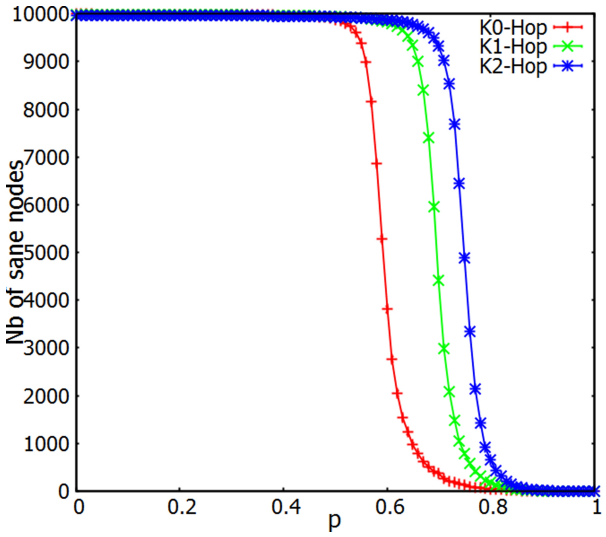


Fig. 5.3 Comparison of $K0$-Hop, $K1$-Hop, $K2$-Hop – Number of sane nodes

Conclusion 3 *All killing strategies can contain the propagation of a virus in a square*

torus provided that the probability of successful infection is less than a certain threshold.These thresholds increase with the number of killed nodes. However killing many nodes is not always the best strategy; when the probability of propagation is low, killing many nodes is counterproductive, especially for a small *torus since there remain few sane nodes.*

5.1.3 Cutting strategy $C0$

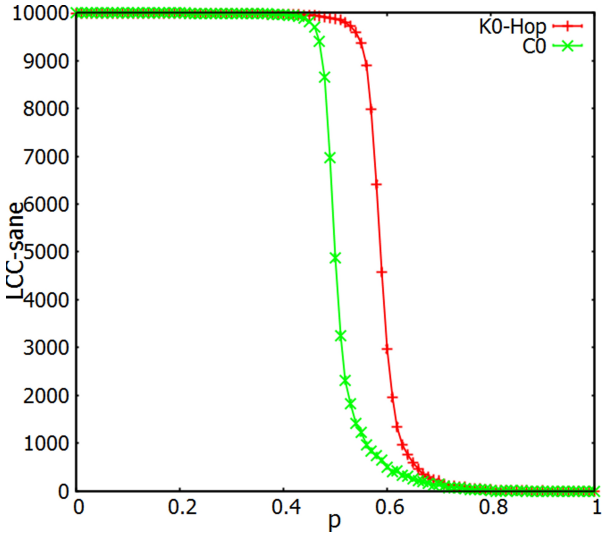


Fig. 5.4 Comparison of $K0$-Hop, $C0$ – Size of the LCC of sane nodes

To analyze the efficiency of cutting strategies, we need to introduce a new metric. Counting the number of sane nodes is not relevant. Indeed the trivial strategy "cut-all-links" would be optimal under this metric since all nodes would be immediately isolated (and sane). Obviously, this strategy should not be considered as efficient. Therefore, instead of counting the number of sane nodes, we measure the size of the Largest Connected Component (LCC) of sane nodes.

Figure 5.4 shows the evolution of the size of this largest component for strategies $K0$-Hop and $C0$. Similarly to killing strategies, we observe a threshold after which the cutting strategy $C0$ does not contain the propagation anymore. The transition occurs when the probability of infection is around 0.5. Again, it is possible to relate this value with results from the percolation theory; it corresponds exactly to the bond percolation threshold of the square lattice.[2]

[2]In bond percolation, each edge has a probability of being open or closed. Contrarily to the site percolation threshold, the value of the bond percolation threshold is known exactly to be 0.5 for the

Globally, the strategy $C0$ is less efficient than $K0$-Hop (and implicitly $K1$-Hop and $K2$-Hop) since its threshold is lower. Nevertheless, for some very specific cases, the cutting strategy performs slightly better. When the probability of infection p is very low, there is a small advantage to use $C0$ instead of $K0$-Hop. In these situations, $C0$ avoids killing the nodes that detects the infection. However since the probability of detection is high, the propagation stops quickly and therefore the number of spared nodes is quite small.

Conclusion 4 *The cutting strategy $C0$ can contain the propagation in the square torus, but is less efficient than any killing strategy. There is only a small advantage to use $C0$ when the probability of infection is very low.*

5.2 In unit-disk graphs

Contrary to the torus, the unit-disk graphs are randomly generated. We choose the parameters in order to guarantee the connectivity of the network with high probability. Under this constraint, we generate random graphs with average degree of 8 or 10. Since results are the same in both cases, we present the figures only for the average degree of 10.

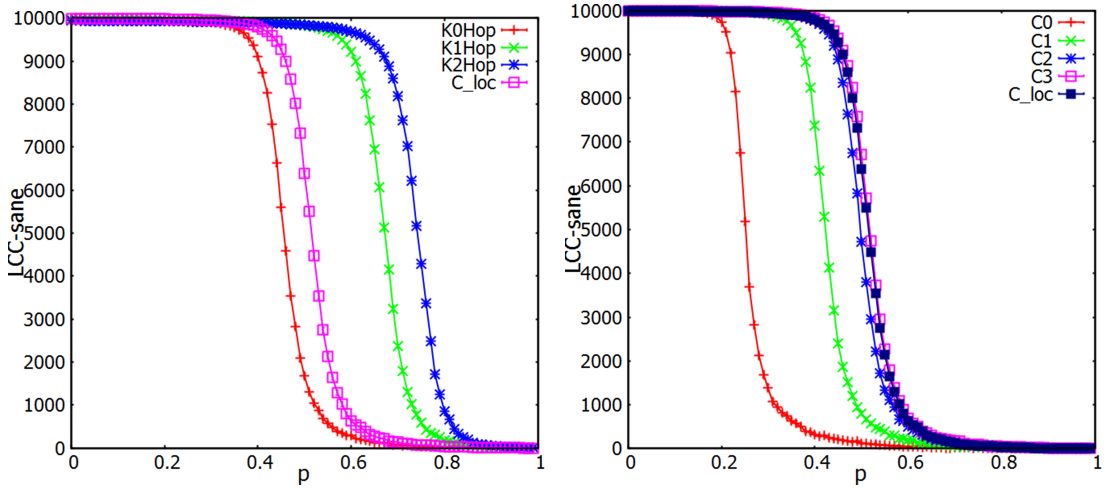
Killing strategies

Figure 5.5a summarizes the simulations of killing strategies in the unit-disk graph $G_{UDG}(10000, 10)$. We observe results that are similar to the ones obtained for the torus. Each strategy can contain the propagation until a certain probability of infection. There exists also a clear phase transition and a threshold after which a given strategy becomes inefficient. As for the square torus, the values of these thresholds are higher for strategies that kill more nodes.

Cutting strategies

Figure 5.5a summarizes the simulations of cutting strategies in the unit-disk graph $G_{UDG}(10000, 10)$. $C0$ and $C1$ give the worse performances than $K0$-Hop. Other analyzed cutting strategies give similar performances. They are better than $K0$-Hop but worse than $K1$-Hop. As for the torus, there is no benefit here to use cutting strategies instead of killing strategies; they globally perform less efficiently.

square lattice (grid).



(a) Killing strategies and C_{loc}

(b) Cx and C_{loc}

Fig. 5.5 Comparison of all strategies in the unit disk graph $G_{UDG}(10000, 10)$

Conclusion 5 *The results for the unit disk graphs are similar to those obtained for the square torus. More generally, we can expect that it will be the same for all locality-based topologies.*

5.3 In small-world networks

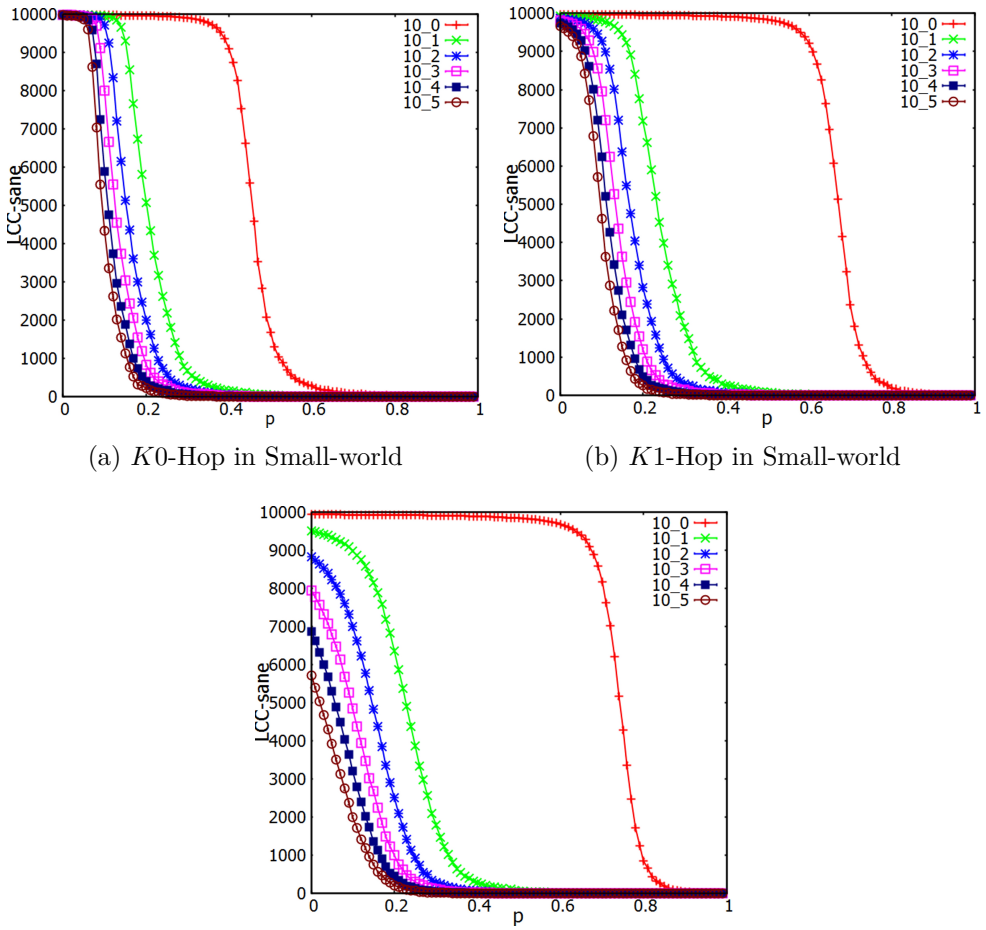
Simulation parameters We study the effects of different strategies on the Small-world graph defined in Section 6.1 to see how the long-links change the strength of the propagation and the strategies to isolate propagation. Based on the existing unit-disk graphs, we vary the number of long edges for each node. The generated small-world graphs are $SW(10000, 8, \ell), \ell = 1..6, 8$ and $SW(10000, 10, \ell), \ell = 1..5, 10$.

