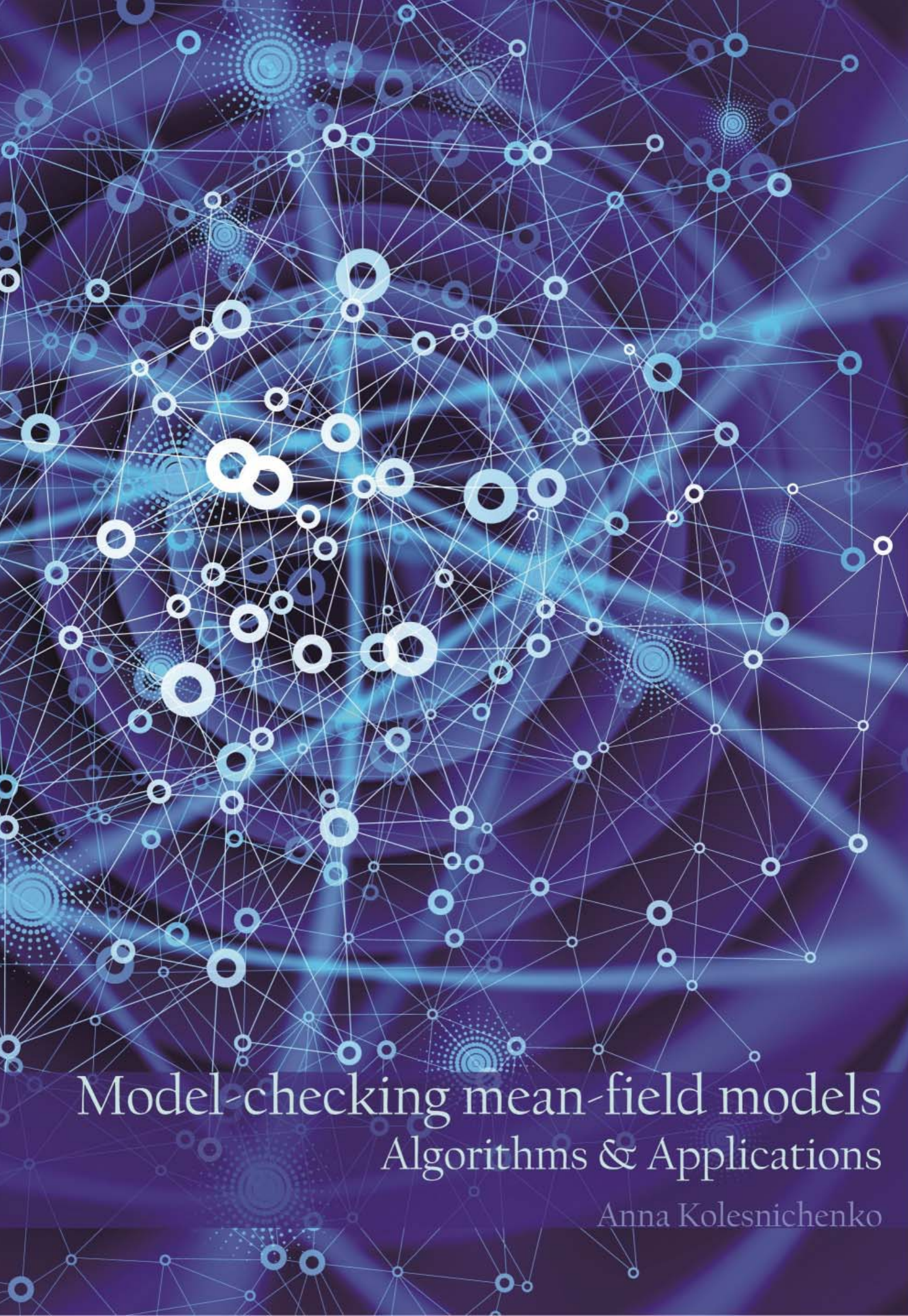


Model-checking mean-field models

Algorithms & Applications

Anna Kolesnichenko

Model-Checking Mean-Field Models: Algorithms & Applications

Anna Kolesnichenko

Graduation committee:

Chairman:	Prof.dr. Peter M.G. Apers
Promoter:	Prof. dr. ir. Boudewijn R. Haverkort
Promoter:	Prof. dr. Anne Remke
Assistant promoter:	Dr. ir. Pieter-Tjerk de Boer

Members:

Prof. dr. ir. Joost-Pieter Katoen	University of Twente
Prof. dr. Hans van den Berg	University of Twente
Prof. dr. Peter Buchholz	Technical University of Dortmund
Prof. dr. Jeremy Bradley	Imperial College London
Prof. dr. William H. Sanders	University of Illinois

CTIT Ph.D. - thesis Series No. 14-341
Centre for Telematics and Information Technology
University of Twente
P.O. Box 217, NL – 7500 AE Enschede

ISSN 1381-3617
ISBN 978-90-365-3821-3
DOI 10.3990/1.9789036538213
http://dx.doi.org/10.3990/1.9789036538213

Type set with LaTeX. Printed by Gildeprint Drukkerijen.
Cover illustration: www.derekdesign.ru

MODEL-CHECKING MEAN-FIELD MODELS: ALGORITHMS & APPLICATIONS

PROEFSCHRIFT

ter verkrijging van
de graad van doctor aan de Universiteit Twente,
op gezag van de rector magnificus,
prof. dr. H. Brinksma,
volgens besluit van het College voor Promoties,
in het openbaar te verdedigen
op woensdag 17 december 2014 om 16.45 uur

door

Anna Victorovna Kolesnichenko

geboren op 13 mei 1985
te Volgograd, Rusland

Dit proefschrift is goedgekeurd door:
Prof. dr. ir. Boudewijn R. Haverkort (promotor)
Prof. dr. Anne Remke (promotor)
Dr. ir. Pieter-Tjerk de Boer (assistent-promotor)

To my family
Моей семье

Abstract

Large systems of interacting objects are highly prevalent in today's world. Such system usually consist of a large number of relatively simple identical objects, and can be observed in many different field as, e.g., physics (interactions of molecules in gas), chemistry (chemical reactions), epidemiology (spread of the infection), etc. In this thesis we primarily address large systems of interacting objects in computer science, namely, computer networks. Analysis of such large systems is made difficult by the state space explosion problem, i.e., the number of states of the model grows exponentially with the number of interacting objects.

In this thesis we tackle the state-space explosion problem by applying mean-field approximation, which was originally developed for models in physics, like the interaction of molecules in a gas. The mean-field method works by not considering the state of each individual object separately, but only their average, i.e., what fraction of the objects are in each possible state at any time. It allows to compute the exact limiting behaviour of an infinite population of identical objects, and this limiting behaviour is a good approximation, even when the number of objects is not infinite but sufficiently large. In this thesis we provide the theoretical background necessary for applying the mean-field method and illustrate the approach by a peer-to-peer Botnet case study.

This thesis aims at formulating and analysing advanced properties of large systems of interacting objects using fast, efficient, and accurate algorithms. We propose to apply model-checking techniques to mean-field models. This allows (i) defining advanced properties of mean-field models, such as survivability, steady-state availability, conditional instantaneous availability using logic; and (ii) automatically checking these properties using model-checking algorithms. Existing model-checking logics and algorithms can not directly be applied to mean-field models since the model consist of two layers: the *local level*, describing the behaviour of a randomly chosen individual object in a large system, and the *global level*, which addresses the overall system of all

interacting objects. Therefore, we motivate and define two logics, called Mean Field Continuous Stochastic Logic (MF-CSL), and Mean-Field Logic (MFL), for describing properties of systems composed of many identical interacting objects, on both the local and the global level. We present model-checking algorithms for checking both MF-CSL and MFL properties, and illustrated these algorithms using an extensive example on virus propagation in a computer network. We discuss the differences in the expressiveness of these two logics as well as their possible combination.

Additionally, we combine the mean-field method with parameter fitting techniques in order to model real-world large systems, and obtain a better understanding of the behaviour of such systems. We explain how to build a mean-field model of the system, and how to estimate the corresponding parameter values, so as to find the best fit between the available data and the model prediction. We also discuss a number of intricate technical issues, ranging from the additional (preprocessing) work to be done on the measurement data, the interpretation of the data to, for instance, a restructuring of the model (based on data unavailability), that has to be performed before applying the parameter estimation procedures. To illustrate the approach we estimate the parameter values for the outbreak of the real-world computer worm Code-Red.

The techniques presented in this thesis allow an involved analysis of large systems of interacting objects, including (i) obtaining parameter values of mean-field model using measurements; (ii) defining advanced properties of the model; and (iii) automatically checking such properties.

Аннотация

В современном мире большое распространение получили системы, состоящие из большого количества сравнительно простых и идентичных взаимодействующих объектов. Такие системы встречаются, например, в физике (взаимодействие частиц газа), химии (химические реакции), эпидемиологии (распространение инфекций) и так далее. В данной диссертации мы фокусируемся главным образом на системах, состоящих из большого количества взаимодействующих объектов (далее больших системах) в теории вычислительных машин и систем, а именно, в компьютерных сетях. Анализ таких систем усложняется так называемым явлением взрыва пространства состояний (state-space explosion), что означает, что количество состояний растет экспоненциально вместе с количеством взаимодействующих объектов. Традиционно эта проблема решается при помощи так называемого mean-field method, что можно перевести как *метод среднего* или *самосогласованного поля*, который был первоначально разработан для моделирования взаимодействия молекул газа в физике. Аппроксимация с помощью метода среднего поля не позволяет анализировать состояние каждого объекта в системе, вместо этого анализируется доля (количество) объектов в каждом из возможных состояний. В данной диссертации мы приводим теоретические основы, необходимые для использования метода среднего поля, и иллюстрируем применение метода на примере анализа распространения так называемого децентрализованного ботнета (peer-to-peer Botnet).

Цель этой диссертации - создание быстрых, эффективных и точных методов для формулирования и анализа нетривиальных свойств (спецификаций) больших систем взаимодействующих объектов. Мы предлагаем использовать *методы проверки на моделях* (model-checking) для моделей среднего поля. Это позволит, во-первых, сформулировать нетривиальные свойства моделей, используя язык формальной логики, и, во-вторых, автоматически проверить удовлетворяет ли заданная модель системы фор-

мальным спецификациям. Существующие языки формальной логики и алгоритмы автоматической проверки не позволяют сформулировать и проверить свойства моделей среднего поля из-за того, что такие модели включают два уровня: *локальный*, описывающий произвольный объект в системе, и *глобальный*, описывающий всю систему. Это послужило причиной для создания нами двух новых языков формальной логики, которые называются Mean Field Continuous Stochastic Logic (MF-CSL) и Mean-Field Logic (MFL), что может быть переведо как *Непрерывная Стахостическая Логика Среднего Поля* и *Логика Среднего Поля.* Эти языки формальной логики позволяют формулировать спецификации для больших систем взаимодействующих объектов на обоих, локальном и глобальном, уровнях. Мы приводим алгоритмы автоматической проверки на модели и иллюстрируем их, используя пример распространения вируса в компьютерной сети. Мы также обсуждаем разницу между двумя предложенными языками и их возможную комбинацию.

Для того, чтобы лучше изучить данную реальную большую систему взаимодействующих объектов, мы предлагаем дополнить моделирование реальных систем за счет комбинации метода среднего поля и методов для поиска значений параметров модели (parameter estimation). В этой части диссертации мы объясняем, как данные, полученные при измерении системы, могут быть использованы для определения значений параметров модели этой системы. Мы также обсуждаем возможные технические сложности и необходимость предварительной обработки данных перед началом анализа. Для иллюстрации предложенного подхода мы определяем значения параметров модели широко известного компьютерного вируса (червя) Код-Красный (Code-Red worm), используя данные, полученные в 2001 году.

Методы, предложенные в этой диссертации, позволяют проводить широкий анализ реальных больших систем взаимодействующих объектов, что включает в себя, во-первых, определение значений параметров модели среднего поля такой системы, во-вторых, формулировку нетривиальных свойств полученной модели и, в-третьих, автоматическую проверку сформулированных свойств.

Contents

1

Introduction

Globalization is a process of international integration that aims at connecting different parts of the world. Globalization allows ideas, knowledge, people, and goods to move more easily around the globe. The new technologies, such as communication networks, both physical and digital, are being designed in order to support this process. Examples of such networks are wireless sensor networks for civil or military surveillance purposes, distributed peer-to-peer file sharing applications or malicious self-aggregating botnets, transport networks, etc. These communication networks play a critical role in a modern world, therefore, analysis of the behaviour of these systems is of our best interest. Typical problems addressed during such analysis include reliability, survivability, long run behaviour, speed of propagation, etc. In order to perform such analysis we first note that communication systems have a similar structure, namely, they often consist of a very large number of interacting objects. If each node is modelled explicitly, a formal performance or dependability evaluation of the system is limited to the restricted case where only a few objects participate since the global model for a realistic number of nodes most probably will suffer from state-space explosion.

Recently, much work has been done on the analysis of such large systems of interacting objects. Markovian Agents have been used to predict the propagation of earth quake waves [25] or the behaviour of sensor networks [46]. The dissemination of gossip information [7] and disease spread between islands [8] was analysed using mean-field approximation. Hybrid approaches, combining mean-field analysis and simulation, have been proposed for general systems of interacting objects [72], but also to predict predator and prey behaviour [54]. Ordinary differential equations (ODEs) have been used to analyze the behaviour of intracellular signalling pathways [23] and for epidemiological models [28] by using Performance Evaluation Process Algebra (PEPA).

Out of the many available approaches, in this thesis we have chosen the

mean-field method which allows for a quick and accurate analysis of systems consisting of a large number of interacting objects, while avoiding the state-space explosion problem. The mean-field approximation is based on the continuous representation of the a discrete system. Instead of the behaviour of the individual nodes in the system the behaviour of the whole system is addressed. The whole system behaviour is described via the average behaviour of the individual objects (nodes).

1.1 The aim of the thesis and research questions

To be able to perform analysis of a large systems of interacting objects we aim to formulate and analyse advanced properties of such systems using fast, efficient, and accurate algorithms. The above goal can be divided into three sub-goals, namely:

- modelling such systems at a reasonable level of abstraction;
- parametrizing the models with realistic values;
- describing and checking properties of the models.

By dividing the aim of the thesis into three parts we obtained the three research questions which will be addressed in this thesis.:

> $\mathcal{Q}1$**: Can mean-field method be used for fast, efficient, and accurate analysis of large systems of interacting objects?**
>
> $\mathcal{Q}2$**: How to obtain realistic parameter values for mean-field models?**
>
> $\mathcal{Q}3$**: How to express and automatically check advanced properties of mean-field models?**

In the following section we discuss a recent example of a real large system of interacting objects as in [64], namely, the Stuxnet virus. We will use this example as an illustration of the objectives that are of interest in this thesis, and to illustrate issues that might arise while dealing with such large systems.

1.2 Research questions: Illustration

Stuxnet is known as one of the most complex computer viruses; it was primarily written to target Industrial Control Systems (ICSs). Recently, many papers and reports were published on the analysis of Stuxnet's code, [40], [61]. Moreover, the Boolean logic Driven Markov Processes have been used to model the fundamental mechanisms of the Stuxnet attack [68]. Quantitative analysis of Stuxnet has not been done so far, mostly because the necessary information was not readily available. However, quantitative analysis can be very useful, for example to obtain better insight in the spreading process and to analyse the efficiency of counter-measures.

Stuxnet was first discovered in July 2010; however, it had been operating without being noticed for at least one year prior to its detection. The virus uses both known and unknown Windows vulnerabilities to install and propagate. During the propagation phase, Stuxnet behaves similarly to known worms and botnets. Once it reaches its target, it sabotages the system by reprogramming Programmable Logic Controllers (PLCs), which can lead to a disaster, e.g. damage to the production of the centrifugal machines in Iranian nuclear enrichment facilities.

The behaviour of Stuxnet consists of three phases: spreading, obtaining access to the PLC, and sabotage. In the present example (and other examples in this thesis), we only address the spreading phase. Modelling of the attacking and sabotaging phases is of less interest, since once the target is reached, Stuxnet accomplished its mission almost surely.

Stuxnet has the ability to propagate using different methods. We classify them for further discussion as follows (see Table 1):

- propagation via USB flash drives and other removable media;
- propagation via a network;
- propagation via shared folders.

Copying itself to removable drives is the main method of propagation, since ICSs are usually programmed through computers that are not connected to a network. Operators use removable drives to exchange data, and once the infected removable drive is inserted into a new computer, Stuxnet will copy itself and its supporting files. The newly compromised computer can infect other USB drives afterwards.

	Local	Remote
Manual	Removable drives	Shared folder
Automatic	–	Network

Table 1.1: Classification of propagation mechanisms

Propagation via a network can be seen as botnet or worm spread, which have been recently studied and modelled, e.g., [22], [42]. Note that network propagation is the only fully automatic way of spreading.

The third way of propagation includes infection via shared folders or network drives, and print spooler services. For example, Stuxnet will execute on each computer where a compromised folder is used.

Stuxnet spreads mainly within company networks. However, propagation between networks of different companies is possible if, for example, the compromised computer has a VPN connection to an outside network, or an infected USB stick is taken to the outside network (and used there).

The behaviour of Stuxnet is controlled remotely. After installation, the virus contacts a command and control (C&C) server and sends information about the compromised computer. The C&C servers are mostly used for spreading new versions of the virus. However, the ability to receive information from outside can be used by attackers to help the worm propagate through specific target networks or, alternatively, stop propagation.

Given the above description of the Stuxnet behaviour one can proceed with the quantitative analysis of the virus. The valuable question now is: how can we analyse this system?

$\mathcal{Q}1$: Can mean-field method be used for fast, efficient, and accurate analysis of such large systems?

Out of many approaches, in this thesis we select the *mean-field method*. The main idea of the mean-field analysis is to describe the evolution of a population that is composed of many similar objects via a deterministic behaviour. The full description on how to built a mean-field model of such systems will follow in Chapter 2 of the thesis.

With respect to the Stuxnet example, we have shown that it is possible to build such a model in [64]. However, in order to conduct a meaningful quantitative analysis of Stuxnet and similar large systems values for many

model parameters (such as infection rates) are needed. Hence, another relevant research question we want to address is the following.

$\mathcal{Q}2$: How to obtain realistic parameter values for mean-field models?

Unfortunately, obtaining such parameter values is not trivial. The automatic spreading via the network is probably the easiest to parametrize, since it does not involve humans. One could obtain values for these parameters analysing the Stuxnet code, or by doing measurements on live infected computers. This is not trivial for several reasons:

- it either needs a sufficiently large test-bed, or a real target environment;
- accurate measurements may take a long time since Stuxnet does not tend to spread very quickly;
- results may be inaccurate due to the "synthetic" environment.

Aspects that involve humans are even harder to parametrize; in the case of Stuxnet, this includes the propagation via shared folders and removable USB drives, and the influence of the C&C server. Such parameters are in general difficult to obtain, since they require knowledge of a large part of the internet community. However, in 2010 a report has been published that takes into account the human factor in cybercrime [94].

Having the mean-field model built (and parametrized) one can obtain knowledge about transient and, possibly, stationary behaviour of the system. Moreover, an automated way to express and check the advanced properties of the model, e.g., survivability, steady-state availability, conditional instantaneous availability, etc, might be of a great interest. Therefore, the last question we have is as follows:

$\mathcal{Q}3$: How to express and automatically check advanced properties of mean-field models?

Such properties can be expressed by using model-checking techniques, therefore, introducing a logic and algorithms for describing and automatic checking properties of the mean-field model of a large system, like Stuxnet, might be beneficial. However, this is not a trivial task due to the nature of the mean-field model. We describe the challenges of model-checking mean-field model in the following section.

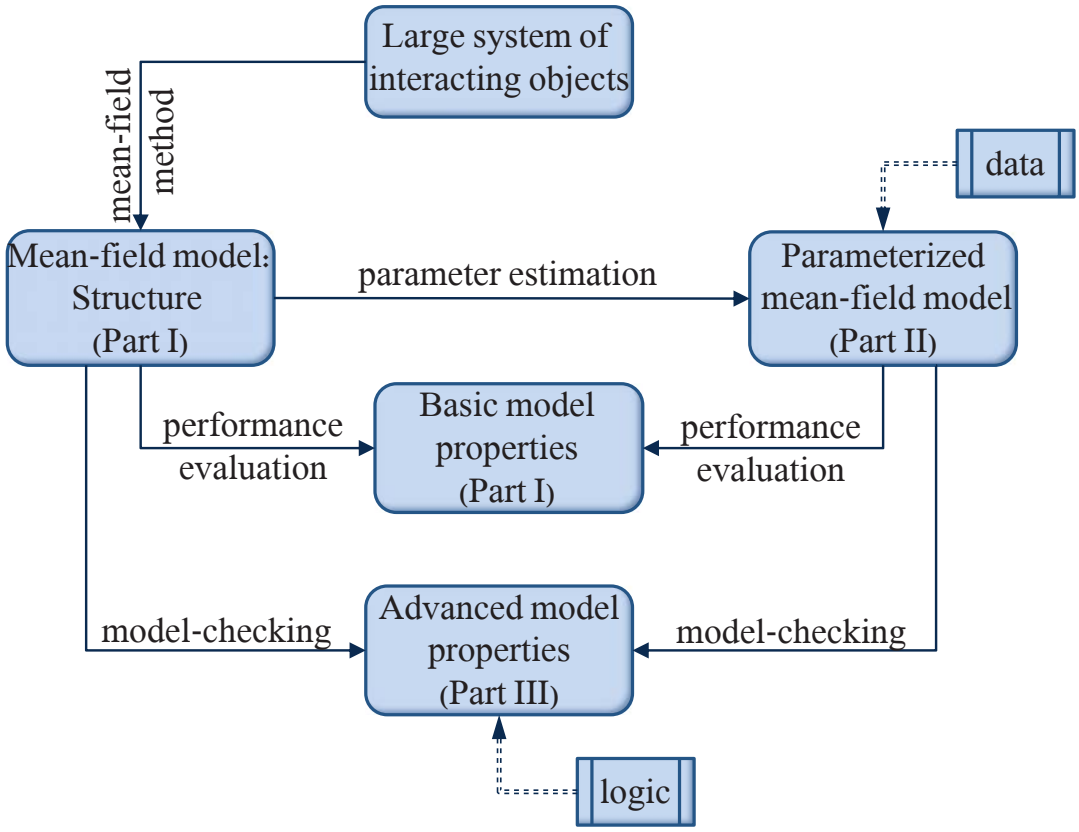


Figure 1.1: The structure of the proposed approach and the outline of the thesis.

1.3 Approach

In the previous section we defined three main questions which will be addressed in this thesis. When these questions are combined the approach towards the aim of the thesis becomes apparent.

Figure 1.1 illustrates the proposed approach. The analysis of a large system starts with building a mean-field model. This model can then either be parametrized using real data, or used without assigning realistic parameters values. Even without complete knowledge of the parameter values, potentially interesting results may be obtained. For example, by trying different values for the unknown parameters, the sensitivity of the final results to them can be studied, and possibly upper and lower bounds obtained. Moreover, advanced

properties of such model can be checked using model-checking algorithms. If the model can be parametrized, obtaining basic (e.g., transient behaviour, stationary behaviour, bounds, etc.) or advanced (e.g., survivability, availability, etc.) properties yields potentially even more realistic results.

As one can see, both complete and partial combinations of answers to the above questions can yield interesting results, e.g.,

- combination $\mathcal{Q}1$, $\mathcal{Q}2$ and $\mathcal{Q}3$ allows building and parametrizing a mean-field model, and automatically check advanced properties of this model;
- combination $\mathcal{Q}1$ and $\mathcal{Q}2$ allows building and parametrizing a mean-field model, and obtaining basic properties of this model;
- combination $\mathcal{Q}1$ and $\mathcal{Q}3$ allows building a mean-field model, and automatically checking advanced properties of this model.

In the following we briefly introduce the three main topics of this thesis. Each of the presented topics allows us to answer one of the research questions, and will be covered in more detail further in the respective part of the thesis.

1.3.1 Mean-field method

Mean-Field Approximation originated in statistical physics [4] and is a technique developed within the field of probability theory. This technique is useful to study the behaviour of stochastic processes with a very large state space, e.g., in the study of systems with a large number of particles, where Monte Carlo simulations are impractical. Beyond physics, this approximation technique has been applied in studies of, e.g., epidemics models [63], queueing theory [15], [4], and network performance [72], [26].

Classical applications of this technique generally require two abstractions. The first is that when studying the system, one abstracts the objects' identities, and instead of capturing the behaviour of each object instance, the system's behaviour is observed at the level of populations [59]. The second abstraction suggests that the spatial distribution of the objects across the system locations is ignored, and the "particles" are assumed to be uniformly spread across the system space (in chemistry this idea is embodied in the notion of well-stirred chemical reaction [44], [98]).

Process algebra is a high-level formalism, which is being widely used in performance modelling due to the well defined and convenient structure. Continuous

Time Markov Chains (CTMCs) are often used to provide a stochastic semantics to process algebra used in performance modelling of computer systems [56]. However, stochastic process algebra models of realistic size can easily result in very large and intractable state-spaces. In that context a technique called *fluid-flow approximation* [57] has been used to construct a continuous state-space representation of the underlying discrete state-space, and ordinary differential equations are used to describe their dynamics. This technique corresponds to results on mean-field approximation of CTMCs [97], [59], [51]. Indeed, the notion of fluid approximation has been used in various contexts such as Petri nets, and relies on the idea that a discrete variable can be approximated using a continuous variable [89].

Applying mean-field analysis from the computer science perspective requires the following major steps: (1) describing how a large population of interacting objects evolves by means of a system of differential equations, (2) finding the emergent deterministic behaviour of the system by solving such differential equations, and (3) analysing properties of this behaviour. Moreover, a *local mean-field model* can be obtained based on the mean-field model of the whole population, which allows to study the behaviour of individual objects within the whole system in an efficient way.

1.3.2 Parameter estimation

Model-based evaluation is widely used for real systems, however, it is often difficult to obtain realistic parameter values for the models. This is particularly the case for large scale distributed systems, like applications running in the internet, for which a structured measurement set-up is very difficult to achieve, or even impossible to obtain. To ensure that the model complies with the system, parameters can be assigned by, for example, one (or a combination) of the following methods: (1) predefined experimental settings; (2) analysis of a real system; (3) measurements. In this thesis we follow the third approach: we discuss how measured data can be used to obtain the parameters of a mean-field model. Doing this we expand the application of mean-field approximation to real-world systems by estimating realistic parameters of the obtained model.

Parameter estimation techniques are widely used in application areas such as biochemical reactions [76], computer vision [106], cosmology [73], etc. In this thesis we combine a *mean-field* model of worm behaviour with *parameter fitting* techniques, and illustrate their combination on the case of the *Code-Red*

worm. We will use two well-known parameter estimation methods, namely, *squared error* [1] and *maximum likelihood* [78] in this thesis. We build a mean-field model of Code-Red worm and obtain the parameter values based on the real data using the above two estimation techniques.

1.3.3 Model-checking

In the course of the last few years the mean-field method was widely used for the analysis of large systems of interacting objects. In the past the method was mainly used for performance evaluation. In this thesis we propose to apply *model-checking* techniques to mean-field models.

Model-checking means checking whether a system state satisfies certain properties. It was initially introduced for finite deterministic models, for validation of computer and communication systems, and later extended towards stochastic models and models with continuous time [5], [6]. Model-checking models of large systems is made difficult by the state-space explosion problem. Since the mean-field method avoids this problem, mean-field models can potentially be checked using model-checking techniques. However, the direct application of model-checking techniques to mean-field models is challenging due to the following reasons:

- there is no readily available techniques which can directly be applied to model-check a mean-field model;
- the mean-field model has two layers (global and local), therefore it is essential to be able to formulate properties on both levels;
- as we will see, the local mean-field model is a *time-inhomogeneous* Markov chain (ICTMC), therefore the results of the model-checking procedure depend on time;
- the state-space of the global mean-field model is infinitely large, hence, capturing exact satisfaction set is difficult.

In this thesis we face the above challenges and introduce and motivate two logics, called *Mean Field Continuous Stochastic Logic* (MF-CSL), and *Mean-Field Logic* (MFL), for describing properties of systems composed of many identical interacting objects. The two logics have been defined to be able to express timed properties on both local and global levels. MF-CSL first

expresses the property of a random node in a system (including timed properties) and then lifts this to the system level using *expectation operators*. In contrast, MFL expresses the property of the overall system directly and it does not take into account the behaviour of the individual objects. The new model-checking algorithms are presented and illustrated using an extensive example on virus propagation in a computer network. We discuss the differences in the expressiveness of these two logics as well as their possible combination.

1.4 Structure of the thesis

The thesis consists of eleven chapters, which are grouped into three parts, where each part corresponds to one of the research questions, presented above.

Part I provides theoretical background on the mean-field method, which is illustrated by a case-study on peer-to-peer Botnet spread.

> *Chapter 2* defines the mean-field model, discusses the mean-field convergence theorem, and practical extensions to the theorem, including behaviour in the stationary regime and a definition of the local mean-field model.
>
> *Chapter 3* discusses how to build the mean-field model of a peer-to-peer Botnet. It compares the results obtained by the mean-field approach to those obtained from simulation. In addition, it provides examples of more advanced studies that can be performed using mean-field analysis.

Part II discusses how to obtain the parameters of a mean-field model using real data on the Code-Red worm example.

> *Chapter 4* motivates the performed case-study. It discusses background information on the parameter estimation methods and related work.
>
> *Chapter 5* provides the set-up of the case-study. It discusses background of the Code-Red worm, the available data, and the mean-field model of the worm behaviour.
>
> *Chapter 6* contains the results of the case-study and concluding remarks.

Part III introduces model-checking techniques for mean-field models.

Chapter 7 provides a motivation and discusses related work.

Chapter 8 defines the logic MF-CSL for checking properties of mean-field models. It discusses model-checking algorithms, whicch can be used to check MF-CSL properties.

Chapter 9 introduces the logic MFL together with the corresponding model-checking algorithms, and discusses the satisfaction set development.

Chapter 10 provides a comparison of the two logics, and discusses the combination of the two logics.

Chapter 11 summarizes the content of the thesis and indicates future research directions.

Part I

Mean-Field Method

The main idea of the mean-field analysis is to describe the evolution of a population that is composed of many similar objects via a deterministic behaviour. It states that under certain assumptions on the dynamics of the system and when the size of the population grows, the ratio of the system's *variance* (standard deviation) to the size of the state space of the whole population tends to zero. Therefore, when the population is large, the stochastic behaviour of the system can be studied through the *unique solution* of a system of Ordinary Differential Equations (ODE)s defined by using the limit dynamics of the whole system.

In this part we first provide the theoretical background on the mean-field method and illustrate the usability of the approach by a case-study on peer-to-peer Botnet spread. We define overall and local mean-filed models and recall the mean-field convergence theorem in Chapter 2. The application of the mean-field method to a peer-to-peer Botnet is discussed in Chapter 3.

2

Mean-field method

In this chapter we provide the theoretical background on the mean-field method based on [67]. The presented way of reasoning is slightly different from, for example, [20], but is more useful for the further developments, presented in this thesis. We describe how to construct the *local* and *global* mean-field models and use the reformulation of the classical Kurtz's Theorem [69], instead of defining *population models*. We provide small examples for each essential part to illustrate the practical application of the theoretical material.

This chapter is further organized as follows. Section 2.1 defines overall mean-field model. The convergence theorem is discussed in Section 2.2. Finally, the valuable extensions to the convergence theorem are described in Section 2.3.

2.1 Model definition

Let us start with a random individual object which is part of a large population. We assume that the size N of the population is constant; furthermore, we do not distinguish between classes of individual objects for simplicity of notation. However, these assumptions can be relaxed, see, e.g., [26].

The behaviour of a single object can be described by defining the state-space $S^l = \{s_1, s_2, \dots s_K\}$ that contains the states or "modes" this object may experience during its lifetime, the labelling of the state space $L : S^l \to 2^{LAP}$ that assigns *local atomic propositions* from a fixed finite set of Local Atomic Properties (LAP) to each state; and the transitions between these states.

Example 2.1.1. *We introduce the model defining the modes of an individual computer, which is exposed to the infection. Such a machine can be* not-infected*,* infected and active *or* infected and inactive*. An infected computer is active when it is spreading the virus and inactive when it is not. This results*

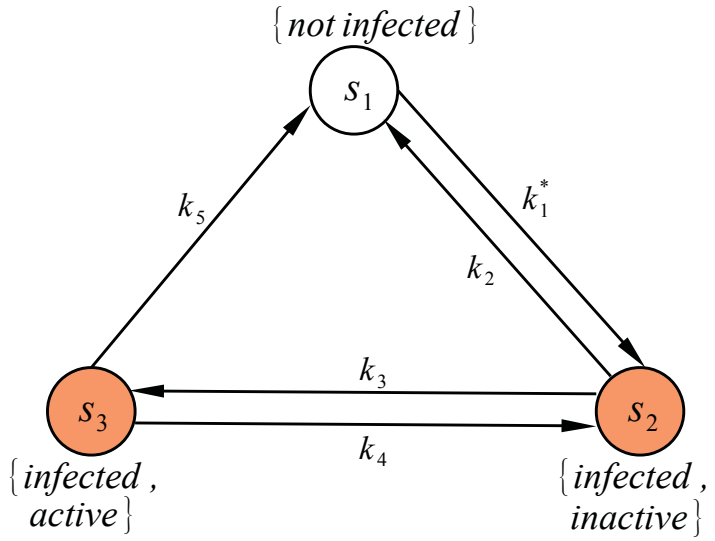


Figure 2.1: The model describing computer virus spread.

in the finite local state space $S^l = \{s_1, s_2, s_3\}$ *with* $|S^l| = K = 3$ *states. These states are labelled as* infected, not infected, active *and* inactive, *as indicated in Figure 2.1. Formally,* $L(s_1) = \{$*not infected*$\}$*;* $L(s_2) = \{$*infected, inactive*$\}$*;* $L(s_3) = \{$*infected, active*$\}$*; and the set of local atomic properties is given by* $LAP = \{$*not infected, infected, inactive, active*$\}$*.* ○

In the following we will consider a large population of N objects, where each individual is modelled as described above, and denoted as $\mathcal{M}_i$ for $i \in [1, N]$. Let us first try to preserve the identity of each object and build the model, describing the behaviour of N objects individually. It is easy to see that when the population grows linearly the size of the state-space of the model grows exponentially. For example, when the system is composed of three computers (as in Example 2.1.1), the size of the state space is $3^3 = 27$ states, where each state represents the mode of all three computers individually, and is labelled accordingly as, e.g., *{{ not infected}, {not infected}, {not infected}}*. As one can see, for a large number of computers N such model with 3^N states might be too large to handle. Fortunately, the mean-field approach allows modelling such a system of indistinguishable objects and avoids exponential growth of the

state-space (*state space explosion*). In the following we describe the method in more detail and explain how it can be applied to the computer virus example.

Given the large number of objects, where each individual is modelled by $\mathcal{M}$, we proceed to build the overall model of the whole population. We assume that all objects behave identically, therefore, we can move from the *individual* representation to a *collective* one, that does not reason about each object separately, but gives number (fraction) of individual objects in a given state of the model $\mathcal{M}$. It is done by taking the following steps:

Step 1. Lump the state space. When preserving the identity of the objects in a population $(\mathcal{M}_1, \mathcal{M}_2 \ldots \mathcal{M}_N)$ the sequence of the models of individual objects can be considered as a model of the population. However, the size of such sequence depends on N. Due to the identical and unsynchronized behaviour of the individual objects, a *counting abstraction* (or transition from the individual to a collective representation) is used to find a smaller stochastic process, denoted as $\mathbf{M}^{(N)}$, whose states capture the number of the individual objects across the states of the local model $\mathcal{M}$:

$$\mathbf{M}_j^{(N)} = \sum_{i=1}^{N} \mathbb{1}\{\mathcal{M}_i = j\}.$$

The state of $\mathbf{M}^{(N)}$ at time t is a counting vector $\overline{M}(t) = (M_1(t), M_2(t), \ldots, M_K(t))$, where $M_i \in \{0, \ldots, N\}$, and $\sum_{i=1}^{K} M_i = N$. The initial state is denoted as $\overline{M}(0)$.

Step 2. Defining transition rates. Given $\mathbf{M}^{(N)}$ and $\overline{M}(0)$ as defined above the Continuous Time Markov Chain (CTMC) $\mathcal{M}^{(N)}(t)$ can be easily constructed. The transition rates are defined as follows [15]:

$$\mathbf{Q}_{i,j}(\overline{M}(t)) = \begin{cases} \lim_{\Delta \to 0} \frac{1}{\Delta} \mathrm{Prob}\Big\{ \mathcal{M}(t+\Delta)) = j \,| \\ \qquad\qquad \mathcal{M}(t) = i, \overline{M}(t) \Big\}, & \text{if } M_i(t) > 0, \\ 0, & \text{if } M_i(t) = 0, \\ -\sum_{h \in S^l, h \neq i} \mathbf{Q}_{i,h}(\overline{M}(t)), & \text{for } i = j, \end{cases}$$

where $\mathcal{M}(t)$ indicates the state of the individual object at time t. The transition matrix depends on time via $\mathcal{M}(t)$.

Step 3. Normalize the population. For the construction of the mean field model which does not depend on the size of the population the state vector is normalized as follows:

$$\overline{m}(t) = \frac{\overline{M}(t)}{N},$$

where $0 \leq \overline{m}_i(t) \leq 1$ and $\sum_{i=1}^{K} m_i = 1$.
When normalizing, first we have to make sure that the related transition rates are scaled appropriately. The transition rate matrix for the normalized population is given by:

$$Q(\overline{m}(t)) = \mathbf{Q}(N \cdot \overline{m}(t)).$$

Secondly, the initial conditions have to scale appropriately. this is commonly called *convergence of the initial occupancy vector* [29], [30]:

$$\overline{m}(0) = \frac{\overline{M}(0)}{N}.$$

The overall mean-field model can then be constructed as follows.

Definition 2.1.2 (Overall mean-field model)**.** *An overall mean-field model $\mathcal{M}^{\mathcal{O}}$ describes the limit behaviour of $N \to \infty$ identical objects and is defined as a tuple (S^o, Q), that consists of an infinite set of states:*

$$S^o = \{\overline{m} = (m_1, m_2, \ldots, m_K) | \forall j \in \{1, \ldots, K\},$$
$$m_j \in [0,1] \wedge \sum_{j=1}^{K} m_j = 1\},$$

where $\overline{m}$ is called occupancy vector, and $\overline{m}(t)$ is the value of the occupancy vector at time t; m_j denotes the fraction of the individual objects that are in state s_j of the model $\mathcal{M}$. The transition rate matrix $Q(\overline{m}(t))$ consists of entries $Q_{s,s'}(\overline{m}(t))$ that describe the transition rate of the system from state s to state s'.

□

Note that for any finite N the occupancy vector $\overline{m}$ is a discrete distribution over K states, taking values in $\{0, \frac{1}{N}, \frac{2}{N}, \ldots, 1\}$, while for infinite N, the m_i are real numbers in $[0, 1]$.

To illustrate the relation between the model of a single object and the overall mean-field model of the whole system, we continue to develop the mean-field model for the virus spread example.

Example 2.1.3. *We assume that all computers in the system behave according to the model described in Example 2.1.1. Given a system of N computers, we can model the limiting behaviour of the whole system through the overall mean-field model, which has the same underlying structure as the individual model (see Figure 2.1), however, with state space* $S^o = \overline{m} = (m_1, m_2, m_3)$*, where* m_1 *denotes the* fraction *of not-infected computers, and* m_2 *and* m_3 *denote the fraction of inactive and active infected computers, respectively. For example, a system without infected computers is in state* $\overline{m} = (1, 0, 0)$*; a system with 50% not infected computers and 40% and 10% of inactive and active infected computers, respectively, is in state* $\overline{m} = (0.5, 0.4, 0.1)$.

The transition rates $k_1^*,\ k_2,\ k_3,\ k_4,\ k_5$ *represent the following: the infection rate* k_1^**, the recovery rate for an inactive infected computer* k_2*, the recovery rate for an active infected computer* k_5*, and the rates with which computers become active* k_3 *and return to the inactive state* k_4*. Rates* $k_2, k_3, k_4,$ *and* k_5 *only depend on the properties of the modelled computer virus and do not depend on the overall system state. The infection rate* k_1^* *does depend on the fraction of infected and active computers, and the fraction of not-infected computers. We discuss the generator matrix in the next section.* ◯

2.2 Mean-field analysis

Here we express a reformulation of Kurtz's theorem [69] which relates the behaviour of the sequence of models $\mathcal{M}_1, \mathcal{M}, \ldots, \mathcal{M}_N$ with increasing population sizes to the limit behaviour. We reformulate the theorem to make it more applicable to the further chapters of this thesis.

Before the theorem can be applied one has to check whether the overall mean-field model satisfies the following two conditions:

1. the model preserves the so-called *density dependence* condition in the limit $N \to \infty$ for all $N > 1$. This means that transition rates scale together with the model population, so that in the normalized models they are independent of the size of the population.

2. The rate functions are required to be Lipschitz-continuous (informally it means that rate function are not too steep).

When the three steps for constructing the mean-field model are taken and the above mentioned conditions are satisfied Kurtz's theorem can be applied, which can be reformulated as follows: *For increasing values of the system size ($N \to \infty$) the sequence of the individual models* converges almost surely *[14] to the occupancy vector $\overline{m}$, assuming that functions in $Q(\overline{m}(t))$ are Lipschitz-continuous and for increasing values of the system size, the initial occupancy vectors converge to $\overline{m}(0)$.* The above statement can be formally rewritten as in [15]:

Theorem 2.2.1 (Mean-field convergence theorem)**.** *The normalized occupancy vector $\overline{m}(t)$ at time $t < \infty$ tends to be deterministic in distribution and satisfies the following differential equations when N tends to infinity:*

$$\frac{d\overline{m}(t)}{dt} = \overline{m}(t) \cdot Q(\overline{m}(t)), \text{ given } \overline{m}(0). \tag{2.1}$$

□

The ODE (2.1) is called the *limit ODE*. It provides the results for the population of size $N \to \infty$, which is often an unrealistic assumption for real-life systems. When the number of objects in the population is finite but sufficiently large, the limit ODE provides an accurate approximation and the mean-field method can be successfully applied.

The transient analysis of the overall system behaviour can be performed using the above system of differential equations (2.1), i.e., the fraction of objects in each state of $\mathcal{M}$ at every time t is calculated, starting from some given initial occupancy vector $\overline{m}(0)$, as illustrated in the following example.

Example 2.2.2. *In the following we apply the mean-field method to the virus spread model, as given in Example 2.1.3. We explain how to obtain the ODEs, which describe the behaviour of the system and compute performance measures.*

As was discussed in the example, all transition rates of a single computer model are considered to be constant, except for k_1^*. This rate depends on how often a computer that is not infected yet is attacked. In this example we assume that the virus is "smart enough" to attack not infected computers only. The infection rate then can be seen as the number of attacks performed by all active infected computers, distributed evenly over all not-infected computers:

$$k_1^*(\overline{m}(t)) = k_1 \cdot \frac{m_3(t)}{m_1(t)}, \tag{2.2}$$

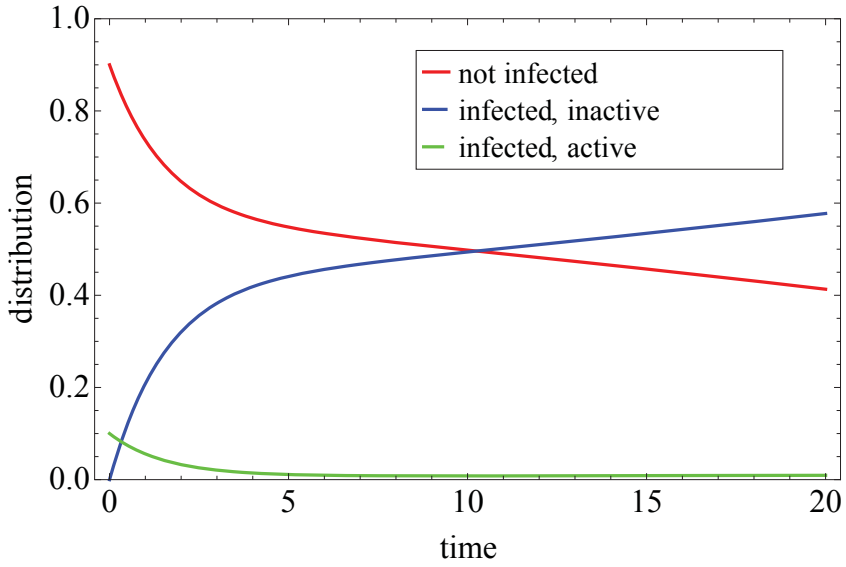


Figure 2.2: Distribution of the computers over the states of the model. Red, blue, and green lines show the number of not infected, infected inactive and infected active computers respectively.

where $\overline{m}(t) = (m_1(t), m_2(t), m_3(t))$ *represents the fraction of computers in each state of the local model* $\mathcal{M}^l$ *at time* t*, and* k_1 *is the attack rate of a single active infected computer.*

The transition rates are collected in the generator matrix:

$$Q(\overline{m}(t)) = \begin{pmatrix} -k_1^*(\overline{m}(t)) & k_1^*(\overline{m}(t)) & 0 \\ k_2 & -(k_2+k_3) & k_3 \\ k_5 & k_4 & -(k_4+k_5) \end{pmatrix}. \tag{2.3}$$

Then Theorem 2.2.1 is used to derive the system of ODEs (2.1):

$$\begin{cases} \dot{m}_1(t) &= -k_1 \cdot m_3(t) + k_2 \cdot m_2(t) + k_5 \cdot m_3(t), \\ \dot{m}_2(t) &= (k_1 + k_4) \cdot m_3(t) - (k_2 + k_3) \cdot m_2(t), \\ \dot{m}_3(t) &= k_3 \cdot m_2(t) - (k_4 + k_5) \cdot m_3(t). \end{cases} \tag{2.4}$$

To obtain the distribution of objects over the states of the model at a given time, the above ODEs have to be solved. Before the computation the parameters have to be assigned to the model. Note that while in the following we assume that a set of meaningful parameters is available, this is not necessarily always

the case. In case parameters are not readily available they can be obtained by a number of methods, we refer to Part II of this thesis for more details. In this example we assume the following parameters:

$$k_1 = 2.5, k_2 = 0.02, k_3 = 0.01, k_4 = 0.3, k_5 = 0.3.$$

Moreover, the initial conditions have to be fixed: $\overline{m}(0) = (0.9, 0, 0.1)$.

Figure 2.2 depicts the fraction of not infected, infected and inactive, and infected and active computers in the system over time. As one can see, in this example the virus managed to infect more than half of the population even though the fraction of actively infecting computers remains very low.

Note that for the sake of simplicity in this example the total number of machines in the population N was not assigned, as we directly moved to the normalized model. In practice, however, the normalization step has to be taken, as discussed in Section 2.1.

○

2.3 Beyond Kurtz's theorem

In the following we discuss a couple of topics, which lie beyond the discussed above convergence theorem. We first explain how the behaviour of individual objects within the overall population can be modelled. Secondly, the possible relaxation on the assumption made when this theorem is formulated are discussed. Then the behaviour in the stationary regime is briefly recalled.

Local model. The rates of the model for an individual object within the population may depend on the overall system state (see, e.g., Equation (2.2)), which means that the local model is a *Time-Inhomogeneous Continuous Time Markov Chain* (ICTMC). To formally describe the behaviour of a single individual in the population the *asymptotic decoupling* of the system is used, and the result is often referred to as *Fast Simulation* [30, 43]. The main idea of this method lies in the fact that every single object (or group of objects) behaves independently from other objects, and can only sense the *mean* of the system behaviour, which is described by $\overline{m}(t)$. The model of one object within the population is called *"local mean-field model"* in the following and is defined as:

Definition 2.3.1 (Local model)**.** *A local model* $\mathcal{M}^l$ *describing the behaviour of one object is defined as a tuple* $(S^l, \mathbf{Q}, L)$ *that consists of a finite set of K*

local states $S^l = \{s_1, s_2, ..., s_K\}$*; an infinitesimal generator matrix* $\mathbf{Q} : (S^l \times S^l) \to \mathbb{R}$*; and the labelling function* $L : S^l \to 2^{LAP}$ *that assigns local atomic propositions from a fixed finite set of Local Atomic Properties (LAP) to each state.*

□

Relaxing assumptions. For models considered in practice the assumption of density dependence may be too restrictive [30]. Furthermore, also the assumption of (global) Lipschitz continuity of transition rates can be unrealistic [16]. Therefore, these assumptions can be relaxed and a more general version of the mean-field approximation theorem, having less strict requirements and which is applied to *prefixes* of trajectories rather than to full model trajectories, can be obtained. We will not focus on this reformulation of the convergence theorem here, instead we refer to [20].

Moreover, the mean-field approach has recently been expanded to a class of models with both Markovian and deterministically-timed transitions, as introduced for *generalized semi-Markov processes* in [50]; and generally-distributed timed transitions for *population generalized semi-Markov processes* [52]. In addition, the extension towards hybrid Markov population models has recently been made in [92] and [91].

Stationary behaviour. The convergence theorem does not explicitly cover the asymptotic behaviour, i.e., the limit for $t \to \infty$. However, when certain assumptions hold, the mean-field equations allow to perform various studies including steady-state analysis. In the following we briefly recall how to assess the steady-state behaviour of mean-field models as in [71].

The stochastic process $(\mathbf{M}^{(N)})$, which was approximated by the mean-field model, has to be studied in order to find out whether the stationary distribution exists. It has been shown that, if the stochastic process is reversible, the fixed point approximation addressing the limiting behaviour of the overall mean-field model is indeed valid. Fixed-point is an approximation of the stationary behaviour of the stochastic process by the stationary points of the mean-field (fluid) limit [71]. The reversibility of the stochastic process implies that any limit point of its stationary distribution is concentrated on stationary points of the mean-field limit. If the mean-field limit has a unique stationary point, it is an approximation of the stationary distribution of the stochastic process. The stationary distribution $\widetilde{m} = \lim_{t\to\infty} \overline{m}(t)$, if it exists, then is the solution

of:

$$\widetilde{m} \cdot Q(\widetilde{m}) = 0. \tag{2.5}$$

For some models the above equation can not be applied straight-forwardly and more advanced methods are required in order to approximate the stationary distribution or its bounds. This, however, lies out of the scope of this thesis; for more details we refer to [11].

Error bounds. As a further remark we want to point out that Theorem 2.2.1 allows to establish that, in the limit of the population size, the error of the deterministic approximation goes to zero. However, we are not able to *quantify* the error committed considering an intermediate system size. Details on worst-case bounds on this error can be found, e.g. in [49], [17].

3

Botnet case-study

To illustrate the usability of the mean-field method, we present a model of a peer-to-peer botnet. Using the obtained model we perform a model-based evaluation of the botnet, similar to [65]. We compare the results, obtained by a mean-field analysis to earlier results obtained by [99] for the same peer-to-peer botnet.

While in this chapter we are not directly interested in obtaining new insights on botnet behaviour, our goal is to show how a quick method of analysis can be used to obtain different measures of interest that cannot be readily obtained using simulation. The comparison shows that the mean-field method is much faster than simulation, therefore, it allows to quickly address more complicated and resource consuming questions, such as how the botnet spreads in different environments. We show that we can obtain deeper insight into the botnet behaviour, by taking into account the costs for running anti-malware software and costs that occur due to computers being infected. Furthermore, we discuss the differences between the mean-field method and simulation and their respective suitability in different settings.

The chapter is further organized as follows. In Section 3.1 we give a short description of peer-to-peer botnets. In Section 3.2 the simulation settings are discussed. The mean-field model of a peer-to-peer botnet is built in Section 3.3, and the results are provided in Section 3.4. Section 3.5 provides examples of more advanced studies, that can be performed using mean-field analysis. Finally, Section 3.6 concludes this chapter.

3.1 Peer-to-peer botnet

In the following we give a short definition of the peer-to-peer botnet, based on [47]:

- *A peer-to-peer network* is a network in which any node in the network can act as both a client and a server. In a peer-to-peer architecture, there is no centralized point for *command-and-control* (C&C).

- *A bot* is a program that performs user centric tasks automatically without any interaction from a user.

- *Botnet*, also known as *zombie army* and *Web robots*, is the generic name given to any collection of compromised PCs controlled by an attacker remotely [41] or a network of malicious bots, that illegally control computing resources.

Nodes in a peer-to-peer network act as both clients and servers such that there is no centralized coordination point that can be incapacitated, which make the botnet less vulnerable to the detection of a single bots. If nodes in the network are taken off-line, the network continues to operate under the control of the attacker. Different malicious botnets have been formed in the past, some of these used existing peer-to-peer protocols for spreading (e.g., Peacomm, Phatbot) while others have developed custom protocols (e.g., SpamThru, Sini).

A peer-to-peer botnet can be seen as a very large population (possibly all computers in the Internet) of interacting components (peers), where infected nodes infect more and more other computers. Due to the large number of (potentially) active components, the analysis of the spreading of such large-scale systems is time consuming and computationally expensive. In this chapter we use mean-field approximation for the fast and accurate analysis of a generic peer-to-peer botnet.

3.2 SAN model

A Stochastic Activity Network (SAN) [87] model has been introduced for peer-to-peer botnets in [99]. It models how the infection spreads through an infinite population of computers. As illustrated in Figure 3.1 the model closely reflects the states a computer goes through after the initial infection has taken place. The original SAN model consists of:

- one place for each phase of infection a system can be in, that can each hold an unbounded number of tokens, representing the number of computers per phase;

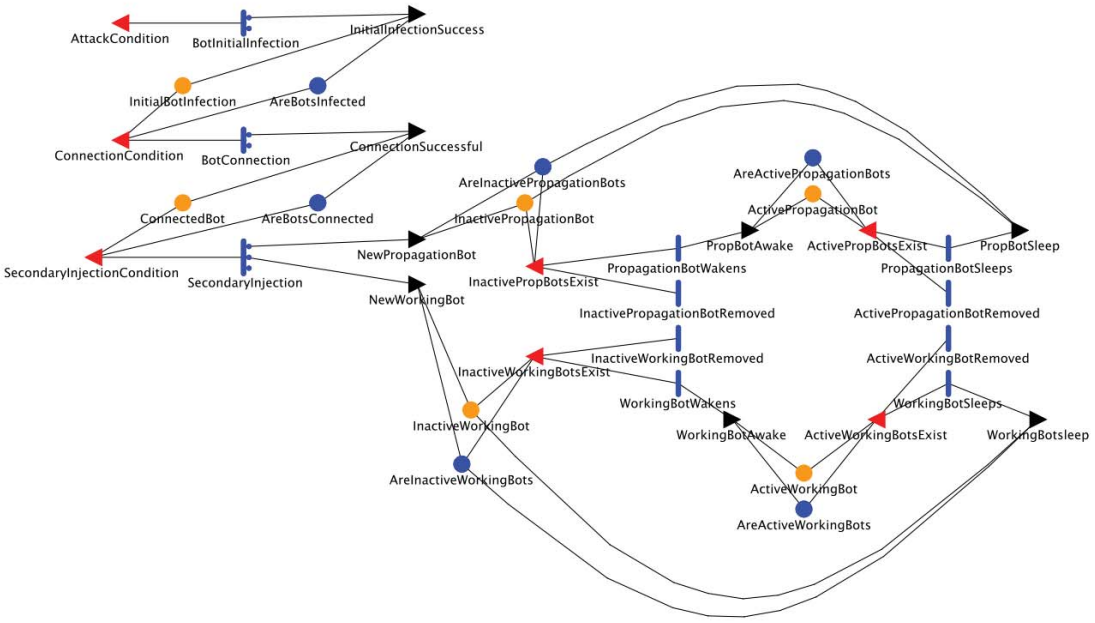


Figure 3.1: SAN model of the botnet as presented in [99] .

- transitions, which move tokens from place to place, as the infection spreads, modelling change in the number of computers in each place.

The SAN model represents the entire population of infected computers, hence, the number of computers in each state (phase) can be directly derived from the model. However, as the population of computers can be very large or even infinite, it is only possible to derive measures of interest from the SAN model using simulation. The main focus in [99] has been the computation of the mean numbers of computers in each of places, which has been obtained by simulating the system 100 times. This is very time consuming and computationally expensive. Therefore, we propose to apply the mean-field method to model the botnet spread in order to obtain faster results and an extended knowledge of the system behaviour.

3.3 Mean-field model of the botnet behaviour

In the following we explain how to build a mean-field model of botnet propagation. We first develop an individual model, which reflects the behaviour of a single computer. This model is based on the SAN model from [99]. The states

of the mean-field model mirror the states of the SAN model with one exception. To be able to represent the whole population we add a state, which corresponds to a non-infected computer. The local model is depicted in Figure 3.2 and the corresponding transition rates can be found in Table 3.1. In the following we provide more information on the botnet spread as modelled in this chapter.

A computer which is not infected yet (state 1) enters the *InitialInfection* state (state 2) with rate k_1^* and becomes initially infected. Then, it connects to the other bots in the botnet, downloads the next part of the malware and possibly moves to state *ConnectedBot* (state 3) with rate k_2. If the computer for any reason is not able to download the malware it returns to the state *NotInfected* with rate k_3.

After downloading the malware, the computer joins the botnet as either *InactiveWorkingBot* (state 4) or as *InactivePropagationBot* (state 6) with rates k_4 and k_5, respectively. If downloading the malware is not possible, for example, because the connection has failed, the computer moves back to the *NotInfected* state with rate k_6. Once the bot becomes either an *InactiveWorkingBot* or an *InactivePropagationBot* it never switches between *Working* or *Propagation*. *Propagation* bots spread infections, that is they try to infect as many new computers as possible. *Working* bots, on the other hand, do not spread infection, but work on harming the target, e.g., sending spam or performing denial-of-service attacks.

In order not to be detected, the bot is inactive most of the time and only becomes active for a very short period of time. Transitions from *InactivePropagationBot* (state 7) to *ActivePropagationBot* (state 5) and back occur with rates k_7 and k_8, respectively. The transition rates for moving from *InactiveWorkingBot* to *ActiveWorkingBot* and back are denoted k_9 and k_{10}, correspondingly.

The computer can recover from its infection, e.g., if an anti-malware software discovers the virus, or if the computer is physically disconnected from the network. It then leaves the *InactivePropagationBot* state or the *ActivePropagationBot* state and moves to the *NotInfected* state with rates k_{13}, k_{14}, correspondingly. The same holds for the *working bots*; the transition rates of *InactiveWorkingBot* or *ActiveWorkingBot* recovery are k_{11}, k_{12}, respectively.

The transition rates for the local model are constant, with the only exception of k_1^*, which depends on the number of active propagation bots in the environment, as more computers are actively spreading the virus the more often an infection occurs. We provide more information on the infection rate

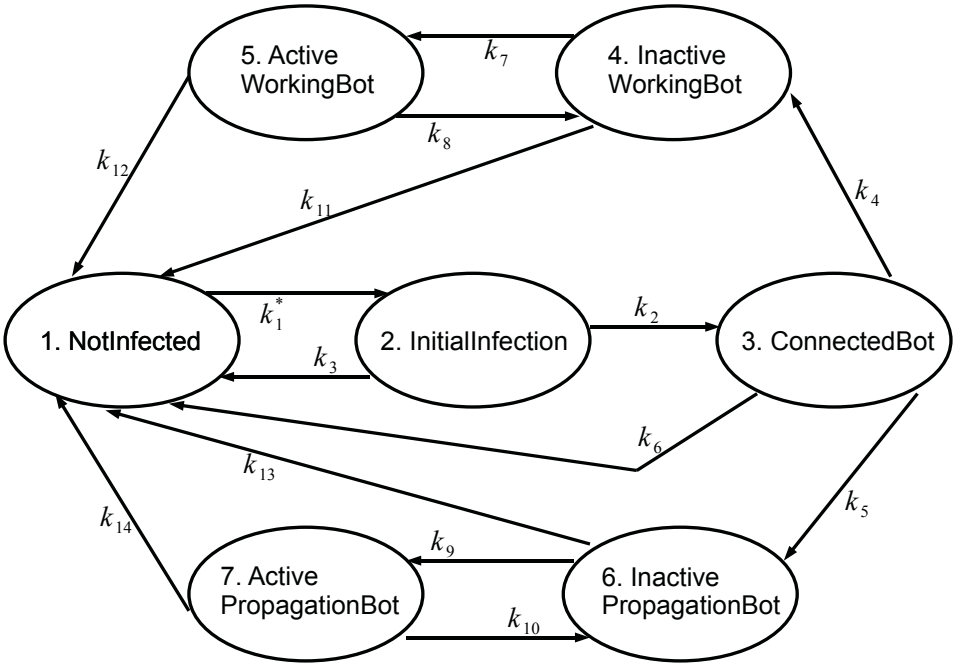


Figure 3.2: A local model $\mathcal{M}^l$ of an individual computer.

when the global mean-field model is built and the information on the whole population is available.

The obtained local model consists of seven states ($S^l = \{s_1, \ldots, s_7\}$), where each state represents a certain phase of the infection of a single computer. The states are labeled as follows:
$L(s_1) = \mathit{NotInfected}$, $L(s_2) = \mathit{InitialInfection}$, $L(s_3) = \mathit{ConnectedBot}$, $L(s_4) = \mathit{InactiveWorkingBot}$, $L(s_5) = \mathit{ActiveWorkingBot}$,
$L(s_6) = \mathit{InactivePropagationBot}$, $L(s_7) = \mathit{ActivePropagationBot}$. The rate matrix R of the local model $\mathcal{M}^l$ is as follows:

$$R = \begin{pmatrix} 0 & k_1^* & 0 & 0 & 0 & 0 & 0 \\ k_3 & 0 & k_2 & 0 & 0 & 0 & 0 \\ k_6 & 0 & 0 & k_4 & 0 & k_5 & 0 \\ k_{11} & 0 & 0 & 0 & k_7 & 0 & 0 \\ k_{12} & 0 & 0 & k_8 & 0 & 0 & 0 \\ k_{13} & 0 & 0 & 0 & 0 & 0 & k_9 \\ k_{14} & 0 & 0 & 0 & 0 & k_{10} & 0 \end{pmatrix} \tag{3.1}$$

The generator matrix Q of size $|S^l| \times |S^l|$ can be computed from the rate matrix

k_1	RateOfAttack·ProbInstallInitialInfection
k_1^*	Rate depends on k_1 and the environment
k_2	RateConnectBotToPeers·ProbConnectToPeers
k_3	RateConnectBotToPeers·(1-ProbConnectToPeers)
k_4	RateSecondaryInjection·ProbSecondaryInjectionSuccess·(1-ProbPropagationBot)
k_5	RateSecondaryInjection·ProbSecondaryInjectionSuccess·ProbPropagationBot
k_6	RateSecondaryInjection·(1-ProbSecondaryInjectionSuccess)
k_7	RateWorkingBotWakens
k_8	RateWorkingBotSleeps
k_9	RatePropagationBotWakens
k_{10}	RatePropagationBotSleeps
k_{11}	RateInactiveWorkingBotRemoved
k_{12}	RateActiveWorkingBotRemoved
k_{13}	RateInactivePropagationBotRemoved
k_{14}	RateActivePropagationBotRemoved

Table 3.1: Transition rates for the model of a single computer.

as follows: for states $s_1 \neq s_2$ Q_{s_1,s_2} is equal to the transition rate R_{s_1,s_2}, i.e. the probability to move from state s_1 to state s_2. For $s_1 = s_2$ Q_{s_1,s_1} is equal to the negative sum of all the rates in row s_1.

Recall that together the state space S^l, the generator matrix Q and the labelling function define the local mean-field model according to Definition 2.3.1. The transition rates are fully available (including k_1^*) when the global model is built.

Once the model of a single computer is built, the overall mean-field model $\mathcal{M}^{\mathcal{O}}$ can be constructed, as described in Section 2.1. The structure of the model remains unchanged (see Figure 3.2), however, the state of the overall model $\overline{m} = (m_1, m_2, \ldots, m_7)$ represents the fraction of computers in each state of the local model, where m_1 corresponds to the fraction of *NotInfected* computers, etc.

Given the overall model definition, the time or population-dependent transition rate can be chosen as in Example 2.2.2, where the botnet is "intelligent enough" to target only not infected computers uniformly[1]:

$$k_1^*(\overline{m}(t)) = k_1 \cdot \frac{m_7(t)}{m_1(t)}.$$

The system of ODEs, describing the transient behaviour of the global mean-field model $\mathcal{M}^{\mathcal{O}}$ can be obtained based on Theorem 2.2.1 as follows:

$$\begin{cases} \dot{m}_1(t) &= k_3 m_2(t) + k_6 m_3(t) + k_{11} m_4(t) \\ & \quad + k_{12} m_5(t) + k_{13} m_6(t) + (k_{14} - k_1) m_7(t), \\ \dot{m}_2(t) &= -(k_2 + k_3) m_2(t) + k_1 m_7(t), \\ \dot{m}_3(t) &= k_2 m_2(t) - (k_4 + k_5 + k_6) m_3(t), \\ \dot{m}_4(t) &= k_4 m_3(t) - (k_7 + k_{11}) m_4(t) + k_8 m_5(t), \\ \dot{m}_5(t) &= k_7 m_4(t) - (k_8 + k_{12}) m_5(t), \\ \dot{m}_6(t) &= k_5 m_3(t) - (k_9 + k_{13}) m_6(t) + k_{10} m_7(t), \\ \dot{m}_7(t) &= k_9 m_6(t) - (k_{10} + k_{14}) m_7(t). \end{cases} \tag{3.2}$$

The initial conditions $\overline{m}(0)$ have to be chosen before the calculation can be started. We fix initial conditions later in this chapter.

[1]Note that the above modelling decision was made to match the existing SAN model and may not completely reflect realistic botnet spreading.

3.4 Mean-field versus simulation

In this section we discuss the results that have been obtained for this model using the mean-field method in detail and compare them to the simulation results we obtained by reproducing the SAN model given in [99]. We carried out a similar series of experiments as in [99]; the chosen parameters for all these experiments are given in Table 3.2.

As was mentioned before, the goal of this chapter is not to study the growth of botnets under different conditions, but to compare the results obtained from mean-field approximation with those obtained from simulations. Hence, we compare results for a representative selection of experiments in order to discuss the advantages and disadvantages of both approaches.

3.4.1 Simulation set-up

The model was simulated using the Möbius tool [31]. The initial conditions for each experiment are as follows: 200 computers are located in the place *ActivePropagationBots* in the SAN, and all the other places are empty. The transition rates can be found in Table 3.2.

Note that the simulation results shown here differ from those in [99]. In consultation with the authors of [99] we found a small mistake in the simulator settings they used: because the rates in the SAN model are marking dependent, a flag has to be set in the Möbius tool to ensure that the rates are updated frequently. Not setting this flag can result in inaccurate numbers of propagation bots, as illustrated in Figure 3.3. The blue dashed line corresponds to the mean number of propagation bots obtained from the unflagged simulation for the baseline experiment (see Table 3.2). When the flag is not set the number of computers in each place is not updated, which results in the overestimation of the number of infected computers. Once the flag is set correctly the results of the Möbius simulation match the mean-field results, as will be shown later.

We performed a number of experiments (see Table 3.2) in order to compare the simulation of the SAN model with mean-field results. Each experiment covered one week of simulated time and was replicated 1000 times. The mean values and 95% confidence intervals of the measures of interest have been obtained.

Parameter	Experiments						
	baseline	1	2	3	4	5	6
ProbInstallInitialInfection	0.1	**0.06**	**0.04**	0.1	0.1	0.1	0.1
ProbConnectToPeers	1	1	1	1	1	1	1
ProbSecondaryInjectionSuccess	1	1	1	1	1	1	1
ProbPropagationBot	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RateOfAttack	10.0	10.0	10.0	10.0	10.0	10.0	10.0
RateConnectBotToPeers	12.0	12.0	12.0	12.0	12.0	12.0	12.0
RateSecondaryInjection	14.0	14.0	14.0	14.0	14.0	14.0	14.0
RateWorkingBotWakens	0.001	0.001	0.001	0.001	0.001	0.001	0.001
RateWorkingBotSleeps	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RatePropagationBotWakens	0.001	0.001	0.001	0.001	0.001	0.001	0.001
RatePropagationBotSleeps	0.1	0.1	0.1	0.1	0.1	0.1	0.1
RateInactiveWorkingBotRemoved	0.0001	0.0001	0.0001	0.001	0.001	0.001	0.001
RateActiveWorkingBotRemoved	0.01	0.01	0.01	0.01	0.01	0.01	0.01
RateInactivePropagationBotRemoved	0.0001	0.0001	0.0001	0.001	0.001	0.001	0.001
RateActivePropagationBotRemoved	0.01	0.01	0.01	**0.07**	**0.04**	**0.02**	**0.015**

Table 3.2: The setups for the different experiments. Bold font indicates difference w.r.t. baseline experiment.

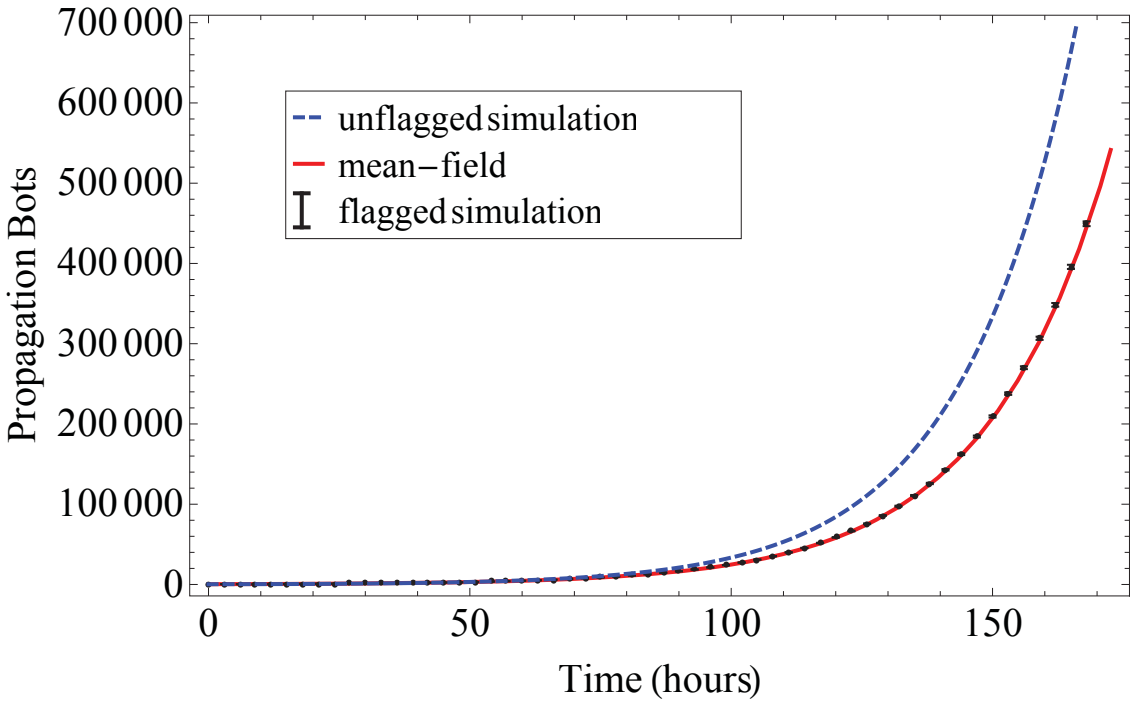


Figure 3.3: Number of propagation bots over time: blue solid line represents the mean value obtained from the Möbius simulation before the rates-updating flag was set; black bars correspond to the 95% confidence intervals obtained from the Möbius simulation with the flag set; red solid line shows the results obtained from mean-field approximation.

3.4.2 Mean-field setup

We use Wolfram Mathematica [103] to obtain solutions for the set of differential equations (3.2) coupled with the transition rates from Table 3.2. To obtain the same initial conditions for the mean-field model as for the SAN model we need to take the *NotInfected* state in the local model into account. As in the SAN model the population size is not bounded, the number of not infected computers has to be set to *infinity*, however, in our model we limit the size of the population, which is closer to the reality, as in practice the total number of computers is finite. Therefore, we set the size of the population for mean-field model large, but finite. Given an overall population of $N = 10^{10}$, the fraction of computers in the state *NotInfected* is initialized as $m_1(0) = (N - 200)/N$, the fraction of computers in the state *ActivePropagationBot* is initialized as

$m_7(0) = 200/N$, and the fractions of computers in all other states are initialized as zero.

3.4.3 Number of propagation bots (baseline)

Figure 3.3 shows the number of the propagation bots in a botnet. The number of propagation bots (both active and inactive) has been taken as measure of interest since they actively infect "healthy'' computers.

The red solid line depicts the mean-field results of the baseline experiment together with the 95% confidence intervals of the "flagged" Möbius simulation. As can be seen, the mean-field results are very accurate in this case, since they lie mostly within the confidence intervals, even though the confidence intervals are very narrow.

The blue dashed line represents the mean value of the original Möbius simulation from [99]. Comparing the original Möbius results with the new results from the correct simulator setting reveals that the number of propagation bots (both *active* and *inactive*) differs from the results stated in [99]. During the first fifty hours the unflagged simulation provides slightly lower results (about 20%), however, on the scale of the figure this difference is hardly noticeable. Starting from fifty hours, the unflagged simulation over-estimates; after a week (168 hours) the difference is about 42% (754755 versus 440073). Note that the simulation takes longer than 5 days of runtime, as opposed to 1 second for the mean-field method. We come back to this at the end of the section.

3.4.4 User factor (Experiments 1-2)

To investigate the influence of users on the growth of botnets, Experiments 1 and 2 have been done in [99]. The *ProbInstalInfection* is reduced to 60% and 40%, respectively, as compared to the baseline experiment, to represent a lower probability of, e.g., opening infected files. The results are presented in Figure 3.4, together with those from the baseline experiment. A logarithmic scale has been chosen for the number of propagation bots, in order to better visualize the exponential growth. For both experiments, the results obtained with the mean-field model are again very accurate and lie well within the confidence intervals most of the time.

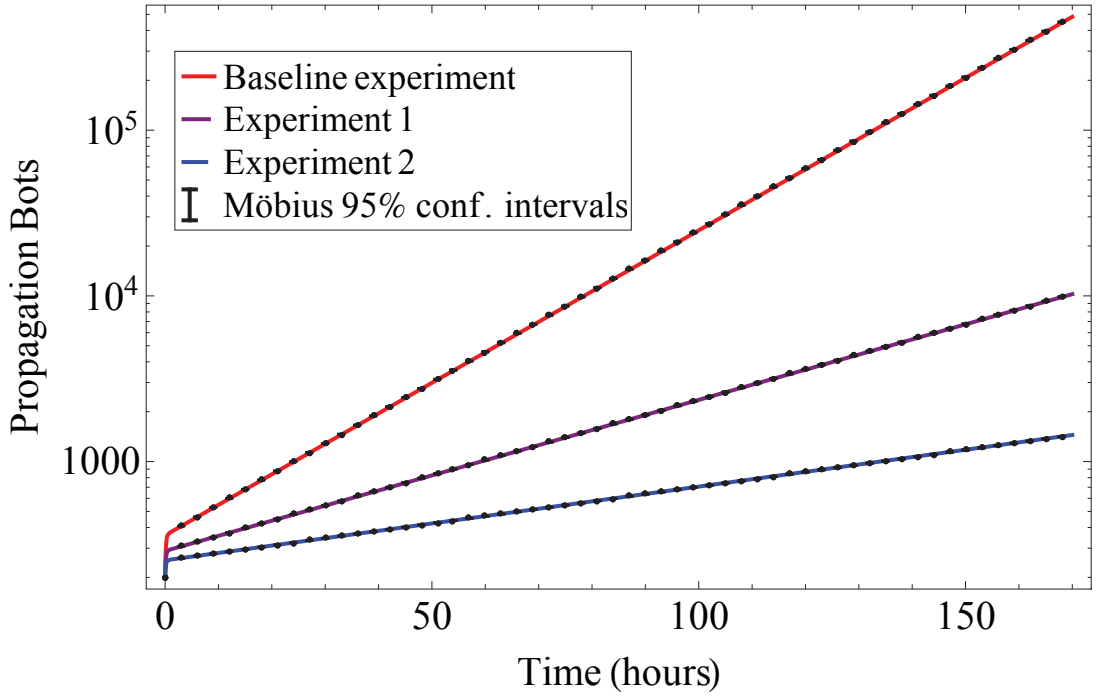


Figure 3.4: Number of propagation bots over time in the baseline experiment and in Experiment 1 and 2 obtained from Möbius simulation and from mean-field analysis.

3.4.5 Removal rate (Experiments 3-6)

To investigate how efficiently anti-malware software can control or even stop the botnet spread, experiments with increased removal rates have been done in [99]. To compare both of the approaches, we conducted a series of experiments, where the removal rate of active propagation bots varies between 0.01 and 0.1. The mean-field approximation provides an explicit result, which in most of the cases lies well within the 95% confidence intervals (see Figure 3.5).

3.4.6 Observation about the method

At first sight the high accuracy of the analytical results might be surprising, since the underlying assumption of mean-field approximation is that the number of interacting components is large. However, apparently in Experiment 3 (cf. Figure 3.5) the initial set of active propagation bots hardly gets a chance to infect more computers before being disinfected themselves. In terms of the

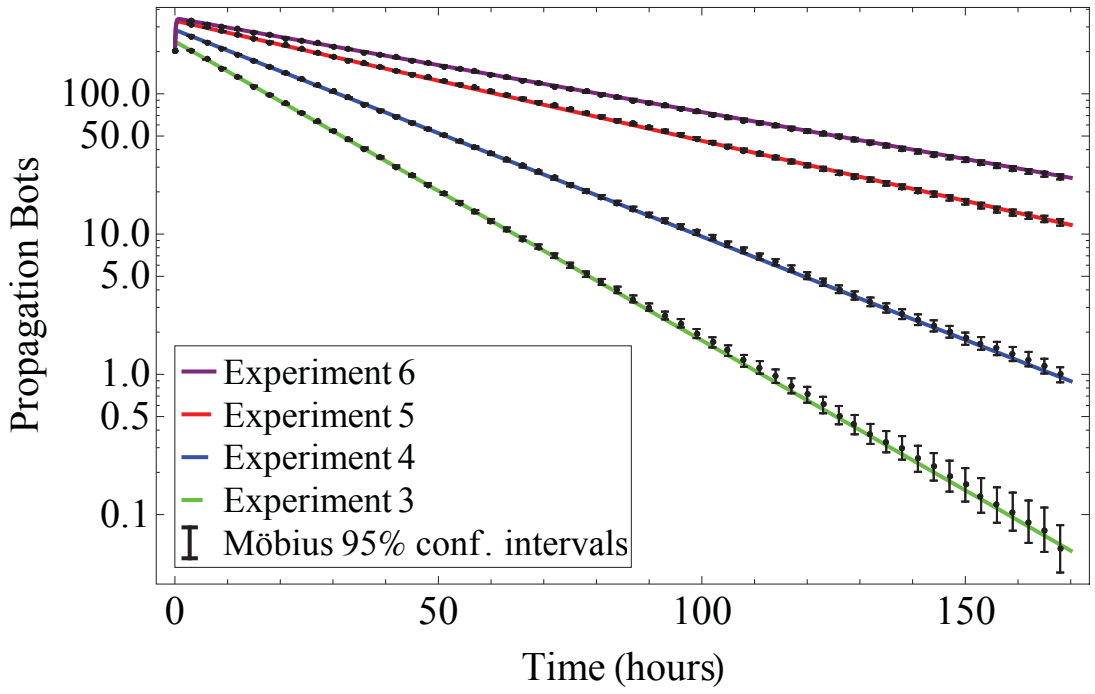


Figure 3.5: Number of propagation bots over time in Experiments 3, 4, 5, and 6, obtained from Möbius simulation and from mean-field analysis.

local model, it means that the transition from the state *NotInfected* to the state *InitialInfection* is taken by (almost) none of the computers. This transition happens to be the only one whose rate depends on the environment; if we remove it from the local model, we obtain a CTMC with constant rates. The ODEs (3.2) of the global model that result from such population-independent local model are easily seen to be the Kolmogorov differential equations, whose solution is the probability distribution over the states of the CTMC as a function of time.

Also, removing this transition in the SAN simulation model reduces it to a set of many *independent* CTMCs. Taking the number of markings (number of objects) per state as a function of time, and dividing by the total, obviously results in an unbiased estimate of the probability distribution of the local model in the course of time. Thus, clearly the two approaches should match, as they in fact do and this explains why the mean-field results are still accurate, even though in this case the overall number of components is small.

Experiment	Simulation	Mean-field
Baseline	5 d 3 h 25 min	1 sec
Exp. 1	9 h 51 min	1 sec
Exp. 2	5 h 37 min	1 sec
Exp. 3	31 min	1 sec
Exp. 4	40 min	1 sec
Exp. 5	45 min	1 sec
Exp. 6	36 min	1 sec

Table 3.3: Time spent on simulation and mean-field approximation.

It is interesting that the confidence intervals in Experiment 6 are much narrower than the ones in Experiment 3. As the average number of propagation bots decreases over time, the confidence intervals seem to get wider on the logarithmic scale (see Figure 3.5). In fact, however, the absolute width of the intervals gets smaller, but less quickly than the estimate itself. The reason for this is that the actual number of propagation bots always is a non-negative integer; therefore, when the estimated *average* decreases far below 1, it must be the average of many 0's and a few 1's (or even fewer higher integers). Such an estimate inherently has a large coefficient of variation; in fact, this is the main problem of *rare-event simulation*, cf. [53].

Another thing to remark about these experiments, is that when the number of propagation bots reaches 0, and there are also no bots in the states *InitialInfection* and *ConnectedBot* anymore, no new infections can occur. The number of propagation bots will then remain 0. Thus, when the graph indicates that after a week the *average* number of propagation bots is about 0.01, this means that in most (about 99%) of the simulation runs the botnet is extinct and will stay so, while only a few runs still have some botnet activity.

3.4.7 Run time

In Table 3.3, the computer run times for the simulation and for the mean-field computation are compared. The results have been obtained on a Core-i7 processor with 3 GB RAM and 4 cores and hyper threading. One sees that the simulation can take a very long time, namely up to several days, while the mean-field approximation is always done within one second. The difference between the simulation time for the different experiments is due to

the marking dependency of the rates. For example, in the baseline experiment the number of *ActivePropagationBots* is large, hence, the rate of infection becomes very large and more time is needed to simulate the resulting large number of events. The time spent on the simulation of the experiments with lower numbers of computers involved is reasonably smaller; however the mean-field approximation is still much faster in all cases. In any case, the simulation times should only be taken as indications, since the simulations have not been run completely independently, but in parallel pairs on a 4-core computer, so the jobs may have interfered with each other.