The long links reduce the average diameter of network from 80 in $G_{UDG}(10000, 10)$ to 7 in $SW(10000, 10, 1)$. Moreover, ℓ long edges added for each node of initial unit-disk graph increases 2ℓ the average degree of network.

The lack of efficient strategy

The figures 5.6a, 5.6b, 5.6c show the simulation results of $K0$-Hop, $K1$-Hop and $K2$-Hop respectively in unit-disk graph $G_{UDG}(10000, 10)$ and different small-world graphs $SW(10000, 10, \ell)$, where $\ell = 1..5$. When long edges are added into unit-disk graph, the phase transitions of Killing strategies $K0$-Hop, $K1$-Hop, $K2$-Hop are significantly

shifted to the left. We also make the simulation for $SW(10000, 8\ell)$, where $\ell = 1..6, 8$, the result is similar to figures above.



(a) $K0$-Hop in Small-world

(b) $K1$-Hop in Small-world

(c) $K2$-Hop in Small-world

Fig. 5.6 The difference of killing strategies between Unit-disk graph and Small-world graph

The simulation results confirm the theorem in 6.4 that no strategies can mitigate the propagation when the probability increases more than 0.2.

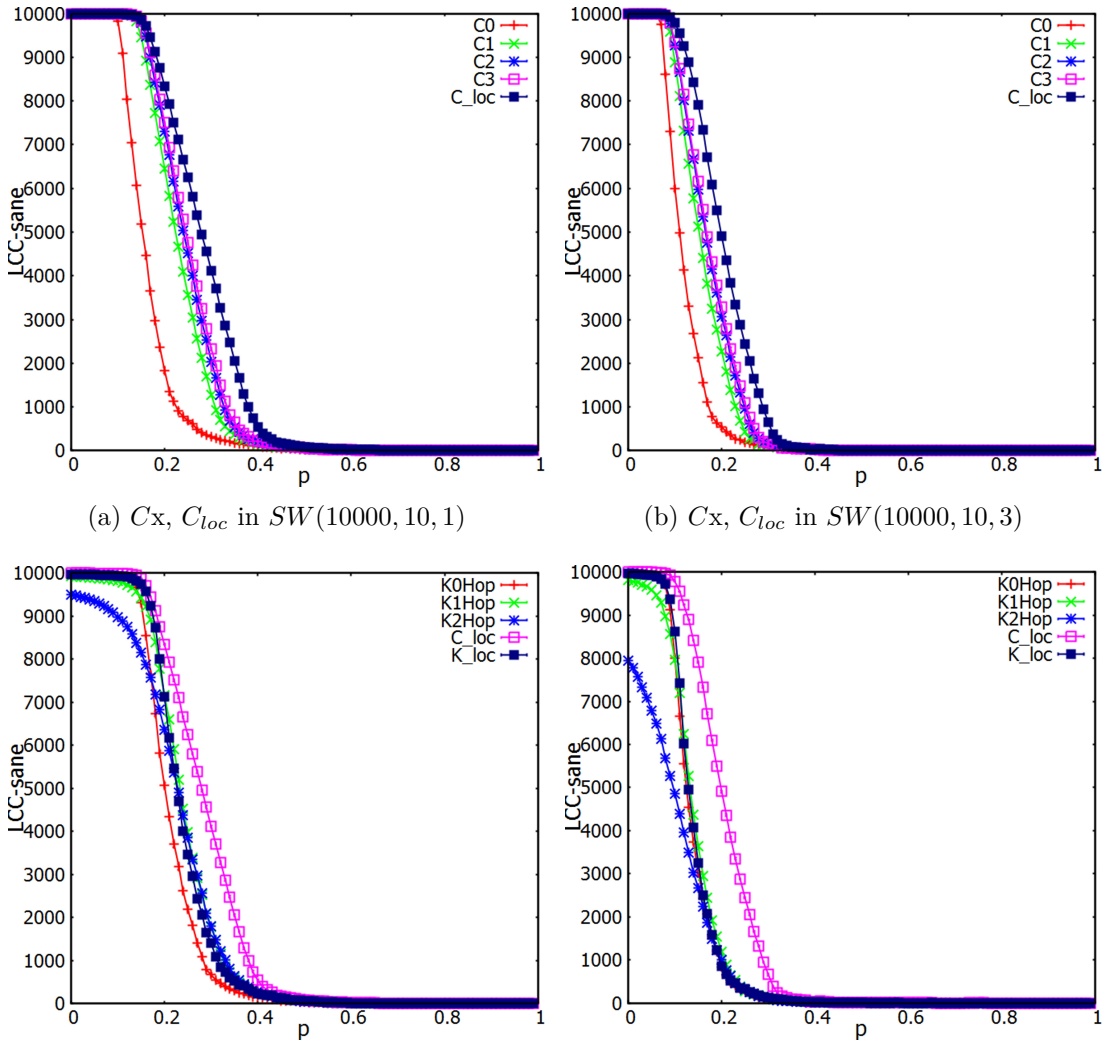
Conclusion 6 *Contrary to the unit-disk graph and torus, when the infection probability greater than* 0.18*, no killing strategy introduced in section 4.4.1 can contain the infection.*

Cutting local-edges is better than killing

On the other point of view, because of the small average diameter of small-world graph, the number of killed nodes in each round is a big fraction of the total number of nodes

in network. When the infection probability is small, for every attack, a node has higher chance to detect than to be infected. Another question is arisen that whether it is interesting to save some neighbors of detector from deactivation.

Besides the cutting strategies introduced in section 4.4.2, strategy K_{loc} using 2-hop neighborhood information to kill all local neighbors and keep long-edge neighbors alive is considered. With 2-hop neighborhood information, each node knows all neighbors of its 1-hop neighbors.



(a) Cx, C_{loc} in $SW(10000, 10, 1)$

(b) Cx, C_{loc} in $SW(10000, 10, 3)$

(c) K-strategies and C_{loc} in $SW(10000, 10, 1)$

(d) K-strategies and C_{loc} in $SW(10000, 10, 3)$

Fig. 5.7 The comparison between $K0$-Hop, $K1$-Hop, K_{loc} and C_{loc} in small-world

Figures 5.7a and 5.7b show the simulation results of cutting strategies in two small-world graphs $SW(10000, 10, 1)$ and $SW(10000, 10, 3)$ respectively. We can see that the

curve of C_{loc} is always the last in the right side and the least sloping when compared with other cutting strategies.

Figures 5.7c and 5.7d show the comparison between killing strategies $K0$-Hop, $K1$-Hop, $K2$-Hop, K_{loc} and C_{loc} small-world graphs $SW(10000, 10, 1)$ and $SW(10000, 10, 3)$ respectively. We can see that, in both graphs, C_{loc} has less sloping curve than the other strategies and its curve also in the right side of other curves. The difference between C_{loc} and killing strategies in $SW(10000, 10, 3)$ is more significant than in $SW(10000, 10, 1)$. While K_{loc} does not bring any benefit than other killing strategies.

It means that in small-world:

- All local edges should be cut to have the most effective performance in mitigating the propagation among the cutting strategies.
- The cutting strategy C_{loc} significantly improves `LCC−sane` of the network. Consequently, keeping the detectors and the long-edges that connects detectors with other components except the one having infected nodes help to slowdown the phase transition.
- If there are more long edges on the top of initial unit-disk graph, the improvement of cutting strategy C_{loc} is more significant.

Conclusion 7 *When the infection probability is small, cutting all local edges, keeping detector and all long edges from detector active (except infection edges) by using 2-hop neighborhood information helps the small-world graph to improve the performance on* `LCC−sane` *especially when the number of long-edges is large.*

5.4 Summary of simulation results

We have studied the propagation of a virus in three kinds of graphs (square torus, unit disk, and small-world) and how local strategies can help contain this propagation.

After verifying the existence of a threshold and a phase transition for all strategies, we observe that the choice of a strategy introduces a shift of that threshold.

In particular, for the two locality-based topologies, we confirm the intuition that sacrificing a larger number of nodes helps with containing the propagation, which is achieved even for higher infection rates (or lower detection probabilities).

Surprisingly, we find that strategies based on cutting communication links are not necessarily better since, although they may save more nodes (*e.g.* by not killing the

detectors), it has no global effect as soon as the probability of infection is not very small since the saved nodes can be infected from other neighbors.

We find that using some mitigation strategy is somewhat effective in the case of the torus and the unit disk graph. The choice of strategy is important and killing strategies are significantly more effective in these two kinds of graphs.

The result is not the same in small-world graphs. The fact that small-world graphs have a very small diameter clearly plays against them, and little can be done to contain the propagation. Containment can be achieved only for a small infection probability. In that case, killing strategies are, in fact, particularly harmful, and only strategies using some additional topology information about neighbors seem to provide any help, albeit a small one.

Chapter 6

Isolating Propagating Fault in Infinite Graphs

Mathematical reasoning may be regarded rather schematically as the exercise of a combination of . . . intuition and ingenuity.
— Alan Turing (1912–1954)

Each potential infinite, if it is rigorously applicable mathematically, presupposes an actual infinite.
— Georg Cantor (1845–1918)

In the previous chapter, the simulation results give us some brief conclusion about the situation of mitigating a grid in some basic graphs. The result leads to some conclusion that the critical threshold exists for torus. In square torus and unit-disk graphs, the threshold is shifted into higher infection probability when the killing hop increases. In contrary, in small-world graph, increasing killing hop is helpless because the small diameter. The question is that whether we can formalize the observation. The limitation on the size of network does not allow us to consider the extreme case of the spread and also the countermeasures. To answer the question in formulation, we study the extreme circumstance of both sides of the game in infinite graphs where there is no limitation on the size of system.

6.1 Infinite graphs

This chapter considers the propagating faults in three types of infinite graphs: grid, unit-disk graph, and small-world. While the grid is a regular topology, the other two graphs are random topologies. It means that they do not denote a specific graph but correspond to a family of graphs satisfying common properties.

Infinite grid The infinite grid corresponds to the graph $G = (\Pi, \Gamma)$ where nodes are located on the square lattice. Each node is connected to the nodes at distance one in each of the four cardinal directions.

$$\begin{cases} \Pi = \mathbb{Z} \times \mathbb{Z} \\ \Gamma = \{e_{i,j} \mid i \in \Pi \wedge j \in \Pi \wedge dist(i,j) = 1\} \end{cases}$$

Infinite unit-disk graph For a given density $d > 0$, an infinite unit-disk graph corresponds to a graph $G = (\Pi, \Gamma)$ where nodes are deployed uniformly at random on the two-dimensional plane ($\Gamma \subset \mathbb{R} \times \mathbb{R}$) with mean density d. There exists an edge between two nodes if and only if they are at Euclidean distance at most 1:

$$\Gamma = \{e_{i,j} \mid i \in \Pi \wedge j \in \Pi \wedge dist(i,j) \leq 1\}$$

For a given density d, let us define $\Delta(d)$ the average number of neighbors of a node. When there is no ambiguity, we will simply use Δ.