3.5 Exploiting the speed-up

In the previous section we have shown how fast and efficient the mean-field method is (cf. Table 3.3) and that the obtained approximation is quite accurate. This allows to use the-mean field method to address problems which are not feasible using simulation. In this section we discuss two types of such problems:

1. The dependence of botnet spread on two (or more) parameters.
2. The cost constraints involved in the botnet behaviour.

In the following we illustrate how mean-field method can be used to analyse broad variety of properties.

3.5.1 Removal rates of active and inactive bots

The removal (disinfection) mechanisms for inactive and active bots differ, and while active bots are relatively easy to identify, detection of inactive bots is a bigger challenge. However, infected bots are programmed to stay active very shortly and remain inactive most of the time, therefore, increased detection rate of active bots while inactive bots left out of the control, might not be enough to control the speed of the botnet propagation. Hence, the question of how to distribute the effort/money in controlling botnet stays open. We will investigate three possibilities:

- improve the deactivation of the active bots (relatively easy);
- improve the deactivation of the inactive bots (more difficult);

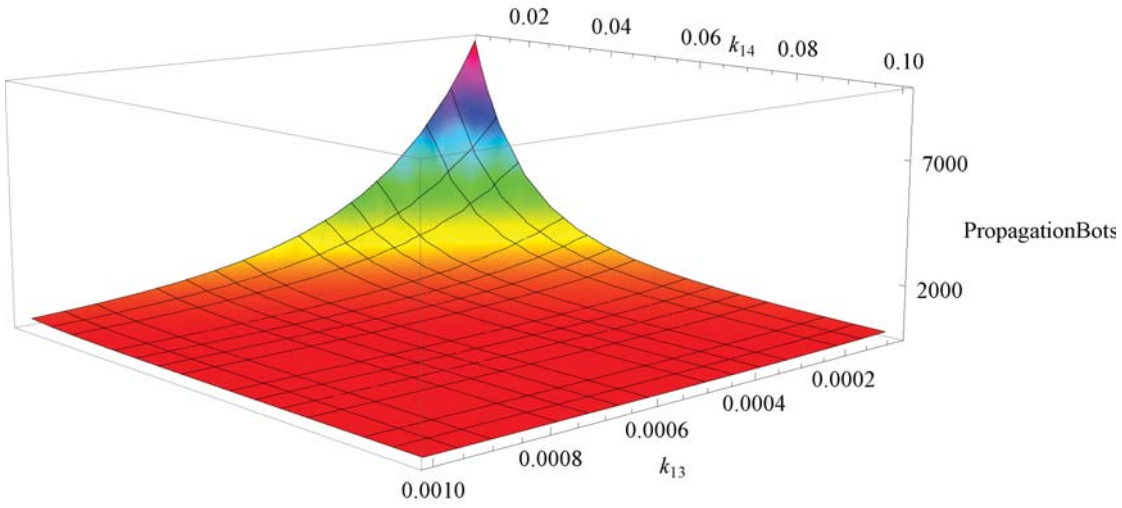


Figure 3.6: Number of propagation bots for $(k_{13}, k_{14}) \in [8 \cdot 10^{-5}; 10^{-3}] \times [8 \cdot 10^{-3}; 10^{-1}]$ at time $T = 3$ days, all other parameters are the same as for the baseline experiment (see Table 3.2).

- find the best combination of the two previous ways.

The authors of [99] used time-consuming simulation to show in a couple of examples that there is no considerable difference in increasing the detection/deactivation of active or inactive bots (namely increasing the removal rates k_{11}, k_{13} or k_{12}, k_{14}). The mean-field method allows to make the analysis faster and to obtain more information. We, therefore, are able to access the dependence of number of propagation bots as defined by Equation (3.2) on two removal rates. It is done by fixing the initial conditions and twelve parameters to the baseline values (see Table 3.2), and let the two remaining parameters change, namely $k_{13} \in [8 \cdot 10^{-5}; 10^{-3}]$ (for inactive propagation bots) and $k_{14} \in [8 \cdot 10^{-3}; 10^{-1}]$ (for active propagation bots). Figure 3.6 depicts the dependence of number of propagation bots in the network in three days after the botnet started to spread on two parameters, namely k_{13} and k_{14}.

As one can see, Figure 3.6 is symmetric with respect to the two chosen parameters, therefore, there is indeed no considerable difference in a system behaviour when either of the two parameters is slightly increased. The peak in the Figure corresponds to the rates in the baseline experiment, and increasing removal rates k_{13} and k_{14} have an identical impact on (decreasing) the number

of propagation bots in the system in three days time.

As was discussed before, inactive computers are much harder to detect (increasing k_{13} is more difficult), therefore the above results might help the developers of anti-virus software to choose a better strategy for botnet removal, namely, the first strategy, which focuses on the easier task of detecting active bots.

The results presented above have been obtained using Wolfram Mathematica, and the execution time equals 2.61 minutes, while only one simulation for a given set of parameters might take between 36 minutes and 5 days (see Table 3.3).

3.5.2 Cost introduced by the botnet

Next, we introduce a cost concept to analyse the economical side of infection spread. Two types of costs are considered:

- the cost of a computer being infected, that occurs for example due to the loss of information or productivity;
- the cost of more frequent checking with antivirus software or updating it.

On the one hand the number of infected computers and, hence, the cost grows if computers are not frequently checked. On the other hand, if computers are checked too often the botnet is not growing, but running the antivirus software becomes very expensive. We analyse this trade-off in more detail in the following. We calculate the cumulative cost as follows:

$$C(t_0, t_1, RR, D_1, D_2) = \int_{t_0}^{t_1} (D_1 \cdot \text{InfBots}(t, RR) + D_2 \cdot RR \cdot N)\, dt, \tag{3.3}$$

where (from the left to the right)

- RR is the change in removal rates $k_{11}, ..., k_{14}$ with respect to the rates in the baseline experiment, i.e. $k_{11} = RR \cdot k_{11,baseline}$ and similarly for k_{12}, k_{13}, k_{14};
- D_1 is the cost of infection;
- $\text{InfBots}(t, RR)$ is the number of infected computers for a given RR, at time t, including active and inactive working and propagation bots;

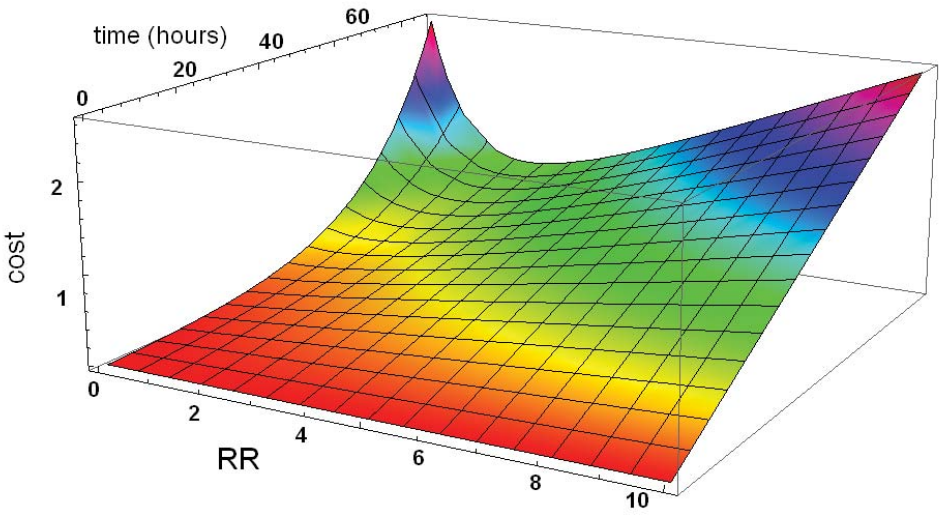


Figure 3.7: Cost of the system performance for $D_1 = 0.01, D_2 = 4 \cdot 10^{-5}$, and $RR \in [0.001; 5]$.

- D_2 is the cost of checking one computer, which probably is much lower than the cost of infection (D_1);
- N is the number of all computers in the system.

Figure 3.7 depicts how the introduced cumulative cost changes over time for varying removal rates, that change with factor $RR \in [0.001; 5]$. The involved costs are chosen as such in order to illustrate that infection is "more expensive" than running/updating antivirus, namely, $D_1 = 0.001$ and $D_2 = 4 \cdot 10^{-5}$.

For low removal rates (RR between 0 and 1) the cost grows exponentially with time due to the fact that the number of infected computers is large as they are not removed from the network fast enough. The loss of productivity (cost) is highest for $RR = 0.001$.

High removal rates (RR between 5 and 10) introduce a linear dependency of the cumulative cost on time and RR since the network is checked "too often".

The "valley" between $RR = 2$ and $RR = 4$ corresponds to the best strategy, when the network is checked frequently enough to prevent the spread of the botnet, but not "too often" to introduce extra cost (loss of productivity).

We see that the mean-field method can be easily used for finding the removal rates which minimize the cost at a given moment of time. It can help network managers with careful decision-making, based on the situation at hand.

Even though not all parameters might be known in reality, such analysis can help to obtain a better understanding of the characteristics of botnet spread. The speed-up that was gained by applying mean-field method is again confirmed by the time needed for the calculation. Although the time spent on integration of the cost function is 20.5 minutes, it is indeed a great improvement compared to the simulation time as in Table 3.3, where the simulation for a fixed set of parameters took at least 36 minutes.

3.6 Concluding remarks

In this chapter we compared different approaches for evaluating a peer-to-peer botnet spreading. We have shown that the mean-field approach is much faster than simulation, taking about 1 second instead of minutes to days of computation time. The results from the mean-field analysis match those from the simulation very well, being mostly inside the 95% confidence interval.

Due to the speed-up of the mean-field method we have been able to address various questions which cannot practically be answered with simulation, such as questions involving cost trade-offs; this is useful in typical engineering applications. One can think of other questions to address, however our aim was to show the potential of the method by addressing problems which can not be solved using simulation.

In general, the mean-field method is only a first-order approximation to the real Markov chain model, which becomes better as the number of entities involved increases. However, in the present case-study we did not observe any significant difference between the mean-field results and the simulation results of the full model. In contrast to the mean-field approximation, the precision of the simulation results suffers when the mean number of bots being estimated gets closer to zero. This is because the standard deviation does not go to zero as fast as the mean value.

The present research shows the usefulness of the mean-field approach, as it is able to provide very accurate results very quickly. However, even in cases where mean-field results are less accurate for small population numbers, it can be useful as a quick check of the simulation. In fact, the simulator setting problem discussed in Section 3.4 was found due to the mismatch with the mean-field results.

Part II

Parameter Fitting

Model-based system evaluation allows to study system behaviour prior to implementation. However, it often suffers from the lack of realistic system parameters. This is particularly so for large scale distributed systems, like applications running in the internet, for which a structured measurement set-up is very difficult to achieve, or even impossible to obtain. This is particularly the case when studying attacks launched via the internet, like computer worms.

This part aims to obtain a better understanding of the spreading phase of a computer worm, and does so by combining a *mean-field* model of worm behaviour with *parameter fitting* techniques, and illustrates this on the case of *Code-Red* worm. We explain how to build a mean-field model of the worm, and how to estimate the corresponding parameters, so as to find the best fit between the available data and the model prediction. We also discuss a number of intricate technical issues, ranging from the additional (preprocessing) work to be done on the measurement data, the interpretation of the data; to, for instance, a restructuring of the model (based on data unavailability), that has to be performed before applying the parameter estimation procedures. The presented model and parametric study is, as far as we know, the most detailed study of the spreading phase of Code-Red.

This part is further organized as follows. In Chapter 4 the motivation of the case-study, background information on parameter estimation, and related work are presented. Chapter 5 provides the set-up of the case-study, including history of the Code-Red, available data, and mean-field model of Code-Red spread. Chapter 6 provides the results of the Code-Red case-study and concluding remarks.

4

Parameter estimation for mean-field models

With this chapter we provide background information on the case-study based on Code-Red worm. We first motivate the case-study in Section 4.1. Then in Section 4.2 the techniques, used for the parameter estimation, are described. The overview of related work is done in Section 4.3. Section 4.4 concludes this chapter.

4.1 Motivation

In Part I of this thesis we performed a (mean-field) model-based evaluation of a peer-to-peer botnet. In doing so, we provided insight in the behaviour of the botnet. Over the last few years, there has been an interest in better understanding the way computer viruses spread, for instance using simulation or simple differential equation models, cf. [88], [99], [108]. Following such a model-based approach, in general, helps to understand systems under varying circumstances, e.g., the transient behaviour of the system, periodicity, the long run system behaviour, how stable the system is under the influence of the environment, etc. Sometimes also effects of countermeasures can be evaluated, by changing the parameters that impact the effectiveness of countermeasures (see, e.g., Sections 3.4 and 3.5).

Despite the fact that model-based evaluation is widely used for real systems, it often suffers from the lack of realistic parameters. The latter is due to the fact that parameter values closely (or absolutely) reflecting the real system behaviour are difficult to obtain. To ensure that the model complies with the system, parameters can be assigned by, for example, one (or combination) of the following methods:

1. *Predefined experimental settings.* The experiment is set in a predefined environment with the predefined parameters, i.e., the system is set up to behave in a certain way. In this case the assigned model parameters exactly reflect the system behaviour. However, the system is evaluated in an experimental, or so-called "synthetic", settings (e.g., in a local network), which might be not 100% realistic.

2. *Analysis of a system.* Obtaining inside information on the system, for example, by analysing the code of the computer virus. In most cases such analysis does not allow assigning values to all parameters of the model of this system, as information is not sufficient. It is possible to see how the system is supposed to behave, however, the influence of the environment can not be seen.

3. *Using measurements.* In this case the real system behaviour is measured and the parameters are approximated based on the obtained data. Various estimation techniques can be used to assign realistic parameter values.

In this part of the thesis we follow Method 3, that is, we discuss how measured data can be used to obtain the parameters of a mean-field model. In general terms, there exist a number of parameter estimation techniques, such as, least squared error [1], maximum likelihood [78], generalized maximum spacing estimates [38], generalized method of moments [48], etc. These methods are widely used in application areas such as biochemical reactions [76], computer vision [106], cosmology [73], etc. We will use two well-known parameter estimation methods, namely, *least squared error* [1] and *maximum likelihood* [78] for this case-study. The above two methods are chosen because they are commonly used in the fields, where models, similar to the mean-field model are used, for example, in system biology or chemistry.

Unfortunately, many real systems can not be properly measured, which limits the possibilities to match the model output to the real system behaviour. Moreover, even if measurement data is available in some form, it is often difficult to obtain meaningful parameters, as the data is often only partially available, or the performed measurements do not match the needs of the model. This is exactly the challenge we encountered when trying to parametrise our mean-field models, and about which we report in this part of the thesis.

We will use the *mean-field approach* to model the spreading phase of a computer worm, in particular, the Code-Red worm [77], and measurement data in combination with the two aforementioned parameter fitting techniques to assign the parameters of the obtained mean-field model.

The aim and contribution of the current case-study is threefold. First of all, the study shows how a single behavioural model of a virus can be used as basis for modelling virus spread, using the mean-field approach. Secondly, the study illustrates that such a model can be parametrized well, based on measurements performed during the outbreak, using standard parameter estimation techniques, provided the measurements are very carefully dealt with. Third, and maybe most importantly, this part of the thesis discusses the challenges encountered when performing such a detailed modelling study, in which the measurements, when studied in detail, show all sorts of artefacts that are easily overlooked, but do have a substantial impact on the estimation procedure. Hence, the contribution of this case-study not only lies in the final fitted model, but also in the description of the process to come to this model.

4.2 Parameter estimation procedures

In this section two parameter estimators are discussed. Let us first introduce some notation.

The measured data is represented by the trace $\mathcal{O}(t_r)$, where t_r is a time stamp for each element of the trace, i.e., the time when value $\mathcal{O}(t_r)$ was observed; $r \in \{1, \ldots, R\}$, and R is the size of the data trace. For simplicity of notation we assume that there is one trace of data available. In Table 4.1 an example of a data trace is presented, where the size of the data trace $R = 8$, the data was measured once per unit of time, therefore, $t_1 = 1, t_2 = 2 \ldots t_R = 8$, and the observed values are $\mathcal{O}(t_1) = 1, \mathcal{O}(t_2) = 2 \ldots \mathcal{O}(t_R) = 9$ as given in the right hand column.

For validation of the obtained results we introduce the vector of number of hosts in each state $\overline{M}(t)$ instead of the fractions $\overline{m}(t)$. In this case we reverse the normalization step (see Section 2.1) in order to come back to the actual number of hosts per state of the local model, which makes it easier to compare the real data with the output of the model:

$$\overline{M}(t) = N \cdot \overline{m}(t). \tag{4.1}$$

Timestamps	Observation
1	1
2	2
3	6
4	2
5	8
6	1
7	3
8	9

Table 4.1: The example of the trace of a data.

Note, however, that vector $\overline{M}(t)$ does not necessary contain integer numbers, as it is obtained from the mean-field model, where fractions of objects are evaluated. The above notation will be used when addressing the measured data.

The data trace $\mathcal{O}(t_r)$ is compared to the output of the mean-field model $\mathcal{M}^O$, more precisely, to the number of objects in a corresponding state of the local model, $M_i(t_r)$ at the time, the observation was made. The parameters of the mean-field model to be estimated are denoted as $\lambda_1, \ldots, \lambda_P$.

There exist several measures of quality, which describe how good observation data fits to a model. One of the simplest and therefore popular measures is the *squared error* [1]. The squared error measures the squared euclidean distance between the model prediction and the actual data at the respective observation time points, and is defined as follows:

$$\mathcal{E} = \sum_{r=1}^{R} ||\mathcal{O}(t_r) - M_i(t_r)||^2. \tag{4.2}$$

The squared error obtained from Equation (4.2) has one disadvantage, namely, it represents the result in the actual values, which might be inconvenient due to the different orders of magnitude of the observations. For example, a squared error of 35 is quite big when the mean of the observed data is in the order of 10^2, however, the same value is considered very small for data with mean value in order of 10^9. Therefore, it is more useful if the measure of interest would not show the difference between the observation and model output, but how big

this difference compared with the mean value of the observed data. To obtain such measure of interest, which allows us to better understand how close the data is compared to the prediction of the model, is the *relative squared error* ($0 \leq \mathcal{E}_{\text{rel}} \leq 1$), defined as in [102]:

$$\mathcal{E}_{\text{rel}} = \frac{\sum_{r=1}^{R} ||\mathcal{O}(t_r) - M_i(t_r)||^2}{\sum_{r=1}^{R} ||\mathcal{O}(t_r) - \widehat{\mathcal{O}}||^2}, \tag{4.3}$$

where $\widehat{\mathcal{O}}$ is the squared error of the default (simplest) predictor (or just the average of the actual data) [102]. Note that the relative squared error becomes independent of the nature of the data, unlike the squared error in (4.2).

To guarantee the best fit the parameters that minimize the relative squared error have to be found, i.e., we need to find

$$\lambda_1^*, \ldots, \lambda_P^* = \underset{\lambda_1,\ldots,\lambda_P}{\operatorname{argmin}} \mathcal{E}_{\text{rel}}. \tag{4.4}$$

Another way to measure the estimation quality is the so-called *likelihood* [78]. Assume that the observations contain a normally distributed error with mean zero and variance σ^2. Formally, we then have

$$\mathcal{O}(t) = M_i(t) + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, \sigma)$, and therefore $\mathcal{O}(t)$ is distributed as $\mathcal{N}(M_i(t), \sigma)$. Thus, the density of observing $\mathcal{O}(t)$ given a real value $m_i(t)$ is given by

$$f(\mathcal{O}(t) \mid M_i(t)) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\mathcal{O}(t) - M_i(t))^2}{2\sigma^2}}. \tag{4.5}$$

If in addition we assume that all observations are statistically independent, we can define the *likelihood* $\mathcal{L}$ as the density of observing the data given the current model output:

$$\mathcal{L} = \prod_{r=1}^{R} f(\mathcal{O}(t_r) \mid M_i(t_r)). \tag{4.6}$$

To obtain the parameter valuess which will fit the model output best according to the observed data, the likelihood has to be maximized, i.e., we have to find

$$\lambda_1^*, \ldots, \lambda_P^* = \underset{\lambda_1,\ldots,\lambda_P}{\operatorname{argmax}} \mathcal{L}. \tag{4.7}$$

In order to simplify and unify the computation, we define the *negative log-likelihood* $\mathcal{L}^*$ as

$$\mathcal{L}^* = -\frac{\sum_{r=1}^{R} \log f(\mathcal{O}(t_r) \mid \overline{M}(t_r))}{R}. \tag{4.8}$$

The negative log-likelihood is divided by the length of the data trace R in order to make it independent of the length of the data. We minimize it in order to find the best estimates:

$$\lambda_1^*, \ldots, \lambda_P^* = \underset{\lambda_1,\ldots,\lambda_P}{\operatorname{argmin}} \mathcal{L}^*. \tag{4.9}$$

Compared to $\mathcal{E}_{\text{rel}}$, $\mathcal{L}^*$ has an extra factor, namely the variance σ^2. If the variance (or standard deviation) is set to 1 we expect that the optimization of both measures of quality yields the same results, however, the variance can be seen as an extra parameter in the optimization process, which potentially might help to yield better results. In this case we have to find

$$\lambda_1^*, \ldots, \lambda_P^*, \sigma^* = \underset{\lambda_1,\ldots,\lambda_P,\sigma}{\operatorname{argmin}} \mathcal{L}^*. \tag{4.10}$$

We discuss this possibility in Section 6.3.

Given the two measures of quality, the minimization is the last step in finding the model, which fits best. There exist a number of methods which can be used to find the global minimum of a given function [78]:

- *Analytical method.* With this method we find the derivatives of the measure of quality with respect to the parameters and set the resulting gradient to zero. After finding extrema we find which of these is a minimum using the second derivative. This method is rarely used, as the analytical solution is often not available.

- *Grid search method.* With this method one repeatedly tries each possible value of the parameters in a predefined region, and finds the parameters which give the minimum value of the measure of quality. This method is not practical when the number of parameters grows beyond 2 or 3.

- *Numerical method.* The most common methods employ numerical algorithms for finding the minimum, such as available in, e.g., MATLAB's optimization toolbox [75] or Wolfram Mathematica Optimization Tools [105]. These tools use more advanced methods, than mentioned above grid search.

In the following we use numerical methods for finding the minimum squared error and the negative log-likelihood. We exploit the Wolfram Mathematica Optimization function `NMinimize` [104] to minimize the measures of quality. This function always attempts to find a global minimum of a given measure of quality subject to the given constraints, and it can employ one of the following numerical methods: Nelder–Mead method [79]; differential evolution [93]; simulated annealing [60]; or random search [107]. The accuracy and number of iterations can be altered in order to obtain the best solution and the desired precision. We discuss the application of the above methods to the Code-Red case study in Capter 6.

4.3 Related work on parameter estimation

4.3.1 Differential equation model

To the best of our knowledge, parameter estimation techniques have not yet been applied to mean-field models before. However, such techniques have been used for estimating parameters of differential equations. For example, in [101] differential equations are promoted as an important tool in studying chemical reactions. Note that, the assumptions made for obtaining such equations are very similar to these made when building models for large population system using the mean-field approach. The author of [101] proposes to use techniques based on squared error to estimate parameters of the obtained differential equations and presents evidence that this method works well for these equations.

More advanced techniques for obtaining the parameters of differential equations were proposed. For example, [84] proposes the *parameter cascade algorithm* that combines data smoothing and a generalization of profiled estimation [10]. The proposed algorithm is shown to yield results at least as good as other approaches, and rather better than estimation based on squared error. Another advantage of the algorithms is that initial conditions of the differential equations do not need to be set in advance.

4.3.2 Hybrid Markov population models

Authors of [91] have illustrated the way to assess the arrival rates for system of many servers with a time varying load from real data. Although estimating the parameters of the model was not the main focus of the above paper, we briefly discuss it here, as it is closely related to the goals of our case-study.

The mean-field method was used to model the behaviour of a system with a large number of services. A widely analysed dataset of all entries to the World Cup 1998 website [3] shows how the website was accessed during its lifetime. This dataset was used to approximate the arrival rates for the mean-field model, which were set to be a piecewise constant function with values equal to the *average number* of accesses during each hour (as was available in a dataset).

Although the approach used in [91] shows how some of the parameters of a mean-field model can be estimated using measurements, it is different from the methods used in our case-study. It uses the data directly, by obtaining the mean values, while in this thesis we perform parameter estimation by minimizing the difference between the model prediction and the measured data. Note that the approach used in [91] is applicable for a limited number of case-studies and only for some of the parameters, while using parameter estimation techniques all parameters values can be assigned if enough data is available.

4.3.3 Code-Red worm

The data used in our study was collected and studied by The Cooperative Association for Internet Data Analysis (CAIDA); the results were presented in [77] where the background of the worm as well as the details of the data collection were discussed. This paper provides insights in the infection and deactivation processes. Moreover, geographical locations and types of the different infected hosts were studied. The authors conclude that at July 19th between 11:00 and 16:30 UTC the infection grows exponentially (citing the work of Staniford [88]). However, no formal model of the worm is provided.

In the following chapters we will introduce the mean-field model for the Code-Red worm and use parameter estimation techniques to assign meaningful parameters to the obtained model. In the past other researchers have modelled this worm with different methods, however, there are similarities in the assumptions made, which confirms the suitability of the proposed model.

4.3.3.1 Epidemiological model

In the work of Staniford [88], a model is proposed to explain the infection rate of the Code-Red worm. The proposed model is very similar to an epidemiological

model, and is built on the same principles as explained in Chapter 2:

$$da/dt = K \cdot a \cdot (1 - a),$$

where K is the infection rate of one compromised machine, and a is the fraction of compromised machines. This model is explained as follows: each of the a infected machines is able to compromise K machines per unit of time while only $1 - a$ machines are not infected yet. Moreover, the author provides a *manually made* fit to the data, obtained from [37]. The parameter K was obtained manually, and no formal explanation was provided.

4.3.3.2 Two-factor worm model

Another model of Code-Red worm propagation was proposed by Zou *et al.* [108]. The proposed model was based on the classical epidemiological model, which is modified in order to provide a better accuracy. The so-called *two-factor worm* model includes two additional aspects:

- human countermeasures against worm spreading (patching, rebooting, etc.);
- slowing down of the worm infection rate due to the worm's impact on internet traffic and infrastructure.

Adding these factors allowed to explain the slowing down of the worm spread before midnight of July 19, 2001 (we discuss this in more detail in Section 5.4). The two-factor worm model is as follows:

$$\begin{cases} dS(t)/dt &= -\beta(t) \cdot S(t) \cdot I(t) - dQ(t)/dt, \\ dR(t)/dt &= \gamma \cdot I(t), \\ dQ(t)/dt &= \mu \cdot S(t) \cdot J(t), \\ \beta(t) &= \beta_0 \cdot [1 - I(t)/N]^n, \end{cases}$$

where $N = S(t) + I(t) + R(t) + Q(t)$, $I(0) = I_0 \ll N; S(0) = N - I_0$; $R(0) = Q(0) = 0$, and $S(t)$, $I(t)$, $R(t)$, and $Q(t)$ are the number of susceptible, infectious; removed from infected, and from susceptible hosts[1], respectively.

This model was compared against the data collected on July 19, 2001 (again for the number of infected machines only). The outcome of the model did fit the data well for the proposed parameter set. However, no evidence on the source of the chosen parameters was provided.

[1]The rest of the notation can be found in Table 1 of [108].

4.4 Summary

In this chapter we argued that existing model-based evaluation techniques might benefit when combined with parameter-estimation methods. We briefly described two parameter estimation techniques, which will be used in this part of the thesis. We provided related work on parameter estimation for differential equations, and mean-field models. Moreover, we discussed related work on the Code-Red virus, which will be used is this case-study.

5

Code-Red worm model and available data

In this chapter we present a case-study, based on the Code-Red worm. In Section 5.1 we recall the main properties of the worm and its history. A mean-field model which describes the Code-Red attack is presented in Section 5.2. In Section 5.3 we discuss the data provided by The Cooperative Association for Internet Data Analysis (CAIDA). The proposed model is re-assessed in Section 5.5 in order to match the available data and the model.

5.1 Code-Red. Introduction

On June 18, 2001, information about a buffer-overflow vulnerability in Microsoft's Internet Information Server IIS web servers was released by eEye [13], which was followed by a Microsoft patch eight days later [36]. On July 12, 2011, the Code-Red worm version 1 (further referred as CRv1) started to spread by exploring this vulnerability. There was no direct damage from CRv1, except for the following message:

Welcome to http://www.worm.com! Hacked by Chinese!

which was displayed on the home pages of some infected servers. CRv1 did not spread widely due to the static seed in the pseudo-random number generator, which caused the worm to scan (try to infect) a list of machines (servers) which was identical for each infected host. The only tangible effects were visible in local networks due to the additional resources consumed on infected hosts; the impact on global resources was negligible.

Following CRv1 on July 19, 2011, at approximately 10:00 UTC, Code-Red version 2 (further referred as CRv2) started to spread. It appears that unlike CRv1, CRv2 used a random seed in its random number generator. Therefore, each of the infected machines tried to infect a different list of randomly generated IP addresses at an observed rate of, approximately, 11 probes per second. Although the worm did not cause any direct damage, again apart from the "Hacked by Chinese" message, CRv2 had a major impact on productivity loss due to the huge number of infected hosts and probes sent. It infected between 1 and 2 million computers out of a possible 6 million, which is approximately the number of existing IIS servers at that time [100]. Moreover, since the lists of IP addresses to infect were drawn randomly, CRv2 was sending probes not only to vulnerable IIS web-servers, but to all kinds of hosts; although these could not be infected as such, they could crash or reboot under the attack. It was the most costly malware of 2001, which resulted in total cost of approximately $2.75 billion [100], including costs of clean-up. Code Red was deemed by the FBI to be so dangerous that it could bring down the entire Internet due to the increased traffic from the scans.

Both versions of Code-Red were programmed to take identical actions when infecting new machines. First the system clock was checked and then one of the following actions was chosen accordingly:

- **Spreading phase**. If the date is between the 1st and the 19th of each month the worm generates a random list of IP addresses and tries to infect as many machines in this list as possible by performing port 80 TCP SYN scanning. The request contains code that exploits a known buffer overflow vulnerability in the indexing software in Microsoft's IIS, allowing the worm to run code from within the IIS server.

- **Attacking phase**. If the date is between the 20th and the 28th the worm stops spreading and starts a Denial-of-Service attack against the following IP address: `198.137.240.91`, which used to be the IP address of `www.whitehouse.gov`. These attacks flooded the servers with so much useless data that they were unable to function properly [12]. Luckily, the attackers addressed the `whitehouse.gov` by IP address and not by URL. The problem was solved by moving the website to another IP address. If the attackers would address `whitehouse.gov` by the URL then URL would have had to be changed to mitigate the attack.

- **Inactive phase**. The worm is inactive after the 28th of each month. It cannot be wakened and stays in sleep mode unless deliberately executed.

Note that the employed system clock call returns UTC time [90], therefore, all hosts switch between these three phases simultaneously, unless a host is malfunctioning, e.g., the system clock is set wrong. The Code-Red worms are both vulnerable to system rebooting, which means that the infected machine is disinfected when rebooted, however, the machine remains vulnerable. The only way to protect a machine is by applying a patch, which was made available as of June 26, 2001.

The CRv2 worm was able to cause major damage within the 14 hours it was spreading; at midnight of July 20th it stopped spreading, as it was programmed to. On August 1, 2001, the worm started to spread again, and by midnight approximately 275,000 unique hosts were infected (as indicated by the available dataset [96]).

5.2 CRv2 mean-field model: first attempt

In the following we present a model for the *spreading phase* of the CRv2 worm which takes place between the 1st and the 19th of each month. This model is based on the description of the worm behaviour as given in the previous section.

Let us first build the local model which reflects the spreading behaviour of a single host. From the description of the worm we conclude that there are three modes the machine can be in while the worm spreads: *Vulnerable, Infected,* and *Patched*. This results in the finite local state space $S^l = \{s_1, s_2, s_3\}$ with $|S^l| = K = 3$ states. They are labelled as *Vulnerable, Infected*, and *Patched*, as indicated in Figure 5.1. The transition rates are as follows:

- A *Vulnerable* machine becomes *Infected* with rate k_1^*, which increases if the number of infected hosts in the environment grows.
- An *Infected* machine is rebooted and returns back to the *Vulnerable* state with constant rate k_2.
- A patch might be installed to a *Vulnerable* or *Infected* machine, which happens with rates k_4^* and k_3^*, respectively. These rates depend on the awareness of the worm existence, i.e., as the number of infected machine grows, the awareness of system administrators increases as well.

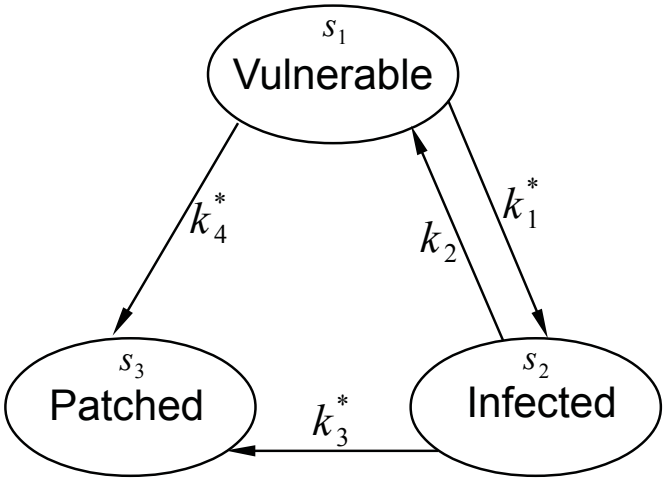


Figure 5.1: The model of the CRv2 propagation for a single server.

- A *Patched* machine can not be infected and stays in that state for the remaining time.

Given a network of N nodes (where N is assumed to be large, which is a reasonable assumption in this context), we can model the overall average behaviour of this network under the CRv2 attack through the global mean-field model (see Section 2.1 for more details). It has the same underlying structure as the individual model (see Figure 5.1), however, its state space is $S^o = \{\overline{m} = (m_1, m_2, m_3)\}$, where m_1 is used to denote the fraction of *Vulnerable* machines, and m_2 and m_3 correspond to the fraction of *Infected* and *Patched* machines, respectively, and the occupancy vector denotes the distribution of machines over the states of the local model. To address the state of the model at a given time t the occupancy vector is equipped with a notion of time: $\overline{m}(t) = (m_1(t), m_2(t), m_3(t))$.

After defining the global model the transition rates need to be specified. The rates k_1^*, k_3^*, k_4^* depend on the number of infected hosts. The infection rate can be expressed as follows:

$$k_1^*(t) = k_1 \cdot m_2(t),$$

where k_1 is the infection rate of a single machine. This representation of the infection rate is quite accurate, as it takes into account each of the m_2 infected computers, that each spread the worm with an identical constant rate

k_1. This representation is different from the Botnet case-study presented in Chapter 3, and is commonly used, for example, in epidemiological models. It is more appropriate for this case-study, since CRv2 targets randomly chosen IP addresses in contrast to only targeting not infected machines (as in the Botnet case-study).

The patching rates are difficult to describe as the human factor plays a big role in this process [64]. We assume here that each infected machine increases the awareness of the system administrators (more system administrators hear about CRv2), which leads to a patch application. Given the above description of the patching rate we first introduce k_3 and k_4: the rates of patching for one infected or vulnerable host, respectively. In our model we assume that the number of patch applications changes with the fraction of infected machines, similar to the infection rate:

$$k_3^*(t) = k_3 \cdot m_2(t), \text{ and } k_4^*(t) = k_4 \cdot m_2(t).$$

Once we have defined the rates, we can apply Theorem 2.2.1 to derive the system of ODEs that describes the transient behaviour of the mean-field model,

$$\begin{cases} \dot{m}_1(t) &= k_2 \cdot m_2(t) - k_1 \cdot m_2(t) \cdot m_1(t) - k_4 \cdot m_1(t) \cdot m_2(t), \\ \dot{m}_2(t) &= k_1 \cdot m_2(t) \cdot m_1(t) - k_2 \cdot m_2(t) - k_3 \cdot m_2(t) \cdot m_2(t), \\ \dot{m}_3(t) &= k_4 \cdot m_1(t) \cdot m_2(t) + k_3 \cdot m_2(t) \cdot m_2(t), \end{cases} \tag{5.1}$$

with initial conditions $\overline{m}(0) = \{m_1(0), m_2(0), m_3(0)\}$. The way to assign initial conditions will be determine in the next chapter.

5.3 Code-Red data sets

The data used in this case-study is the CAIDA Dataset on the Code-Red Worms - July and August 2001 [96]. This dataset consists of a publicly available set of files that contain summarized information that does not identify infected hosts individually.

5.3.1 July 2001

Let us first describe the data sets available for the first outbreak of CRv2 in July, as in [96]. The dataset includes data from the following three data-sources (see Figure 5.2):

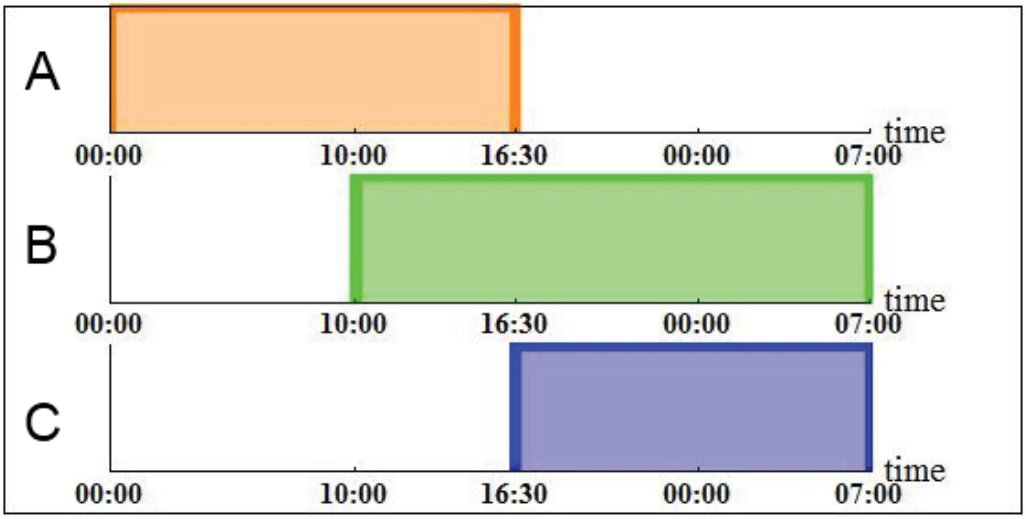


Figure 5.2: Time intervals where each datasets **A** (orange), **B** (green), and **C** (blue) are available for the first outbreak of CRv2 on July 19-20, 2001.

A Packet headers collected from a /8 Telescope Network [95] at the University of California, San Diego (UCSD), between midnight and 16:30 UTC on July 19. The UCSD Network Telescope is a globally routed /8 network, where an /8 network is defined by the 8 leftmost bits of the 32-bit IP address, and contains $2^{32-8} = 2^{24} = 16\ 777\ 216$ addresses. Because the network telescope includes one out of every $256(2^8)$ IPv4 addresses, it receives approximately one out of every 256 probes from hosts infected with randomly scanning worms. The availability of this dataset is depicted by the orange rectangle on Figure 5.2.

B Timestamp/IP address pairs for TCP SYN packets (caused by port 80 scanning) received by two /16 networks at Lawrence Berkeley Laboratory (LBL) [81]. This data consists of probe information collected by Bro [82] on the LBL networks between 10:00 UTC on July 19, 2001, and 7:00 UTC on July 20, 2001, which is illustrated by the green rectangle on Figure 5.2.

C Sampled net-flow data from a router upstream of the /8 Telescope Network at UCSD [95], which is collected after 16:30 UTC on July 19. It was not possible to capture IP packet headers after 16:30 UTC (as in data-source **A**), due to a filter, which was put into place upstream of the

monitor. This filter was installed on a campus router to reduce congestion caused by the worm, it blocked all external traffic to this network. However, an UCSD data set consisting of *sampled* net-flow output from the filtering router was available at the UCSD site throughout the 24 hour period. The sampled net-flow data can be seen as an approximation of the individually captured packet headers, as in the data-source **A**. This dataset is depicted by the blue rectangle in Figure 5.2.

These three data sources are used to maximize coverage of the expansion of the worm. Note that the datasets were merged by CAIDA to produce the anonymized Code-Red July dataset [96], therefore, we can not access them separately or obtain IP addresses of infected machines.

5.3.2 August 2001

The dataset of the second outbreak of CRv2 (August 2001) includes packet headers collected from a /8 Telescope Network at UCSD (as in data-source **A**). This dataset does not include any external data (as in data-source **B**) from outside the Telescope network. However, this dataset is sufficiently large (see [95]) and can be used for further analysis.

5.3.3 Available data

Both datasets (collected in July and August, respectively) consist of Unix time stamps (UTC time) and counters (number of hosts). The time stamp represents the time a new IP address is counted, and the counter keeps track of all IP addresses registered (see Table 5.1). Thus each line in Table 5.1 corresponds to one newly registered IP address.