Infinite small-world For a given density $d > 0$ and a given integer $\ell > 0$, an infinite small-world (denoted by $SW(d, \ell)$) corresponds to a graph $G = (\Pi, \Gamma)$ built on top of an infinite unit-disk graph of density d in which each node is connecting to ℓ new neighbors (uniformly randomly chosen). Since we consider undirected graphs, the average number of neighbors of a node equals $\Delta + 2\ell$.

6.2 Equivalence between site percolation and strategy $K0$-Hop

In this section, we show that site percolation and strategy $K0$-Hop have the same threshold, below which propagation is contained with high probability and above which it is not. Thus, finding the value of the threshold for one is equivalent to finding the threshold for the other. After defining the basic concepts and terminology of site percolation and strategy $K0$, we prove the equivalence.

Site percolation Given an infinite graph and a probability p such that any site (*i.e.*, node) is *occupied* with probability p (resp., *empty* with probability $1-p$), let us consider the following random variable X_p that equals 1 if there is an infinite connected component of occupied nodes and 0 otherwise.

There exists a threshold τ (called the percolation threshold) such that [10, 46]:

$$\begin{cases} \mathbb{P}[X_p = 1] = 0 & \text{if } 0 \leq p < \tau \\ \mathbb{P}[X_p = 1] = 1 & \text{if } \tau < p \leq 1 \end{cases}$$

In other words, with probability 1, there is an infinite connected component of occupied nodes when the probability p is above τ. Respectively, there is no such component when p is below the threshold. When p exactly equals the threshold, the situation is unclear.

The value of the percolation threshold depends on the class of graph under consideration. While it is known for some classes of graphs, even after half a century of research, no exact value is known for the threshold in the case of the infinite square lattice (grid). The currently best-known approximation for the grid is $\tau = 0.59274598(4)$ obtained through Monte-Carlo simulation.

Instead of considering the existence of an infinite connected component, one may consider a given occupied node x and the following random variable Z_p^x that equals 1 if x is in an infinite connected component of occupied nodes and 0 otherwise.

This new random variable is closely related to X_p previously defined. Indeed, if there is no infinite connected component ($\mathbb{P}[X_p = 1] = 0$), it is impossible for node x to belong to such an infinite component (thus, $\mathbb{P}[Z_p^x = 1] = 0$). Conversely, if there is a non-zero probability that x belongs to an infinite component ($\mathbb{P}[Z_p^x = 1] > 0$), then necessarily there exists an infinite component ($\mathbb{P}[X_p = 1] = 1$). Therefore:

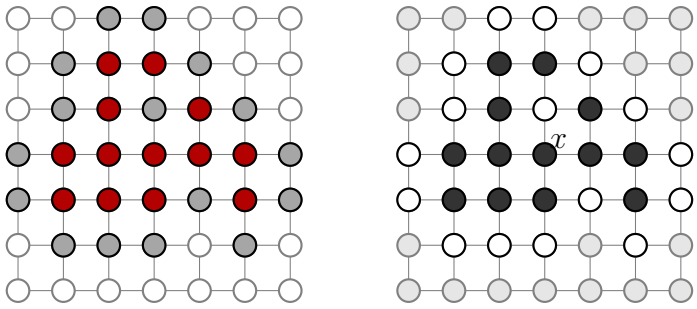
$$\begin{cases} \mathbb{P}[Z_p^x = 1] = 0 & \text{if } 0 \leq p < \tau_0 \\ \mathbb{P}[Z_p^x = 1] > 0 & \text{if } \tau_0 < p \leq 1 \end{cases}$$

Strategy $K0$-Hop Given a probability p of infection and an infinite graph, consider a spread when $K0$-Hop is used as the containment strategy. Let Y_p be a random variable that equals to 1 if the spread never stops and 0 if the spread is contained. For the two extreme values of p, the distribution of Y_p is trivial: when $p = 0$, the spread is immediately contained and $\mathbb{P}[Y_0 = 1] = 0$; conversely, when $p = 1$, all nodes are infected and $\mathbb{P}[Y_1 = 1] = 1$. Moreover, by a simple coupling argument, it is also clear that the function $p \mapsto P[Y_p = 1]$ is non-decreasing. Therefore, there exists a unique threshold τ_0 such that:

$$\begin{cases} \mathbb{P}[Y_p = 1] = 0 & \text{if } 0 \leq p < \tau_0 \\ \mathbb{P}[Y_p = 1] > 0 & \text{if } \tau_0 < p \leq 1 \end{cases}$$

strategy $K0$	site-percolation
infected node	occupied node
killed node	empty node
sane node	— *(no equivalent)*
initial infected node	any occupied node x

(a) Intuition link between $K0$ and site-percolation



(b) Strategy $K0$-Hop (c) Site-percolation

Fig. 6.1 "Equivalence" between strategy $K0$-Hop and site-percolation model

It is important to note the inequality. Contrary to the random variable X_p associated to site percolation, Y_p does not "jump directly from 0 to 1". In fact, Y_p always equals 1 only for $p = 1$. For any probability of infection p strictly lower than one, there is a non-zero probability that strategy $K0$-Hop contains the spread. Indeed, if all neighbors of the initially infected node detect the infection, the propagation stops immediately. This case happens with probability $(1-p)^\delta$, where $\delta > 0$ is the degree of the initial node, and thus $\mathbb{P}[Y_p = 1] \leq 1 - (1-p)^\delta$.

Theorem 3 *Site percolation and strategy $K0$ have the same thresholds: $\tau = \tau_0$.*

Proof: The proof consists in showing that Y_p and Z_p^x follow the same distribution for any p and any occupied node x. To prove the claim, consider the following (intuitive) link between both models in Table 6.1a.

With site percolation, *all* nodes are randomly set to the occupied state (with probability p) or to the empty state (with probability $1-p$). With strategy $K0$-Hop, *some* nodes are randomly set to the infected state (with probability p) or to the killed state (with probability $1-p$) , but *some other* nodes are not set in one of these states. Indeed, any node that is neither an infected node, nor a neighbor of an infected node, remains in a sane state (see Section 4.3).

However, sane nodes do not have any effect on the random variable Y_p since they are disconnected from the initially infected node. The value of Y_p depends only on the infected (red nodes on Figure 6.1b) and killed nodes (gray nodes): all infected nodes are connected and therefore Y_p reflects the infiniteness of this infected component.

With site percolation, for a given occupied vertex x, the same observation applies to the random variable Z_p^x. It depends only on the occupied nodes (black nodes on Figure 6.1c) that are connected to x or the empty nodes (white nodes) that are around these occupied nodes. Other nodes (light gray nodes) do not have any effect on Z_p^x; they could be occupied or empty; it will not change the value of Z_p^x.

Since (1) nodes are infected or occupied with the same probability p starting from an initial infected node or an arbitrary occupied node x, and (2) $Y_p = Z_p^x$, it follows that $\tau = \tau_0$. □

Corollary 2 *Given a graph $G = (\Pi, \Gamma)$, finding the site percolation threshold and the propagation threshold with strategy $K0$-Hop are equivalent.*

6.3 Propagation always contained in the infinite grid

This sections proves that it is always possible to contain the propagation in an infinite grid, provided that we use a killing strategy that sacrifices enough nodes (*i.e.*, a strategy Kh-Hop where h large enough). The proof is done by reduction to strategy $K0$-Hop so that we can apply Theorem 3 and use the existence of a percolation threshold for a specific topology.

6.3.1 Definitions and explanations

General idea of the proof Based on the infinite grid, we define the notion of *super nodes* to encompass a squared subset of nodes. Super nodes are considered as infected, killed, or sane depending on the states of their internal nodes. By choosing an appropriate size for the super nodes and an adequate containment strategy, we can show that, at the super node level, the propagation of the virus behaves similar to when strategy $K0$-Hop is executed. From there, we can deduce that the infection is contained at the super node level, which then implies containment at the lower level.

Super node Given a strictly positive integer h and the infinite grid $G(\Pi, \Gamma)$ with $V = \mathbb{Z} \times \mathbb{Z}$, we partition the nodes into *super nodes of size h*. Each super node is a

subset of h^2 nodes organized in a square of side h. The super node at column x and row y, denoted $g(x, y)$, corresponds to the following subset of V:

$$g(x, y) = \{(hx + i, hy + j) \mid 0 \leq i, j < h\}$$

Similarly to nodes, super nodes may be infected, killed, or sane. We define a super node as infected if all nodes in at least one of its sides are infected. Otherwise, we define that a super node is killed or sane whether it contains killed node(s) or not.

Containment strategy Given a super nodes of size h, we study the strategy $K2h$-Hop. We consider two additional assumptions compared to the definitions of Sections 4.3 and 4.4:

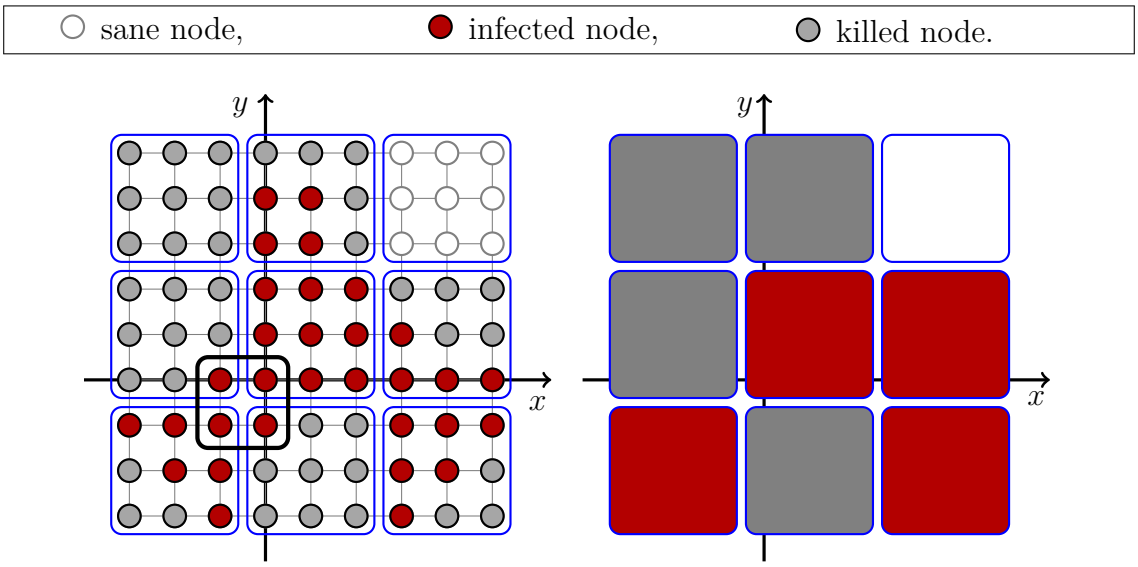
1. A killing strategy affects only nodes contained in the same super node as the detector. Nodes do not send killing messages to neighbors belonging to different super nodes.