There are two traces of data available: (a) start and (b) end times of hosts performing port 80 TCP SYN scanning (cf. Table 5.1), which can be interpreted as follows:

(a) **Unique newly infected hosts** or hosts starting to spread infection. Hosts were considered to be infected if they sent at least two TCP SYN packets on port 80 to non-existent hosts on these networks, which helps to eliminate random source denial-of-service attacks from the Code-Red data. The term *unique* here means that the host was added to this dataset only once, when it first became infected. If the host gets reinfected it will not be counted a second time.

Unix timestamps	Counter
995500872.242891	1
995500873.623339	2
995500876.419235	3
995500876.964408	4
995500881.546861	5
995500885.53822	6
995500892.317562	7
995500897.297761	8

(a) Unique newly infected hosts

Unix timestamps	Counter
995500873.623339	1
995500876.964408	2
995500918.699153	3
995500933.063657	4
995500935.712072	5
995500947.16169	6
995500948.64369	7
995500965.217833	8

(b) Hosts stopped being infected

Table 5.1: First elements of the data set by CAIDA. This data consists of the UNIX timestamps when the unique IP addresses (a) started spreading the CRv2, (b) stopped spreading the CRv2; and the counter.

(b) **Hosts stopped being infected (*inactive*)**. A host, which was previously infected is considered to be inactive after there was no further unsolicited traffic observed from it. Note than if the host was disinfected, and got infected again it will not appear in this dataset, as it starts to scan again. Only the hosts which remain inactive are collected in this dataset.

Note that since the data represents only a sample of all probes sent by infected machines (approximately every 256 probes) it provides a lower bound on the number of the infected (and inactive) machines at any given time. In the following we graphically represent the available data for the purpose of further analysis.

5.4 Code-Red data analysis

To obtained a detailed view on the data we accumulate the data into buckets of 1 minute, similar to [77], i.e., we calculate the number of infections (disinfections) occurring every minute.

5.4.1 July 2001

Figure 5.3 depicts the number of newly infected (red) and disinfected (blue) hosts per minute on July 19-20, 2001. In Figure 5.3a we observe a peak of activity of 2000 newly infected hosts per minute, which occurred almost at the same time when the passive monitor data (data-source **A**) became unavailable, which makes the duration of this activity unknown. Moreover, the largest peak of 7000 is registered at around 17:21, however, according to [77] it does not correspond to the CRv2 hyperactivity, but to a gap in the data collection. The latter caused the detection of all hosts infected during the gap at the time when the collection resumed (all infections between 16:51 and 17:21). The infection slowed down before midnight, when the worm was programmed to stop the spreading phase. The reason why the rate of infection declines has been attributed in the literature to various reasons:

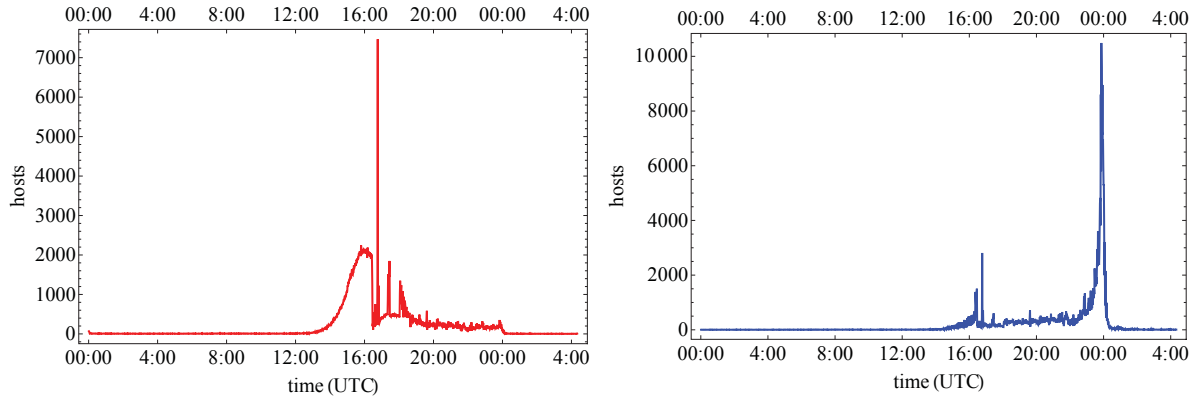
- *CRv2 caused the network overflow.* This explanation is provided in [108], where the *two-factor model* for the worm was equipped with the extra factor in order to address this phenomena. Note that there is no other evidence available to support this version.

- *Many vulnerable hosts were infected or patched.* As was mentioned above, the large amount of hosts were infected, which indeed might make "finding" vulnerable hosts more difficult as suggested by [77].

- *The machines in some time zones were switched off.* According to [77] a daily pattern was observed in the infection rate, therefore, many of the infected machines were actually office desktops, whose users were not aware that they are running an active web server. Therefore, we argue that the slow down might be due to the fact that more and more computers are switched off in each time-zone when the working day ended and, hence, do not contribute to the propagation of CRv2.

Figure 5.3b depicts the number of inactive computers per minute. The peak at around 16:30 is caused by the same gap in data collection, as mentioned above. The activity peak at the end of the day of July 19, 2001, is due to the switching from the spreading phase to the attack phase.

The cumulative total of the unique infected and inactive hosts is calculated in order to access the total number of infected and inactive hosts over time. Figure 5.4 depicts the number of unique newly infected hosts (red solid line)



(a) Newly infected hosts

(b) Disinfected (inactive) hosts

Figure 5.3: Number of (a) infected and (b) inactive hosts per minute for CRv2 on July 19-20, 2001, as presented in [77].

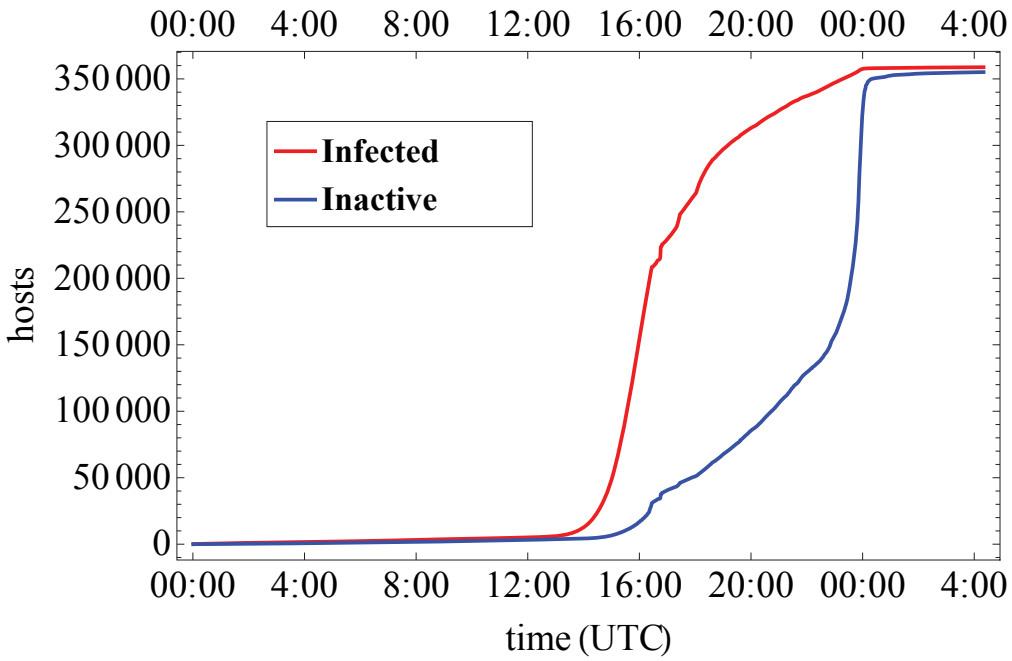
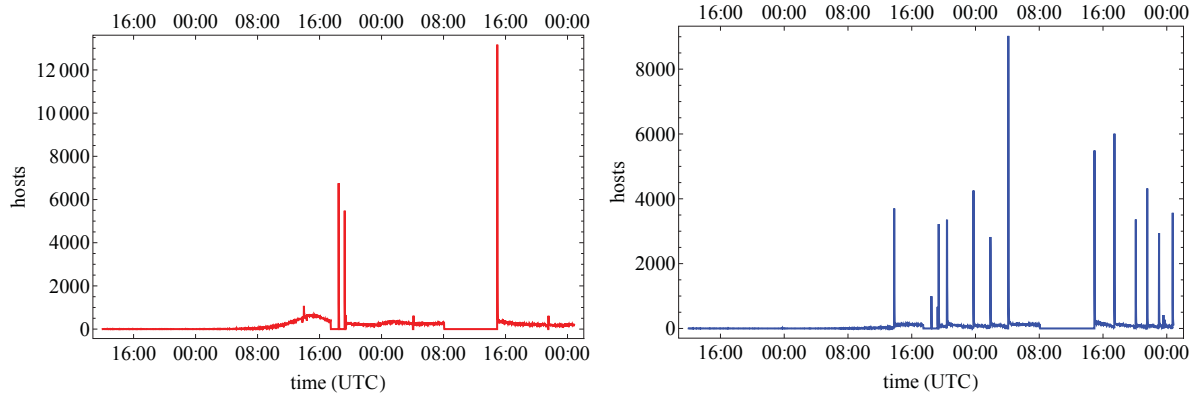


Figure 5.4: The cumulative total of unique infected (red) and inactive (blue) hosts on July 19-20, 2001, as presented in [77].



(a) Newly infected hosts

(b) Desinfected (inactive) hosts

Figure 5.5: Number of (a) infected and (b) inactive hosts per minute for CRv2 on July 31, August 1 and 2, 2001.

and inactive hosts (blue solid line) over time on July 19-20, 2001. As one can see in Figure 5.4, the growth is smooth until at around 16:30 the dataset is affected by the gap in the data collection as was described above. The number of infected (and inactive) hosts is also affected by the data collection problem and it stops growing at midnight because the worm was programmed to stop spreading.

5.4.2 August 2001

Figure 5.5 depicts the number of newly infected (red) and disinfected (blue) hosts per minute on August 1 and 2, 2001. The number of infected hosts (Figure 5.5a) grows smoothly during the first hours of August 1, 2001. At approximately 17:30, August 1st, the growth stops and the first big peak appears, followed by the second peak after approximately 1 hour. There is no evidence which explains this behaviour, however, it clearly looks like a gap in the data collection, similar to the one registered by CAIDA in the data from July. After the gaps the number of infected hosts per minute does not grow, which, probably, means that the number of vulnerable hosts was very low, and it took more time to locate such hosts to infect them. In Figure 5.5b we observe the gaps in the data collection as well. The first peak appears earlier than in the previous figure, that is between 13:40 and 14:00 on August 1, 2001.

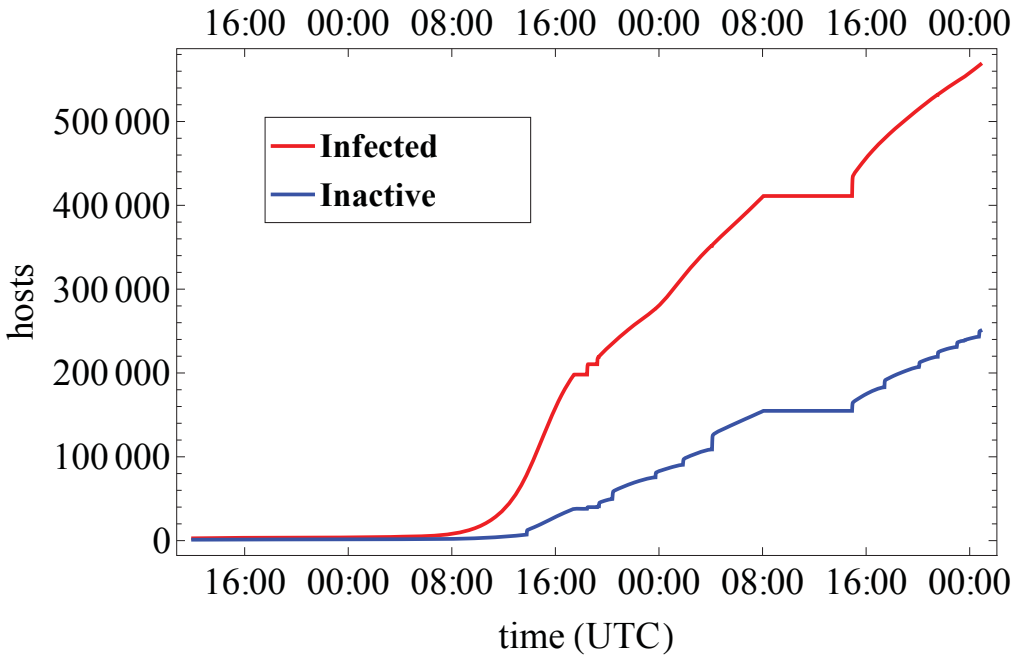


Figure 5.6: The cumulative total of unique infected hosts (red) cf. [77], and inactive hosts (blue) on July 31 and August 1-2, 2001.

The cumulative total of infected (red solid line) and inactive (blue solid line) hosts for data between August 1 and 2, 2001, is depicted in Figure 5.6. It is again visible that this data is affected by, probably, measurement problems, as discussed above. However, there is no evidence available to provide the exact reason for this ambiguity.

Given the information above we can now try to find the correspondence between the state of the mean-field model and the available traces of data.

5.5 CRv2 mean-field model: reconsideration

In Section 5.2 the mean-field model for CRv2 was built based on the worm description only. The next step is finding the parameters of this model which provide the best fit to the data. However, an extra-step has to be taken in order to make sure that the model not only reflects the behaviour of the system (network under CRv2 attack), but also fits the available data. In case of CRv2, the model has to be changed to match the available data. In the following we motivate the changes and build the new model.

5.5.1 Rebooting

First of all, the rebooting transition from the infected host back to vulnerable has to be reconsidered. According to the data description rebooting was not captured in the dataset. A rebooted host can contribute to the dataset in two different ways:

- a rebooted host does not get infected again, and is added to the dataset with inactive hosts,
- a rebooted host gets infected again, however, it will not be counted as *unique* infected host for this second infection, so it is not counted.

Since the rebooting transition can not be captured using the available data we eliminate it from the model. This will, however, influence the fitted parameters in the following way:

- The propagation speed of CRv2 is found to be very large, therefore, rebooted hosts were very likely to be reinfected again, which would mean that the actual number of infections is bigger than the measured data shows.
- On the other hand, if a host has been rebooted but has not been infected again, the actual number of disinfections is lower than the one measured (inactive hosts), since a rebooted host was counted as inactive.

Given the reasoning above, we conclude that if the rebooting transition is eliminated the infection rate will be under-approximated, while the disinfection rate will be over-approximated. However, the number of rebooting events is significantly lower compared to actual infections, therefore, the over/under-approximation is relatively insignificant.

5.5.2 Patched machines

As was discussed in the previous section the data captures only the inactive machines (patched after being infected), therefore, the patching transition in the mean-field model has to be reconsidered. The patched machines have to be split into two groups: (i) the machines, which became inactive after being infected, and (ii) the machines, which were never infected before getting patched. This change will not influence the fitted parameters, since each patched host will be counted once.

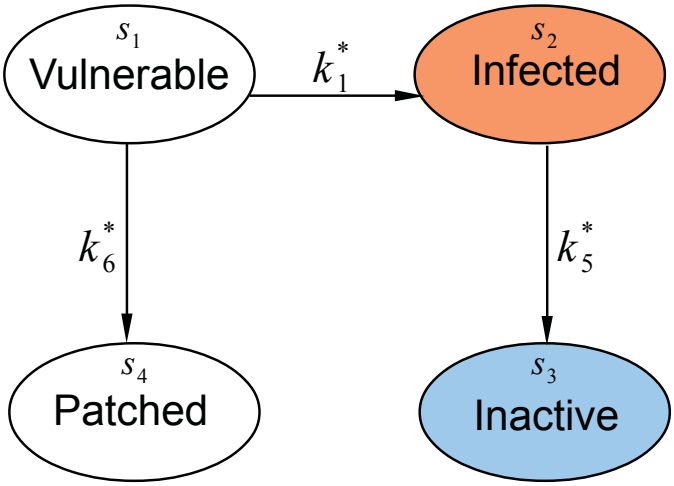


Figure 5.7: The rethought model of CRv2 propagation

5.5.3 Refined mean-field model

Given the above considerations, a fourth state is added to the model of CRv2. The states are labelled as *Vulnerable, Infected, Inactive, Patched*, where the *Inactive* state reflects patching ***after*** being infected (as present in the dataset); patching ***before*** being infected belongs to the *Patched* state (see Figure 5.7). The model then has a finite local state space $S^l = \{s_1, s_2, s_3, s_4\}$ with $|S^l| = K = 4$ states, and the transition rates are as follows:

- A *vulnerable* machine becomes *Infected* with rate $k_1^*(t) = k_1 \cdot m_2(t)$, as discussed in Section 5.2.

- *Infected* machines are patched (and become *Inactive*) with rate $k_5^*(t) = k_5 \cdot m_2(t)$, where k_5 is the patching rate of one infected host.

- *Vulnerable* machines are patched with rate $k_6^*(t) = k_6 \cdot m_2(t)$, where k_6 is the patching rate of one vulnerable host.

Note that, the assumptions for the transition rates are kept the same as in Section 5.2.

Given a system of N such hosts, the overall mean-field model $\mathcal{M}^O$ is built as in Section 5.2. The set of ODEs, describing the transient behaviour of the

overall model is as follows:

$$\begin{cases} \dot{m}_1(t) &= -k_1 \cdot m_2(t) \cdot m_1(t) - k_6 \cdot m_1(t) \cdot m_2(t), \\ \dot{m}_2(t) &= k_1 \cdot m_2(t) \cdot m_1(t) - k_5 \cdot m_2(t) \cdot m_2(t), \\ \dot{m}_3(t) &= k_5 \cdot m_2(t) \cdot m_2(t), \\ \dot{m}_4(t) &= k_6 \cdot m_1(t) \cdot m_2(t), \end{cases} \tag{5.2}$$

with initial conditions $\overline{m}(0) = \{m_1(0), m_2(0), m_3(0), m_4(0)\}$. The initial conditions are not available at the moment, we provide a way to obtain them in the next chapter.

Note that there is a couple of other possible changes which might be considered after studying the dataset, for example:

- *Daily patterns of the infected machines.* This factor is not included due to the nature of the available data. Since the data was anonimized there is no information available about location (time-zone) of the host, therefore, the adjustments can not be made. Moreover, by considering this factor we would add more unknown parameters in the model, while only two data traces are available for fitting. This might have a negative impact on the results (see chapter Chapter 6).

- *Network overload.* We do not include this into the case-study due to the fact that this will over-complicate the model. Moreover, there is no proof that network overload influenced the spread of CRv2. Finally, due to the gaps in the data collection we will only use part of the data, where the network overload could not play a big role.

5.5.4 Adapted view on the data

The available measurement data (as shown in Figures 5.4 and 5.6) shows the total number of infected and patched hosts, neither of which corresponds directly to state s_2 of the model. To obtain data corresponding to s_2 of the overall model, i.e., the number of hosts still infected at a given time, the number of inactive hosts has to be subtracted from the number of infected hosts. This results in Figures 5.8 and 5.9, which depict this modified view on the data, for July and August 2001, respectively. This data corresponds directly to the states of the extended model: (i) the number of still infected hosts (orange solid line) is reflected by $m_2(t)$ (and state s_2), (ii) the number of inactive hosts (blue solid line) corresponds to $m_3(t)$ (and state s_3), and this data will be used for the fitting procedure in the next chapter.

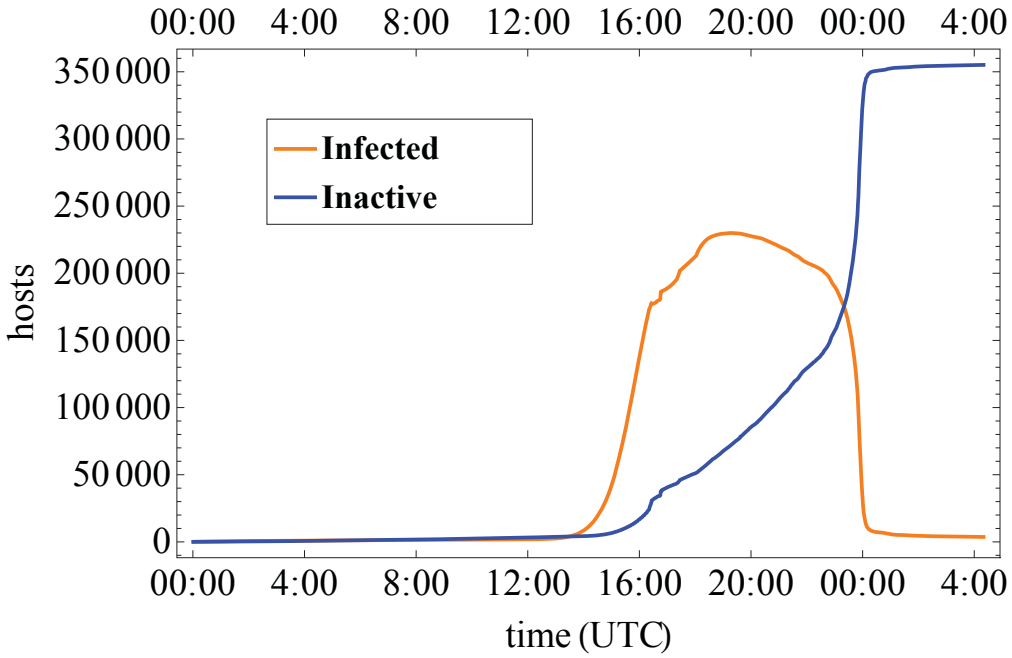


Figure 5.8: The number of hosts, still infected at time t (orange); and inactive hosts (blue) on 19th and 20th of July 2001

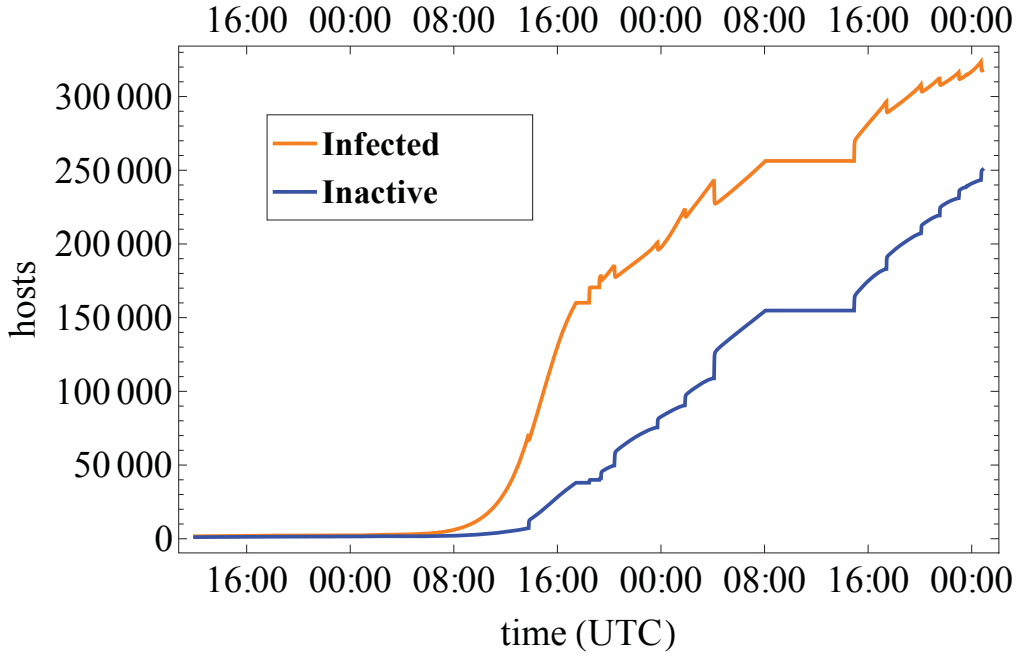


Figure 5.9: The number of hosts, still infected at time t (orange); and inactive hosts (blue) on July 31 and August 1-2, 2001

5.6 Summary

In this chapter we have described the behaviour of the Code-Red worm and built the mean-field model of the spreading phase of the worm behaviour. Moreover, the available data has been discussed, which led to the adaptation of the obtained model. We discussed the relation between the original and the adapted model in more details in Section 6.6, however, it is important to note that the fitting could not be done without adapting the model.

6

Code-Red case study. Results

In this chapter we discuss the fitting procedure, and present best fitting parameters of the mean-field model, we analyse both the first and the second outbreaks of the CRv2, and show that the mean-field model is able to capture the real-life worm behaviour. Moreover, we discuss the possible issues, which can appear during the fitting process.

6.1 Parameter-fitting applied to CRv2

In Section 4.2 two methods for parameter estimation were presented, namely, minimization of the relative squared error and the negative log-likelihood. In the following we apply these methods to the CRv2 dataset with two traces, denoted as $\mathcal{O}(t_r) = \{\mathcal{O}_2(t_r), \mathcal{O}_3(t_r)\}$. In the following we match these traces to the corresponding states of the mean-field model $\overline{m}(t)$, using the mapping as in (4.1):

- $\mathcal{O}_2(t_r)$ denotes the number of infected hosts over time, which corresponds to $M_2(t_r)$,
- $\mathcal{O}_3(t_r)$ represents the number of inactive hosts, which corresponds to $M_3(t_r)$,
- t_r is an observational points in time, where $t_r - t_{r-1} = 1$ minute, because the data was collected in one minute buckets (see Section 5.4).
- $0 \leq r \leq R$, where R is the number of available data observations (the size of the dataset). The size of the dataset is different for the first and the second outbreak of CRv2.

Data corresponding to $M_1(t)$ and $M_4(t)$ is not available, so these can not be included into the fitting procedure. Note that the partial availability of the data may lead to unsuccessful parameter estimation.

The relative squared error and the negative log-likelihood are calculated according to Formulas (4.3) and (4.8) as follows:

$$\mathcal{E}_{\text{rel}} = \frac{\sum_{r=1}^{r=R}(\mathcal{O}_2(t_r) - M_2(t_r))^2 + (\mathcal{O}_3(t_r) - M_3(t_r))^2}{\sum_{r=1}^{r=R}(\mathcal{O}_2(t_r) - \widehat{\mathcal{O}_2})^2 + (\mathcal{O}_3(t_r) - \widehat{\mathcal{O}_3})^2}, \tag{6.1}$$

where $\widehat{\mathcal{O}_2}$ and $\widehat{\mathcal{O}_3}$ are averages of the data traces $\mathcal{O}_2$ and $\mathcal{O}_3$ respectively.

$$\mathcal{L}^* = \frac{1}{R}\sum_{r=1}^{R} \log 2\pi\sigma^2 + \frac{(\mathcal{O}_2(t_r) - M_2(t_r))^2 + (\mathcal{O}_3(t_r) - M_3(t_r))^2}{2\sigma^2}. \tag{6.2}$$

As one can see, in both estimators the Euclidean Distance is being minimized, which means that both relative squared error and negative log-likelihood in this case-study yield the same results. Therefore, exploiting variance as an extra parameter will not lead to a better result for our model. In the following we will use both to check whether these estimators predict the same parameter values.

6.2 Setting initial conditions

The initial conditions for the mean-field model need to be set, as these are critical in order to define the model. There exist a couple of possible ways to determine the initial conditions:

1. The exact information about the initial system state is obtained by measurements or the a-priory known set-up of the experiment.

2. There is circumstantial evidence about the initial system state obtained from the system description or any other additional resources. In this case justifiable assumptions have to be made.

3. Adding the initial conditions as an extra parameters to be estimated.

Knowing the exact initial setting is always beneficial, since inaccurate assumptions can lead to an incorrect fit. However, in most of the cases this information is not available. In what follows, we will make use of each of the above three approaches, thus also showing their advantages and disadvantages.

6.3 CRv2 outbreak in July 2001

We first estimate parameters for the data of July 19th 2001. In order to make sure that we only use the data which can be captured by the proposed model we have to omit the slowing down phase, which starts when the desktops in the eastern time zones start to switch off, and/or the network is overloaded by the large number of probes infected hosts are performing. Note that in general these factors can be included in the mean-field model, however, the data, which was made publicly available by CAIDA is not sufficient enough to do so. However, when more data is available it is possible to extend the mean-field model of the spreading phase and necessary.

As was mentioned in Section 5.4 the slowing down can clearly be seen after 16:20 (see Figure 5.8), and at around this time the gap in the data collection occurred and the dataset **A** become unavailable. Therefore, we use the data collected before 16:20 UTC on July 19th, to avoid (i) the strong effect of the hosts switching off and, (ii) usage of data which was collected in two different ways; and (iii) the gap in the data collection (see Figure 5.3).

6.3.1 Setting the initial conditions for the July outbreak

We need to set the initial conditions for the model. In [19] it was mentioned that CRv2 infected between 1 and 2 million out of a potential number of 6 million hosts. Therefore, to start with, we set the initial number of vulnerable hosts $M_1(0)$ equal to 6 million, minus the number of nodes initially infected, patched or inactive. We set the initial number of patched nodes to $M_4(0) = 0$, whereas for the initial number of infected and patched nodes we take the corresponding number from the trace at 10:00 o'clock (the generally assumed starting time of the outbreak of CRv2). That is, we set $M_2(0) = 4181$ and $M_3(0) = 2528$. Note that these could be "left-overs" from the CRv1 outbreak, hence, we cannot be sure whether this is a good choice.

Figure 6.1 depicts the (fitted) model prediction of the number of infected (orange solid line) and inactive (blue solid line) hosts, and the measured number of infected (orange dashed line) and inactive (blue dashed line) hosts. The model parameters are obtained by minimising the relative squared error and the negative log-likelihood. As one can see, the fitted model (solid lines) does not fit the measurement data (dashed lines) very well. It overestimates the number of infected hosts in the beginning by 10 to 20 thousand hosts, and underestimates it after 15:30 UTC. This is also reflected in the relative squared

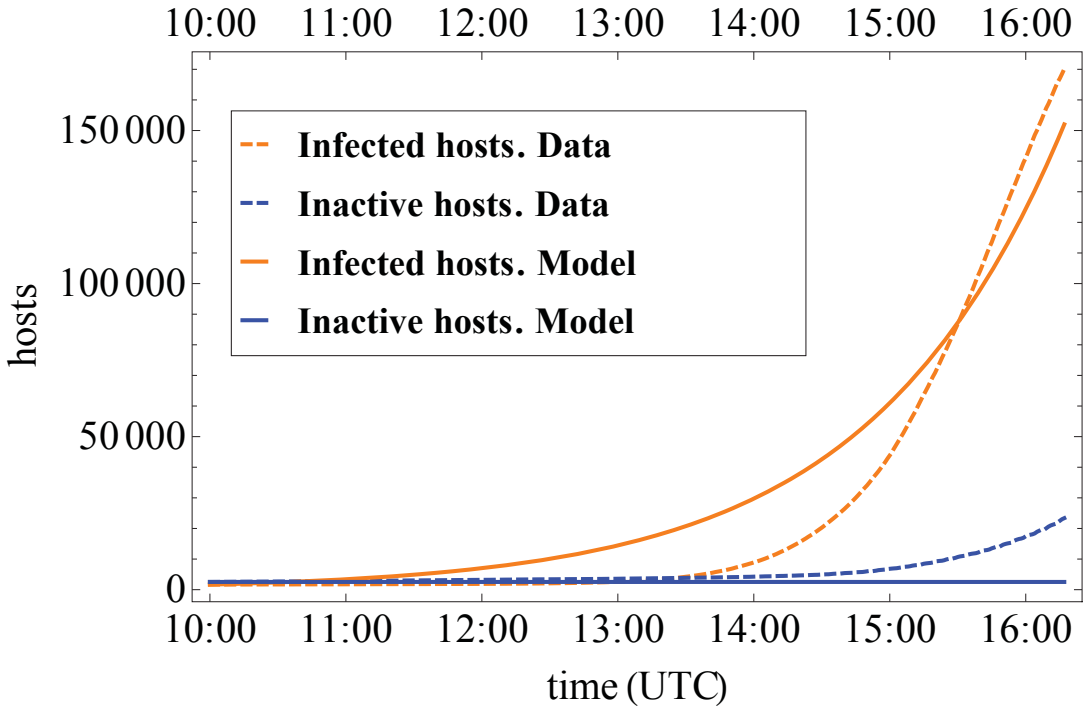


Figure 6.1: Fitting results for the number of infected hosts at time t (orange solid lines), and inactive hosts at time t (blue solid lines) for the dataset (dashed lines) of July 19, 2001; initial condition $\overline{M}(0) = (6 \cdot 10^6 - 6709, 4181, 2528, 0) = (5\,993\,291, 4181, 2528, 0)$.

error of the fitting procedure, which is approximately 9.9%, and negative log-likelihood, which equals 265 308.

Apparently, there is a factor we did not take in account well enough. Most probably, the parameter k_1 (the speed of the virus propagation) is underestimated (see results after 15:30), whereas the initial overestimation might be due to the incorrect initial settings (the number of the initially infected hosts is too big).

6.3.2 Reconsidering initial conditions

The reason for the bad fit might lie in the fact that the activity of CRv1 and other unsolicited SYN probes were already registered before CRv2 started to spread (see [77]). Because of this, all infections that took place before 10:00 UTC have to be subtracted; which leaves only 3 infected hosts at 10:00 UTC. In

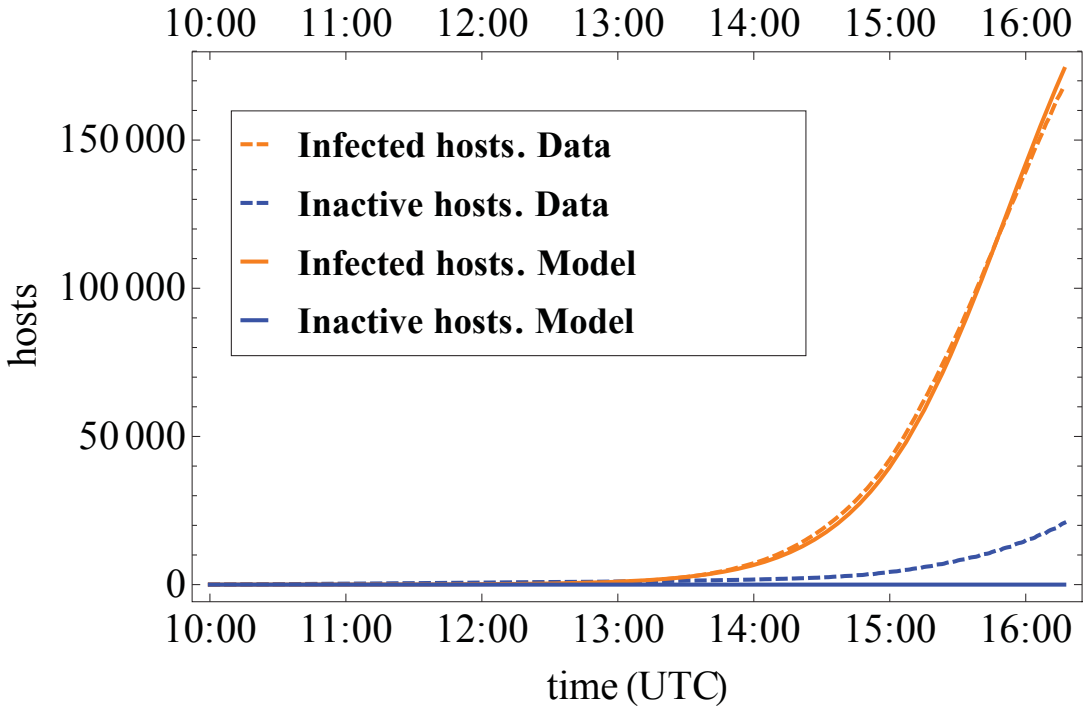


Figure 6.2: Fitting results for the number of infected hosts at time t (orange solid lines), and inactive hosts at time t (blue solid lines) for the dataset (dashed lines) of July 19, 2001; initial condition $\overline{M}(0) = (6 \cdot 10^6 - 3, 3, 0, 0)$.

this context, note that the initial number of hosts infected with CRv1 is known to be 3 [77]; it appears consistent that CRv2, as direct "improvement" of CRv1 starts from the same number of initially compromised hosts. Similarly, all the hosts which were registered as inactive at 10:00 UTC have to be eliminated since they were not counted due to the CRv2 activity and are not captured by the model. Hence, we refit the model, but now with the data, which was corrected accordingly, namely, $\mathcal{O}_1(t_r) = \mathcal{O}_1(t_r) - 4181 + 3$; $\mathcal{O}_2(t_r) = \mathcal{O}_1(t_r) - 2528$, and initial conditions $M(0) = (6 \cdot 10^6 - 3; 3; 0; 0)$.

The results of the new parameter fitting are shown are Figure 6.2. The fit for the number of infected hosts is quite good: the observed data (orange dashed line) is almost indistinguishable from the number predicted by the model (solid orange line). Also the relative squared error of the fitting procedure has reduced to 1.6%. Minimizing the negative log-likelihood yields the same result, and $\mathcal{L}^* = 5114.22$, while in the previous experiment it was 265 308. The negative likelihood still looks large, however, compared to the

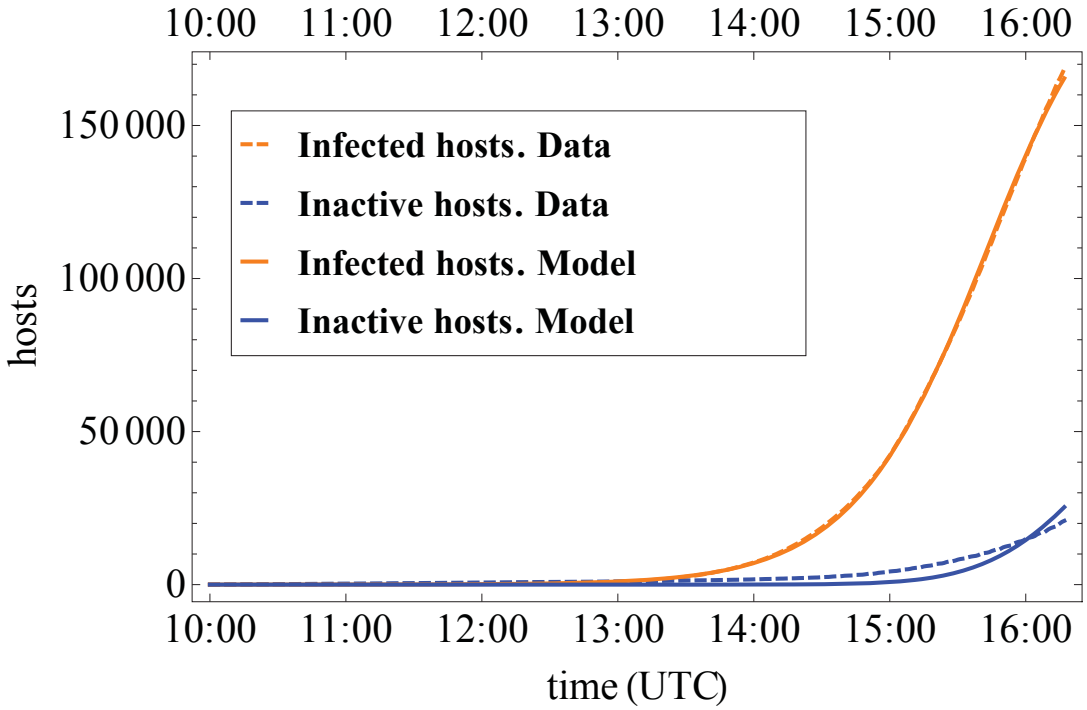


Figure 6.3: Fitting results for the total number of infected hosts (orange solid line), inactive hosts (blue solid line) and the corresponding observed data points (dashed lines, same colour), for July 19, 2001. Initial condition $\overline{M}(0) = (2 \cdot 10^6 - 3, 3, 0, 0)$.

total size of the population and the number of infections it is relatively small.

To a large extent, the remaining uncertainty is due to the estimation of the number of inactive hosts. This can be explained by the fact that the inactive hosts are difficult to model because it involves modelling human behaviour: as explained in Section 5.2 we have assumed for simplicity that the rate of patching hosts is linearly proportional to the number of infected hosts.

6.3.3 Double-checking assumptions

We now go further into the choice of the initial conditions, in particular, the number of initially vulnerable hosts ($M_1(0)$ in our model). We performed 60 fitting experiments, where we took $M_1(0) = I - M_2(0)$, with $I \in \{5 \cdot 10^5; 6 \cdot 10^5; \ldots; 6.4 \cdot 10^6\}$, $M_2(0) = 3$ and $M_3(0) = M_4(0) = 0$. We then find that when the number of initially vulnerable hosts is taken in the range from $5 \cdot 10^5$

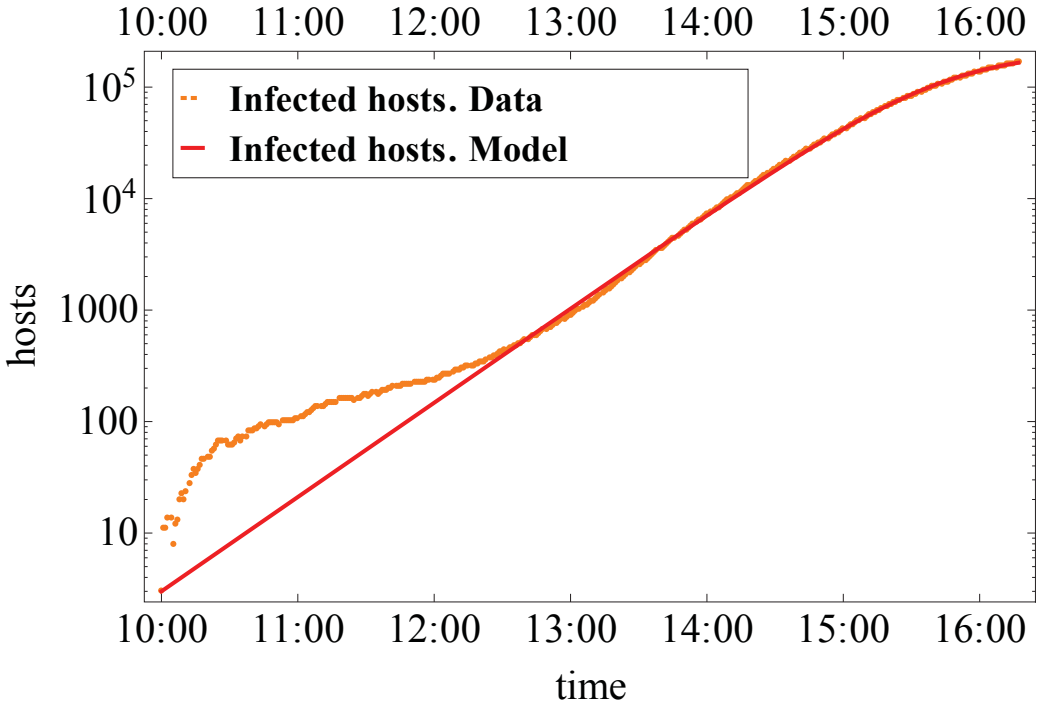


Figure 6.4: Fitting results for the total number of infected hosts (red solid line) and the corresponding observed data points (dotted orange line), for July 19, 2001. Initial condition $\overline{M}(0) = (2 \cdot 10^6 - 3, 3, 0, 0)$.

to $2 \cdot 10^6$, the relative error is smallest, and does almost not change (with value close to 0.2%). In case the initial number of vulnerable hosts is larger than 2 million, the relative error increases. Hence, without relying on any textual source, the (purely numerical) optimal choice for the initial number of hosts lies in the smaller range (and is smaller than the 6 million assumed previously).

Figure 6.3 presents the fitting results for initial conditions $M(0) = (2 \cdot 10^6 - 3; 3; 0; 0)$. As one can see, the model prediction is hardly distinguishable from the measured data, however, the quality of the fit during the early hours of the spread (between 10:00 and 13:00 UTC) is difficult to judge. To be able to evaluate the quality of the fit during the whole time interval Figure 6.4 depicts the number of infected hosts as predicted by the fitted model and measured by CAIDA in logarithmic scale. When the number of infected hosts is small, we observe the "big" difference between the model prediction and the observed data. Recall that in the data we use the activity of CRv1 and random port 80 probes can be measured together with the CRv2 activity. Even though we

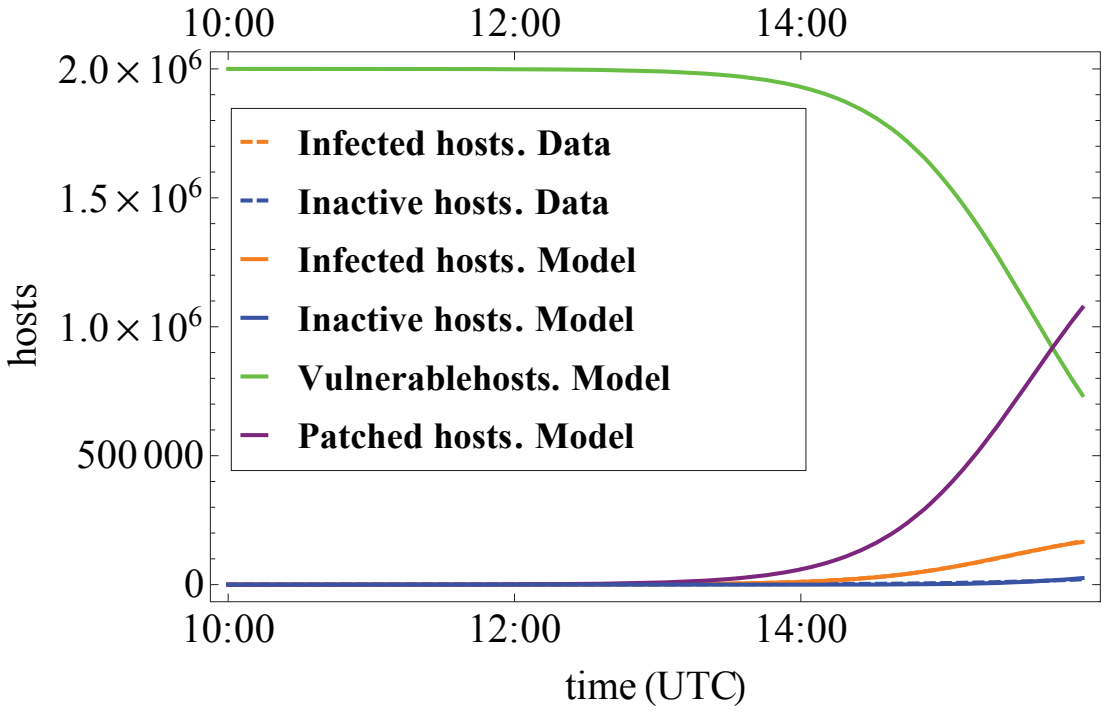


Figure 6.5: Fitting results for the total number of infected hosts (orange solid line), inactive hosts (blue solid line) and the corresponding observed data points (dashed lines, same colour; almost completely overlapping with the solid lines), as well as the vulnerable hosts (green line) and the patched hosts (purple line), all for July 19, 2001. Initial condition $\overline{M}(0) = (2 \cdot 10^6 - 3, 3, 0, 0)$

subtracted the unrelated probes which happen before 10:00 UTC, CRv1 has been active during whole day of July 19th. Therefore, the underestimation seen in Figure 6.4 during the first hours of the CRv2 spread might be due to the fact that these unrelated activities are still "visible" and have an impact on the data. However, this impact is limited (due to the static seed of CRv1), and once the number of host infected by CRv2 exceeds by far the unrelated activity, the fit becomes excellent. Note, moreover, that the underestimation does no exceed 100 hosts, therefore, it does not have a significant impact on the estimators (relative squared error and negative log-likelihood), hence a good fit can still be obtained.

Finally, Figure 6.5 depicts the fitted model behaviour for all four states including the two states for which no data was available. The number of patched hosts is estimated to be quite high, and the number of vulnerable

hosts is getting smaller, which explains the slowing down of the infection.

Earlier in this section we discussed that (1) the fitting procedure for CRv2 is done on partially available data, and (2) we made assumptions about the number of vulnerable hosts in the population. Both these factors influence the obtained result as follows:

(1) As the number of vulnerable and patched hosts are not available the fitting procedure is *free* to choose these numbers for minimizing the squared error. If there is more data available, the fitting procedure would probably yield a different parameter set, however, given the available information the obtained fit is the best possible.

(2) As we discussed above, the correct choice of the initial conditions might influence the results. We performed the experiments in order to find out what would be the best initial conditions for the given case-study (see Figure 6.3). Moreover, we checked how these changes would influence the obtained parameters. The infection rate k_1 remains constant in all 60 experiments, while the patching rate k_6 increases when the population grows, which leads to a large number of patched hosts.

6.4 CRv2 outbreak in August 2001

The parameter fitting for the measurements from August 1, 2001 will be addressed in this section. As before, also for the August dataset the initial conditions are unknown. It seems reasonable to assume that the number of vulnerable hosts did not change dramatically after the first outbreak of CRv2; only a limited number of hosts was patched during the attacking phase of CRv2, as also suggested in [77]. We therefore set $M_1(0) = 1.5 \cdot 10^6$. It is again difficult to find good values for the initial number of infected machines and inactive machines, due to the fact that all the activity of CRv2 before 00:00 UTC has to be taken into account, and, in addition, all the nodes being affected by background activity have to be eliminated.

In this section we illustrate the third way (cf. Section 6.2) of dealing with unknown initial conditions, that is, we add them as an extra parameters to the fitting procedure. Notice that although this way seems to be the most straightforward, adding extra degrees of freedom to the fitting procedure can lead to a worse result. Therefore, using this method while having limited data

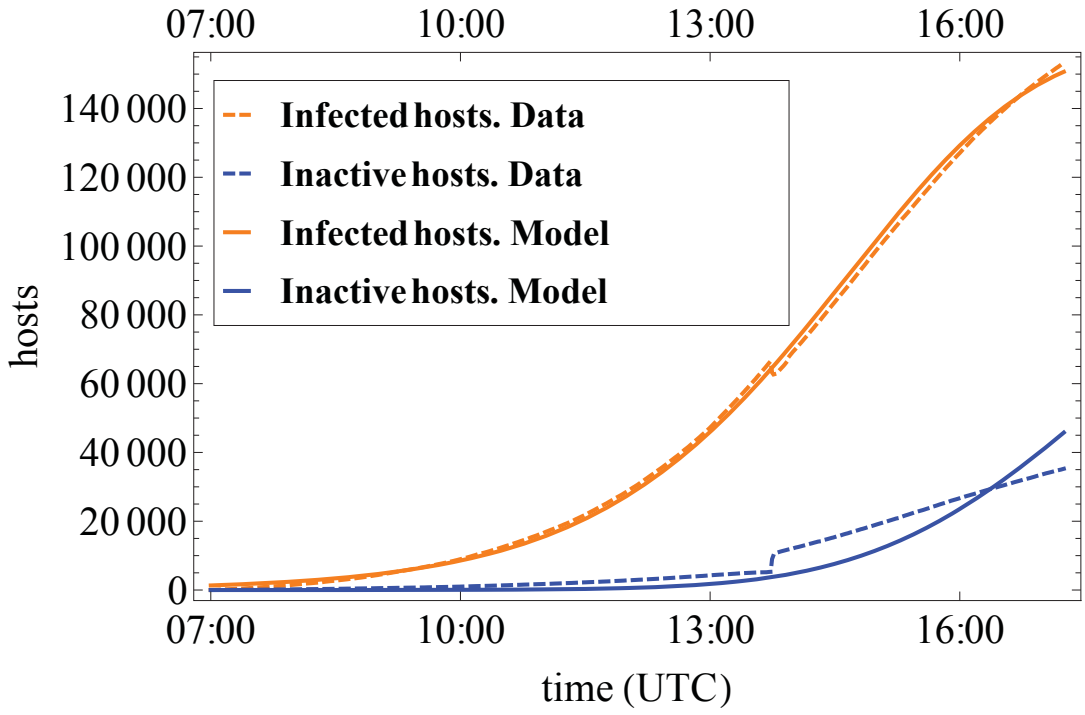


Figure 6.6: he observed data points and fitting results (dashed lines) for the number of infected hosts at time t (orange); and inactivated hosts (blue) on August 1, with initial conditions $\overline{M}(0) \approx (1\,498\,669, 1331, 0, 0)$ (rounded values).

traces (as in CRv2 case) is a "last resort" rather than an obvious solution. Here we take the initial numbers of infected nodes ($M_2(0)$) as extra parameters to the parameter fitting procedure; the number of inactive (M3(0)) and patched hosts ($M_4(0)$) can again be set to zero, as the patching before midnight was not due to the activity of CRv2 for this outbreak. Given the above assumptions, we need to find the parameters k_1,k_5, k_6, and initial condition $M_2(0)$ that minimise the relative squared error between the model prediction and the data.

Figure 6.6 depicts the result of the model fitting procedure for the initial conditions $\overline{M}(0) \approx (1\ 498\ 669; 1\ 331; 0; 0)$. These initial conditions allow us to gain the best fit with a squared error of approximately 0.7%. Note that the number of initially infected hosts is actually not an integer number (but 1 331.16) due to the fact that the mean-field model is a model that addresses fractions of objects, but not every object is modelled independently. As one can see, the fitted model reflects the number of infected hosts during the second

outbreak of CRv2 quite well, despite a still unresolved problem with the data collection around 13:00 UTC.

6.5 Summary

In this chapter the parameter estimation procedure was illustrated. We were able to obtain parameters which ensure the relative squared error of 0.2% and 0.7% between the model prediction and the real data of July and August, respectively. Moreover, a number of possible issues which can emerge when working with real data were addressed, namely:

- how to obtain accurate initial conditions (three possible ways);
- how to make sure that only the data which corresponds to the modelled activity is used;
- how to double-check the influence of the assumptions made before the fitting procedure is started.

The following section concludes the case study providing final remarks and directions.

6.6 Concluding remarks

This case-study provides a full account of the fitting process required to obtain a fully parameterised model of virus spread. We base our model and fitting procedure on publicly known insight in the operation of CRv2, and on publicly available data sets. This part foremost shows the challenges encountered when trying to parameterise a simple model of a large-scale distributed system based on publicly available data sets.

Starting from the characterisation of the system itself, i.e., CRv2, we proposed an initial model describing the state of each host in the network, and subsequently discussed why the unavailability of certain measurement data leads to a slightly adapted model. We also presented why the measurement data has to be handled very carefully, e.g., due to the fact that certain measurement intervals are missing or incomplete, e.g., due to sampling or gaps in the data collection. Furthermore, the available data only reflects part of the system under study, that is, the data does not provide information on the

number of rebooted and patched hosts nor on the total number of vulnerable hosts. For these reasons, the fitting procedure has been a very tedious process, which cannot be easily automated, and from which no final "once-and-for-all recipe" can be given.

The model with the fitted parameter reflects closely the behaviour of the modelled system, which gives the possibility of gaining better understanding of this phenomena, for example, using model-based evaluation techniques (see next part). Note that the original mean-field model differs from the reconsidered model, therefore, some properties of the original model can not be assessed. This is a draw-back of the partially available data. For example, any properties related to rebooting of the infected host are out of the scope of the final model. However, the fitted model reflects most of the properties of the original model, and without the proposed changes the model could not be parametrized, and, hence, studied.

Despite these facts, we have shown that it is possible to find a model and a set of parameters that closely captures the first part of the virus spreading. Whether these are the "ultimate correct parameters" cannot be concluded, simply because we are missing ground truth for statements like that. The models we fitted allowed us to obtain parameters which ensure a relative squared error of 0.2% and 0.7% between the model prediction and the measurement, for the July and August outbreaks, respectively.

Part III

Model-Checking

Recently, many systems that consist of a large number of interacting objects have been analysed using the mean-field method, which allows a quick and accurate analysis of such systems, while avoiding the state-space explosion problem. So far, the mean-field method has mostly been used for performance evaluation. In this part, we discuss model-checking mean-field models. We define and motivate two logics, called MF-CSL and MFL, for describing properties of systems composed of many identical interacting objects. We discuss the differences in the expressiveness of these two logics and possible combination of them.

This part is organized as follows. Chapter 7 provides motivation, related work and describes a running example, used through the part. In Chapter 8 the logic MF-CSL is introduced together with the model-checking algorithms and examples, illustrating checking an MF-CSL property of a computer virus model. The logic MFL is described in Chapter 9; we present syntax and semantics of MFL, model-checking algorithms, and discuss the satisfaction set development. Chapter 10 provides a comparison of the two logics and discusses the combination of MFL and MF-CSL. Concluding remarks are presented in Section 10.3.

7

Model-checking mean-field models

With this chapter we provide an introduction to the model-checking of mean-field models. We motivate the performed study in Section 7.1. An overview of the related work is presented in Section 7.2. The running example used in this part of the thesis is introduced in Section 7.3.

7.1 Motivation

In the previous chapters we showed how a mean-field model can be built and how the model parameters can be obtained. Thus far, the mean-field method was mostly used for performance evaluation of systems using measures like transient and stationary behaviour, maximum (minimum) or mean values, however, also more involved measures are of interest. Therefore, developing methods for efficient and automated *model-checking* of such non-trivial properties is essential and not trivial.

One challenge lies in the fact that the model has two layers, therefore it is essential to be able to formulate properties on both levels. Another challenge is that the local model is a *time-inhomogeneous* Markov chain (ICTMC), therefore the results of the model-checking procedure depend on time. Finally, the state-space of the global mean-field model is infinitely large, hence, finding the satisfaction set is difficult.

In this part of the thesis we introduce two logics, namely, *Mean Field Continuous Stochastic Logic* (MF-CSL), and *Mean-Field Logic* (MFL) together with full model-checking algorithms. The reason for introducing two logics lies in the different types of properties that can be expressed and checked using one or the other. The logic MF-CSL expresses properties of a random

node in a system (including timed properties) and then lifts these up to the overall system level using new *expectation* operators. In contrast, the logic MFL expresses properties of the overall system directly; it does not take into account the behaviour of the individual objects. We compare the logic MF-CSL with the logic MFL, and motivate the existence of both. The possible combination of both logics is also discussed and illustrated by an example.

7.2 Related work

Model-checking means checking whether a system state satisfies certain properties. It was initially introduced for finite deterministic models, for the validation of computer and communication systems, and later extended towards stochastic models and models with continuous time. Checking models of large systems is made difficult by the state-space explosion problem. Hence, model-checking mean-field models is considered a valuable continuation. For an overview we refer to [58].

Before the first steps towards model-checking mean-field models were taken, more general problems of (i) HML (Hennessy-Milner logic) and (ii) LTL (Linear Temporal logic) model-checking of ICTMCs were addressed in [62] and [27]. First an approximate algorithm for verifying stochastic variant of HML on piecewise constant ICTMCs was developed. Then the basis for the LTL model-checking of ICTMCs was provided. Even though the local mean-field model is an ICTMC these algorithms will not be used in this thesis, since different types of properties, namely CSL properties, are checked on the local level of MF-CSL.

The first work on model-checking mean-field models has been presented in [18, 66]. In [18] first steps towards approximate model-checking of bounded CSL properties of an individual object (or group of objects) in a large population model are presented. The Fast Simulation Theorem [30] is used for the characterisation of the behaviour of a single object via the average system behaviour, as defined by the mean-field approximation. The proposed method is called *fluid model-checking*, it has been supplemented with next and steady-state operators in [19].

In contrast to fluid model checking, [66] focuses on the properties of the overall population and proposes the logic MF-CSL (as in Chapter 8). MF-CSL formulas consist of two layers, namely, the local CSL formula, which describes the property of the individual object, and a global *expectation* formula de-

scribing the fraction of objects, satisfying the local formula. The algorithm for checking the until operator on the local level is based on the algorithm presented in [18]. An extra-layer on top of local CSL properties allows describing global properties of the whole system. We discuss this logic in more detail later in this part.

Another two-layer logic is introduced in [21]. The time-bounded local properties are described using 1-clock Deterministic Timed Automata. Then, a global probability operator is introduced on top of that for estimating the probability that a certain fraction of local objects satisfies the local property, instead of the fractions of objects satisfying a certain property as in [66] and Chapter 8.

A different approach for model-checking mean-field models has been proposed in [70]. The authors focus on the properties of an individual object, which is modelled as a discrete-time model in contrast to previously mentioned works. The, so called, *on-the-fly* model-checking approach examines only those states that are required for checking a given property, instead of constructing the whole state-space before starting the actual model-checking procedure.

Another way of looking at a mean-field model is by considering the behaviour of the whole system without addressing its local behaviour. Having in mind the representation of the global behaviour as real-valued signal, one can use Signal Temporal Logic (STL)-like properties [74], as will be discussed further in Chapter 9 of this thesis. Moreover, a great effort has recently been made in enhancing temporal properties with a real value, which is addressed as *quantitative semantics* or *robustness degree* [34, 35, 39, 86]. These methods can be successfully applied to mean-field models, as well. The robustness of stochastic systems (including mean-field or fluid models) has been discussed in [9]. In addition, the design problem has been addressed, where the parameters of the model can be optimized in order to maximize robustness. Several tools are available, which allow calculating the robustness degree [2, 24, 32].

7.3 Running example

Let us introduce a running example now, which will be used to illustrate the expressivity of the two logics and the corresponding model-checking algorithms.

Example 7.3.1. *We address a model of virus spread in a system of interacting computers (see Figure 7.1). We divide the whole system into three groups (e.g.,*

different departments or geographical locations). We name these groups X, Y, and Z. Each group has a fixed number of nodes (computers) N_X, N_Y and N_Z, respectively, where $N = N_X + N_Y + N_Z$. Communication between the groups is possible, but less probable than within a group. The system we model has three different locations and two different types of hardware (or software), where for each type the same computer virus would behave differently, i.e., computers in group X differ from the computers in groups Y and Z. The model of computer behaviour has to take into account the possibility of being (i) in each of the three groups, and (ii) being in each of the states of infection. Let us now consider the spread of the infection.

States represent the modes of an individual computer, which can be not-infected, infected and active *or* infected and inactive *in all three groups; in addition, in group X a computer has to first stay in the* initially connected *state before the virus is fully embedded in the computer system and the computer is infected. An active infected computer spreads the virus, while an inactive computer does not.*

Given this system description, the state-space of the local model has to be extended to incorporate the possibility of being in each of the three groups. This results in the finite local state-space

$$S^l = \{s_{X,1}, s_{X,2}, s_{X,3}, s_{X,4}, s_{Y,1}, s_{Y,2}, s_{Y,3}, s_{Z,1}, s_{Z,2}, s_{Z,3}\},$$

with $|S^l| = K = 10$ *states. They are labelled as* infected_X, not infected_X, $\text{initial connection}_X$, active_X *and* inactive_X, *etc., as indicated in Figure 7.1.*

The transition rates for computers in the first group are as follows: the infection rate $k^*_{X,1}$ *is the rate to become initially connected; after that a computer has to first try to pass the initial connected state with rate* $k_{X,3}$ *or return to the not infected state with rate* $k_{X,2}$. *The recovery rate for an inactive infected computer is* $k_{X,5}$, *the recovery rate for an active infected computer is* $k_{X,7}$, *the rate with which computers become active is* $k_{X,4}$ *and they return to the inactive state with rate* $k_{X,6}$. *Rates* $k_{X,2}, k_{X,3}, k_{X,4}, k_{X,5}, k_{X,6}$ *and* $k_{X,7}$ *are specified by the individual computer and the properties of the computer virus and* <u>do not</u> *depend on the overall system state. The infection rate* $k^*_{X,1}$ <u>does</u> *depend on the rate of attack* $k_{X,1}$, *the fraction of computers that is infected and active and, possibly, the fraction of not-infected computers. The dependence on the overall system state is intended to reflect real-world scenarios and might be different for different situations.*

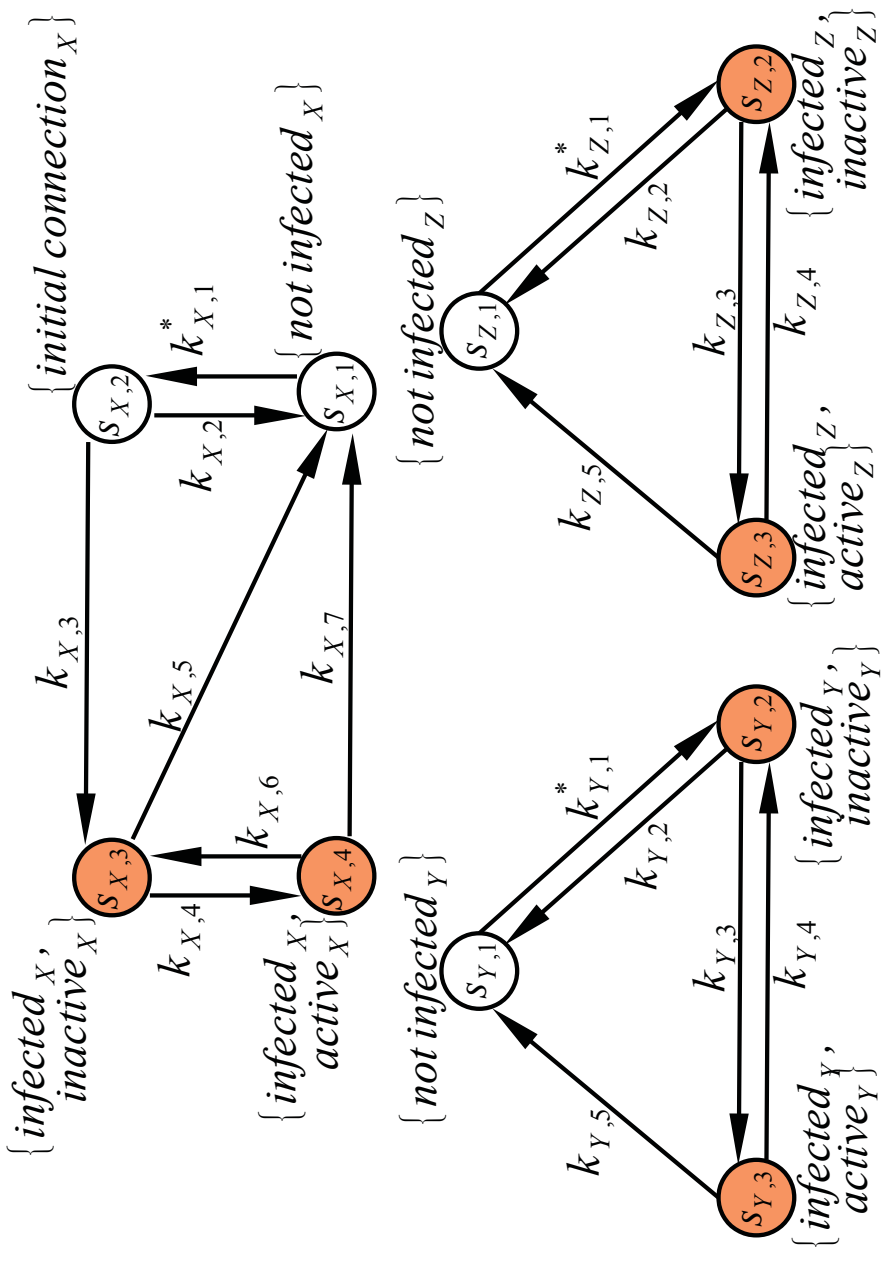


Figure 7.1: The model describing computer virus spread in three groups of computers.

Given a system of N computers, we can model the average behaviour of the whole system through the global mean-field model, which has the same underlying structure as the individual model (see Figure 7.1), however, with state-space

$$S^o = \{m_{X,1}, m_{X,2}, m_{X,3}, m_{X,4}, m_{Y,1}, m_{Y,2}, m_{Y,3}, m_{Z,1}, m_{Z,2}, m_{Z,3}\},$$

where $m_{X,1}$ denotes the fraction of not-infected computers in group X, $m_{X,2}$ represents the fraction of computers in the initially connected state, and $m_{X,3}$ and $m_{X,4}$ denote the fraction of active and inactive infected computers in group X, etc.

The infection rate can then be seen as the number of attacks performed by all active infected computers in group X, which is uniformly distributed over all not-infected computes in a chosen group, that is,

$$k_{X,1} \cdot \frac{m_{X,4}(t)}{m_{X,1}(t)}.$$

Note that we assume here that computer viruses are "smart enough" to only attack computers which are not yet infected, see [99], [65]. As was discussed above, the computers from the different groups might interact with a certain probability. In the context of our virus spread model, this interaction plays a role when infected computers from groups Y and Z might contact not infected computers in group X and vice versa. In this example model we describe a virus which chooses one of the groups of computers with probability $p_{X,X}, p_{X,Y}, p_{X,Z}$ (for an infected computer from group X). The complete infection rates are composed by multiplying the above rates for one group with the probability of choosing a given group and accumulating all possible interactions between the three groups. Given the reasoning above, the infection rate in group X is

$$k^*_{X,1} = p_{X,X} \cdot k_{X,1} \cdot \frac{m_{X,4}(t)}{m_{X,1}(t)} + p_{Y,X} \cdot k_{Y,1} \cdot \frac{m_{Y,3}(t)}{m_{X,1}(t)} + p_{Z,X} \cdot k_{Z,1} \cdot \frac{m_{Z,3}(t)}{m_{X,1}(t)}.$$

The infection rates of groups Y and Z are constructed in a similar way:

$$\begin{aligned} k^*_{Y,1} &= p_{Y,Y} \cdot k_{Y,1} \cdot \tfrac{m_{Y,3}(t)}{m_{Y,1}(t)} + p_{X,Y} \cdot k_{X,1} \cdot \tfrac{m_{X,4}(t)}{m_{Y,1}(t)} + p_{Z,Y} \cdot k_{Z,1} \cdot \tfrac{m_{Z,3}(t)}{m_{Y,1}(t)}, \\ k^*_{Z,1} &= p_{Z,Z} \cdot k_{Z,1} \cdot \tfrac{m_{Z,3}(t)}{m_{Z,1}(t)} + p_{X,Z} \cdot k_{X,1} \cdot \tfrac{m_{X,4}(t)}{m_{Z,1}(t)} + p_{Y,Z} \cdot k_{Y,1} \cdot \tfrac{m_{Y,3}(t)}{m_{Z,1}(t)}. \end{aligned}$$

Theorem 2.2.1 is used to derive the following system of ODEs, that describes the mean-field model:

$$
\left\{\begin{array}{rcl}
\dot{m}_{X,1}(t) &=& -p_{X,X}\cdot k_{X,1}\cdot m_{X,4}(t) - p_{Y,X}\cdot k_{Y,1}\cdot m_{Y,3}(t) - \\
&-& p_{Z,X}\cdot k_{Z,1}\cdot m_{Z,3}(t) + \\
&+& k_{X,2}\cdot m_{X,2}(t) + k_{X,5}\cdot m_{X,3}(t) + k_{X,7}\cdot m_{X,4}, \\
\dot{m}_{X,2}(t) &=& p_{X,X}\cdot k_{X,1}\cdot m_{X,4}(t) + p_{Y,X}\cdot k_{Y,1}\cdot m_{Y,3}(t) + \\
&+& p_{Z,X}\cdot k_{Z,1}\cdot m_{Z,3}(t) + \\
&+& (k_{X,2}+k_{X,3})\cdot m_{X,2}(t), \\
\dot{m}_{X,3}(t) &=& k_{X,3}\cdot m_{X,2}(t) - (k_{X,4}+k_{X,5})\cdot m_{X,3}(t), \\
\dot{m}_{X,4}(t) &=& k_{X,4}\cdot m_{x,3} - (k_{X,6}+k_{X,7})\cdot m_{X,4}, \\
\dot{m}_{Y,1}(t) &=& -p_{Y,Y}\cdot k_{Y,1}\cdot m_{Y,3}(t) - \\
&-& p_{X,Y}\cdot k_{X,1}\cdot m_{X,4}(t) - \\
&-& p_{Z,Y}\cdot k_{Z,1}\cdot m_{Z,3}(t) + \\
&+& k_{Y,2}\cdot m_{Y,2}(t) + k_{Y,5}\cdot m_{Y,3}(t), \\
\dot{m}_{Y,2}(t) &=& p_{Y,Y}\cdot k_{Y,1}\cdot m_{Y,3}(t) + p_{X,Y}\cdot k_{X,1}\cdot m_{X,4}(t) + \\
&+& p_{Z,Y}\cdot k_{Z,1}\cdot m_{Z,3}(t) + \\
&+& k_{Y,4}\cdot m_{Y,3}(t) - (k_{Y,2}+k_{Y,3})\cdot m_{Y,2}(t), \\
\dot{m}_{Y,3}(t) &=& k_{Y,3}\cdot m_{Y,2}(t) - (k_{Y,4}+k_{Y,5})\cdot m_{Y,3}(t), \\
\dot{m}_{Z,1}(t) &=& -p_{Z,Z}\cdot k_{Z,1}\cdot m_{Z,3}(t) - p_{X,Z}\cdot k_{X,1}\cdot m_{X,4}(t) - \\
&-& p_{Y,Z}\cdot k_{Y,1}\cdot m_{Y,3}(t) + \\
&+& k_{Z,2}\cdot m_{Z,2}(t) + k_{Z,5}\cdot m_{Z,3}(t), \\
\dot{m}_{Z,2}(t) &=& p_{Z,Z}\cdot k_{Z,1}\cdot m_{Z,3}(t) + \\
&+& p_{X,Z}\cdot k_{X,1}\cdot m_{X,4}(t) + p_{Y,Z}\cdot k_{Y,1}\cdot m_{Y,3}(t) + \\
&+& k_{Z,4}\cdot m_{Z,3}(t) - (k_{Z,2}+k_{Z,3})\cdot m_{Z,2}(t), \\
\dot{m}_{Z,3}(t) &=& k_{Z,3}\cdot m_{Z,2}(t) - (k_{Z,4}+k_{Z,5})\cdot m_{Z,3}(t),
\end{array}\right. \tag{7.1}
$$

Let us define the parameters of the mean-field model as follows:

$$
\begin{array}{lll}
p_{X,X}=0.93, & p_{Y,Y}=0.94, & p_{Z,Z}=0.97, \\
p_{X,Y}=0.05, & p_{Y,X}=0.05, & p_{Z,X}=0.02, \\
p_{X,Z}=0.02 & p_{Y,Z}=0.01, & p_{Z,Y}=0.01, \\
k_{X,1}=0.2, & k_{Y,1}=0.9, & k_{Z,1}=0.25, \\
k_{X,2}=0.01, & k_{Y,2}=0.005, & k_{Z,2}=0.001, \\
k_{X,3}=0.2, & k_{Y,3}=0.01, & k_{Z,3}=0.001, \\
k_{X,4}=0.0001, & k_{Y,4}=0.1, & k_{Z,4}=0.05, \\
k_{X,5}=0.0001, & k_{Y,5}=0.06. & k_{Z,5}=0.001. \\
k_{X,6}=0.005, & & \\
k_{X,7}=0.005. & &
\end{array}
$$

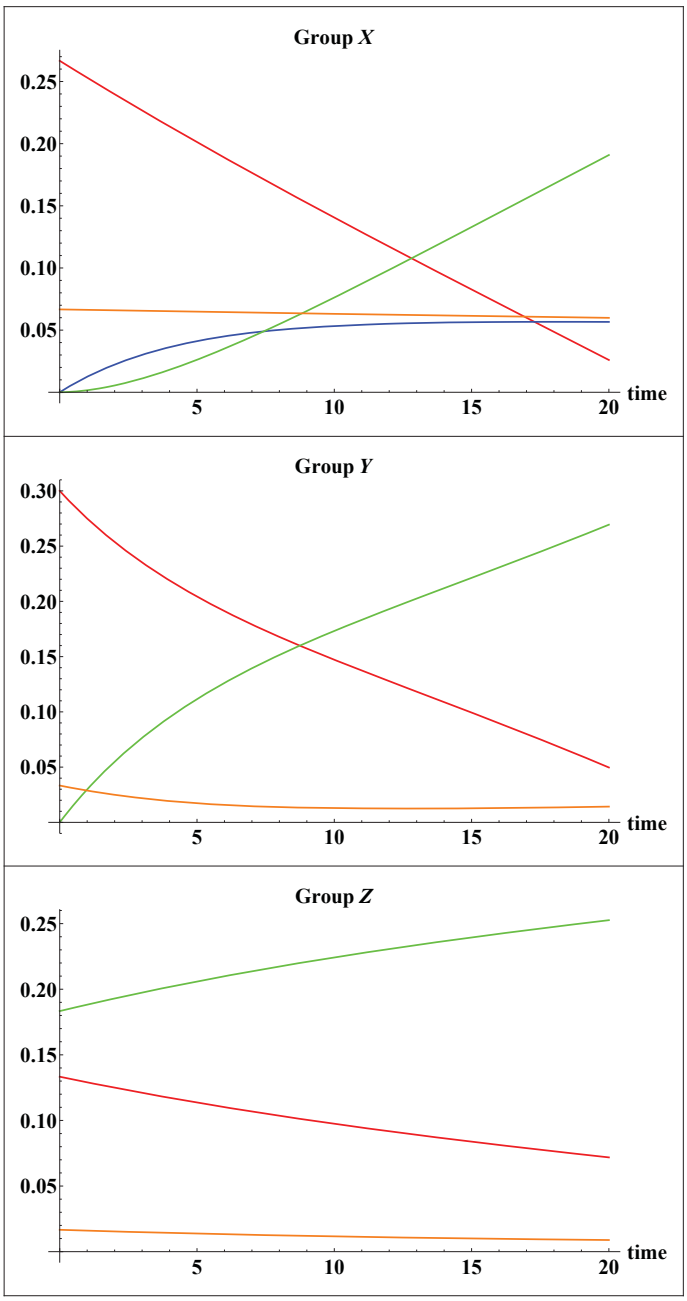


Figure 7.2: Distribution of the computers over the states of the model for each group. Red, blue, green and orange lines show the fraction of not infected, initially infected, infected inactive and infected active computers respectively.

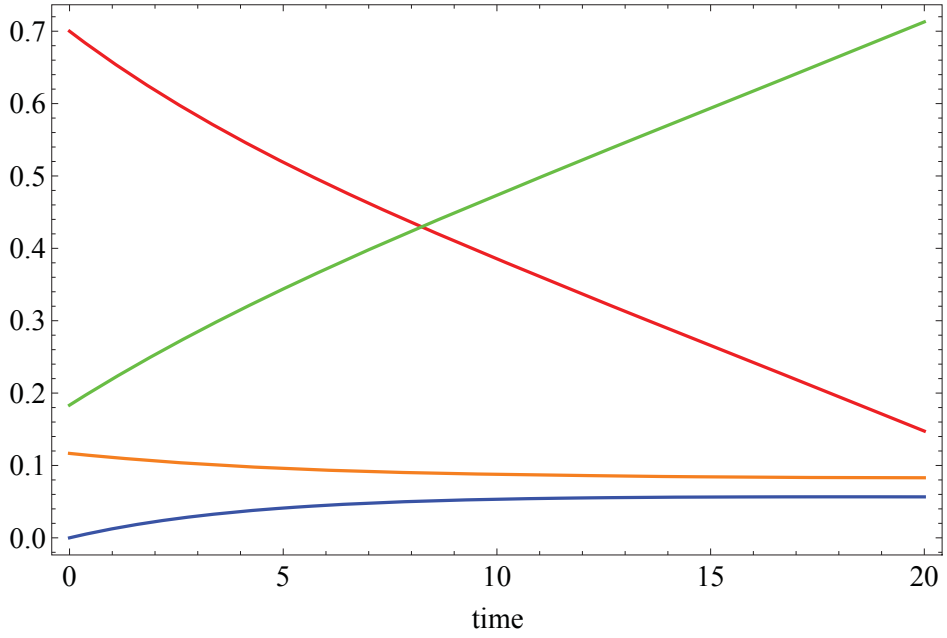


Figure 7.3: The total distribution of the computers over the states of the model for all three groups. Red, blue, green and orange lines show the fraction of not infected, initially infected, infected inactive and infected active computers respectively.

To illustrate the behaviour of the modelled system the ODEs (7.1) can be solved numerically, given the following initial conditions:

$$\overline{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\}).$$

The distribution of objects over three groups in this example is even, where $N_1 = N_2 = N_3 = \frac{N}{3}$. *Figure 7.2 shows the distribution of the objects between the states of the model over time. We obtained these results using Wolfram Mathematica [103]. As one can see, for the given parameters the virus manages to spread over the population.*

In groups X *and* Y *virus spreads comparably fast, despite the fact that the rate of infection* $k_{X,1}$ *in group* X *is* 4.5 *times lower than* $k_{Y,1}$ *in group* Y. *This is due to the high removal rates for active infected computers in group* X, *and the fact that infected machines stay in active state less often.*

In group Z *the spreading is much slower due to the relatively low infection rate* $k_{Z,1}$ *and due to the fact that many computers were already infected, when*

we started to evaluate the system in group Z. Moreover, the communication between groups X and Y is more probable, therefore, group Z is more "isolated", and infection spreads (almost) only within this group.

To evaluate the behaviour of the whole system the total fraction of not infected, infected and inactive, infected and active, and initially infected computers in all three groups was obtained by simply summing up the respective fractions from the overall model state. As one can see on Figure 7.3, around 70% of the whole population was infected after twenty time units, which confirms the fast infection spread patterns within all three groups.

The fraction of the initially infected computers is relatively low since these machines are only found in one group. The fraction of the active infected computers tends to stay low as the viruses are programmed to be as less detectable as possible, which is confirmed by both Figures 7.2 and 7.3.

○

8

Mean-Field Continuous Stochastic Logic

In this chapter we propose a new logic for expressing properties of mean-field models. Before introducing a new logic one has to formally describe the types of questions of interest. Since a mean-field model is used to reason about the fractions of objects in a given state one can express the expected fractions of object, which satisfy a given local property. Given that type of specification, a two-layer logic can be introduced: (i) on the local level the property of a random object is specified; (ii) on the global level the fraction of objects satisfying this local property is expressed. In the following we introduce Mean-Field Continuous Stochastic Logic (MF-CSL) and provide the algorithms for checking MF-CSL properties of a global mean-field model.

The chapter is further organized as follows. Section 8.1 provides syntax and semantics of MF-CSL. Model-checking properties of the local model is discussed in Section 8.2. Section 8.3 describes algorithms for model-checking MF-CSL properties on the global level and illustrates both sets of algorithms using the running example. A summary of the chapter in provided in Section 8.4.