2. Within a super node, strategy messages take precedence over infection messages. It means that if, during a round, a susceptible node receives both an infection message from an infected neighbor and a strategy message from another neighbor, it always reacts according to the strategy message and sacrifices itself (after having potentially forwarded killing messages).

The first assumption favors the propagation; it weakens the containment strategy and helps the virus propagate and therefore has no impact on our result: if it is possible to contain a propagation with this assumption, it is also possible to contain a propagation without the assumption. It is used only for convenience of the proof. Conversely, the second assumption favors the containment; it weakens the propagation of the virus and is necessary for our proof. Indeed, with this assumption, it is guaranteed that inside a super node, nodes on the sides cannot be attacked from internal nodes (see Lemma 6).

Figure 6.2 provides an example of infection with corresponding nodes and super nodes. It is worth noting that, unlike what happens at node level, at the super node level, an infection may be transmitted in "diagonal": in the figure, the central super node has infected the super node located at the lower left corner.

Small-Four Based on the previous observation, we introduce the notion of "Small Four". For any group of four super nodes arranged in a 2×2 grid, let us call "Small-Four" the group of four normal nodes that connect these four super nodes by their



(a) at node level (b) at super node level

Fig. 6.2 Example of infection

corners. In Figure 6.2a, the group of four nodes $(0,0),(0,-1),(-1,0),(-1,-1)$ locates in the small black square is an example of the "Small-Four" that connects the four super nodes $g(0,0), g(0,-1), g(-1,0), g(-1,-1)$.

Since an infection can be propagated via the diagonals, the infection in super nodes can be seen as an infection in the lattice where every node has 8 neighbors. This lattice is called Moore's neighborhood [48]. In this section, the neighbors of a super node are the neighbors according to Moore's neighborhood and the neighbors of normal node are the four neighbors in the default model (Von Neumann's neighborhood).

6.3.2 Proof

Lemma 6 *With an original infected node at* $(0,0)$*, at any time, in any side of any super node, if a side node is infected then (1) this node is at a corner, or (2) one of its two neighbors on this side was already infected in the previous round.*

Proof: The Lemma is proved by induction on the round number. At time $t = 0$, $(0,0)$ is an infected node in a side of $g(0,0)$ and it is a corner. At time $t = 1$, there are four nodes could be infected $(-1,0),(0,-1),(0,1)$ and $(1,0)$. Among these four nodes, $(-1,0)$ and $(0,-1)$ are corner nodes while $(0,1)$ and $(1,0)$ have infected neighbor $(0,0)$ that is infected at previous round $t = 0$. Hence, the Lemma holds with $t \leq 1$.

Assume that the claim is correct for any $t < n - 1$, we must prove that it holds for $t = n$.

Without loss of generality, consider a newly infected side node $v_C = (hx, hy + j)$ of super node $g(x, y)$ at time $t = n$. With $j = 0 \vee j = h - 1$, v_C is a corner, the claim holds. With $0 < j < h - 1, v_C$ has four neighbors $v_B = (hx, hy + j - 1), v_U = (hx, hy + j + 1), v_L = (hx - 1, hy + j)$ and $v_R = (hx + 1, hy + j)$ at the bottom, the upper, the left and the right side respectively. We now prove that the infection cannot come only from v_L and v_R.

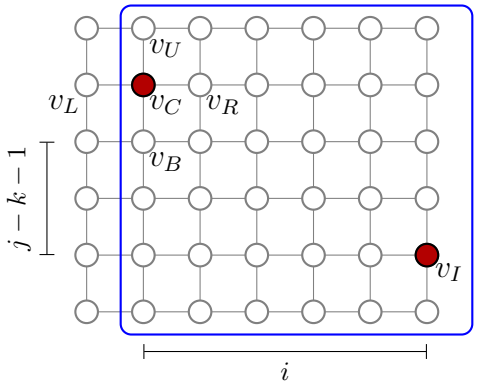


Fig. 6.3 An infected node in a side

If node v_L is infected at time $t = n - 1$, according to the induction step, one of two neighbors $(hx - 1, hy + j - 1)$ and $(hx - 1, hy + j + 1)$ of v_L must be infected at time $t = n - 2$. Assume that it is $(hx - 1, hy + j + 1)$, then node v_U is attacked by $(hx - 1, hy + j + 1)$ at time $t = n - 1$. Consequently, v_U is either infected or killed at time $t = n - 1$. If v_U is killed at $t = n - 1$ then according to the assumption that killing messages have priority over infection messages, v_C is killed at time $t = n$. If v_U is infected, then the infection comes from both v_L and v_U.

If node v_R is infected at time $t = n - 1$, then let $v_I = (hx + i, hy + k)$ be the first node in super node $g(x, y)$ that is infected from $(hx + 1, hy + j)$. Let t_I be the time when v_I is infected, we have $t_I < n - 1$. Let us call the distances from v_i to v_c, v_b, v_u, v_r are $d_{IC} = i + |k - j|, d_{IB} = i + |k - j + 1|, d_{IU} = i + |k - j - 1|, d_{IR} = |i - 1| + |k - j|$ respectively. Moreover, v_i must be in one of four sides of $g(x, y)$:

- If $i = 0$, then $d_{IR} = 1 + |k - j|$, while $d_{IC} = |k - j| < d_{IR}$. d_{IR} is the smallest latency for v_R to be infected from v_I, so v_R is infected at time $t \geq t_I + |k - j| + 1$. Consider, the attack from v_I to v_C, $d_{IC} = |k - j| \leq h$,

If there is any detection by a node between v_I and v_C, by applying $K2h$-Hop, all v_C is either killed at time $t_I + |k - j|$ or it is killed or infected by an earlier attack at time $t < t_I$. Therefore, v_C is either infected or killed before v_R.

- If $(i > 0 \wedge k \leq j - 1)$, then $d_{IR} = d_{IB} = i + j - k - 1 < d_{IC} = i + j - k < d_{IU} = i + j - k + 1$. In this case, it is the earliest time for v_R to be infected from v_I is $t_I + d_{IR}$. Consider the state of v_B according to the attack from v_I, $d_{IB} = i + j - k - 1 \leq h + h - 1 = 2h - 1$,
  - If there is any detection by a node having coordinates $(hx+r, hy+s), \forall r, \forall s, 0 \leq r \leq i, k \leq s \leq j - 1$, then by strategy $K2h$-Hop, v_B is killed at time $t_I + d_{IB} = t_I + d_{IR}$. This is because all nodes on the shortest path between v_I and v_B are killed (there are always less than $2h$ such nodes).

    It means that v_B is killed at the same time or before v_R is killed or infected. Then, by the assumption that killing messages are faster than infection messages, v_C is killed by a killing message from v_B. A contradiction.
  - If there no detection, then v_B is infected at time $t_I + d_{IB} = t_I + d_{IR}$; the same time as v_R. Then, v_C is infected by both v_R and v_B.
- If $(i > 0 \wedge k \geq j + 1)$, then $d_{IR} = d_{IU} = i - j + k - 1 < d_{IC} = i - j + k < d_{IB} = i - j + k + 1$. Similar to the previous case, either v_C is killed by a killing message from v_U or it is infected by both v_R and v_U.
- If $(i > 0 \wedge k = j)$, then $d_{IR} = i - 1 < d_{IC} = i < d_{IB} = d_{IU} = i + 1$. In this case, with $0 < j < h - 1$, v_I is a non-corner node in a side of $g(x, y)$ and $v_I = (hx + h - 1, hy + j)$. v_I is a side node infected at time $t_i < n - 1$. According to the assumption of the induction step, one of the two neighbors $(hx+h-1, hy+j-1)$ and $(hx + h - 1, hy + j + 1)$ must be infected at time $t_i - 1$. A contradiction.

The claim holds for $t = n$, thus proving the induction. □

Corollary 3 *Given a super node, a side node is only infected by external attacks starting at corners. Other external attacks are irrelevant since they infect no sooner.*

Proof: If a side node is infected, by Lemma 6, either it is a corner or one of its neighbors on the same side is infected in the previous round. It implies that, with any non-corner node on a side of a supper node, the attack from its neighbor on the same side occurs always first. Recursively, we can deduce that the attack originates from a corner. □

Corollary 4 *For any node of a super node, either (1) it is attacked by an attack originating at a corner or (2) it is killed.*

Proof: According to Corollary 3, for any side node, it is only infected by external attacks starting at a corner. Necessarily, for any non-side node, if it is not killed, it must be attacked from an infection starting at a side node. Consequently, all nodes of a super node are either attacked by an infection starting at corners or killed. □

Lemma 7 *Except for the "Small-Four" containing the first infected node* $(0,0)$*, the first infected node in a "Small-Four" must belong to an infected super node.*

Proof: Consider the "Small-Four", $(0,0),(0,-1),(-1,0),(-1,-1)$, because the first infected node is at $(0,0)$, the three direct super node neighbors $(-1,0),(0,-1),(-1,-1)$ could be infected while super node $g(0,0)$ is not infected (i.e., no entire side of $g(0,0)$ consists uniquely of infected nodes).

Consider any "Small-Four" other than the "Small-Four" discussed in the previous case, and call it $(hx,hy),(hx-1,hy),(hx-1,hy-1)$, and $(hx,hy-1)$. Assume that (hx,hy) is the first infected node among the four nodes. Because the three other nodes are not infected, the infection must come from inside super node $g(x,y)$. According to Corollary 4, all nodes inside a super node must be attacked by an infection starting from a corner, thus another corner node was infected. This means that, at least one side of super node $g(x,y)$ must consist entirely of infected nodes. Hence, $g(x,y)$ is infected. □

Lemma 8 *A super node may be infected only if it is a direct or diagonal neighbor of* $g(0,0)$ *or of a super node infected previously. Moreover, in this situation, the probability of being infected is bounded by* $4p^h$*.*

Proof: A super node $g(x,y)$ can only be firstly attacked at a corner (Corollary 3), let this corner be (hx,hy). This means that the attack comes from one of its neighbors in the "Small-Four" $(hx-1,hy),(hx-1,hy-1),(hx,hy-1)$.

According to Lemma 7, one of the three neighbors of node (hx,hy) must belong to an infected super node (or to $g(0,0)$). Finally, $g(x,y)$ must be attacked either from $g(0,0)$ or from an infected super node from Moore's neighborhood.

Let us recall that p is the probability that a normal node in the grid is infected. To become infected, a super node must have one of its sides entirely infected. For a given side, the probability that it becomes infected is bounded by p^h since any of the h nodes has at most a probability p of being infected. Since there are four sides, and even if the

probabilities are not independent, for any super node, it is still possible to bound by $4p^h$ the probability of infection. □

Theorem 4 *Given a virus spread with infection probability p, where $0 \leq p < 1$, there exists always a killing strategy that contains the virus spread.*

Proof: According to Lemma 8, only an infected super node can attack another super node, so every infected super node is connected to the infected component.

Since every super node neighbor to the infected component has a bounded probability to be infected, the infection spreads through super nodes can be seen as strategy $K0$ on an infinite lattice with Moore's neighborhood.

According to Theorem 3, percolation and strategy $K0$ are equivalent in the sense that they have the same critical probability (or percolation threshold). According to Malaz and Galam [48], there exists a percolation threshold in an infinite lattice with Moore's neighborhood. Hence, $K0$-Hop in an infinite lattice with Moore's neighborhood necessarily has the same threshold.

Let ρ_c^{Moore} be the percolation threshold in an infinite lattice with Moore's neighborhood.

Let $h \in \mathbb{N}$ be such that $h > \frac{\log(\rho_c^{\text{Moore}}/4)}{\log p}$. The virus spreading with infection probability p against the $K2h$-Hop strategy can be seen as the propagation of virus via super nodes of size h with probability $\rho < 4p^h < \rho_c^{\text{Moore}}$ against strategy $K0$-Hop.

In a model with super nodes of size h, the main connected component of the percolation with infection probability $\rho < \rho_c^{\text{Moore}}$ is finite, therefore the virus propagation with infection probability p is contained by strategy $K2h$-Hop in an infinite grid. □

6.4 Analysis for the infinite small-world networks

Theorem 5 *Given a small-world graph $SW(d, \ell)$ and the infection probability $p > \frac{\sqrt{(\ell-1)^2+2\Delta\ell}-(\ell-1)}{2\Delta\ell}$, there is no killing strategy such that propagation is contained with probability 1.*

Proof: According to the definition of small-world graphs given in Section 6.1, each node $SW(d, \ell)$ is connected on average to $\Delta + 2\ell$ other nodes, decomposed into two sets: (1) Δ nodes are *short-neighbors* coming from the underlying unit-disk graph, and (2) (2ℓ) nodes are *long-neighbors*, randomly selected. Long-neighbors are randomly chosen from

an infinite set, so these two sets intersect with probability 0. We construct the argument based on infections that occur through long-neighbors.

Let us consider an arbitrary execution starting from an initial infected node n. As any node, it has on average 2ℓ long-neighbors. Since each attack is independent, in expectation, $2\ell p$ long-neighbors are infected during the first timeslot. These newly infected node are "independent" (*i.e.* for any distance $d < \infty$, there is a probability 0 that any two of them are at Euclidean distance less than d). During the second timeslot, each of these newly infected nodes infects on average $(2\ell-1)p$ of its long-neighbors, which means that, on average, there are $2\ell p \times (2\ell - 1)p$ newly infected long-neighbors at the end of the timeslot. The same behavior repeats forever, and the average number of newly infected nodes increases when $(2\ell - 1)p > 1$, which implies that the propagation can not be contained if $p > \frac{1}{2\ell-1}$.

We now refine the bound by also considering the possible infections of the long-neighbors of short-neighbors. Indeed, in addition to the $(2\ell - 1)p$ "long-infections", a node infected from one of its long-neighbor also infects Δp of its short-neighbors in expectation. In turn, each of these short-neighbors infects $2\ell p$ of its long-neighbors in expectation. Therefore, a node infected from one of its long-neighbor propagates the virus an expected number of $(2\ell-1)p+\Delta p \times 2\ell p$ different nodes. Solving the inequality $2p(\ell - 1) + 2\Delta p^2 \ell > 1$ gives us

$$p > \frac{\sqrt{(\ell-1)^2 + 2\Delta\ell} - (\ell - 1)}{2\Delta\ell}.$$

Hence, if $p > \frac{\sqrt{(\ell-1)^2+2\Delta\ell}-(\ell-1)}{2\Delta\ell}$, it is impossible, with probability 1, to contain the propagation of the virus. Since the killing strategy is irrelevant to the argument, this proves the theorem. □

6.5 Summary of propagation in infinite graphs

In this chapter, by considering the problem in some infinite graphs, we confirm and formalize the observations that are risen in chapter 5 from the simulation. Firstly, we prove the equivalence between site percolation and strategy $K0$-hop in the arbitrary infinite graphs. Secondly, we prove that the spread of infection is always possible to contain in an infinite grid, provided that we use a killing strategy that sacrifices enough nodes. Finally, we prove that there is no killing strategy that always contain the propagation when infection probability $p > \frac{\sqrt{(\ell-1)^2+2\Delta\ell}-(\ell-1)}{2\Delta\ell}$.

The interesting question arises that what is the situation of the two-side game in a real networks such as the Internet, email network and social network? Is it close to the regular locality graph (like grid or unit-disk graph) or close to the small diameter graph with long-edges that are not created by locality?

Chapter 7

An application in the Internet

It is the weight, not numbers of experiments that is to be regarded.
— Isaac Newton (1643–1727)

Nowadays, our society is built upon many kinds of overlay networks such as social networks, e-banking systems, health care sensor networks, ubiquitous network and cloud-based systems. All of these networks are built on the top of the Internet's infrastructure. In itself, the Internet turns out to be vulnerable to attacks because it facilitates an infection starting from some particular location to quickly propagate and corrupt the whole network. Moreover, when a mechanism can attack different types of networks, due to their heterogeneity, recovering from the attack is more difficult. So, planetary-scale systems based on the Internet are vulnerable, not only because of their infrastructure, but also because of their heterogeneity and complexity.

Following the increasing impact of computer into the human life, many self-replicating computer viruses have been created. Among them, worms are a very serious threat to Internet security. For example, the CodeRed [53] and Nimda incidents of 2001 pointed out how vulnerable our Internet can be and how fast a worm can propagate through it. More recently, the Heartbleed issue in OpenSSH introduces an even more serious threat because many kinds of devices, servers, and clients relied on vulnerable versions of this protocol. At the scale of nations, cyber-attacks become a weapon in warfare between different countries. From the viewpoint of companies, an infection occurring at a world-scale of a big company can threaten or at least affect millions (or even billions) of users. The more heterogeneous and complex a system is, the more difficult it is to defend it against virus attacks and to prevent cascading failures.

An interesting question that arises how the two-side probabilistic game between propagation vs countermeasure is in the realistic network like the Internet. Is whether there exists a "last-line-of-defense" mechanism to contain the virus propagation at a local scale, provided that failed attack attempts can be detected, but not necessarily with perfect accuracy? And, if it is, then under what conditions are required?

This chapter introduces the experiment with a real trace data of the Internet topology collected by Caida project [15].

7.1 The Caida topology

This section focuses on a real network given by the IPv4 Internet topology collected by CAIDA project. We consider the network topology at router level in which a node in the network represents to a router on the Internet.

The topology consists of about 3 million IPv4 routers. Among them, the highest node degree is about 13 thousand, and there are more than 55 thousand nodes that have a degree higher than 50. We call these nodes *"high-degree"* nodes. On the other hand, there are more than 2 million nodes with degree lower than 3, among which more than 1.5 million nodes have degree 1. We designate nodes having degree smaller than 3 as *"low-degree"* nodes.

7.2 The experiment

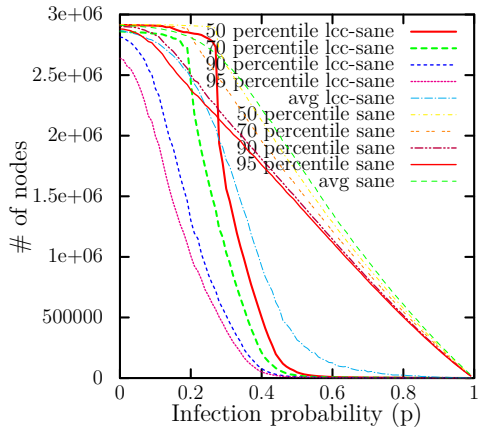
7.2.1 Simulation settings

With the discrete simulation written in C++ we study our containment strategies. We measure the number of infected, sacrificed, and sane nodes at the end of the spread. Because of the wide variance of the node degree, we analyze the difference when the infection starting from a *'low-degree'* node and from a *'high-degree'* node.

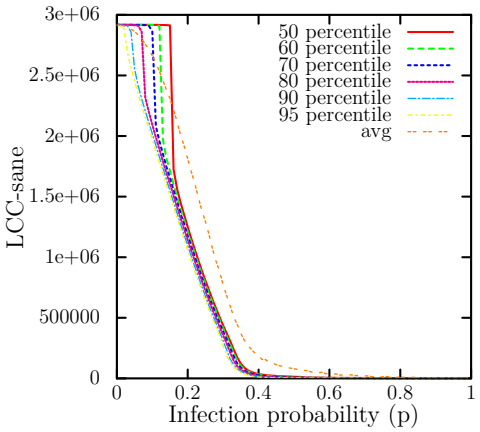
The probability of infection p and the probability of detection q are changed from 0 to 1 by 0.01 steps. The results are stated at the 95% confidence interval level for absolute value.

7.2.2 Choice of a metric

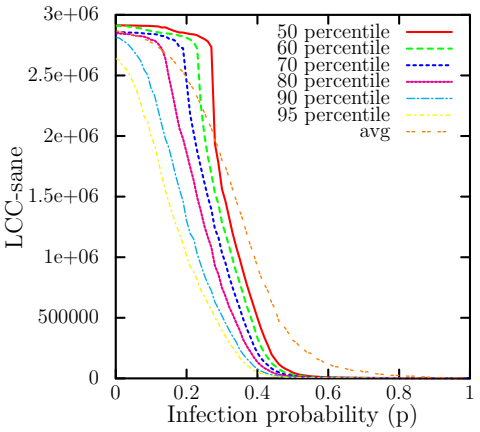
Figure 7.1a shows the performance of strategy K1-Hop in term of the number of sane nodes and the largest connected component of sane nodes (*LCC-sane*) at the end of the propagation starting from a *'high-degree'* node. The number of sane nodes is much greater than the largest connected component. It means that there are many small independent components as islands when the number of *'low-degree'* nodes is large. There are several reasons. Firstly, with the large number of low-degree nodes, there is a