8.1 CSL and MF-CSL

Let us first recall how the properties of a random object can be expressed and checked. As discussed in Section 2.1, the model of one object in a mean-field system is an ICTMC. Therefore, the logic CSL [6] for checking properties of CTMCs can be used for describing properties on the local level. In the following we recall the syntax and semantics of CSL, and explain how CSL properties of the local model can be checked.

Definition 8.1.1 (Syntax of CSL)**.** *Let $p \in [0,1]$ be a real number, $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator, $I \subseteq \mathbb{R}_{\geq 0}$ a non-empty bounded time interval and LAP a set of local atomic propositions with $lap \in LAP$. CSL* ***state formulas*** *Φ are defined by:*

$$\Phi ::= tt \mid lap \mid \neg\Phi \mid \Phi_1 \wedge \Phi_2 \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\phi),$$

where ϕ is a ***path formula*** *defined as:*

$$\phi ::= \chi^I \Phi \mid \Phi_1 \; U^I \; \Phi_2.$$

□

To define the semantics of path formulas we first recall the notion of a path as in [6]. An *infinite path*[1] σ is a sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} ...$ with, for $i \in \mathbb{N}$; $s_i \in S^l$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{Q}_{(s_i,s_{i+1})}(\overline{m}(\sum_{j=0}^{i} t_j)) > 0$ for all i. A finite path σ is a sequence $s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} ...s_{h-1} \xrightarrow{t_{h-1}} s_h$ such that s_h is absorbing, and $\mathbf{Q}_{(s_i,s_{i+1})}(\overline{m}(\sum_{j=0}^{i} t_j)) > 0$ for all $i < h$. For an infinite path σ, $\sigma[i] = s_i$ denotes for $i \in \mathbb{N}$ the $(i+1)st$ state of path σ. The time spent in state s_i is denoted by $\delta(\sigma; i) = t_i$. Moreover, with i the smallest index with $t \leq \sum_{j=0}^{i} t_j$, let $\sigma@t = \sigma[i]$ be the state occupied at time t. For finite paths σ with length $h+1$, $\sigma[i]$ and $\delta(\sigma; i)$ are defined in the way described above for $i < h$ only and $\delta(\sigma; h) = \infty$ and $\delta@t = s_h$ for $t > \sum_{j=0}^{h-1} t_j$. $Path^{\mathcal{M}^l}(s_i, \overline{m})$ is the set of all finite and infinite paths of the local model $\mathcal{M}^l$ that start in state s_i given the state $\overline{m}$ of the overall model $\mathcal{M}^O$, and $Path^{\mathcal{M}^l}(\overline{m})$ includes all (finite and infinite) paths of $\mathcal{M}^l$, which depends on the overall system state $\overline{m}$ (global time) if $\mathcal{M}^O$ is time-inhomogeneous. A probability measure $Pr(\overline{m})$ on set of paths can be defined as in [6].

When the local model is time-homogeneous the semantics of CSL formulas is well known. However, in any non-trivial mean-field model, the transition rates of the local model $\mathcal{M}^l$ are not constant. According to Definition 2.3.1 the rates of the local model $\mathcal{M}^l$ may depend on the state of the global model $\mathcal{M}^O$, which changes with time. There are two ways to formalize this: the local rates depend on (i) the *current state* $\overline{m}$, which changes with time, or (ii) the *global* time. While the first is more intuitive, it does not allow transition rates

[1] Note that $\overline{m}(\sum_{j=0}^{i} t_j)$ is the global state of the overall model $\mathcal{M}^O$ at the time of the i'th transition.

to depend explicitly on global time. For ease of notation, in the following we restrict ourselves to models that only depend on the overall state. Note that our approach can easily be extended to models that explicitly depend on global time and the proposed algorithms can handle both cases.

Since the local model changes with the overall system state, the satisfaction relation for a local state or path depends on the global state $\overline{m}$, therefore, we introduce the satisfaction relation $\models^{\overline{m}}$, which is given as follows:

Definition 8.1.2 (Semantics of CSL)**.** *Satisfaction of CSL state and path formulas for local mean-field model $\mathcal{M}^l$ is given as follows:*

$$
\begin{array}{ll}
s \models^{\overline{m}} tt & \forall s \in S^l, \\
s \models^{\overline{m}} lap & \text{iff } lap \in L(s), \\
s \models^{\overline{m}} \neg\Phi & \text{iff } s \not\models^{\overline{m}} \Phi, \\
s \models^{\overline{m}} \Phi_1 \wedge \Phi_2 & \text{iff } s \models^{\overline{m}} \Phi_1 \text{ and } s \models^{\overline{m}} \Phi_2, \\
s \models^{\overline{m}} \mathcal{S}_{\bowtie p}(\Phi) & \text{iff } \pi^{\mathcal{M}^l}(s, Sat(\Phi, \widetilde{m})) \bowtie p, \\
s \models^{\overline{m}} \mathcal{P}_{\bowtie p}(\phi) & \text{iff } Prob^{\mathcal{M}^l}(s, \phi, \overline{m}) \bowtie p, \\
\sigma \models^{\overline{m}} \chi^I \Phi & \text{iff } \sigma[1] \text{ is defined, and} \\
 & \sigma[1] \models^{\overline{m}(\delta(\sigma,0))} \Phi \wedge \delta(\sigma, 0) \in I, \\
\sigma \models^{\overline{m}} \Phi_1 \; U^I \; \Phi_2 & \text{iff } \exists t' \in I : (\sigma@t' \models^{\overline{m}(t')} \Phi_2) \\
 & \wedge (\forall t'' \in [0, t')(\sigma@t'' \models^{\overline{m}(t'')} \Phi_1)),
\end{array}
$$

where

- *$\overline{m}$ is the occupancy vector (state of the overall model) at time 0, and $\overline{m}(t)$ is the occupancy vector at time t;*
- *s is a state of the local model $\mathcal{M}^l$;*
- *$I \subseteq \mathbb{R}_{\geq 0}$ is a non-empty bounded time interval;*
- *$Sat(\Phi, \overline{m}) = \{s' \in S^l : s' \models^{\overline{m}} \Phi\}$ is the satisfaction set of the CSL formula Φ defined as in* [6]*;*
- *$\pi^{\mathcal{M}^l}(s, Sat(\Phi, \widetilde{m})) = \sum_{s_j \in Sat(\Phi, \widetilde{m})} \pi^{\mathcal{M}^l}(s, s_j, \widetilde{m})$, describes the steady state probability to be in a state from $Sat(\Phi, \widetilde{m})$, where $\widetilde{m}$ is the stationary distribution of the global model, cf.* [6]*;*

- *$Prob^{\mathcal{M}^l}(s, \phi, \overline{m})$ is defined as in* [6]*, i.e., the probability measure of all paths $\sigma \in Path^{\mathcal{M}^l}(s, \overline{m})$ that satisfy ϕ when the system is starting in state s:*

$$Prob^{\mathcal{M}^l}(s, \phi, \overline{m}) = Pr\{\sigma \in Path^{\mathcal{M}^l}(s, \overline{m}) | \sigma \models^{\overline{m}} \phi\}.$$

□

Although $\overline{m}$ is referred to as the $\overline{m}$ vector at time 0, this is only for ease of discussion, without loss of generality. In fact, the $\overline{m}$ argument to $\models$ is just the global state at the time at which one checks the satisfaction relation. This is illustrated in the above definitions for next and until, in which satisfaction at a future time t' is denoted by writing $\models^{\overline{m}(t')}$. Throughout the definition, $\overline{m}(t)$ is the occupancy vector at future time t, which can be obtained by solving the ODEs (2.1) for time t with $\overline{m}$ as initial condition.

Recall that not in all models the mean-field approximation is valid for the steady-state; clearly, the steady-state operator $\mathcal{S}$ should only be used for models in which it is (see Section 2.3).

The properties of interest of the overall mean-field model differ from the properties which can be described by CSL. Therefore, in order to reason at the level of the overall model in terms of fractions of objects we introduce an extra layer "on top of CSL" that defines the logic *CSL for mean-field models*, denoted MF-CSL. The latter is able to describe the behavior of the overall system in terms of the behavior of random local objects.

Definition 8.1.3 (Syntax of MF-CSL)**.** *Let $f \in [0, 1]$ be a real number, and $\bowtie \in \{\leq, <, >, \geq\}$ a comparison operator. MF-CSL formulas Ψ are defined as follows:*

$$\Psi ::= tt \mid \neg\Psi \mid \Psi_1 \wedge \Psi_2 \mid \mathbb{E}_{\bowtie f}(\Phi) \mid \mathbb{ES}_{\bowtie f}(\Phi) \mid \mathbb{EP}_{\bowtie f}(\phi),$$

*where Φ is a **CSL state formula** and ϕ is a **CSL path formula**.*

□

In this definition we introduce three expectation operators: $\mathbb{E}_{\bowtie f}(\Phi)$, $\mathbb{ES}_{\bowtie f}(\Phi)$ and $\mathbb{EP}_{\bowtie f}(\phi)$, with the following interpretation:

- $\mathbb{E}_{\bowtie f}(\Phi)$ denotes whether the fraction of objects that are in a (local) state satisfying a general CSL state formula Φ fulfills $\bowtie f$;

- $\mathbb{ES}_{\bowtie f}(\Phi)$ denotes whether the fraction of objects that satisfy Φ in steady state, fulfills $\bowtie f$;
- $\mathbb{EP}_{\bowtie f}(\phi)$ denotes whether the probability of a random object to satisfy path-formula ϕ fulfills $\bowtie f$.

Let us consider the virus spread example to illustrate the expressive power of MF-CSL for mean-field models.

Example 8.1.4. *In order to express the property that not more than 5% of the computers are* infected_Y *(i.e., these computers belong to the group* Y *and are infected), the following formula is used:*

$$\mathbb{E}_{\leq 0.05}\left(\ infected_Y\right).$$

The percentage of all computers which happen to belong to the group X *and have a probability lower than 10% of going from not infected to active infected state within 3 hours is greater than 40%:*

$$\mathbb{E}_{>0,4}\left(\mathcal{P}_{<0.1}(not\ infected_X\ U^{[0,3]}\ active_X)\right).$$

If one wants to ensure that the probability of a computer to get infected in group Z *within two hours from now is less than 50%:*

$$\mathbb{EP}_{<0.5}(tt\ U^{[0,2]}\ infected_Z).$$

Note that in the formula above the initial state of the individual is not taken into account. If the percentage of not infected computers in the group Z *which will become infected within the next two hours is of interest, the formula has to be changed accordingly:*

$$\Psi = \mathbb{EP}_{<0.5}(not\ infected_Z\ U^{[0,2]}\ infected_Z).$$

If in the long run the system has to have a low probability (less than 2%) for a random computer to be infected in the group Z *we express this as:*

$$\mathbb{ES}_{<0.02}(\ infected_Z).$$

○

The formal definition of the MF-CSL semantics is as follows:

Definition 8.1.5 (Semantics of MF-CSL)**.** *The satisfaction relation $\models$ for MF-CSL formulas and states $\overline{m} = (m_1, m_2, \ldots, m_K) \in S^o$ of the overall mean-field model is defined by:*

$$
\begin{aligned}
&\overline{m} \models tt && \forall\ \overline{m}\ \in\ S^o,\\
&\overline{m} \models \neg\Psi && \text{iff } \overline{m} \not\models \Psi,\\
&\overline{m} \models \Psi_1 \wedge \Psi_2 && \text{iff } \overline{m} \models \Psi_1 \wedge \overline{m} \models \Psi_2,\\
&\overline{m} \models \mathbb{E}_{\bowtie f}(\Phi) && \text{iff } \left(\sum_{j=1}^{K} m_j \cdot Ind_{(s_j \models^{\overline{m}} \Phi)}\right) \bowtie f,\\
&\overline{m} \models \mathbb{ES}_{\bowtie f}(\Phi) && \text{iff } \left(\sum_{j=1}^{K} m_j \cdot \pi^{\mathcal{M}^l}(s_j, Sat(\Phi, \overline{m}))\right) \bowtie f,\\
&\overline{m} \models \mathbb{EP}_{\bowtie f}(\phi) && \text{iff } \left(\sum_{j=1}^{K} m_j \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m})\right) \bowtie f,
\end{aligned}
$$

where $Sat(\Phi, \overline{m})$, $\pi^{\mathcal{M}}(s, Sat(\Phi, \overline{m}))$, $Prob^{\mathcal{M}^l}(s, \phi, \overline{m})$ are defined as in Definition 8.1.2; and $Ind_{(s_j \models^{\overline{m}} \Phi)}$ is an indicator function, which indicates whether a local state $s_j \in S^l$ satisfies formula Φ for a given overall state $\overline{m}$, i.e.,

$$
Ind_{(s_j \models^{\overline{m}} \Phi)} = \begin{cases} 1, & \text{if } s_j \models^{\overline{m}} \Phi,\\ 0, & \text{if } s_j \not\models^{\overline{m}} \Phi. \end{cases}
$$

□

To check an MF-CSL formula at the global level, the local CSL formula has to be checked first, and the results are then used at the global level. As discussed above, the local model $\mathcal{M}^l$ is a time-inhomogeneous CTMC, i.e., transition rates vary with the state of the overall model $\mathcal{M}^O$, which makes model-checking at the local level non-trivial. We discuss model-checking CSL formulas at the local model $\mathcal{M}^l$ in the next section. The algorithms to compute the satisfaction of the MF-CSL formulas at the global model are presented in Section 8.3.

8.2 Checking CSL formula at the local level

In this section we first recall algorithms for model-checking CSL on *time-homogeneous* Markov chains in Section 8.2.1. The CSL operators which require a different approach for the time-inhomogeneous local model are discussed in

Sections 8.2.2, 8.2.3, and 8.2.4. The satisfaction set development for a given CSL formula on a local model $\mathcal{M}^l$ is addressed in Section 8.2.5.

8.2.1 CSL for local mean-field models

All CSL operators can be divided into two groups:

- *time-independent operators*: lap[2], $\neg$, $\wedge$.
- *time-dependent operators*: $\mathcal{P}_{\bowtie p}$ and $\mathcal{S}_{\bowtie p}$, where $\mathcal{P}_{\bowtie p}$ includes path operators X and U.

The CSL operators can be nested according to Definition 8.1.1. Model-checking of the CSL formula is done by building the *parse tree* and performing the satisfaction set development of the individual operators recursively, as described in [6].

All time-independent CSL operators can be checked using the standard methods (see [6]) due to the independence of the results on time. Therefore, model-checking these operators is not addressed any further.

Properties that include the Next operator are rarely used in a real-life scenarios, therefore, we omit the discussion of such formulas and refer to [18] for algorithms for checking the CSL Next operator on the local time-inhomogeneous CTMC. A discussion on the steady-state operator on the local mean-field model $\mathcal{M}^l$ will be provided in Section 8.2.4.

Since the main challenge lies in model-checking the time-dependent operators, and in the context of this thesis in checking the until formula, let us recall the interval until formula $\Phi_1 U^{[t_1,t_2]} \Phi_2$ for an arbitrary time-homogeneous CTMC $\mathcal{M}$, as in [6].

For model-checking such a CSL formula, we need to consider all possible paths, starting in a Φ_1-state at the current time and reaching a Φ_2-state during the time interval $[t_1, t_2]$ by only visiting Φ_1-states on the way. We can split such paths in two parts:

- the first part models the path from the starting state s to a Φ_1-state s_1;
- the second part models the path from s_1 to a Φ_2-state s_2 only via Φ_1-states.

[2]Note that the atomic property could be defined as a time-dependent operator, however, according to Definition 2.1.2, it belongs to the time-independent group.

In the first part of the path, we only proceed along Φ_1-states, thus if any state that does not satisfy Φ_1 is visited the path will not satisfy the formula, therefore, these states can be made absorbing. As we want to reach a Φ_2-state via Φ_1-states in the second part, we can make all states that do not fulfill Φ_1 and satisfy Φ_2 absorbing. We therefore need two transformed CTMCs[3]: $\mathcal{M}[\neg\Phi_1]$ and $\mathcal{M}[\neg\Phi_1 \vee \Phi_2]$, where $\mathcal{M}[\neg\Phi_1]$ is used in the first part of the path, for $t \in [0, t_1]$, and $\mathcal{M}[\neg\Phi_1 \vee \Phi_2]$ is used in the second, for $t \in [t_1, t_2]$.

In order to calculate the probability for such a path, we accumulate the multiplied transition probabilities for all triples (s, s_1, s_2), where $s_1 \models \Phi_1$ is reached before time t_1, and $s_2 \models \Phi_2$ is reached within time $t_2 - t_1$ (note that this is only applicable for time-homogeneous CTMCs), i.e.,

$$\begin{aligned}&\mathrm{Prob}^{\mathcal{M}}(s, \Phi_1 U^{[t_1,t_2]} \Phi_2) = \\ &\sum_{s_1 \models \Phi_1} \sum_{s_2 \models \Phi_2} \pi_{s,s_1}^{\mathcal{M}[\neg\Phi_1]}(t_1) \cdot \pi_{s_1,s_2}^{\mathcal{M}[\neg\Phi_1 \vee \Phi_2]}(t_2 - t_1).\end{aligned} \tag{8.1}$$

Hence, CSL until formulas can be checked as a combination of two reachability problems, as shown in Equation (8.1), namely $\pi_{s,s_1}^{\mathcal{M}[\neg\Phi_1]}(t_1)$ and $\pi_{s_1,s_2}^{\mathcal{M}[\neg\Phi_1 \vee \Phi_2]}(t_2 - t_1)$ that can be computed by performing a transient analysis on the transformed CTMCs. Note that Equation (8.1) is valid for $t_1 > 0$ and $t_2 > 0$, if $t_1 = 0$ the first part of the path ($\pi_{s,s_1}^{\mathcal{M}[\neg\Phi_1]}(t_1)$) can be omitted.

In the following we discuss the methods for checking until formulas for a local mean-field model. Since the results of the model-checking procedure for an ICTMCs may depend on time we first provide the algorithms for checking the formula for a given moment of time, namely, $t = 0$. And afterwards the calculation of the time-dependent probability for an until formula to hold will be addressed. We start the description of the algorithms with the single (non-nested) until operator in Section 8.2.2, then we proceed with the more involved nested case in Section 8.2.3 .

8.2.2 Single until

Due to the time-inhomogeneity of the local mean-field model, standard methods for model-checking timed operators can not be used. Recently, model-checking algorithms for a time-bounded fragment of CSL were proposed in

[3]We reuse the notation for a modified CTMC from [6], where the formula in brackets refers to the set of states which are made absorbing.

[18]. We adapt the model-checking algorithms presented in [18] for use on the local model $\mathcal{M}^l$ of a mean-field model. In [18] explicit dependence of the local rates on time was chosen, while in MF-CSL the dependence on the current global state $\overline{m}$ is of interest. Therefore, the approach from [18] has to be modified in order to reason about the newly introduced satisfaction relation $\models^{\overline{m}}$.

In the following we discuss model-checking of non-nested CSL interval until formulas (where the sub-formulas Phi_1 and Phi_2 are atomic properties and logical combinations) on a time-inhomogeneous CTMC. This can occur in MF-CSL when using any of the three expectation operators:

$$\mathbb{E}_{\bowtie f}(P_{\bowtie p}(\Phi_1 U^{[t_1,t_2]} \Phi_2)),$$

$$\mathbb{ES}_{\bowtie f}(P_{\bowtie p}(\Phi_1 U^{[t_1,t_2]} \Phi_2)),$$

$$\mathbb{EP}_{\bowtie f}(\Phi_1 U^{[t_1,t_2]} \Phi_2).$$

Note that due to the restriction to single until formulas here, the validity of Φ_1 and Φ_2 does not depend on time.

The core idea of CSL model-checking of until formulas as explained in the previous section remains unchanged for time-inhomogeneous CTMCs. However, due to time-inhomogeneity it is not enough to only consider the time *duration*; the *exact time* at which the system is observed must be taken into account. Hence, we add time t' to the notation of a time-inhomogeneous reachability probability $\pi^{\mathcal{M}^l}_{s,s_1}(t', T)$ to denote that we start in state s at time t' and have to reach state s_1 within $T - t'$ time units.

The until formula $\Phi_1 U^{[t_1,t_2]} \Phi_2$ is then again addressed by computing two reachability problems on the transformed local models $\mathcal{M}^l[\neg\Phi_1]$ and $\mathcal{M}^l[\neg\Phi_1 \vee \Phi_2]$, respectively; the results are combined similarly to 8.1:

$$\begin{aligned} &\mathrm{Prob}^{\mathcal{M}^l}(s, \Phi_1 U^{[t_1,t_2]} \Phi_2, \overline{m}) = \\ &\sum_{s_1 \models^{\overline{m}} \Phi_1} \sum_{s_2 \models^{\overline{m}} \Phi_2} \pi^{\mathcal{M}^l[\neg\Phi_1]}_{s,s_1}(0, t_1) \cdot \pi^{\mathcal{M}^l[\neg\Phi_1 \vee \Phi_2]}_{s1,s2}(t_1, t_2 - t_1). \end{aligned} \tag{8.2}$$

Note that the first reachability probability always has zero as a starting time since we assume that $\overline{m}$ is the starting distribution[4] at time $t = 0$.

[4]For models which depend on global time this needs to be $\pi^{\mathcal{M}^l[\neg\Phi_1]}_{s,s_1}(t_0, t_1)$, where t_0 indicates that the system is observed at global time t_0.

8.2.2.1 Reachability probability at a given time

In the following we describe how to compute the reachability probability $\pi^{\mathcal{M}^l}_{s,s_1}(t', T)$ for an arbitrary modified local model and a given occupancy vector $\overline{m}$ that is observed at time $t = 0$. Note that this can be used for both modified local models $\mathcal{M}^l[\neg\Phi_1]$ or $\mathcal{M}^l[\neg\Phi_1 \vee \Phi_2]$, as needed in (8.2).

Let $\Pi'(t', t' + T)$ be the probability matrix of the modified local model, where $\Pi'_{s,s_1}(t', t'+T)$ is the probability of being in state s_1 at time $t'+T$, given that we were in state s at time t'. In order to find this transient probability, the forward Kolmogorov equation is solved with the identity matrix as initial condition, i.e., $\Pi(t', t' + 0) = \mathbb{1}$,

$$\frac{d\Pi'(t', t' + T)}{d(T)} = \Pi'(t', t' + T) \cdot Q'(\overline{m}(t' + T)), \tag{8.3}$$

where $Q'(\overline{m}(t' + T))$ is the rate matrix of the modified local model.

Due to the modifications made in the local model, the transient probability matrix $\Pi'(t', t' + T)$ contains the reachability probabilities $\pi^{\mathcal{M}^l}_{s,s_1}(t', T)$ for all possible states s and s_1.

Once the reachability probabilities $\pi^{\mathcal{M}^l[\neg\Phi_1]}_{s,s_1}(0, t_1)$ and $\pi^{\mathcal{M}^l[\neg\Phi_1\vee\Phi_2]}_{s_1,s_2}(t_1, t_2 - t_1)$ have been calculated using (8.3), the probability $\text{Prob}^{\mathcal{M}^l}(s, \Phi_1\, U^{[t_1,t_2]}\Phi_2, \overline{m})$ can be computed according to Equation (8.2), which allows to check the satisfaction relation for a given occupancy vector $\overline{m}$ according to Definition 8.1.2.

8.2.2.2 Reachability probability as a function of time

Keeping the occupancy vector $\overline{m}$ and time t' as initial conditions of the mean-field model, the validity of a CSL formula may change when it is evaluated at a later moment in time $t \in [t', \theta]$, where θ is a predefined upper bound of the evaluation time. In the following we discuss how the reachability probability $\pi^{\mathcal{M}^l}_{s,s_1}(t, T)$ depends on its evaluation time t while T is kept constant.

First, the probability matrix $\Pi'(t', t' + T)$ is derived according to Equation (8.3), where t' is predefined. Next, the ODE describing the dependence of the transient probability on time t is derived by combining the forward and backward Kolmogorov equations using the chain rule:

$$\begin{aligned}\frac{d\Pi'(t, t + T)}{dt} &= -Q'(\overline{m}(t)) \cdot \Pi'(t, t + T) \\ &\quad + \Pi'(t, t + T) \cdot Q'(\overline{m}(t + T)).\end{aligned} \tag{8.4}$$

Finally, the time-dependent probability matrix $\Pi'(t, t+T)$ can be obtained by solving Equation (8.4) with initial condition $\Pi'(t', t'+T)$. This can be done either analytically or numerically, e.g., with the tool Wolfram Mathematica [103] as used in this thesis.

As mentioned above, the validity of a local CSL until formula may change when the system is evaluated at a later moment in time t due to the changing overall state. The time-dependent probability

$$\text{Prob}^{\mathcal{M}^l}(s, \Phi_1 U^{[t_1,t_2]} \Phi_2, \overline{m}, t)$$

to take a $(\Phi_1 U^{[t_1,t_2]} \Phi_2)$-path in $\mathcal{M}^l$, when starting in state s at time t, can be computed similar to Equation (8.2), by taking into account the time t that has elapsed since the initial condition $\overline{m}$ was observed:

$$\begin{aligned}&\text{Prob}^{\mathcal{M}^l}(s, \Phi_1 U^{[t_1,t_2]} \Phi_2, \overline{m}, t) = \\ &\sum_{s_1 \models^{\overline{m}} \Phi_1} \sum_{s_2 \models^{\overline{m}} \Phi_2} \pi^{\mathcal{M}^l[\neg\Phi_1]}_{s,s_1}(t, t+t_1) \cdot \pi^{\mathcal{M}^l[\neg\Phi_1 \vee \Phi_2]}_{s_1,s_2}(t+t_1, t+t_2-t_1). \end{aligned} \tag{8.5}$$

Note that using Kolmogorov equations for solving reachability problems on the local models $\mathcal{M}^l$ is efficient due to the fact that the local state space is usually quite small (see [18]).

8.2.3 Nested until

The method described in the previous section can be used when both Φ_1 and Φ_2 do not depend on time, i.e., when we do not have nested until formulas. In the following let us consider the following nested until formula:

$$\mathcal{P}_{\bowtie f}\left(\Phi_1 U^{[t_0,T]}\left(\mathcal{P}_{\bowtie p}(\Phi_2 U^{[t_1,t_2]} \Phi_3)\right)\right).$$

In order to evaluate a nested until formula the corresponding parse tree is built, as in the time-homogeneous case, and the satisfaction sets of all sub-formulas need to be computed. The satisfaction set of the sub-formula $\Gamma = \mathcal{P}_{\bowtie q}(\Phi_2 U^{[t_1,t_2]} \Phi_3)$, however, changes with time. To compute this set for a given $t \in [t_0, T]$, first $\text{Prob}^{\mathcal{M}^l}(s, \Phi_2 U^{[t_1,t_2]} \Phi_3, \overline{m}, t)$ needs to be computed for all $s \in \mathcal{S}^l$ according to Equation (8.5). Then the time-dependent satisfaction set of Γ is given as:

$$Sat(\Gamma, \overline{m}, t) = \{s \in \mathcal{S}^l \mid \text{Prob}^{\mathcal{M}^l}(s, \Gamma, \overline{m}, t) \bowtie p\}. \tag{8.6}$$

More details on the computation of the satisfaction set are provided in Section 8.2.5. Having computed this set then in principle allows to model-check the nested until formula as a combination of two reachability problems, as in Equation (8.2). When replacing Φ_2 by Γ in this equation, it becomes clear that model-checking a nested until formula requires computations on the modified local model $\mathcal{M}^l[\neg\Phi_1 \vee \Gamma]$. This is far from trivial, since the satisfaction set of Γ is time-dependent which results in a modified local model that also changes with time.

Therefore, similar to the single until we provide first the algorithm for computing the reachability probability for a given time $t = 0$ (see Section 8.2.3.1), then in Section 8.2.3.2 we discuss how to calculate the time-dependent reachability probability for $t \in [t', \Theta]$. Note, however, that in this case the reachability problems are solved for the time-varying sets of safe and goal states.

8.2.3.1 Time-varying set reachability

In the following, we describe how in general a time-bounded reachability problem $\pi^{\mathcal{M}[\neg\Gamma_1 \vee \Gamma_2]}(t', T)$ with time-dependent formulas Γ_1 and Γ_2 can be evaluated, similar to [18]. Note that t' indicates the starting time and T the duration of the time interval we are interested in.

At first we have to determine the so-called discontinuity points, i.e., the time points $T_0 = t' \leq T_1 \leq T_2 \leq \cdots \leq T_k \leq T_{k+1} = T + t'$, where at least one of the satisfaction sets $Sat(\Gamma_1)$ or $Sat(\Gamma_2)$ changes. After that, we can integrate separately on each time interval (T_i, T_{i+1}) for $i = 0, ..., k$.

To ensure that only Γ_1 states are visited before a Γ_2 state is reached, we need to modify the local model $\mathcal{M}^l$ for each time interval, as follows:

- A new goal state s^* which remains the same for all time intervals is introduced.
- All $\neg\Gamma_1$ and Γ_2 states are made absorbing.
- All transitions leading to Γ_2 states are redirected to the new state s^*.

Given the modified local model $\overline{\mathcal{M}^l}$, the transient probability matrix for each time interval $\overline{\Pi'}(T_i, T_{i+1})$ is found using the forward Kolmogorov equation, according to Equation (8.3).

Upon "jumps" between time intervals (T_{i-1}, T_i) and (T_i, T_{i+1}) it is possible that a state that satisfied Γ_1 in the previous time interval does not satisfy Γ_1 in

the next. In this case the probability mass in this state is lost, since this path does not satisfy the reachability problem anymore. In the case that a state remains satisfying Γ_1 or a Γ_1-state is turned into a Γ_2-state, the probability mass has to be carried over to the next time interval. This is described by a matrix $\zeta(T_i)$ of size $(|S^l|+1)\times(|S^l|+1)$ constructed in the following way:

- For each state $s \in S^l$ which satisfies $\neg\Gamma_1 \wedge \neg\Gamma_2$ before and after T_i, we set $\zeta(T_i)_{s,s} = 1$.
- For each state $s \in S^l$ which satisfies $\neg\Gamma_1 \wedge \neg\Gamma_2$ before T_i and Γ_2 after T_i, we set $\zeta(T_i)_{s,s*} = 1$.
- For the new goal state s^* the entry always equals one, that is, $\zeta(T_i)_{s*,s*} = 1$.
- All other elements of $\zeta(T_i)$ are set to 0.

The probability to reach a Γ_2 state before time T has passed when starting in a $\neg\Gamma_2$ state at time t' is then given by the matrix $\Upsilon(t', t'+T)$:

$$\Upsilon(t', t'+T) = \overline{\Pi'}(t', T_1)\cdot\zeta(T_1)\cdot\overline{\Pi'}(T_1, T_2)\cdot\zeta(T_1)\ldots\zeta(T_k)\cdot\overline{\Pi'}(T_k, t'+T). \quad (8.7)$$

The probability to reach the goal state s^* is unconditioned on the starting state by adding 1 for all Γ_2-states, i.e., for the cases, when the starting state satisfied Γ_2:

$$\pi^{[\neg\Gamma_1\vee\Gamma_2]}_{s,s^*}(t', t'+T) = \Upsilon_{s,s^*}(t', t'+T) + \mathbb{1}\{s \in Sat(\Gamma_2, \overline{m}, t')\}. \quad (8.8)$$

The way of calculating reachability probabilities as described above is based on the method proposed in [18]. The only difference lies in the way of handling the probability of reaching the goal state. In [18] the state space is doubled and all goal states are considered separately, which increases the computational complexity and does not add any extra information. In our approach, only one extra state is added in order to simplify the calculations.

Another way of reducing the computational complexity would be to lump all Γ_2-states and all $\neg\Gamma_1$-states in the model itself. However, in the case when the satisfaction sets of Γ_1 and Γ_2 change with time, the state space of the modified local model will change at each discontinuity point, which would require a more complicated calculation of (8.7) and (8.8).

8.2.3.2 Reachability probability as a function of time

To evaluate a nested until formula for varying points in time $t \in [t'; \theta]$, in the following we adapt the two components of Equation (8.8) to allow for varying evaluation points.

Since only the first and the last component of $\Upsilon(t', t'+T)$ depend on t', we rewrite Equation (8.7) for ease of notation:

$$\Upsilon(t', t'+T) = \overline{\Pi'}(t', T_1) \cdot \Lambda(T_1, T_k) \cdot \overline{\Pi'}(T_k, t'+T), \tag{8.9}$$

where $\Lambda(T_1, T_k) = \zeta(T_1) \cdot \overline{\Pi'}(T_1, T_2) \cdot \cdots \cdot \overline{\Pi'}(T_{k-1}, T_k) \cdot \zeta(T_k)$.

To explicitly take into account the change of $\Upsilon(t, t+T)$ with time, the following differential equation is constructed using forward and backward Kolmogorov equations:

$$\frac{d\Upsilon(t, t+T)}{dt} = -\overline{Q}(t) \cdot \Upsilon(t, t+T) + \Upsilon(t, t+T) \cdot \overline{Q}(t+T), \tag{8.10}$$

where $\overline{Q}(t)$ is the rate matrix of $\overline{\mathcal{M}^l}$. Then in order to calculate $\Upsilon(t, t+T)$ Equation (8.10) is solved for $t \in [t', \theta]$. Note that when during the integration either t or $t+T$ reaches a discontinuity point T_i, the computation has to be reset, namely, $\Upsilon(t, t+T)$ has to be recomputed; and the computation is resumed and ODE (8.10) is used until the next discontinuity point.

The complete algorithm for this is as follows:

1. All the discontinuity points $t' = T_0 < T_1 < \cdots < T_k < T_{k+1} = \theta + T$ are found. In addition the points $T_i' = T_i + T$ are considered for all $i = 0, 1, \ldots, k$ and $pre(T_i')$ is defined as the largest T_j preceding T_i' and $post(T_i')$ is defined as the smallest T_j after T_i'.

2. The probability matrices $\overline{\Pi'}(T_i, T_{i+1})$ and $\overline{\Pi'}(pre(T_i'), T_i')$ are calculated for all $i \leq k$ using the forward Kolmogorov equation (8.3).

3. For each discontinuity point T_i, the matrix $\zeta(T_i)$ is computed as defined in Section 8.2.3, for all $i = 1, 2, .., k$.

When integrating Equation (8.10) for all $t \in [t', \theta]$, due to the discontinuity points, we may not have a single solution $\Upsilon(t, t+T)$ that can be used for all values of t. For the intervals between discontinuity points, $\Upsilon(t, t+T)$ is given by the solution of ODE (8.10). At each discontinuity point, $\Upsilon(T_i, T_i+T)$ is

recalculated and the integration is resumed until the next discontinuity point is reached.

At the first discontinuity point, i.e., $T_0 = t'$, $\Upsilon(t', t'+T)$ is given by Equation (8.7), and for all $T_0 \leq t \leq t^* = \min\{T_1, post(T_0') - T\}$ $\Upsilon(t, t+T)$ is given by the solution of ODE (8.10). Then $\Upsilon(t^*, t^*+T)$ is recalculated, depending on whether t or $t+T$ hits a discontinuity point T_i, as follows:

- If $t^* = T_i$, then $\Upsilon(T_i; T_i')$ has to be recomputed as follows:

$$\Upsilon(T_i, T_i') = \overline{\Pi'}(T_i, T_{i+1}) \cdot \Lambda(T_{i+1}, pre(T_i')) \cdot \overline{\Pi'}(pre(T_i'), T_i').$$

  The integration of the ODE (8.10) is resumed and $\Upsilon(t, t+T)$ is calculated for $T_i \leq t \leq \min\{T_{i+1}, post(T_i') - T_i' + T_i\}$ until the next discontinuity point is reached.

- If $t^* + T = T_i$, then to account for the changes at the discontinuity point $\Upsilon(T_i - T; T_i)$ has to be multiplied on the right by $\zeta(T_i)$. The integration of the ODE (8.10) is resumed and $\Upsilon(t, t+T)$ is calculated for $T_i - T \leq t \leq \min\{post(T_i - T), T_{i+1} - T\}$ until the next discontinuity point is reached.

This procedure is repeated until the time bound of the evaluation $t = \theta$ is reached. Note that the number of the discontinuity points is limited by the depth of nesting of the until-operator, which is low in practice, therefore the numerical complexity of the algorithm is not a practical issue.

Finally, the time-dependent reachability probability can be computed as follows:

$$\pi_{s,s^*}^{[\neg\Gamma_1 \vee \Gamma_2]}(t, t+T) = \Upsilon_{s,s^*}(t, t+T) + \mathbb{1}\{s \in Sat(\Gamma_2, \overline{m}, t)\}. \tag{8.11}$$

Recall that the second component of this equation is also time-dependent and has to be reconsidered at each discontinuity point.

8.2.4 Steady-state operator

In the following we discuss how to model-check the steady-state operator $\mathcal{S}_{\bowtie p}(\Phi)$ for a given overall distribution $\overline{m}$. Recall that this is only meaningful for mean-field models which are known to be also valid for the long run behaviour (see, e.g., [71]).

Since the long run behavior of the individual object reflects the behavior of the whole model, the stationary distribution $\widetilde{m}$ of the overall model can be used as the steady-state distribution of the local model $\pi^{\mathcal{M}^l}(s, s_j, \widetilde{m})$. Therefore, given the satisfaction set $Sat(\Phi, \widetilde{m})$ of the formula Φ, which can be found as will be explained in the next section, the steady state operator can be checked according to Definition 8.1.2, as follows:

$$\pi^{\mathcal{M}}(s, Sat(\Phi, \widetilde{m})) = \sum_{s_j \in Sat(\Phi, \widetilde{m})} \pi^{\mathcal{M}^l}(s, s_j, \widetilde{m}) = \sum_{s_j \in Sat(\Phi, \widetilde{m})} \widetilde{m}_j. \tag{8.12}$$

The steady-state probability does not depend on time, therefore, the satisfaction relation on the steady-state operator does not depend on time and the probability of the formula to hold remains constant at all times:

$$\pi^{\mathcal{M}}(s, Sat(\Phi, \widetilde{m}), t) = \sum_{s_j \in Sat(\Phi, \widetilde{m})} \widetilde{m}_j. \tag{8.13}$$

8.2.5 Satisfaction set of the local model $\mathcal{M}^l$

The satisfaction set of a CSL formula on a time-inhomogeneous CTMC is constructed using a parse tree [6], as in the time-homogeneous case. First the satisfaction sets of the sub-formulas have to be developed. For time-independent operators the procedure is the same as explained in [6] and the satisfaction set does not change with time, therefore we do not discuss these operators here. For time-dependent operators both the satisfaction set for a given time t' and the time-dependent satisfaction set for a given time interval $[t', \theta]$ can be computed, as follows.

For a given time t' and the overall system state $\overline{m}$ we obtain the satisfaction set of the probability operator:

$$Sat(\mathcal{P}_{\bowtie p}(\phi), \overline{m}) = \{s \mid Prob^{\mathcal{M}^l}(s, \phi, \overline{m}) \bowtie p\}, \tag{8.14}$$

where $Prob^{\mathcal{M}^l}(s, \phi, \overline{m})$ is given by Equation (8.2).

According to Equation (8.12) the satisfaction set of the steady-state operator is as follows:

$$Sat(\mathcal{S}_{\bowtie p}(\Phi), \overline{m}) = \{s \mid \sum_{s_j \in Sat(\Phi, \overline{m})} \widetilde{m}_j \bowtie p\}. \tag{8.15}$$

The time-dependent satisfaction set of until formula is developed similarly, but Equation (8.5) is used:

$$Sat(\mathcal{P}_{\bowtie p}(\phi), \overline{m}, t) = \{s \mid Prob^{\mathcal{M}^l}(s, \phi, \overline{m}, t) \bowtie p\}, \tag{8.16}$$

The steady-state does not change with time, therefore:

$$Sat(\mathcal{S}_{\bowtie p}(\Phi), \overline{m}, t) = \{s \mid \sum_{s_j \in Sat(\Phi, \overline{m}, t)} \widetilde{m}_j \bowtie p\}. \tag{8.17}$$

8.2.6 Run time

In this section we briefly discuss the complexity of the numerical (approximate) algorithms for checking CSL properties of a local mean-field model, presented above. The full discussion on the decidability and convergence can be found in [19], therefore, we will not provide it here.

As was discussed above, the presented algorithms are computationally demanding, due to the need of iterative computations of the solution of Kolmogorov equations. However, in practice, when using modern tools, e.g., Wolfram Mathematica [103], these computations take a fraction of second to couple of seconds. It is a very large improvement compared to applying the statistical model checking to a large population of interaction objects.

In Table 1 of [19] a comparison of the running times of the algorithms presented above to stochastic model-checking for different population sizes has been presented. The local CSL model-checking appears to be between 500 to 1000 times faster ($0.05-0.1$ sec. vs $5-100$ sec), than statistical model-checking for a population of 10^3 objects, with a negligible loss in accuracy. Moreover, the running time of the CSL model-checking on a local ICTMC does not depend on a size of population. Therefore, a very large population can be analysed using the above algorithms while the statistical model-checkers might fail to converge if the size of the population is too large.

8.3 MF-CSL model-checking at the global level

Model-checking MF-CSL formula consists of two parts: checking the satisfaction relation for individual states and developing the *satisfaction set* of a given MF-CSL formula Ψ. Both parts include CSL model-checking on the local level, as has been discussed in Section 8.2. In this section we proceed with

the satisfaction relation and show how to build the satisfaction set of MF-CSL operators on the overall model $\mathcal{M}^O$.