(a) # of sane nodes and LCC-sane of $K1$-Hop



(b) 'LCC-sane' of $K0$-Hop

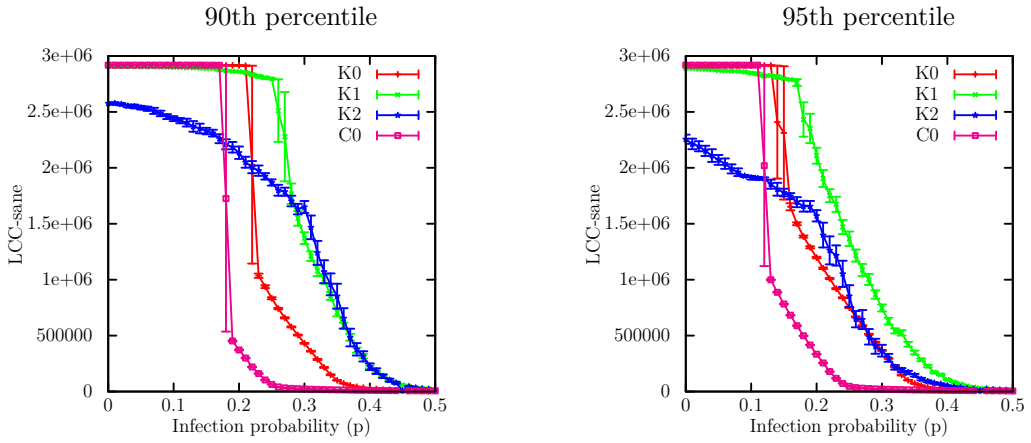


(c) 'LCC-sane' of $K1$-Hop

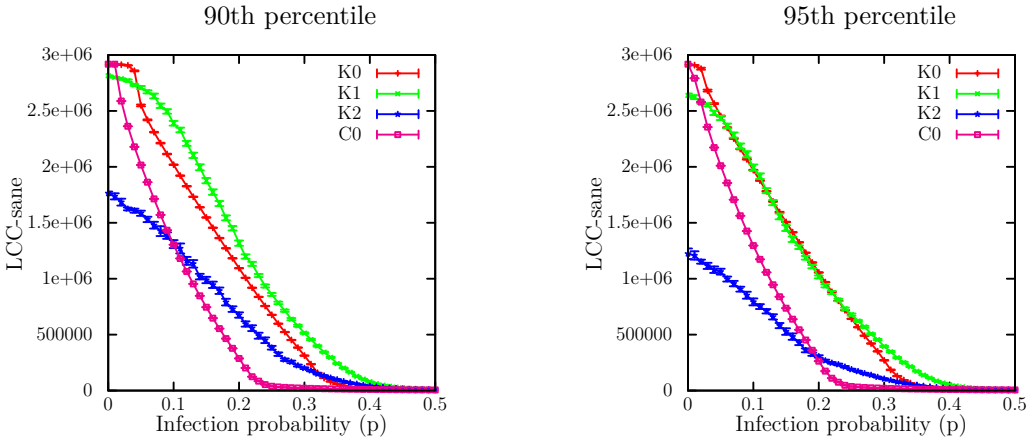
Fig. 7.1 The comparison of metrics when infection starts from a 'high-degree' node

high probability to have the case in which the deactivation of a high-degree node makes all low-degree neighbors that are still alive but disconnected to each other. Secondly, when there is a 'high-degree' infected node connecting to many nodes having degree 1, the probability of any of its neighbor to be infected linearly depends on the infection probability of each attack. Therefore, the number of sane nodes is not an exact metric to evaluate the performance of strategies at the end of the virus propagation. And we use the largest connected component of sane nodes (*LCC-sane*) to analyze the performance of different containment strategies.

Moreover, Figure 7.1b, 7.1c show the wide variant distribution of *'LCC-sane'*. Hence, the average of 'LCC-sane' does not reflect correctly the performance of strategy. The greater number of hops are deactivated, the wider variance of data are. In order to



(a) Infection starts with a 'low-degree' node



(b) Infection starts with a 'high-degree' node

Fig. 7.2 The comparison of different strategies on LCC-sane

increase the reliability of the system evaluation, we consider the 90th-percentile and 95th-percentile of data generated by $40,000$ times of running the simulation. The value of each percentile is stated at the level of 95% confidence interval.

7.3 The effect of countermeasures

Figure 7.2 presents the performance comparison of four strategies $K0$-Hop, $K1$-Hop, $K2$-Hop and $C0$ against the virus propagation in Caida in both cases when the infection starts from a "low-degree" and "high-degree" node. According to that Figure, the only place having a large confidence interval is at the epidemic threshold.

An epidemic threshold exists

We can see that, in both cases, there is an epidemic threshold where *"LCC-sane"* decreases sharply as the infection probability increases. The threshold is however less clear when the infection starts from a "high-degree" node. The reason is that as soon as there are some infected nodes and detectors; a large number of nodes and links will be killed or cut by the containment strategy. Therefore, when the infection probability is small, the infection starting from a "low-degree" node is less harmful than the infection starting from a "high-degree" node. However, when the infection probability increases, the consequences in both cases are similar.

Cutting strategies are ineffective in containing the spread

In other works [54, 79, 81], the most common strategy is used to mitigate virus propagation are blacklisting or cutting communication links with infected nodes, which is similar to strategy $C0$ in this paper. Surprisingly, it is not effective to contain the virus propagation. In other words, blacklisting is not enough to prevent the propagation of virus. Deactivating the detector and its neighbors is more effective than cutting the link carrying the infection.

Killing more than 1-Hop is pointless

In contrast, due to the small diameter of Caida, increasing the number of killed hop up to 2 does not bring benefit in isolating the virus spread. In fact, it kills too much nodes of the network. Figure 7.2b shows that when the infection starts from "high-degree" node, $K2$-Hop strategy kills more than a half of network nodes even the infection probability is around 0. When the infection starts from "low-degree" node, according to Figure 7.2a, the number of killed nodes is reduced but still a lot when compared with other strategies. Therefore, in the Internet, we can strictly limit the killing hop less than or equal to 1 in order to prevent the attack exploiting the killing strategy to do DoS attack.

Killing detector itself effective with small infection probability

When the infection probability is small, by saving a larger connected component at the end of spread, strategy $K0$-Hop is better than cutting strategies and $K1$-Hop. However, the difference between $K0$-Hop and $K1$-Hop is not so significant as the different with cutting strategies.

Killing 1-Hop neighbors is effective with greater infection probability

Figure 7.2 shows that, from the infection probability 0.2 with the infection starting "low-degree" node and 0.05 with the infection starting from "high-degree" node, the "LCC-sane" of $K0$-Hop sharply decreases while $K1$-Hop can still maintain a larger connected component of sane nodes. Interestingly, the difference between two strategies is more significant when the infection starts from the "low-degree" node. It means that $K1$-Hop strategy helps to mitigate the virus propagation by shifting the propagation threshold to higher infection probability.
Therefore, the benefit of using the $K1$-Hop strategy is clearer than the benefit of using the $K0$-Hop strategy. The advantage of $K1$-Hop is less significant when we consider an infection starting from a "high-degree" node with 95th percentile of data.

7.4 Summary of propagation in the Internet

We have studied (1) how last-line-of-defense strategies can contain the spread of a virus against an imperfect virus detector over Caida, and (2) how local strategies can help contain this propagation to prevent cascading failures.

After verifying the existence of a threshold and a phase transition for all strategies, we analyzed the effect of different factors on the choice of an effective strategy to shift that critical threshold to higher infection rate.

Surprisingly, we find that cutting communication links with infected nodes are not effective to isolate the infection. It is better to deactivate the node that detects the attack and possibly also its direct neighbors (K1hop). On the other hand, killing more than 1-hop neighbors from the detector is pointless because of the small diameter of the Caida topology. Killing up to 1-hop neighbors of detector helps mitigate the virus spread. In general, $K1$-Hop brings more significant benefit than $K0$-Hop.

The starting point of an infection seriously affects the chance to mitigate it. An infection starting from a "low-degree" node is less harmful to the network than one starting from a "high-degree" node when infection probability is small. However, when the infection probability increases, the effects are similar in both cases.

Containment can be achieved only for a small infection probability. Therefore, to be robust, a system must necessarily be well constructed in order to increase the detection ability of virus. After all, the mitigating strategies based on deactivating neighbors around the detector node should be considered as a last line of defense, to tolerate the

failure caused by the incompetence of the virus detection.

The tolerance to incompetence fault of a virus attack is concerned when several classes of smart viruses in real systems are introduced. These viruses have a wide range of different attack mechanisms or can always evolve to change the *virus signature* (such as polymorphic virus) such that the virus detector becomes unreliable quickly. Our model aims to model such kinds of viruses. When the virus detection cannot always detect the signature, it leads to a probabilistic model of infection and detection. With the virus having the wide range of attack, the infection probability can be bounded by the successful rate of the strongest attack. In reality, we can use multiple detection mechanisms at different defense layers, the first layer is the weakest mechanism, and the final layer is the strongest mechanism. When a virus can attack the weaker layer, the stronger inner layer that detects the attack can know the strength of the virus. Therefore, detector node will decide whether any strategy should be used to mitigate that virus.

Chapter 8

Conclusion

There is nothing permanent except change.
— Heraclitus (525–475 B.C.)

8.1 Research assessment

This dissertation includes the three following main contributions. The first contribution is the definition of dynamicity property of fault in distributed systems which provides the formal approach to study the change of the set of failed processes in system according to execution time. The second contribution is a fairer and more general model of *Intermittent Malicious Fault* created by malicious mobile agents as well as the tight bound on the number of faulty processes in this model to solve the agreement problem. The third contribution is the two-side probabilistic games between infection propagation against the countermeasure based on the imperfect infection detector. Some fundamental questions of the game are answered by both simulation and mathematical analysis, some other interesting questions are still open for the future research.

8.1.1 Dynamicity of fault

The change of the set of failed processes is systematically defined with three levels including *Permanent fault, Intermittent fault,* and *Propagating fault.* The permanent fault includes most of traditional failure models in term of both processes and communication channel failure. The Intermittent fault is a natural form of failure in communication channel.

Malicious mobile agent We model the malicious fault occurring in process as the malicious mobile agent. The goal is to separate the logical fault and the process. The fault with malicious agent can move from one process to another process. Our malicious

mobile agent is a generalization of the existing one. It has two properties that lead to different models: *the time of movement* in which agent moves from one process to another one; and *the self-replication* property which defines the ability of agent to create the copied versions of itself and attach into all sending messages to other processes. The self-replication defines the difference of Intermittent Malicious Fault and Propagating Fault.

8.1.2 Tight bound on Mobile Byzantine Agreement

In the presence of Intermittent malicious fault, we study Byzantine Agreement problem, a building block of distributed systems. We prove the tight bound on the total number of processes needed to tolerate a given number of malicious mobile agents whose movement time is alternatively defined in different models. In general, the value of this tight bound varies depending on the movement time of t malicious mobile agents:

- A larger number of correct processes ($5t+1$ in total number of processes) is required to tolerate the malicious agents when they move either between *compute/send* steps of two continuous rounds or between *receive/compute* steps of the same round.
- Differently, a smaller number of correct processes ($3t + 1$ in total number of processes) is required when the malicious agent moves between *send/receive* steps of the same round. The tight bound of this model is proved by Buhrman et al. [14].

  The difference is due to the number of processes whose messages can be corrupted by malicious agent in one sending effort.

The proposed model more realistic than the previous models based on the malicious mobile agent because it balances the power of the correct algorithm and the malicious agent. In this model, the malicious fault coming from the mobile agent can only control communication and computation activities of process but do not have the full permission in the memory. It is common in many modern authenticated computation machines and can be applied to some dynamic sensor networks or vehicle network.