8.3.1 Satisfaction for individual states

Given the results on the local level, checking individual states of the global model can be done by straight-forward application of Definition 8.1.5. We briefly discuss checking the satisfaction relation between a given occupancy vector $\overline{m}$ and expectation operators in the following.

For the expectation operator $\mathbb{E}_{\bowtie f}(\Phi)$ the satisfaction set of the local CSL formula is used in essence to define the indicator function, which allows to check the following inequality:

$$\left(\sum_{j=1}^{K} m_j \cdot Ind_{(s_j \models^{\overline{m}} \Phi)}\right) \bowtie f.$$

If the inequality holds for a given occupancy vector $\overline{m}$, this occupancy vector satisfies the expectation formula.

For the expected probability operator $\mathbb{EP}_{\bowtie f}(\phi)$ we check

$$\left(\sum_{j=1}^{K} m_j \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m})\right) \bowtie f,$$

where the probability $\text{Prob}^{\mathcal{M}^l}(s, \Phi, \overline{m})$ is computed as described in Section 8.2.

Since the long run behavior of an individual object reflects the behavior of the whole model, checking the satisfaction of a steady-state MF-CSL formula $\mathbb{ES}_{\bowtie f}(\Phi, \overline{m})$ simplifies to the following expression:

$$\sum_{j=1}^{K} \pi^{\mathcal{M}^l}(s_j, Sat(\Phi, \overline{m}, t)) \cdot m_j = \pi^{\mathcal{M}^l}(s, Sat(\Phi, \overline{m}, t)).$$

Hence, the expected steady-state operator on the global level mirrors the steady-state operator on the local level, when the steady-state exists (see [71]). Therefore the stationary distribution $\widetilde{m}$ of the global model is used as steady-state distribution of the local model and the expected steady-state operator is

checked using Equation (8.12), that is,

$$\left(\sum_{s_j \in Sat(\Phi, \overline{m}, t)} \widetilde{m}_j \right) \bowtie f.$$

The occupancy vector satisfies the given steady-state formula if the above inequality holds.

8.3.2 Satisfaction (time validity) set development

Traditionally, the satisfaction set of a given formula is the set of states of the model which satisfies a given formula. In the context of MF-CSL model-checking, this would result in a set of all occupancy vectors $\overline{m}$ satisfying a given MF-CSL formula. While such a set can be built for time-independent MF-CSL operators, it is not a trivial task for time-dependent operators, since the model-checking on the local model $\mathcal{M}^l$ would have to be done without knowing the initial conditions, i.e., the occupancy vector. Theoretically speaking, in some cases the analytical solution of the ODEs (2.1) can be used, however, in practice these solutions are not easy (or even impossible) to find. Furthermore, the procedure of model-checking time-dependent CSL operators often includes numerical evaluation, therefore, using the general solution seems not feasible.

When, for example, discretization of the infinite state space of the overall mean-field model is done, and the validity of a given formula is checked in each discrete point, the full satisfaction set can be approximated. However, the complexity of such an approximation grows with the number of local states, and the fineness of the grid. Moreover, even though the algorithms for checking CSL properties of the local model are indeed more efficient, than, e.g., statistical model-checking (see Section 8.2.6), this gain in efficiency might not be enough for approximating the full satisfaction set.

Despite the above arguments, once the initial occupancy vector is fixed, the time instances where a MF-CSL formula holds when evaluated at later times $t \in [0, \theta]$ can be found. From this point of view, the conditional satisfaction set or the *Time Validity Set* of the MF-CSL formula is defined as follows.

Definition 8.3.1 (Time Validity Set)**.** *Let $\theta > 0$ be a predefined time bound, Ψ be an MF-CSL formula, and initial conditions of the mean-field model $\mathcal{M}^O$ be an occupancy vector $\overline{m}(0) = \overline{m}$. The* ***Time Validity Set (TVS)*** *contains*

all time intervals, where Ψ *holds, for a given* $\overline{m}$*, and* θ*:*

$$TVS(\Psi, \overline{m}, \theta) = \{t \in [0, \theta] \mid \overline{m}(t) \models \Psi\}.$$

□

In the following, we discuss how to develop the time validity set for the MF-CSL expectation operators. Table 8.1 summarizes the equations which define this set for the expectation operators. The algorithms for calculating the corresponding TVS are developed in the remainder of this section.

For some of the MF-CSL operators the development of the TVS is straightforward:

$$\begin{array}{lcl} \mathrm{TVS}(tt, \overline{m}, \theta) & = & [0, \theta], \\ \mathrm{TVS}(\neg\Psi, \overline{m}, \theta) & = & [0, \theta] \setminus \mathrm{TVS}(\Psi, \overline{m}, \theta), \\ \mathrm{TVS}(\Psi_1 \wedge \Psi_2, \overline{m}, \theta) & = & \mathrm{TVS}(\Psi_1, \overline{m}, \theta) \cap \mathrm{TVS}(\Psi_2, \overline{m}, \theta). \end{array}$$

To calculate TVS of the expectation operator the set of inequalities has to be built using Definition 8.1.5 as presented in Table 8.1. In the following we discuss how these inequalities are built for each operator separately.

A set of inequalities defines the constraints on the validity set of the expectation operator $\mathrm{TVS}(\mathbb{E}_{\bowtie f}(\Phi), \overline{m}, \theta)$. To construct these inequalities one has to find the satisfaction set $Sat(\Phi, \overline{m}, t)$ of the local CSL state formula Φ (see Section 8.2.5). When the time dependent satisfaction set of the CSL formula Φ is found, the following steps have to be taken in order to find $\mathrm{TVS}(\mathbb{E}_{\bowtie f}\Phi, \overline{m}, \theta)$:

1. Discontinuity points $0 < \tau_1 < \tau_2 < \cdots < \tau_h < \theta$, where at least one state of the local model leaves or enters the satisfaction set $Sat(\Phi, \overline{m}, t)$, have to be found.

2. The indicator function $Ind^{[\tau_i;\tau_{i+1}]}_{s_j \models^{\overline{m}}\Phi}(\overline{m}, t)$, which shows whether a local state s_j satisfies formula Φ, is then defined on each time interval $[\tau_i; \tau_{i+1}]$.

3. The inequality is constructed at each time interval $[\tau_i; \tau_{i+1}]$ according to Definition 8.1.5, as shown in the first row of Table 8.1.

4. For each time interval, the constraints on the occupancy vector $\overline{m}(t)$ are found by solving the respective inequalities, where $t \in [\tau_i; \tau_{i+1}]$.

MF-CSL operator	Set of inequalities to build TVS$(\Psi, \overline{m}, \theta)$	Computation on $\mathcal{M}^l$
$\Psi = \mathbb{E}_{\bowtie p}(\Phi)$	$\left(\sum_{j=1}^{K} m_j(t) \cdot Ind_{s_j \models^{\overline{m}} \Phi}^{[\tau_i;\tau_{i+1}]}(\overline{m}, t)\right) \bowtie f, \quad \forall \tau_i \in [0, \theta]$	$Sat(\Phi, \overline{m}, t)$
$\Psi = \mathbb{ES}_{\bowtie p}(\Phi)$	$\left(\sum_{s_j \in Sat(\Phi, \widetilde{m}, t)} m_j(t) \cdot \widetilde{m}_j\right) \bowtie f$	$Sat(\Phi, \widetilde{m}, t)$
$\Psi = \mathbb{EP}_{\bowtie p}(\phi)$	$\left(\sum_{j=1}^{K} m_j(t) \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m}, t)\right) \bowtie f$	$Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m}, t)$

Table 8.1: Time validity set development for the MF-CSL formulas

5. Recall that ODEs (2.1) define all possible occupancy vectors in the time interval $[\tau_i; \tau_{i+1}]$. These are checked against the above constraints, and the time intervals at which the occupancy vector satisfies the inequalities are added to the time validity set.

As mentioned in Section 8.3.1, the steady-state operator at the global level mirrors the steady-state operator at the local level[5]. Therefore, according to Equation (8.13) the probability $\pi^{\mathcal{M}}(s, Sat(\Phi, \widetilde{m}), t)$ is used for the computation of TVS$(\mathbb{ES}_{\bowtie f}(\Phi), \overline{m}, \theta)$. The overall distribution, given by the ODEs (2.1), is then checked against the above inequalities, and all the time instances where the inequalities are satisfied are added to the time validity set.

TVS$(\mathbb{EP}_{\bowtie f}(\phi), \overline{m}, \theta)$ is found in a similar way as TVS$(\mathbb{E}_{\bowtie f}(\Phi), \overline{m}, \theta)$, however, instead of obtaining the satisfaction set of the CSL formula at the local level, the probability of taking path ϕ has to be calculated for each local state s_j, as given in Equation (8.5).

The inequalities, describing the TVS$(\mathbb{EP}_{\bowtie f}(\phi), \overline{m}, \theta)$, are constructed according to Definition 8.1.5 for the whole time interval $[0, \theta]$. These inequalities are solved together with ODEs (2.1) to find TVS$(\mathbb{EP}_{\bowtie f}(\phi), \overline{m}, \theta)$, as was discussed above.

[5]Recall that for a time-inhomogeneous local CTMC, the steady-state operator can only be used in a limited number of cases, because the stationary distribution of mean-field models can be approximated using stationary points of the ODEs (2.1) only if the model is well-behaved (for more information see e.g. [71]).

By nesting formulas, more complex measures of interest can be specified. Model-checking of nested MF-CSL formulas does not differ from CSL model-checking [6], and the parse tree of the MF-CSL formula Ψ is built as for CSL formulas and the model-checking procedure is invoked recursively.

We now proceed with an example of model-checking MF-CSL formulas, to provide insight in both model-checking formula against a given occupancy vector $\overline{m}$, and in finding the time validity set. We use the model as in Example 7.3.1 to illustrate the model-checking procedure.

Example 8.3.2. *Let us consider the following formula*

$$\Psi = \mathbb{EP}_{\leq 0.2}(\textit{not infected}_Y \; U^{[0,3]} \; \textit{infected}_Y)$$

and a predefined occupancy vector

$$\overline{m} = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\}).$$

In order to check the satisfaction relation $\overline{m} \models \Psi$ the following three steps need to be taken:

1. *Calculate the time-dependent rates of the local model $\mathcal{M}^l$ using the ODEs (7.1).*
2. *Perform CSL model-checking on the local model $\mathcal{M}^l$ in order to compute $Prob^{\mathcal{M}^l}(s, \textit{not infected } U^{[0,3]} \textit{ infected}, \overline{m})$ for all states $s \in S^l$;*
3. *Check the satisfaction relation $\overline{m} \models \Psi$ using Definition 8.1.5.*

The time-dependent rates of the local model are $k^*_{X,1}(t)$, $k^*_{Y,1}(t)$, $k^*_{Z,1}(t)$. These rates are calculated as in Example 7.3.1 using the solution of the ODEs (7.1) with $\overline{m}$ as initial condition.

To find $Prob^{\mathcal{M}^l}(s, \textit{not infected}_Y \; U^{[0,3]} \; \textit{infected}_Y, \overline{m})$ the reachability problem $\pi^{\mathcal{M}^l[\neg \textit{not infected}_Y \vee \textit{infected}_Y]}_{s,s_1}(0,3)$ has to be solved according to the algorithm described in Section 8.2.2. The local model $\mathcal{M}^l$ is modified, namely, all infected_Y*-states are made absorbing along with all states in groups X and Z. Let us*

denote the generator matrix of the modified local model as:

$$Q'(\overline{m}(t)) = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k^*_{Y,1}(t) & k^*_{Y,1}(t) & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Then the Kolmogorov equation (8.3) is used to calculate the transient probability matrix of the modified model:

$$\frac{d\Pi'(0, 0+T)}{d(T)} = \Pi'(0, 0+T) \cdot Q'(\overline{m}(0+T)),$$

with $\Pi(0,0) = \mathbb{1}$. *For* $T = 3$ *the reachability probabilities are as follows:*

$$\Pi'(0,3) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.77 & 0.23 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}.$$

The probability that the until formula

$$\phi = \textit{not infected}_Y \; U^{[0,3]} \; \textit{infected}_Y$$

holds for each starting state is calculated using Equation (8.2), and is as follows:

$$Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}) = \pi^{\mathcal{M}^l[\textit{infected}_Y]}_{s_{Y,1},s_{Y,2}}(0,3) + \pi^{\mathcal{M}^l[\textit{infected}_Y]}_{s_{Y,1},s_{Y,3}}(0,3) = 0.23,$$
$$Prob^{\mathcal{M}^l}(s_{Y,2}, \phi, \overline{m}) = 1,$$
$$Prob^{\mathcal{M}^l}(s_{Y,3}, \phi, \overline{m}) = 1.$$

For all starting states in groups X and Z the probability that this until formula holds equals zero, as the starting state does not satisfy infected_Y.

According to Definition 8.1.5, the weighted sum of the entries of the occupancy vector $\overline{m}$ and the respective probabilities in the local model define the probability of a given occupancy vector $\overline{m}$ to satisfy the expected probability formula $\mathbb{EP}_{\bowtie f}(\phi)$. In this example this probability is calculated as follows:

$$\sum_{j=1}^{K} m_j \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m}) = m_{X,1} \cdot 0 + \cdots + m_{Y,1} \cdot 0.23 + m_{Y,1} \cdot 1 + m_{Y,1} \cdot 1 + \cdots + m_{Z,3} \cdot 0 = 0.3 \cdot 0.23 + 1/30 \cdot 1 = 0.102.$$

As one can see, the probability of the MF-CSL formula to hold is greater than 0.04*, therefore, the occupancy vector $\overline{m}$ satisfies the formula*

$$\mathbb{EP}_{\leq 0.2}(not\ infected_Y U^{[0,3]}\ infected_Y).$$

Example 8.3.3. *As was discussed in Section 8.3, the satisfaction of the global MF-CSL formula may change with time. Let us consider the same formula Ψ and occupancy vector $\overline{m}$, as in Example 8.3.2. In the following we calculate the time validity set $TVS(\Psi, \overline{m}, 15)$ for $\theta = 15$.*

The calculation of the time-dependent probabilities

$$Prob^{\mathcal{M}^l}(s, not\ infected_Y\ U^{[0,3]} infected_Y, \overline{m}, t)$$

is done as described in Section 8.2.2, which requires the following steps:

1. *The local model $\mathcal{M}^l$ is modified by making all* infected_Y*-states, and states in groups X and Z absorbing, and the transient probability $\Pi(0,3)$ is calculated as in Example 8.3.2.*

2. *The ODEs, describing the time-dependent transient probability of the modified model are constructed using both forward and backward Kolmogorov equations (see Equation (8.4)) .*

3. *These ODEs are solved using $\Pi(0,3)$ as initial condition. The solution of the ODEs defines the required reachability probabilities.*

4. *The probabilities $Prob^{\mathcal{M}^l}(s, \phi, \overline{m}, t)$ are computed using Equation (8.5).*

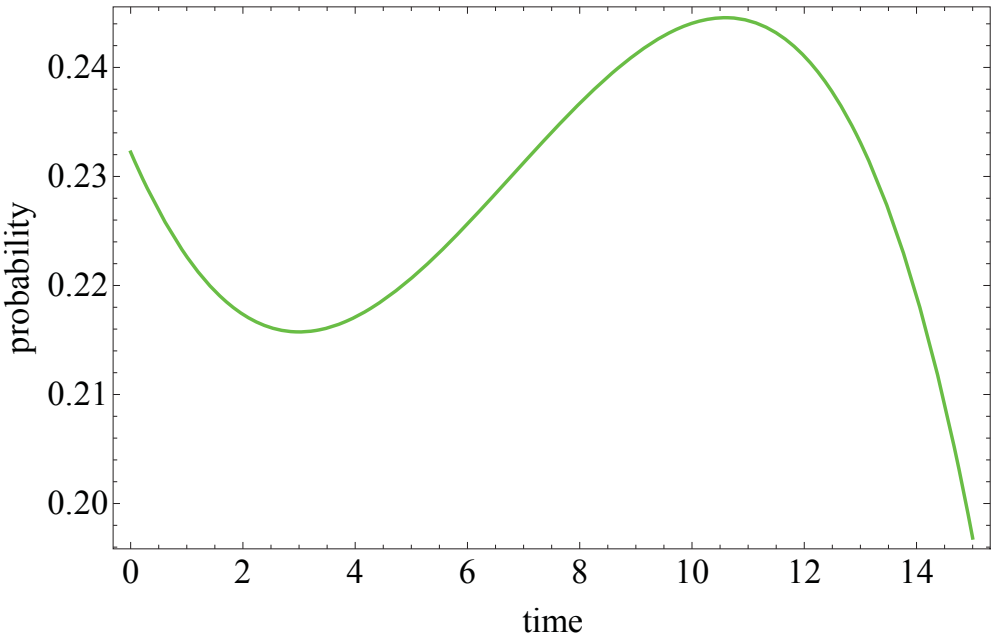


Figure 8.1: $Prob^{\mathcal{M}^l}(s_{Y,1}, \text{not infected}_Y\ U^{[0,3]}\text{infected}_Y, \overline{m}, t)$.

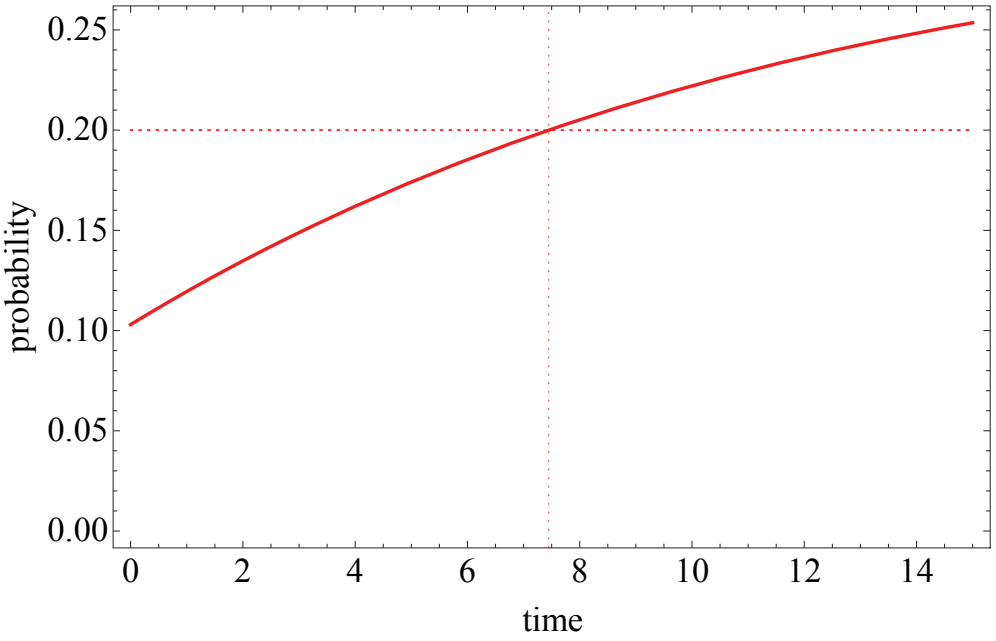


Figure 8.2: The red solid line shows the time-dependent expected probability $\mathbb{EP}(\text{not infected}_Y\ U^{[0,3]}\ \text{infected}_Y)$. Two red dashed lines intersect in the point, where the above probability equals 0.2.

The time-dependent probability $Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}, t)$ *is depicted in Figure 8.1. The time-dependent probabilities* $Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}, t)$ *and* $Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}, t)$ *equal to* 1*, since these states satisfy* infected_Y*, which means that the until formula holds immediately. This probability equals zero for all starting states in groups* X *and* Z*, since these states do not satisfy neither* not infected_Y*, nor* infected_Y*.*

The probability $Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}, t)$ *is quite stable within the time interval* $[0, \theta]$*, which coincides with the behaviour of the population as presented in Figure 7.2. Namely, it decreases initially as active infected computers start to switch off, and slightly increases when the fraction of infected machines decreases, namely, around* $4 < t < 11$ *(note that it is essential that infection has to occur within 3 time units). When the fraction of not infected machines becomes low this probability starts to decrease.*

To calculate the time validity set $TVS(\Psi, \overline{m}, 15)$ *the following steps need to be taken:*

1. *Construct the equation describing the time-dependence of the probability that the MF-CSL formula* $\mathbb{EP}_{\leq 0.2}(not\ infected_Y U^{[0,3]} minfected_Y)$ *holds:*

$$\sum_{j=1}^{K} m_j \cdot Prob^{\mathcal{M}^l}(s_j, \phi, \overline{m}, t) = m_{X,1}(t) \cdot 0 + \cdots + m_{Y,1}(t) \cdot$$
$$\cdot Prob^{\mathcal{M}^l}(s_{Y,1}, \phi, \overline{m}, t) + m_{Y,1}(t) \cdot 1 + m_{Y,1}(t) \cdot 1 + \cdots + m_{Z,3}(t) \cdot 0.$$

   This probability is depicted in Figure 8.2.

2. *Find the time points, where the above probability equals* 0.2*. In the current example, this is the case for only* $t = 7.45$*.*

3. *The time validity set of the formula*

$$\Psi = \mathbb{EP}_{\leq 0.2}(not\ infected_Y\ U^{[0,3]} infected_Y)$$

   consists of all time intervals, where the probability of this formula to hold is less than 0.2*. We, therefore, find that, as illustrated in Figure 8.2,* $TVS(\Psi, \overline{m}, \theta) = [0, 7.45]$*.*

○

8.4 Summary

In this chapter we have introduced a logic and algorithms for doing model-checking of mean-field models.

Since the details of the individual components are no longer visible in a mean-field description, existing logics are not suitable to express their properties. Therefore, we have introduced a new logic, called MF-CSL. This logic expresses properties of the *global* model, in terms of what fraction of objects satisfies *local* properties, where the latter are CSL-like properties of the individual objects.

Checking the local properties is challenging because the models of the individual objects are time-inhomogeneous Markov-Chains, as their parameters depend on the global state. We have adapted results from [18] to obtain algorithms for this, and building on this we have obtained algorithms for checking the global properties for a given global state, and for obtaining the time validity set, the time interval(s) in which a global property is satisfied.

9

Mean-Field Logic

In the previous chapter we introduced a way to express and check properties of the global mean-field model $\mathcal{M}^l$ via properties of a random local object. The properties, which MF-CSL allows to describe include CSL properties of the local model and the expected number of objects for a global model to satisfy these. However, not only this type of properties might be of interest. Timed properties of the global model can not be expressed using MF-CSL, but can be beneficial for understanding the behaviour of large systems. To accommodate this type of measure, we introduce the Mean-Field Logic (MFL) in Section 9.1. In Section 9.2 we describe an algorithm for checking whether a given occupancy vector $\overline{m}$ satisfies a given MFL formula. In Section 9.3 the ways to approximate the *satisfaction set* of an MFL formula and the availability of tooling are discussed. the short summary of the chapter is provided in Section 9.4.

9.1 MFL syntax and semantics

To be able to express timed properties of the overall model, we adapt the existing Signal Temporal Logic (STL) which was developed for monitoring discrete temporal properties of continuous signals [74], [80]. In order to do so we interpret the solution of the ODEs (2.1) for given initial conditions $\overline{m}(0)$ as a signal or a *trajectory* of the overall mean-field model $\mathcal{M}^O$. In the following we express properties of real-valued trajectories of mean-field models $\overline{m}(t)$ using STL-like properties. However, we first have to introduce a, so-called, mapping of the model trajectory to the boolean signal:

Definition 9.1.1 (Global atomic property)**.** *An atomic property GAP of the global model is a characteristic function (Boolean predicate)* $\overline{m} \to \{0, 1\}$*, from occupancy vector* $\overline{m}$ *to a Boolean value.* □

Applying the concept of GAP to a given trajectory of a mean-field model $\overline{m}(t)$ results in a Boolean function of time $GAP(\overline{m}(t))$. In order to guarantee decidability, we require that the output Boolean trajectory $GAP(\overline{m}(t))$ has finite variability, i.e., the number of time points, where $GAP(\overline{m}(t))$ changes value is finite; the output trajectory is a Boolean robust (cadlag[1]) function, see [74]. For simplicity, in the following we use inequalities of the form $\sum_{i \in N} a_i \cdot m_i \bowtie p$ as global atomic properties of mean-field models, where a_i is an indicator factor equal to 1 or 0. However, more advanced functions, satisfying the above requirements, can be used as GAP. Give the definition of a global atomic property, the syntax of Mean-Field Logic (MFL) can now be introduced.

Definition 9.1.2 (Syntax of MFL)**.** *Let $I = [a, b]$, where $0 \leq a < b$, be a non-empty bounded time interval and function GAP defining global atomic properties. MFL formulas Υ are defined as:*

$$\Upsilon ::= tt \;\mid\; GAP \;\mid\; \Upsilon_1 \wedge \Upsilon_2 \;\mid\; \neg\Upsilon \;\mid\; \Upsilon_1 \, \mathcal{U}^I \, \Upsilon_2.$$

□

Using MFL, we can define not only a property of the global model at a given time point, but also the evolution of the model over time, as shown in the following example.

Example 9.1.3. *We first start with the properties of the global model at a given time point (time-independent properties). To define such a property GAPs are combined with the time-independent operators $\neg$ and $\wedge$.*

The following property describes a system in which the fraction of computers that belong to group Y and that are infected is smaller than 0.2*:*

$$\Upsilon_1 = m_{Y,2} + m_{Y,3} < 0.2,$$

where $m_{Y,2}$ and $m_{Y,3}$ denote the number of infected computers in group Y (inactive and active respectively).

The same property can be expressed using atomic properties of the local model as follows:

$$infected_Y < 0.2.$$

[1] A function is cadlag if it is a function defined on the real numbers (or a subset of them) that is everywhere right-continuous and has left limits everywhere.

The lap-*based representation is not required in MFL, and is used here and further in the thesis for the easier interpretation of the meaning of an MFL formula.*

A more involved property can be defined by the conjunction of GAPs. The property that a system has more than 20% *infected computers and less than* 1% *active infected computers-members in group* Z *is formalized as:*

$$\Upsilon_2 = (\mathit{infected} < 0.2) \wedge (\mathit{active}_Z < 0.01).$$

Note that the first part of property Υ_2 *includes all infected computers in all three groups.*

Timed properties of the global system are constructed by combining GAPs (or other MFL formulas) using the until operator. The following property describes the system in which the fraction of computers which belong to group X *and are infected is smaller than* 0.1 *at all times* ***until*** *in the time interval between 3 and 5 time units the fraction of computers that are members of group* Z *and active exceeds* 0.4*:*

$$\Upsilon_3 = (\mathit{infected}_Y < 0.1)\ \mathcal{U}^{[3,5]}\ (\mathit{active}_Z > 0.4).$$

○

Definition 9.1.4 (Semantics of MFL)**.** *The satisfaction relation* $\models$ *for MFL state formulas and state* $\overline{m} \in S^o$ *is defined as:*

$$\begin{array}{lll}
\overline{m} \models tt & \forall & \overline{m} \in S^o, \\
\overline{m} \models GAP & \mathit{iff} & GAP(\overline{m}) = 1, \\
\overline{m} \models \Upsilon_1 \wedge \Upsilon_2 & \mathit{iff} & \overline{m} \models \Upsilon_1 \mathit{and}\ \overline{m} \models \Upsilon_2, \\
\overline{m} \models \neg\Upsilon & \mathit{iff} & \overline{m} \not\models \Upsilon, \\
\overline{m} \models \Upsilon_1 \mathcal{U}^I \Upsilon_2 & \mathit{iff} & \exists t \in I : (\overline{m}(t) \models \Upsilon_2) \wedge (\forall t' \in [0,t]\ \overline{m}(t') \models \Upsilon_1),
\end{array}$$

where $\overline{m} = \overline{m}(0)$ *at time* $t = 0$*, and* $\overline{m}(t')$ *is a solution of the ODEs (2.1) at time* $t = t'$*, with* $\overline{m}$ *as initial condition.* □

The definition of the until formula is different from the usual representation [83], because it requires both Υ_1 and Υ_2 to hold at time t, in order to guarantee closure [74].

As was explained in the previous chapter, in this thesis we discuss mean-field models, where the dependency on time is only implicit (via $\overline{m}(t)$), hence, the entire model trajectory (the solution of the ODEs (2.1)) is defined through

the current system state; the time when this state is reached does not influence the behaviour of the system. The occupancy vector for which the satisfaction relation is checked is therefore denoted as $\overline{m}(0)$, and the time intervals in the until formulas are relative to that. However, all the arguments and algorithms presented further in this Chapter can be generalized to models with an explicit time dependence.

9.2 Checking an MFL property

To check an MFL formula the parse tree has to be built and all sub-formulas checked recursively. Therefore, the algorithms for checking each individual operator have to be introduced. Checking a given occupancy vector $\overline{m}$ against time-independent operators is straightforward, it follows directly from Definition 9.1.4. However, MFL formulas containing the Until operator can not be checked that easily, since the behaviour of the system (trajectory) influences the result. Therefore, we introduce the notion of the *time validity set* for a given MFL formula, mean-field model and an occupancy vector, as has been done in the previous section for MF-CSL formulas (see Definition 8.3.1).

It is easy to see that if the $\mathrm{TVS}(\Upsilon, \overline{m}, \theta)$ contains $t = 0$, the formula holds for $\overline{m}$. The TVS of a general MFL formula is again built recursively by finding the TVSs of sub-formulas. The computation of the TVS for the time-independent operators is straightforward:

$$\begin{array}{lcl} \mathrm{TVS}(tt, \overline{m}, \theta) & = & [0, \theta], \\ \mathrm{TVS}(\mathrm{GAP}, \overline{m}, \theta) & = & \{t \in [0, \theta] \mid \mathrm{GAP}(\overline{m}(t)) = 1\}, \\ \mathrm{TVS}(\neg\Upsilon, \overline{m}, \theta) & = & [0, \theta] \setminus \mathrm{TVS}(\Upsilon, \overline{m}, \theta), \\ \mathrm{TVS}(\Upsilon_1 \wedge \Upsilon_2, \overline{m}, \theta) & = & \mathrm{TVS}(\Upsilon_1, \overline{m}, \theta) \cap \mathrm{TVS}(\Upsilon_2, \overline{m}, \theta). \end{array} \tag{9.1}$$

Computing the TVS for the until operator $(\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2)$ (with $0 \leq a < b$) is more challenging. The algorithm described in the following is based on the method of monitoring temporal properties as in [74], [80]; we refer to these papers for more details and proofs.

To compute the TVS for the until formula $\Upsilon = \Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2$ we first find the sets of time intervals where the sub-formulas Υ_1 and Υ_2 hold. Note that both sets may contain multiple intervals, therefore we denote them as $\mathrm{TVS}(\Upsilon_1, \overline{m}, \theta) = \upsilon_1^1 \cup \upsilon_1^2 \cup \cdots \cup \upsilon_1^{n_1}$, and $\mathrm{TVS}(\Upsilon_2, \overline{m}, \theta) = \upsilon_2^1 \cup \upsilon_2^2 \cup \cdots \cup \upsilon_2^{n_2}$, respectively.

To calculate TVS$(\Upsilon, \overline{m}, \theta)$ one has to obtain all time intervals where $\Upsilon = \Upsilon_1 U^{[a,b]} \Upsilon_2$ holds. We, hence, search for time intervals where both Υ_1 and Υ_2 hold, since these are the time intervals, where the validity of the until formula can be confirmed, in case at least one such time interval lies between a and b. Recall that the time interval in the until formula is relative to the starting point. Therefore, to check whether a given m-vector (state of the overall mean-field model) fulfils the until formula one has to check whether the intersection interval can be reached within the predefined time interval $[a, b]$. Hence, to directly compute the set of all time points from which the formula can be fulfilled we shift $TVS(\Upsilon_1 \cap \Upsilon_2, \overline{m}, \theta)$ backwards, i.e., move the left interval bound back with b and the right with a time units. This is defined for each pair of sub-intervals in TVS$(\Upsilon_1, \overline{m}, \theta)$ and TVS$(\Upsilon_2, \overline{m}, \theta)$ as so-called *backwards shift*, denoted as $\mathcal{BS}^{[a,b]}(v_1^i; v_2^j)$. For each pair of the intervals $(v_1^i; v_2^j)$, the backward shift is computed as follows:

$$\mathcal{BS}^{[a,b]}(v_1^i; v_2^j) = ((v_1^i \cap v_2^j) \ominus [a, b]) \cap v_1^i, \tag{9.2}$$

where $[x_1, x_2] \ominus [a, b] := [x_1 - b, x_2 - a] \cap [0, \infty)$. This backward shift can be understood as follows (from left to right):

1. The intersection $(v_1^i \cap v_2^j)$ defines all time points where both Υ_1 and Υ_2 are valid;

2. The $\ominus$-operation (or backwards shift) makes sure that: (1) the earliest starting point is taken such that after at most b time units one can reach a state where Υ_2 holds; and that (2) the latest starting point is taken such that from that point after, at least a time units, one can still switch to a state in which Υ_2 holds;

3. The intersection with v_1 ensures that on the way to the state where Υ_2 holds, Υ_1 always holds.

After the backwards shift is applied to each pair $(v_1^i; v_2^j)$, the resulting intervals are then combined to find the TVS of the overall until formula, as follows:

$$\text{TVS}(\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2, \overline{m}, \theta) = \bigcup_{i=1}^{n_1} \bigcup_{j=1}^{n_2} \mathcal{BS}^{[a,b]}(v_1^i; v_2^j). \tag{9.3}$$

Note that in practice only the pairs of intervals which actually do intersect have to be considered. Using Equations (9.1-9.3), the TVS of any MFL formula can

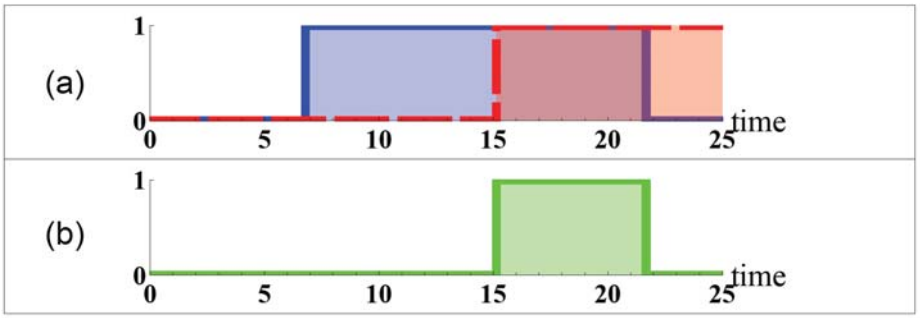


Figure 9.1: (a) TVS of $\Upsilon_1^A = (\text{active}_Y \leq 0.015)$ (blue solid line) and $\Upsilon_2^A = (\text{active}_Z \leq 0.01)$ (red dashed line); (b) intersection of $TVS(\Upsilon_1^A, \overline{m}(0), \theta)$ and $TVS(\Upsilon_2^A, \overline{m}(0), \theta)$.

be found. After the TVS of the formula is found, we can validate whether a formula holds for a given initial occupancy vector $\overline{m}$ by checking whether $t = 0$ lies in the TVS. Note that the TVS can also be seen as an independent measure of interest, if one is looking for the time-slots where the system satisfies a given property, for a given initial state (as in the case of MF-CSL properties). In the following, model-checking MFL formulas is illustrated by an example.

Example 9.2.1. *We again address the model of Example 7.3.1. We explain in detail how to calculate the time validity set* $TVS(\Upsilon, \overline{m}(0), \theta)$ *for both (**Case A**) time-independent and (**Case B**) time dependent formulas, given*

$$\overline{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\}),$$

and $\theta = 25$*. Then we check whether* $0 \in TVS(\Upsilon, \overline{m}(0), \theta)$*, which would indicate that the initial occupancy vector* $\overline{m}(0)$ *satisfies the formula.*

Case A. *We first consider the time-independent property that describes the system in which the fractions of active computers in groups* Y *and* Z *are "sufficiently small", i.e., the fraction of active infected computers in the group* Y *is bounded by* 0.015*, and the fraction of active infected computers in the group* Z *is at most* 0.01*, i.e., in MFL:*

$$\Upsilon^A = (\text{active}_Y \leq 0.015) \wedge (\text{active}_Z \leq 0.01).$$

To check this property the following steps are taken:

1. *The trajectory of the model is obtained by solving the ODEs (7.1) given* $\overline{m}(0)$ *(as depicted Figure 7.2).*

2. *The time validity sets of the sub-formulas* $\Upsilon_1^A = (\text{active}_Y \leq 0.015)$ *and* $\Upsilon_2^A = (\text{active}_Z \leq 0.01)$ *are calculated using a root finding procedure for*

$$m_{Y,3}(t) = 0.015 \quad \textit{and} \quad m_{Z,3}(t) = 0.01.$$

   We find

$$TVS(\Upsilon_1^A, \overline{m}(0), \theta) = [6.79; 21.69]$$

   and

$$TVS(\Upsilon_2^A, \overline{m}(0), \theta) = [15.14, 25]$$

   (see Figure 9.1(a)).

3. *The time validity set of the whole formula then consist of all intervals where both sub-formulas hold, i.e., we have*

$$TVS(\Upsilon^A, \overline{m}(0), \theta) = [15.14; 21.69]$$

   (see Figure 9.1(b)).

4. *The validity of the formula for a given initial vector is checked by verifying whether* $t = 0$ *lies in* $TVS(\Upsilon^A, \overline{m}(0), \theta)$*. It is easy to see that* $0 \notin [15.14, 21.69]$*, therefore,* $\overline{m}(0) \not\models \Upsilon^A$*.*

Case B. *We now consider a time-dependent property of the global model, which describes that the fraction of active infected computers in all three groups together (denoted as* active*) remains smaller or equal to* 0.05 *until within 3 time units the fraction of all not infected computers becomes less or equal to* 0.6*. This property ensures that the virus is "quiet enough" and will not be detected until a sufficient number of computers in the system is infected. In MFL we have:*

$$\Upsilon^B = (\text{active} \leq 0.05)\ \mathcal{U}^{[0,3]}\ (\text{not infected} \leq 0.6).$$

We first find the time validity sets for this formula, then we check whether the initial occupancy vector $\overline{m}(0)$ *satisfies this property:*

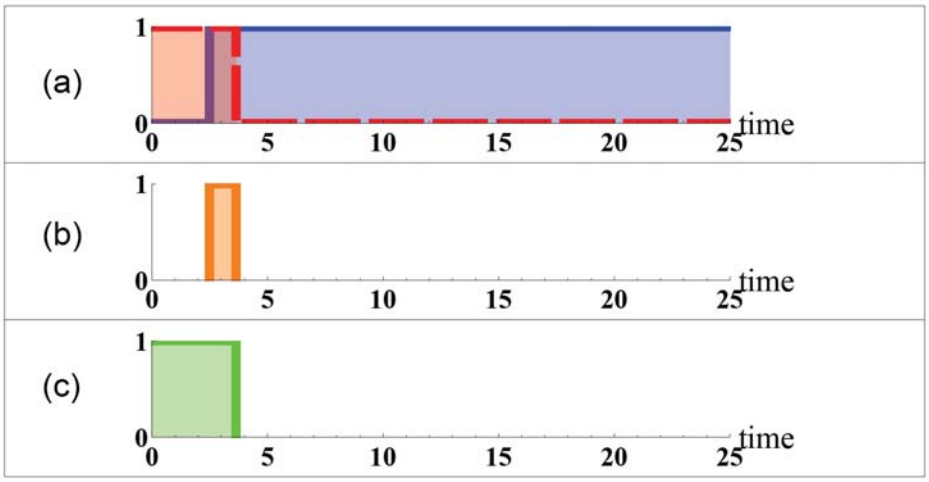


Figure 9.2: (a) TVS of Υ_1^B = active ≤ 0.05 (red dashed line) and Υ_2^B = (not infected ≤ 0.6) (blue solid line); (b) intersection of $TVS(\Upsilon_1^B, \overline{m}(0), \theta)$ and $TVS(\Upsilon_2^B, \overline{m}(0), \theta)$; (c) TVS of $\Upsilon^B = (\text{active} \leq 0.05)\mathcal{U}^{[0,3]}(\text{not infected} \leq 0.6)$.

1. *The time validity sets of the sub-formulas* Υ_1^B = (active ≤ 0.05) *and* Υ_2^B = (not infected ≤ 0.6) *are calculated using a root finding procedure for*

$$m_{X,3} + m_{Y,3} + m_{Z,3} = 0.05 \quad \textit{and} \quad m_{X,1} + m_{Y,1} + m_{Z,1} = 0.6.$$

We find

$$TVS(\Upsilon_1^B, \overline{m}(0), \theta) = [0; 3.64]$$

and

$$TVS(\Upsilon_2^B, \overline{m}(0), \theta) = [2.5, 25]$$

(see Figure 9.2(a)).

2. *Then the intersection of* $TVS(\Upsilon_1^B, \overline{m}(0), \theta)$ *and* $TVS(\Upsilon_2^B, \overline{m}(0), \theta)$ *is found, that is an interval* $[2.5; 3.64]$ *(Figure 9.2(b)).*

3. *To find* $TVS(\Upsilon^B, \overline{m}(0), \theta)$ *the backwards shift is done as in Equation (9.3)* $TVS(\Upsilon^B, \overline{m}(0), \theta) = [2.5-3; 3.64-0] \cap TVS(\Upsilon_1^B, \overline{m}(0), \theta) = [0; 3.64]$ *(see Figure 9.2(c)). Note that the behaviour of the system in the past (before time* $t = 0$*) is not relevant, therefore, the lower bound of the TVS is set to zero.*

4. *Formula* $\Upsilon^B = \big((\textit{active} \leq 0.05)\ \mathcal{U}^{[0,3]}\ (\textit{not infected} \leq 0.6)\big)$ *holds for* $\overline{m}(0)$*, since* $0 \in TVS(\Upsilon^B, \overline{m}(0), \theta)$.