8.1.3 Propagating fault

When malicious agent can self-replicate to duplicate itself into the sending message, the number of failed processes can grow from one to the whole system. In this case, the objective is to save some correct processes to maintain the availability of system.

This circumstance leads to a two-side probabilistic game between infection propagation and the countermeasure based on imperfect detection. The effect of countermeasure in containing the infection propagation is varied in different graph topologies.

The proposed model is the first one that considers a complete situation in the problem of defense an infection propagation. While the other models only consider two extreme cases where infection propagates without the countermeasure or the countermeasure can always detect the infection. This point becomes important especially when the current techniques cannot guarantee the perfect detection for some polymorphism and metamorphism virus.

Regular locality graphs In the graphs where the connectivity between two processes are based on the locality property such as grid/torus and unit-disk graph, the countermeasure based on killing strategy can contain the infection. When the infection probability increases, the increasing of hop to be killed helps to contains the infection. This conclusion is discovered by simulation and proved for the case of infinite grid.

Graph with long-edge Reversely, in the graphs where the connectivities between processes are not based on the locality such as small-world, because of the small diameter and the effect of long-edges, the infection cannot be contained. The similar situation applies for the case information propagation in social network and Internet, it is impossible for countermeasure to contain the spam content of worm.

Therefore, the countermeasures based on an imperfect detection should be only considered as the last line of defense after increasing the filtering of firewall and successful rate of detection.

8.2 Open questions and future research directions

The proposed models of *Intermittent* and *Propagating* faults with malicious mobile agent in this dissertation still have interesting open questions that lead to multiple directions of research.

8.2.1 Separated communication and moving topologies

In both models of Intermittent and Propagating fault, the topology in which messages are exchanged and the topology in which malicious agents move are unified. More specifically, the communication topology is a complete graph, and the graph where

malicious agents move is generalized for arbitrary graph with a simple characteristic that is directly implied from the impossibility proof.

The more general question about the characteristic of the two arbitrary graphs such that the agreement is possible to solve. It is interesting to consider these two graphs separately because in reality, the malicious agent can move independently to the communication graphs. For example, in mobile robot network or mobile sensor network, the agent or robot can move in a specific graph of a stationary graph of sensor or station.

8.2.2 Agreement in asynchronous/anonymous system

In this dissertation, the agreement problem is considered in authenticated synchronous system where the identity of process is unique and is not corrupted by malicious agent; and the communication is executed in synchronous round. The situation of Agreement is completely changed in asynchronous system while in static asynchronous system it is impossible even with one crash failure [33]. Similarly, in the model of malicious mobile agent, the question is that what synchrony level is required to do the agreement. In addition, another open question for anonymous distributed system is that how many correct processes are required to tolerate t malicious agents.

Looking for the weakest synchrony level or the anonymous makes the model become more realistic and enable to create some real application.

8.2.3 Multiple propagations in the same graph

Besides the generalization for separated topologies where malicious agent moves and where messages are exchanged, it would be interesting to see how the containment change if infected processes can recover then propagate that recovery to other infected processes. This additional ability gives more options to the defense side to contain the infection. The two-side game becomes the game of two or more propagations with tunable success probabilities versus one or two countermeasures.

List of Publications

[BDNP14] François Bonnet, Xavier Défago, Thanh Dang Nguyen, and Maria Potop-Butucaru. Tight bound on mobile byzantine agreement. In *Proceedings of the 28th International Symposium on Distributed Computing (DISC)*, pages 76–90 *(authors in alphabetical order)*, October 2014.

[NBD12] Thanh Dang Nguyen, François Bonnet, and Xavier Défago. Byzantine fault tolerant protocols for task management in mobile robots. In *The 18th IEEE Pacific Rim International Symposium on Dependable Computing*, page 3 pages, November 2012.

[NBD14a] Thanh Dang Nguyen, François Bonnet, and Xavier Défago. Analyzing the impact of mitigation strategies on the spread of a virus. Research Report IS-RR-2014-002, Japan Advanced Institute of Science and Technology (JAIST), August 2014.

[NBD14b] Thanh Dang Nguyen, François Bonnet, and Xavier Défago. Mitigating the spread of a virus in the internet. In *The 33rd IEEE Symposium on Reliable Distributed Systems/Workshop on Planetary-Scale Distributed Systems (SRDS/W-PSDS)*, page 6 pages, October 2014.

[NZC11] Thanh Dang Nguyen, Vyacheslav Zalyubovskiy, and Hyunseung Choo. Efficient time latency of data aggregation based on neighboring dominators in wsns. In *Proceedings of 2011 IEEE Global Communications Conference GLOBECOM*, pages 1–6, December 2011.

Bibliography

[1] Aguilera, M.K., Chen, W., Toueg, S.: Failure detection and consensus in the crash-recovery model. In: Kutten, S. (ed.) Distributed Computing, Lecture Notes in Computer Science, vol. 1499, pp. 231–245. Springer Berlin Heidelberg (1998), http://dx.doi.org/10.1007/BFb0056486

[2] Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.P., Porth, C.: Bar fault tolerance for cooperative services. ACM SIGOPS Operating Systems Review 39(5), 45–58 (Oct 2005), http://doi.acm.org/10.1145/1095809.1095816

[3] Anderson, R.M., May, R.M.: Population biology of infectious diseases: Part i. Nature 280(5721), 361–367 (Aug 1979)

[4] Anderson, R.M., May, R.M.: Infectious Diseases of Humans Dynamics and Control. Oxford University Press (1992)

[5] Avizienis, A., Laprie, J.C., Randell, B., Landwehr, C.: Basic concepts and taxonomy of dependable and secure computing. IEEE Transactions on Dependable and Secure Computing 1(1), 11–33 (Jan 2004), http://dx.doi.org/10.1109/TDSC.2004.2

[6] Bailey, N.: The Mathematical Theory of Infectious Diseases and its Applications. Griffin, London (1975)

[7] Banu, N., Souissi, S., Izumi, T., Wada, K.: An improved byzantine agreement algorithm for synchronous systems with mobile faults. International Journal of Computer Applications 43(22), 1–7 (Apr 2012)

[8] Bershad, B.N., Zekauskas, M.J., Sawdon, W.A.: The midway distributed shared memory system. Tech. rep., Pittsburgh, PA, USA (1993)

[9] Blagodurov, S., Zhuravlev, S., Dashti, M., Fedorova, A.: A Case for NUMA-aware Contention Management on Multicore Systems. In: Proceedings of the 2011 USENIX Conference on USENIX Annual Technical Conference. pp. 1–1. USENIXATC'11, USENIX Association, Berkeley, CA, USA (2011), http://dl.acm.org/citation.cfm?id=2002181.2002182

[10] Bollobás, B., Riordan, O.: Percolation. Cambridge University Press (2006)

[11] Bonnet, F., Défago, X., Nguyen, T.D., Potop-Butucaru, M.: Tight bound on mobile byzantine agreement. In: Kuhn, F. (ed.) Distributed Computing, Lecture Notes in Computer Science, vol. 8784, pp. 76–90. Springer Berlin Heidelberg (2014), http://dx.doi.org/10.1007/978-3-662-45174-8_6

[12] Bracha, G.: An o(log n) expected rounds randomized byzantine generals protocol. Journal of the ACM 34(4), 910–920 (Oct 1987), http://doi.acm.org/10.1145/31846.42229

[13] Brumley, D., Newsome, J., Song, D., Wang, H., Jha, S.: Towards automatic generation of vulnerability-based signatures. In: Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P'06). pp. 2–16 (May 2006)

[14] Buhrman, H., Garay, J.A., Hoepman, J.H.: Optimal resiliency against mobile faults. In: Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS'95). pp. 83–88 (1995), http://dl.acm.org/citation.cfm?id=874064.875660

[15] CAIDA: Ipv4 routed dataset. http://data.caida.org/datasets/topology/ark/ipv4/ (Feb 2014)

[16] Castañeda, A., Gonczarowski, Y.A., Moses, Y.: Unbeatable consensus. In: Proceedings of 28th International Symposium on Distributed Computing (DISC'14). pp. 91–106 (2014), http://dx.doi.org/10.1007/978-3-662-45174-8_7

[17] Castro, M., Liskov, B.: Practical byzantine fault tolerance. In: Proceedings of the Third Symposium on Operating Systems Design and Implementation. pp. 173–186. OSDI '99, USENIX Association, Berkeley, CA, USA (1999), http://dl.acm.org/citation.cfm?id=296806.296824

[18] Chakrabarti, D., Wang, Y., Wang, C., Leskovec, J., Faloutsos, C.: Epidemic thresholds in real networks. ACM Transactions on Information and System Security 10(4) (2008)

[19] Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM 43(2), 225–267 (Mar 1996), http://doi.acm.org/10.1145/226643.226647

[20] Charron-Bost, B., Schiper, A.: The heard-of model: computing in distributed systems with benign faults. Distributed Computing 22(1), 49–71 (2009), http://dx.doi.org/10.1007/s00446-009-0084-6

[21] Correia, M., Veronese, G.S., Lung, L.C.: Asynchronous byzantine consensus with $2f+1$ processes. In: Proceedings of the 25th ACM Symposium on Applied Computing (SAC'10). pp. 475–480 (2010), http://doi.acm.org/10.1145/1774088.1774187

[22] Cristian, F.: Understanding fault-tolerant distributed systems. Commun. ACM 34(2), 56–78 (Feb 1991), http://doi.acm.org/10.1145/102792.102801

[23] Daliot, A., Dolev, D.: Self-stabilizing Byzantine agreement. In: Proc. 25th ACM Symp. on Principles of Distributed Computing (PODC'06). pp. 143–152 (2006)

[24] De Marco, G., Rescigno, A.A.: Tighter bounds on broadcasting in torus networks in presence of dynamic faults. Parallel Processing Letters 10(01), 39–49 (2000)

[25] De Marco, G., Vaccaro, U.: Broadcasting in hypercubes and star graphs with dynamic faults. Information Processing Letters 66(6), 321 – 326 (1998), http://www.sciencedirect.com/science/article/pii/S002001909800074X

[26] Dijkstra, E.W.: Self-stabilizing systems in spite of distributed control. Commun. ACM 17(11), 643–644 (Nov 1974), http://doi.acm.org/10.1145/361179.361202

[27] Dobrev, S., Vrt'o, I.: Optimal broadcasting in hypercubes with dynamic faults. Information Processing Letters 71(2), 81–85 (1999), http://www.sciencedirect.com/science/article/pii/S0020019099000939

[28] Dolev, D.: The byzantine generals strike again. Journal of Algorithms 3(1), 14–30 (Mar 1982)

[29] Dolev, D., Fischer, M.J., Fowler, T.R., Lynch, N.A., Strong, H.R.: An efficient algorithm for byzantine agreement without authentication. Information and Control 52(3), 257–274 (Mar 1982), http://dx.doi.org/10.1016/S0019-9958(82)90776-8

[30] Dolev, D., Reischuk, R., Strong, H.R.: Early stopping in byzantine agreement. J. ACM 37(4), 720–741 (Oct 1990), http://doi.acm.org/10.1145/96559.96565

[31] Falliere, N.: Sality: Story of a peer-to-peer viral network. http://www.symantec.com/connect/sites/default/files/sality_peer_to_peer_viral_network.pdf (Jul 2011)

[32] Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM Journal on Computing 26(4), 873–933 (Aug 1997), http://dx.doi.org/10.1137/S0097539790187084

[33] Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM 32(2), 374–382 (Apr 1985), http://doi.acm.org/10.1145/3149.214121

[34] Fischer, M.J.: The consensus problem in unreliable distributed systems (a brief survey). In: Proceedings of the 1983 International FCT-Conference on Fundamentals of Computation Theory. pp. 127–140. Springer-Verlag, London, UK, UK (1983), http://dl.acm.org/citation.cfm?id=647891.739594

[35] Fraigniaud, P., Peyrat, C.: Broadcasting in a hypercube when some calls fail. Information Processing Letters 39(3), 115–119 (Sep 1991), http://dx.doi.org/10.1016/0020-0190(91)90105-Q

[36] Garay, J.A.: Reaching (and maintaining) agreement in the presence of mobile faults. In: Proceedings of the 8th International Workshop on Distributed Algorithms. vol. 857, pp. 253–264. Springer Berlin Heidelberg (1994), http://dx.doi.org/10.1007/BFb0020438

[37] Grassberger, P.: On the critical behavior of the general epidemic process and dynamical percolation. Mathematical Biosciences 63(2), 157–172 (Apr 1983)

[38] Hu, R., J., S., Arantes, L., Sens, P., Demeure, I.: Fair comparison of gossip algorithms over large-scale random topologies. In: Proceedings of the 31st IEEE Intl. Symp. on Reliable Distributed Systems (SRDS'12). pp. 331–340 (Oct 2012)

[39] Kephart, J.O., White, S.R.: Directed-graph epidemiological models of computer viruses. In: Proceedings of the 12th IEEE Symposium on Security and Privacy (S&P'91). pp. 343–361 (1991)

[40] Kephart, J.O., White, S.R.: Measuring and modeling computer virus prevalence. In: Proceedings of the 14th IEEE Symposium on Security and Privacy (S&P'93). pp. 2–15 (1993)

[41] Kermack, W.O., McKendrick, A.G.: A Contribution to the Mathematical Theory of Epidemics. Proceedings of the Royal Society of London. Series A 115(772), 700–721 (Aug 1927)

[42] Kruegel, C., Kirda, E., Mutz, D., Robertson, W., Vigna, G.: Polymorphic worm detection using structural information of executables. In: Proceedings of the 8th

International Conference on Recent Advances in Intrusion Detection (RAID'05). pp. 207–226. Springer-Verlag, Berlin, Heidelberg (Sep 2006)

[43] Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (Jul 1982), http://doi.acm.org/10.1145/357172.357176

[44] Lamport, L.: The part-time parliament. ACM Trans. Comput. Syst. 16(2), 133–169 (May 1998), http://doi.acm.org/10.1145/279227.279229

[45] Lamport, L.: Fast paxos. Distributed Computing 19(2), 79–103 (October 2006), http://research.microsoft.com/apps/pubs/default.aspx?id=64624

[46] Lee, M.J.: Pseudo-random-number generators and the square site percolation threshold. Physical Review E 78(3), 031131–1–11 (2008)

[47] Li, Z., Sanghi, M., Chen, Y., Kao, M.Y., Chavez, B.: Hamsa: fast signature generation for zero-day polymorphic worms with provable attack resilience. In: Proceedings of the 27th IEEE Symposium on Security and Privacy (S&P'06). pp. 32–47 (May 2006)

[48] Malarz, K., Galam, S.: Square-lattice site percolation at increasing ranges of neighbor bonds. Phys. Rev. E 71, 016125 (Jan 2005), http://link.aps.org/doi/10.1103/PhysRevE.71.016125

[49] Manchanda, N., Anand, K.: Non-Uniform Memory Access (NUMA). New York 1

[50] Mans, B., Santoro, N.: Optimal elections in faulty loop networks and applications. IEEE Transactions on Computers 4, 286–297 (1998)

[51] Martin, J.P., Alvisi, L.: Fast byzantine consensus. IEEE Transactions on Dependable and Secure Computing 3(3), 202–215 (Jul 2006), http://dx.doi.org/10.1109/TDSC.2006.35

[52] May, R.M., Anderson, R.M.: Population biology of infectious diseases: Part ii. Nature 280(5722), 455–461 (Aug 1979)

[53] Moore, D., Shannon, C., Brown, J.: Code-red: a case study on the spread and victims of an internet worm. In: Internet Measurement Workshop (IMW) 2002. pp. 273–284. ACM SIGCOMM/USENIX Internet Measurement Workshop, Marseille, France (Nov 2002)

[54] Moore, D., Shannon, C., Voelker, G., Savage, S.: Internet quarantine: requirements for containing self-propagating code. In: Proceedings of the 22nd Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM'03). vol. 3, pp. 1901–1910 (Mar 2003)

[55] Okun, M., Barak, A.: Efficient algorithms for anonymous byzantine agreement. Theory of Computing Systems 42(2), 222–238 (Jan 2008), http://dx.doi.org/10.1007/s00224-007-9006-9

[56] Oliveira, R., Guerraoui, R., Schiper, A.: Consensus in the crash-recover model. Research Report 97-239, Département d'Informatique, Ecole Polytechnique Fédérale, Lausanne, Switzerland (Aug 1997)

[57] Ostrovsky, R., Yung, M.: How to withstand mobile virus attacks (extended abstract). In: Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91). pp. 51–59 (1991), http://doi.acm.org/10.1145/112600.112605

[58] Pastor-Satorras, R., Vespignani, A.: Epidemic dynamics and endemic states in complex networks. Physical Review E 63(6), 066117–1–8 (2001)

[59] Pastor-Satorras, R., Vespignani, A.: Epidemic spreading in scale-free networks. Physical Review Letters 86, 3200–3203 (Apr 2001)

[60] Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM 27(2), 228–234 (Apr 1980), http://doi.acm.org/10.1145/322186.322188

[61] Raynal, M.: Fault-tolerant Agreement in Synchronous Message-passing Systems. Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers (2010)

[62] Reischuk, R.: A new solution for the byzantine generals problem. Information and Control 64(1-3), 23–42 (Jan-Mar 1985)

[63] Santoro, N., Widmayer, P.: Time is not a healer. In: Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89). pp. 304–313 (1989), http://dl.acm.org/citation.cfm?id=646505.694026

[64] Santoro, N., Widmayer, P.: Majority and unanimity in synchronous networks with ubiquitous dynamic faults. In: Proceedings of the 12th International Conference on Structural Information and Communication Complexity (SIROCCO'05). pp. 262–276 (2005), http://dx.doi.org/10.1007/11429647_21

[65] Santoro, N., Widmayer, P.: Distributed function evaluation in the presence of transmission faults. In: Asano, T., Ibaraki, T., Imai, H., Nishizeki, T. (eds.) Algorithms, Lecture Notes in Computer Science, vol. 450, pp. 358–367. Springer Berlin Heidelberg (1990), http://dx.doi.org/10.1007/3-540-52921-7_85

[66] Sasaki, T., Yamauchi, Y., Kijima, S., Yamashita, M.: Mobile byzantine agreement on arbitrary network. In: Proceedings of the 17th International Conference on Principles of Distributed Systems (OPODIS'13). pp. 236–250 (Dec 2013)

[67] Sasson, Y., Cavin, D., Schiper, A.: Probabilistic broadcast for flooding in wireless mobile ad hoc networks. In: Proceedings of the IEEE Wireless Communications and Networking (WCNC'03). pp. 1124–1130 (2003)

[68] Sayeed, H., Abu-Amara, M., Abu-Amara, H.: Optimal asynchronous agreement and leader election algorithm for complete networks with byzantine faulty links. Distributed Computing 9(3), 147–156 (1995), http://dx.doi.org/10.1007/s004460050016

[69] Schmid, U., Weiss, B., Keidar, I.: Impossibility results and lower bounds for consensus under link failures. SIAM Journal on Computing 38(5), 1912–1951 (Jan 2009), http://dx.doi.org/10.1137/S009753970443999X

[70] Schneider, F.B.: Byzantine generals in action: Implementing fail-stop processors. ACM Transactions Computer Systems 2(2), 145–154 (May 1984), http://doi.acm.org/10.1145/190.357399

[71] Schneider, F.B.: What good are models and what models are good? In: Mullender, S. (ed.) Distributed Systems (2nd Ed.), pp. 17–26. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA (1993), http://dl.acm.org/citation.cfm?id=302430.302432

[72] Staniford, S., Paxson, V., Weaver, N.: How to own the internet in your spare time. In: Proceedings of the 11th USENIX Security Symposium (USENIX-Security'02). pp. 149–167. USENIX Association, Berkeley, CA, USA (Aug 2002)

[73] Tel, G.: Introduction to Distributed Algorithms. Cambridge University Press, New York, NY, USA (1994)

[74] Van Mieghem, P.: The N-intertwined SIS epidemic network model. Computing 93(2-4), 147–169 (Dec 2011)

[75] Van Mieghem, P., Omic, J., Kooij, R.E.: Virus spread in networks. IEEE/ACM Transactions on Networking 17(1), 1–14 (2009)

[76] Warns, T.: Structural failure models for fault-tolerant distributed computing. Vieweg + Teubner research : Software engineering research, Vieweg + Teubner, Wiesbaden (2010), http://opac.inria.fr/record=b1131224

[77] Xia, J., Vangala, S., Wu, J., Gao, L., Kwiat, K.: Effective worm detection for various scan techniques. Journal of Computer Security 14(4), 359–387 (Jul 2006)

[78] Xu, W., Zhang, F., Zhu, S.: Toward worm detection in online social networks. In: Proceedings of the 26th Annual Computer Security Applications Conference (ACSAC'10). pp. 11–20. ACM, New York, NY, USA (Dec 2010)

[79] Zhou, L., Zhang, L., McSherry, F., Immorlica, N., Costa, M., Chien, S.: A first look at peer-to-peer worms: Threats and defenses. In: Proceedings of the 4th International Conference on Peer-to-Peer Systems (IPTPS'05). pp. 24–35. Springer-Verlag, Berlin, Heidelberg (2005)

[80] Zou, C.C., Gong, W., Towsley, D.: Code red worm propagation modeling and analysis. In: Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02). pp. 138–147. ACM, New York, NY, USA (2002)

[81] Zou, C.C., Towsley, D., Gong, W.: Modeling and simulation study of the propagation and defense of internet e-mail worms. IEEE Transactions on Dependable and Secure Computing 4(2), 105–118 (Apr 2007)