Wolfram Mathematica [103] was used for calculating the above results. We compared them with results obtained with the Breach toolbox [32], which has been built for checking STL properties, and they coincide. Note that the Breach toolbox is publicly available and can be used for checking MFL-like properties of mean-field models.

◯

9.3 Satisfaction set of an MFL formula

In this section we discuss how to compute the complete satisfaction set of an MFL formula, which is formally defined as follows:

Definition 9.3.1 (MFL Satisfaction Set)**.** *Given an MFL formula* Υ *and a mean-field model* $\mathcal{M}^O$*, the* ***satisfaction set*** *of an MFL formula consists of* ***all*** *occupancy vectors* $\overline{m}$ *that satisfy* Υ*:*

$$Sat(\Upsilon) = \{\overline{m} \mid \overline{m} \models \Upsilon\}.$$

□

The mean-field model has an infinite state-space, therefore, the computation of the satisfaction set is not straight-forward. The ultimate goal is to partition the state-space of the model S^O into two parts: (i) states, satisfying a given formula, i.e., Sat(Υ); (ii) states, which do no satisfy Υ. Since exact methods for computing the satisfaction set of an MFL formula are not available, numerical (approximate) algorithms will be discussed in the following. The latter means that in some cases the partitioning of the state-space into two sets might not be available, hence, a third set, namely, the set of *uncertain states*, might be necessary. In such cases this third set should be as small as possible. The satisfaction set of a given formula is constructed recursively, by building and combining the satisfaction sets of sub-formulas.

9.3.1 Time-independent operators

The computation of satisfaction sets for operators which are not time dependent, is straight-forward and does not imply any additional computations. It

follows directly from Definition 9.1.4 and is formalized as follows:

$$\begin{array}{lcl} \mathrm{Sat}(tt) & = & S^o, \\ \mathrm{Sat}(GAP) & = & \{\overline{m} \mid GAP(\overline{m}) = 1\}, \\ \mathrm{Sat}(\Upsilon_1 \wedge \Upsilon_2) & = & \mathrm{Sat}(\Upsilon_1) \cap \mathrm{Sat}(\Upsilon_2), \\ \mathrm{Sat}(\neg\Upsilon) & = & S^o \setminus \mathrm{Sat}(\Upsilon). \end{array}$$

9.3.2 The until operator

The computation of the satisfaction set of the time-dependent until operator $\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2$ is not trivial and involves additional methods. In the following we discuss three ways that can be used to calculate this set. Two of these are directly applicable to the whole MFL formula, and one is only suitable for a single until operator, hence, the satisfaction sets of the time-independent sub-formulas have to be computed separately (see Section 9.3.1).

9.3.2.1 Discretization of the state-space

One of the ways to approximate the satisfaction set of an MFL formula is to discretize the continuous state-space and check the MFL formula for each point of the obtained discrete states-space, using the standard method (see Section 9.2). The discretization can be done, for example, by a grid-based approach. However, this approach is computationally intensive and produces only an approximation of the satisfaction set. Moreover, the quality of such an approximation and the computational demand depend directly on the fineness of the grid. In addition, the complexity of the problem grows with the number of dimensions (local states). Although the method is applicable for any MFL formula, applying it to a model with a large local state-space and in order to obtain high-quality approximations is simply not feasible.

The Breach toolbox [32] can be used for estimating what parts of the state space would likely be included to the satisfaction set using the algorithms as presented in Section 9.2, and parameter sampling procedure. Grid sampling (for smaller models) or quasi-random sampling (for bigger models) is used for choosing a sufficient number of points of the state space, then for each of these sampled points it is easy to check whether a given MFL formula holds or not. Given a fine enough number of such points qualitative, but not quantitative, characterization of the satisfaction set can be obtained.

Example 9.3.2. *The state-space of the running example in this part of the thesis is ten-dimensional, which makes the graphical representation of the satisfaction set difficult. Moreover, obtaining a fine-enough sampling of the ten-dimensional state-space is very computationally demanding. Therefore, we illustrate the application of the Breach toolbox to the simpler virus spread model, as in Example 2.2.2, or as in one of the groups Y or Z if they are cut off the overall network, become independent of the other two groups, and are analysed separately. Recall, that this model has a three-dimensional state-space, where the local states are labelled as* not infected*,* infective and inactive*, and* infected and active*. We set the parameters of the model as*

$$k_1 = 0.9, k_2 = 0.005, k_3 = 0.01, k_4 = 0.1, k_5 = 0.06,$$

where k_1 is an infection rate, k_2 is the recovery rate for the inactive infected machine, k_3 and k_3 are the rates for the infected machine to become active and return to inactive state, respectively; and k_5 is the recovery rate for an active infected machine. The initial occupancy vector is as follows: $\overline{m}(0) = (0.9; 0.0; 0.1)$.

We perform a quasi random sampling of 10 000 *states from the state space, and calculate the trajectories of the model for each of these states. Note that additional constraints on the sampled states have to be added to guarantee that $m_1 + m_2 + m_3 = 1$. The MFL formula*

$$\Upsilon = (\textit{not infected} > 0.5)\ \mathcal{U}^{[0,2]}\ (\textit{inactive} > 0.3)$$

is checked for each of the chosen states. Figure 9.3 provides the results of the procedure, where the states which satisfy Υ are marked blue, and the states that do not satisfy Υ are depicted in red. As one can see, the formula holds in the area, where the fractions of not infected, inactive and active machines satisfy the following inequalities:

$$0.4 < m_1 < 0.6, \quad 0.2 < m_2 < 0.4, \quad 0 < m_3 < 0.3.$$

The above results are not exact, and more precise algorithms are needed in order to approximate the satisfaction set of a given MFL formula. Moreover, since the sampling is done randomly some areas of the state-space can possibly be covered less than another (see bottom part of Figure 9.3), which can also be avoided when using other techniques.

○

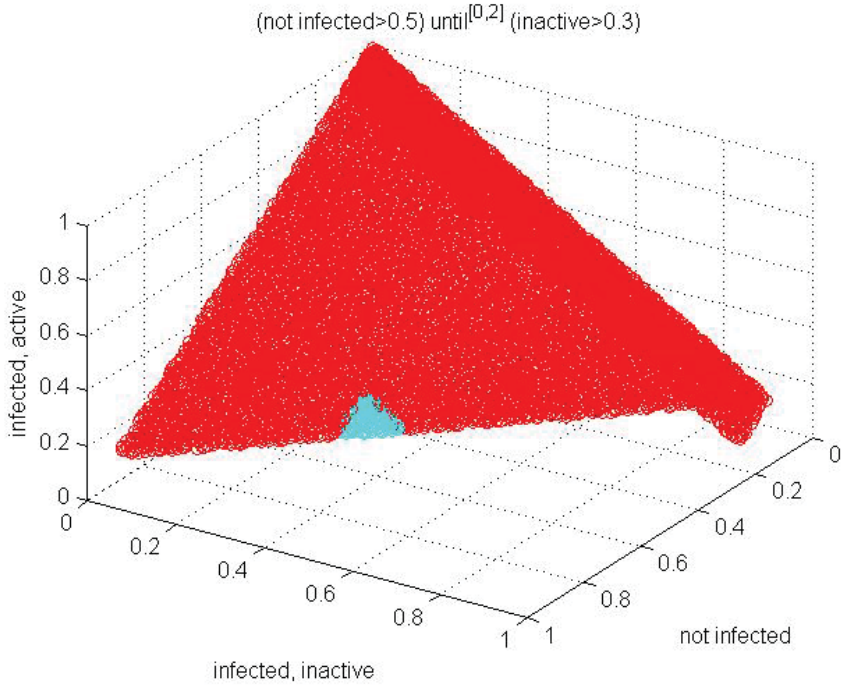


Figure 9.3: The regions where MFL formula holds obtained directly from Breach toolbox.

Even though this method does not provide any guarantees, it can be used as a "quick check" to find the regions where more precise procedures have to be applied to obtain the desired results. For instance, when calculating the satisfaction set of the formula, discussed in Example 9.3.2, only the part of the state space, which includes blue coloured states needs to be checked using more fine discretization and algorithms presented in Section 9.2 to check an MFL formula for each of the chosen initial states. Moreover, when the "regions of interest" are found using the Breach Toolbox, one of the methods presented in the following can be applied to approximate the satisfaction set of an MFL formula.

9.3.2.2 Solving two reachability problems

Another way to numerically develop the satisfaction set of a given until formula would be to divide the formula $\Upsilon_1 \mathcal{U}^{[a,b]} \Upsilon_2$ into two *reachability problems*, similar to standard methods, as, e.g., in [6]. The reachability problems for

mean-field models can be solved using techniques proposed in [33]. Their method partitions the set of all parameters of the ODE-based model into three sets, namely (i) S_{goal}, which comprises all states that satisfy the reachability problem; (ii) S_{bad}, including all states which do not satisfy it; and (iii) S_{unc}, which combines all states for which reachability can not be indicated. Instead of partitioning the parameter set we use the proposed methods for partitioning the state-space of the mean-field model. Note, however, that this approach is only applicable for a single until operator. To compute the satisfaction set of an until formula we need to solve the two reachability problems in reverse order. We first find all states from which we can reach a Υ_2 state in at most $(b-a)$ time units, while visiting only Υ_1 states. We denote the set of all these states S'_{goal}. We then find the set of all states (denoted as S_{goal}), from which we can reach S'_{goal}, while visiting only Υ_1 states. The satisfaction set of the overall until formula then equals S_{goal}.

The general approach of the method, as in [33] is as follows. The reachable set is found using *sensitivity analysis* and the satisfaction set is obtained by using parameter synthesis algorithm based on a *refining partition*. That is to iteratively refine the state-space of the mean-field model, and assign the obtained subsets to one of the three sets, namely S_{goal}, S_{bad}, or S_{unc} by checking the reachability problem for this set. Each refinement introduces only subsets that are strictly smaller than the refined set to guarantee that the process ends. The algorithm is designed to stop when the uncertain set is either empty, or smaller than a predefined size.

To implement this algorithms the Breach Toolbox [32] in [33], however, a tool for the automated solution of the reachability problem is not publicly available. Although the numerical algorithms in [33] can not provide formal guarantees on the correctness of the results, asymptotic guarantees exist, therefore, results can always be improved by decreasing the tolerance factor in the numerical computations.

9.3.2.3 Robustness based method

In the following we discuss another method to obtain the satisfaction set of an arbitrary MFL formula (including nested until operators). The state-space partitioning is again based on refining partition algorithms, however, this approach requires introducing a *quantitative semantics* of MFL. This allows both boolean and real values as an output of a model-checking algorithm ($\mathbb{R} \cup \{\top, \bot\}$). The

result of the model-checking procedure shows that a given occupancy vector satisfies an MFL formula (in case the obtained value is greater then zero), in addition it estimates the robustness of satisfaction. In the following we introduce the quantitative semantics of MFL, similarly to [35], where a quantitative semantics was introduced for STL. For simplicity of notation we use global atomic properties in the form $f(m_1, m_2 \dots m_K) \geq c$, where $c \in \mathbb{R}$.

Definition 9.3.3 (Quantitative semantics)**.** *Given an MFL formula Υ, a mean-field model $\mathcal{M}$, and initial occupancy vector $\overline{m}$, the quantitative semantics for space robustness $\rho(\Upsilon, \overline{m})$ is defined as follows:*

$$
\begin{array}{lcl}
-\top & = & \bot, \\
\rho(tt, \overline{m}, t) & = & \top, \\
\rho(GAP, \overline{m}) & = & f(m_1, m_2 \dots m_K) - c, \\
\rho(\Upsilon_1 \wedge \Upsilon_2, \overline{m}) & = & min(\rho(\Upsilon_1, \overline{m}),\ \rho(\Upsilon_2, \overline{m})), \\
\rho(\neg\Upsilon, \overline{m}) & = & -\rho(\Upsilon, \overline{m}), \\
\rho(\Upsilon_1 \mathcal{U}^I \Upsilon_2, \overline{m}) & = & \sup_{t' \in t+I} \min \Big(\rho(\Upsilon_2, \overline{m}(t')), \\
 & & \inf_{t'' \in [t,t']} \rho(\Upsilon_1, \overline{m}(t'')) \Big),
\end{array}
$$

where $\overline{m} = \overline{m}(0)$, and $\overline{m}(t')$ is a solution of the ODEs (2.1) at time $t = t'$ with $\overline{m}$ as initial condition, $\top$ and $\bot$ are ordered such as $\top = \infty$ and $\bot = -\infty$. □

Time and a space-time robustness of satisfaction for a quantitative semantics is discussed in [35] and $\rho(\Upsilon, \overline{m})$ is called a *robustness estimate*. The robustness estimate is found using an inductive procedure as for model-checking MFL formulas. Efficient algorithms for finding robust estimates are described in [34], and the Breach toolbox [32] can be directly used for that purpose. Given the algorithms for finding the robust satisfaction of an MFL formula, the satisfaction set of such formula, $\mathrm{Sat}(\Upsilon)$, can be calculated by partitioning the state-space S^O of the mean-field model. The latter can be done using refining partition, as proposed in [33, 35].

It is also possible to perform the robustness analysis using tools like S-TaLiRo [2] and BIOCHAM [24]. Note that also here no formal guarantees on the correctness of results can be provided. The advantage of the robustness-based method lies in the fact that the procedure is applicable to any MFL formula (unlike reachability-based methods). Moreover, there are tools available for robustness analysis of the time-series (numerical solutions of the ODEs, or even measured data in the context of satisfaction set development). Finally,

more advanced numerical methods might be applied for the partitioning of the state-space.

9.4 Summary

In this chapter we introduced the logic MFL for checking timed propertied of overall mean-field models. We defined the global atomic properties of the overall mean-field model to be able to reason on the global level directly, in contrast with using the logic MF-CSL presented in the previous chapter.

The algorithms for checking MFL properties against a given occupancy vector and finding the time validity set have been presented and illustrated by the examples. All computations were done in Wolfram Mathematica and compared against results produced by the Breach Toolbox. Both results were consistent. Moreover, three possible ways to estimate the satisfaction set of an MFL formula have been discussed.

10

Relation between MFL and MF-CSL

As we discussed in the previous chapters, there are two ways of describing properties of the mean-field models. One way is to reason about the fraction of objects satisfying a given local property (check, whether this number meets a given threshold) using the logic MF-CSL. Another way is describing the properties of the overall system, including timed properties, which can be done with the logic MFL. In Section 10.1 we discuss the difference between these two logics and argue that both have their value. The possibility of combining both logics is discussed in Section 10.2. Section 10.3 provides the concluding remarks for the material presented in Part III.

10.1 Comparison of MFL and MF-CSL

In Table 10.1 the main differences between MFL and MF-CSL are depicted. As we previously discussed both logics are used in order to describe (and check) properties of mean-field models. Moreover, the time validity set can be defined for MF-CSL, while model-checking MFL properties require the computation of TVSs.

All properties in MF-CSL are based on the structure and labelling of the local model, and *expectation operators* lift these properties to the global level. Conversely MFL expresses properties of the global mean-field model independently from the labelling and structure of the local model. A global atomic property (GAP) can be defined both on the local model structure and labelling, as well as via labelling-independent functions of the occupancy vector $\overline{m}$. For example, such properties as "infected computer in all three groups" can be easily described by MFL, while MF-CSL needs "workarounds" by either introducing

Property	MF-CSL	MFL
Applicable for mean-field models	+	+
Operates on both local and global levels	+	−
Timed property on the local level	+	−
Timed property on the global level	−	+
Local labelling (*lap*)	+	−
Global labelling (*GAP*)	−	+
Expectation relations	+	−
Time Validity Set	+	+
Satisfaction set can be approximated	−/+	+

Table 10.1: MF-CSL versus MFL

different labelling on the local level, or expressing the *infected* properties for each group separately on the local level, then on the global level; and finally combining these properties by concatenation and/or negation.

The largest difference lies in the application of timed properties (see Figure 10.1). Both logics may use the until operator, however, in MFL the until operator is used on the global level, while in MF-CSL only the evolution of an individual random object can be described with the until operator.

Finally, approximating the full satisfaction set is possible for MFL properties, as defined in Section 9.3. The calculations on the local level of MF-CSL are, however, quite demanding [18] and the partitioning of the state-space of the overall mean-field model, as, e.g., discussed in Section 9.3 is impractical. Therefore, the satisfaction set of the MF-CSL formulas is often impossible to obtain.

For some models, e.g., models of a chemical reactions [45], the behaviour of a random individual (one molecule) is not of interest. Therefore, MF-CSL might not be of interest and only the logic MFL would be applicable. Despite that, there are many systems, that can be modelled using the mean-field method, where the behaviour of a random object would be of importance, for example in the virus spread models, as is discussed in this thesis.

Clearly, both logics can be of interest, albeit for different users and different systems. Some properties can be expressed in both logics, however, the majority of properties can only be described in one of the two logics, which explains the necessity of introducing these logics separately.

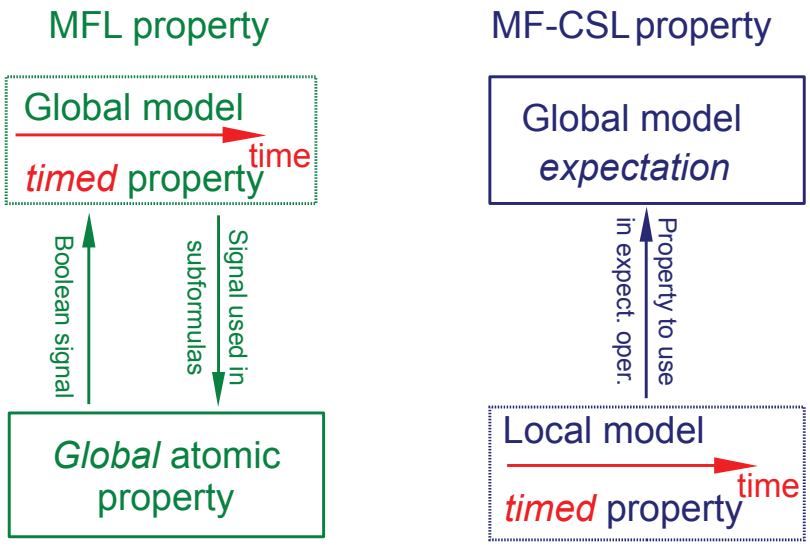


Figure 10.1: MFL property vs MF-CSL property.

10.2 Combination of the two logics

We now discuss the combination of the two logics to achieve the best expressivity. As described in Section 9.1, a Global Atomic Property can be defined by any boolean function, which, when applied to the model trajectory, has as output a robust cadlag function (everywhere right-continuous and has left limits everywhere). Moreover, an arbitrary MF-CSL formula Ψ can be seen as a boolean function $\overline{m} \to \{0,1\}$, where the output signal equals 1 for all $t \in TVS(\Psi, \overline{m}, \theta)$, and 0 otherwise. Therefore, if this output signal is cadlag, the MF-CSL property Ψ can be used to define a GAP of the overall mean-field model. This allows to define combined properties, where both MF-CSL and MFL operators are used. Such combined properties are easier to interpret, since the GAPs of the MFL formulas are defined via the local atomic properties, moreover, using such combined formulas allows, for example, to describe timed properties on both levels. Let us consider the following example:

$$\Upsilon = \underbrace{(\mathbb{E}_{\leq 0.1}\, active_Y)}_{\text{MF-CSL sub-formula}} \quad \underbrace{\mathcal{U}^{[0,5]}}_{\text{MFL until}} \quad \underbrace{\left(\mathbb{EP}_{\leq 0.1}(tt\ U^{[0,3]}\ infected_Y)\right)}_{\text{MF-CSL sub-formula}}$$

This property is useful for a system administrator, who wants to be sure that

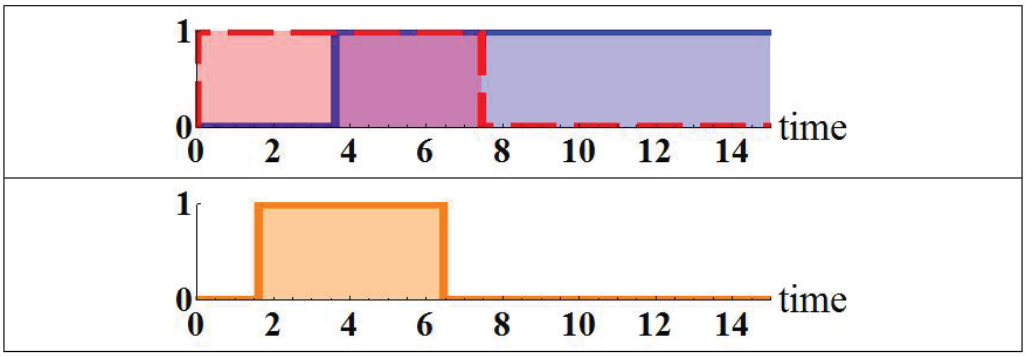


Figure 10.2: (a) TVS of Υ_1 (red dashed line) and Υ_2 (blue solid line); (b) TVS of Υ.

with the current activity of the anti-malware software:

> **(MF-CSL)** not more than 10% of the computers in group Y are infected and active,
>
> **(MFL)** until within 5 time units a system state is reached,
>
> **(MF-CSL)** where the probability that a random computer will become infected within 3 time units is less or equal than 0.1

The example below provides a more detailed explanation on how to check such properties.

Example 10.2.1. *In the following we define and check a combined property of the computer virus model, as given in Example 7.3.1. We will construct a combined property using both logics. Then we find the time validity set for the obtained property given a predefined time interval* $\theta = [0, 15]$*, and initial distribution*

$$\overline{m}(0) = \frac{1}{3}(\{0.8, 0, 0, 0.2\}, \{0.9, 0, 0.1\}, \{0.4, 0.55, 0.05\}).$$

Finally, we check the combined property against the initial occupancy vector $\overline{m}(0)$*. For simplicity we use the MF-CSL property, described in Example 8.3.2 as one of the sub-formulas in the combined property:*

$$\Upsilon_1 = \mathbb{EP}_{\leq 0.2}(\text{not infected}_Y \; U^{[0,3]} \; \text{infected}_Y).$$

We combine Υ_1 *with MFL sub-formula* $\Upsilon_2 = (\text{active} \geq 0.1)$ *using the until operator and obtain the following combined formula:*

$$\Upsilon = \underbrace{\left(\mathbb{EP}_{\leq 0.2}(\text{not infected}_Y \; U^{[0,3]} \; \text{infected}_Y)\right)}_{\textit{MF-CSL sub-formula}} \; \underbrace{\mathcal{U}^{[1,2]}}_{\textit{MFL until}} \; \underbrace{(\text{active} \geq 0.1)}_{\textit{MFL GAP}}.$$

This property describes a system, where the expected probability of a random computer to be from group Y *and to become infected within* 3 *time units is less than or equal to* 0.2 *(MF-CSL), at all times until within time interval* $[1, 2]$ *(MFL) the number of active infected computers in all three groups is less or equal than* 0.1 *(MFL).*

To check such a combined property we first have to find the TVSs of all sub-formulas. From Example 8.3.2 we recall that

$$TVS(\Upsilon_1, \overline{m}, \theta) = [0, 7.45].$$

The time validity set of the MFL sub-formula $TVS(\Upsilon_2, \overline{m}, \theta)$ *is found by solving* $m_{X,3} + m_{Y,3} + m_{Z,3} = 0.1$*:*

$$TVS(\Upsilon_2, \overline{m}, \theta) = [3.61; 15].$$

Figure 10.2(a) displays the time validity sets of sub-formulas Υ_1 *and* Υ_2*.*

The time validity set of the combined formula is computed as described in Section 9.2: first the intersection of $TVS(\Upsilon_1, \overline{m}, \theta)$ *and* $TVS(\Upsilon_2, \overline{m}, \theta)$ *is found:*

$$TVS(\Upsilon_1 \cap \Upsilon_2, \overline{m}, \theta) = [3.61; 7.45].$$

Then $TVS(\Upsilon_1 \cap \Upsilon_2, \overline{m}, \theta)$ *is shifted backwards, as depicted in Figure 10.2(b):*

$$TVS(\Upsilon, \overline{m}, \theta) = [3.61 - 2; 7.85 - 1] \cap TVS(\Upsilon_1, \overline{m}, \theta) = [1.61; 6.45].$$

As one can see, the occupancy vector $\overline{m}$ *does not satisfy the combined property* Υ*, since* $0 \notin [1.61; 6.45]$*.*

○

Full computation of the satisfaction set of such combined properties might be not feasible, due to the high computational costs on the local level of MF-CSL [18].

10.3 Concluding remarks

With this section we provide concluding remarkc for Part III. In this part of the thesis two logics for checking mean-field models have been proposed. The logic MF-CSL [66] uses local CSL properties as a basis for global expectation operators, therefore, it is fully dependent on the structure of the local model. The logic MFL, on the other hand, does not take into account properties of the individual objects and only reasons on the global level. Therefore, time-dependent properties of the global model can be described using MFL, whereas MF-CSL only allows time-dependent properties on the local level.

The algorithms for checking a given formula against an initial occupancy vector and for finding time validity sets were discussed for both logics. For model-checking MF-CSL properties first the local property has to be checked (similar to [18]). The satisfaction set (path probability) of the local model is either built for a given moment in time or the time-dependent satisfaction set (path probability) is found. Then the local satisfaction set is used on the global level for checking MF-CSL formula or building time validity sets. Examples were discussed to illustrate the presented algorithms.

The algorithms for checking MFL properties against a given occupancy vector and finding the time validity set are based on the method of monitoring temporal properties as in [74], [80]. We adapt these methods for checking global properties of the mean-field model, and illustrate the algorithms in examples. All computations were done in Wolfram Mathematica and compared against results, produced by the Breach Toolbox. Both results were consistent

Moreover, three possible ways to calculate the satisfaction set of an MFL formula are discussed. One of these methods relies on the presented boolean semantics of MFL and a discretization of the continuous state-space. The second method makes use of an existing technique for finding the parameters of the model, which satisfy a given *reachability* problem [33] using *sensitivity analysis*. The until formula is solved as a combination of two reachability problems, which are tackled in reverse order to be able to check whether the until formula holds. The *refine partitioning* method is used to partition the states space and find the satisfaction set. The third method adapts an existing notion of *robustness* [35] of a temporal logic and *sensitivity analysis*. This technique is based on defining a *quantitative semantics* of MFL, then the resulting robust estimate is used as an indicator in order to guide the partitioning algorithm.

Furthermore, the expressivity and applicability of the two logics have been

compared. Despite the fact that both logics are applicable to mean-field models, clearly, both are of interest and can not be fully replaced by the other. Another interesting insight in the usage of the presented logics lies in the fact that the two logics can be combined if the global atomic property of the mean-field model is represented by one of the expectation operators. This allows the combination of MF-CSL and MFL properties (including timed properties) on both levels. Such properties can be easily checked for a given occupancy vector, however, the satisfaction set development might not be feasible.

11

Conclusions

In this chapter we summarize the thesis; we first recall the research question stated in Chapter 1 and list the findings of the thesis, addressing each of these questions. Then we discuss the contribution of this thesis and directions for future work.

$\mathcal{Q}1$: Can mean-field method be used for fast, efficient, and accurate analysis of large systems of interacting objects?

In Chapter 3 we showed that the mean-field method is much faster than simulation. Moreover, the approximated results are quite accurate, and lie inside the 95% confidence intervals, as obtained by simulation. In addition we have been able to explore the gain in speed (see Table 3.3), which is achieved by applying the mean-field method. We addressed various questions which cannot practically be answered with simulation, such as questions involving cost trade-offs; this is useful in typical engineering applications. One can think of other questions to address, however, our aim was to show the potential of the method by addressing problems which can not be solved using simulation.

$\mathcal{Q}2$: How to obtain realistic parameter values for mean-field models?

To address this research question we performed a case-study based on a real-world networking phenomenon in Part II. We combined the mean-field approach with parameter-fitting techniques, namely, minimizing relative squared error and negative log-likelihood, to obtain a parametrized mean-field model of the spreading phase of the Code-Red worm. We were able to show that the data measured during the outbreak of the worm in 2001 can be used to parametrize the model, such that it closely reflects the behaviour of the worm.

The models we fitted allowed us to obtain parameters which ensure a relative squared error of 0.2% and 0.7% between the model prediction and the measurement, for the July and August outbreaks, respectively. Last, but not least, we provided a full discussion on the challenges one may encounter when performing similar studies.

$\mathcal{Q}3$: How to express and automatically check advanced properties of mean-field models?

In Part III of this thesis we proposed two logics for model-checking mean-field models, namely, MF-CSL and MFL. We defined the syntax and semantics of both, and provided detailed model-checking algorithms. The logic MF-CSL uses local CSL properties as a basis for global expectation operators, therefore, it is fully dependent on the structure of the local model. The logic MFL, on the other hand, does not take into account properties of the individual objects and only reasons on the global level. Therefore, time-dependent properties of the global model can be described using MFL, whereas MF-CSL only allows time-dependent properties on the local level. We compared these two logics, and provided a discussion on the possible combination of both. We argue that even though both logics are defined for mean-field models, clearly, both are of interest and can not be fully replaced by the other.

Lessons learned

Let us first come back to the Stuxnet example discussed in Chapter 1. We first tried to find out whether quantitative analysis of Stuxnet is possible in 2011 in [64]. Looking back at it after 3 years of experience in the field we can better judge what research questions can be answered for this concrete example, and why some of these can only be addressed partially. A full parametrization of the Stuxnet model is still not possible. There are couple of reasons for it:

- *There is no data available.* Indeed, Stuxnet is very different from Code-Red, as discussed in Part II. It was programmed to spread slowly and imperceptibly, therefore, it was operating for more then a year without being discovered, and measured.

- *Human influence.* As we learned from the Code-Red case-study, proper modelling of human behaviour is a very difficult task. Although some,

e.g., sociological study can help to estimate the parameters describing human influence, it is very difficult to do, and the results might still be unrealistic.

- *Command-and-control.* The influence of C&C is difficult to model since it includes both carefully planned, and unpredictable "on-the-spot" human actions.

If one would want to do quantitative analysis of Stuxnet now, the mean-field model would clearly be possible to obtain, however, the parameter values are still not available. A partial parametrization might be possible, but very difficult to obtain. It will require close collaboration with security experts to obtain experimental data. When fitting the parameters, data measured in a synthetic environment has to be treated differently, than the real data. Although some of the difficulties might be avoided, e.g., setting initial conditions or dealing with measurements that do not relate to the model; the influence of the experimental environment has to be carefully considered. Model-checking of the partially parametrized model might, however, help to better understand the behaviour of Stuxnet.

Next to the application and case-study this thesis presents two new logics and model-checking algorithms for mean-field models. Defining the syntax and semantics of the two logics required us to look deep into the structure of the mean-field model and properties of interest. Moreover, the techniques for model-checking ICTMCs [18] had to be studied and adopted to be used as a part of the MF-CSL model-checking algorithms. The notion of time validity sets for a given formula was introduced in order to stress the dependency of the model-checking results on time. Moreover, global atomic properties have been defined, to be able to express the properties of the overall model by MFL. The logic MFL is closely related to STL [74], therefore, the techniques for checking temporal properties of the continuous signals had to be studied and applied to mean-field models. Model-checking mean-field models closely relates to the field of model-checking infinite-state models, since the state-space of the mean-field model is infinitely large as well. Recently a number of studies were performed in this field, either addressing models with a highly structured state-space [85], or applying abstractions [85], and [55]. The convergence theorem allows to apply an abstraction of the underlying state-space that only considers the deterministic behaviour of a very large population. This, in turn, makes

model-checking of such large-scale models feasible.

Overall this thesis presents an interesting combination of applied case-studies and theoretical techniques, which has led to very practical and relevant insights. In our point of view the field of computer science would benefit from more collaboration between experts from different fields. Such a collaboration will lead to a better understanding of which kind of measurements are necessary to build reliable models and to obtain accurate results. This, in turn, could result in a better understanding of the studied phenomenon.

Future directions

For each part of the thesis there remain open research topics, which, however, lie outside the scope of this thesis. In the following we will mention a couple of them.

Even though there is an increasing body of work that deals with relaxing the conditions of the convergence theorem, e.g., [91], [50], [52], it would be very interesting to see whether it can be applied to other kinds of model, for example, such including non-determinism.

To the best of our knowledge it would be difficult to circumvent the encountered problems with respect to the parameter fitting, due to the partial availability of data and the lack of initial conditions. Even though special parameter fitting method exists that do not require initial conditions, e.g., [84], they are not able to cover the main problem of the partial availability of data.

Model-checking mean-field models is a very young field, and many ways of exploring this field have not yet been taken. The directions of interest include providing formal guarantees for obtained results, and exploring the unbounded properties of the local and global models. These extensions are not easy to achieve since there are many different types of models and properties to be considered, however, they are important for the practical usage of the proposed theoretical findings. Moreover, in this thesis we focused on model-checking continuous-time models, while some of the application, e.g., gossiping protocols, may benefit from developing algorithms for discrete-time models.

Bibliography

[1] Aldrich, J. Doing Least Squares: Perspectives from Gauss and Yule. *International Statistical Review*, 66(1):61–81, 1998.

[2] Annpureddy, Y., Liu, C,. Fainekos, G., and Sankaranarayanan, S. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *TACAS*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[3] Arlitt, M., and Jin, T. A Workload Characterization Study of the 1998 World Cup Web Site. *Network Magazine of Global Internetwkg*, 14(3): 30–37, 2000.

[4] Baccelli, F., Karpelevich, F.I., Kelbert, M.Y., Puhalskii, A.A., Rybko, A.N., and Suhov, Y.M. A mean-field limit for a class of queueing networks. *Journal of Statistical Physics*, 66:803–825, 1992.

[5] Baier, C. and Katoen, J.P. *Principles of model checking.* MIT Press, 2008.

[6] Baier, C., Haverkort, B.R., Hermanns, H., and Katoen, J.P. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(7):524–541, 2003.

[7] Bakhshi, R. and Cloth, L. and Fokkink, W. and Haverkort, B.R. Mean-Field Analysis for the Evaluation of Gossip Protocols. In *QEST*, pages 247–256. IEEE Computer Society, 2009.

[8] Bakhshi, R. and Endrullis, J. and Endrullis, S. and Fokkink, W. and Haverkort, B.R. Automating the mean-field method for large dynamic gossip networks. In *QEST*, pages 241–250. IEEE Computer Society, 2010.

[9] Bartocci, E., Bortolussi, L., Nenzi, L., and Sanguinetti, G. On the robustness of temporal properties for stochastic models. In *HSB*, volume 125 of *EPTCS*, pages 3–19. Open Publishing Association, 2013.

[10] Bellman, R., and Roth, S.R. The use of splines with unknown end points in the identification of systems. *Journal of Mathematical Analysis and Applications*, 34(1):26 – 33, 1971.

[11] Benaïm, M. and Le Boudec, J.Y. A class of mean field interaction models for computer and communication systems. *Performance Evaluation*, 65 (11-12):823–838, 2008.

[12] Berghel, H. The Code Red Worm. *Communications of the ACM*, 44(12): 15–19, 2001.

[13] BeyondTrust, Inc. eEye Digital Security. `http://www.eEye.com`.

[14] Billingsley, P. *Probability and Measure*. Wiley-Interscience, 3rd edition, 1995.

[15] Bobbio, A. and Gribaudo, M. and Telek, M. Analysis of Large Scale Interacting Systems by Mean Field Method. In *QEST*, pages 215–224. IEEE Computer Society, 2008.

[16] Bortolussi, L. Hybrid Limits of Continuous Time Markov Chains. In *QEST*, pages 3–12. IEEE Computer Society, 2011.

[17] Bortolussi, L., and Hayden, R.A. Bounds on the deviation of discrete-time Markov chains from their mean-field model. *Performance Evaluation*, pages 736–749, 2013.

[18] Bortolussi, L. and Hillston, J. Fluid Model Checking. In *CONCUR*, volume 7454 of *LNCS*, pages 333–347. Springer, 2012.

[19] Bortolussi, L. and Hillston, J. Checking Individual Agent Behaviours in Markov Population Models by Fluid Approximation. In *SFM*, volume 7938 of *LNCS*, page 113–149. Springer, 2013.

[20] Bortolussi, L. and Hillston, J. and Latella, D. and Massink, M. Continuous approximation of collective systems behaviour: A tutorial. *Performance Evaluation*, 70(5):317 – 349, 2013.

[21] Bortolussi, L. and Lanciani, R. Model Checking Markov Population Models by Central Limit Approximation. In *QEST*, volume 8054 of *LNCS*, pages 123–138. Springer, 2013.

[22] Bradley T.J., Gilmore, S.T., and Hillston, J. Analysing distributed internet worm attacks using continuous state-space approximation of process algebra models. *Journal of Computer and System Sciences*, 74(6):1013 – 1032, 2008.

[23] Calder, M. and Gilmore, S. and Hillston, J. Automatically deriving ODEs from process algebra models of signalling pathways. In *CMSB*, pages 204–215, 2005.

[24] Calzone, L., Fages, F., and Soliman, S. Biocham: An environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.

[25] Cerotti, D. and Gribaudo, M. and Bobbio, A. Disaster Propagation in Heterogeneous Media via Markovian Agents. *Critical Information Infrastructure Security*, 5508:328–335, 2009.

[26] Chaintreau, A., Le Boudec, J.Y., and Ristanovic, N. The age of gossip: spatial mean field regime. In *SIGMETRICS/Performance*, pages 109–120. ACM, 2009.

[27] Han T. Katoen J.P. Chen, T. and A. Mereacre. LTL Model Checking of Time-Inhomogeneous Markov Chains. In *ATVA*, volume 5799 of *LNCS*, pages 104–119. Springer, 2009.

[28] Ciocchetta, F. and Hillston, J. Bio-PEPA for epidemiological models. *Electronic Notes in Theoretical Computer Science*, 261:43–69, 2010.

[29] Darling, R.W.R. Fluid Limits of Pure Jump Markov Processes: a Practical Guide. *ArXiv Mathematics e-prints*, 2002.

[30] Darling, R.W.R. and Norris, J.R. Differential equation approximations for Markov chains. *Probability Surveys*, 5:37–79, 2008.

[31] Deavours, D.D. and Clark, G. and Courtney, T. and Daly, D. and Derisavi, S. and Doyle, J.M. and Sanders, W.H. and Webster, P.G. The Möbius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.

[32] Donzé, A. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *CAV*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[33] Donzé, A. and Clermont, G. and Legay, A. and Langmead, C.J. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. *Journal of Computational Biology*, 17(3):325–336, 2010.

[34] Donzé, A. and Ferrère, T. and Maler, O. Efficient robust monitoring for STL. In *CAV*, volume 8044 of *LNCS*, pages 264–279. Springer, 2013.

[35] Donzé, A. and Maler, O. Robust satisfaction of temporal logic over real-valued signals. In *FORMATS*, volume 6246 of *LNCS*, pages 92–106. Springer, 2010.

[36] eEye Digital Security. Advisories and Alerts: AD20010618. `https://web.archive.org/web/20010805211728/http://www.eeye.com/html/Research/Advisories/AD20010618.html`, 2001.

[37] Eichmann, K. Handlers Diary Blog. `https://isc.sans.edu/diary/diary.php`.

[38] Ekstrom, M. Consistency of generalized maximum spacing estimates. *Scandinavian Journal of Statistics*, 28(2):343–354, 2001.

[39] Fainekos, G.E., and Pappas, G.J. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410 (42):4262 – 4291, 2009.

[40] Falliere, N.,Murchu, L.O., and Chien, E. W32.Stuxnet Dossier. `http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/w32_stuxnet_dossier.pdf`, 2010.

[41] Fisher, D. What is a Botnet? Kaspersky lab daily| Kaspersky lab official blog. `http://reference.wolfram.com/mathematica/guide/Optimization.html`, 2013.

[42] Garetto, M. and Gong, W. and Towsley, D. Modeling malware spreading dynamics. In *INFOCOM*, pages 1869–1879. IEEE Computer Society, 2003.

[43] Gast, N., and Gaujal, B. A Mean Field Model of Work Stealing in Large-scale Systems. In *ACM SIGMETRICS*, pages 13–24. ACM, 2010.

[44] Gillespie, D.T. Exact stochastic simulation of coupled chemical reactions. *The Journal of Physical Chemistry*, 81:2340–2361, 1977.

[45] Gómez-Marín, A. M. and Hernández-Ortíz, J. P. Mean Field Approximation of Langmuir-Hinshelwood CO-Surface Reactions Considering Lateral Interactions. *The Journal of Physical Chemistry*, 117(30):15716–15727, 2013.

[46] Gribaudo, M. and Cerotti, D. and Bobbio, A. Analysis of on-off policies in sensor networks using interacting Markovian agents. In *PerCom*, pages 300–305. IEEE Computer Society, 2008.

[47] Grizzard, J.B., Sharma, V., Nunnery, C., Kang, B.B., and Dagon, D. Peer-to-peer botnets: Overview and case study. In *HotBots*. USENIX Association, 2007.

[48] Hall, A.R. *Generalized Method of Moments*. Advanced Texts in Econometrics. Oxford University Press, 2005.

[49] Hayden, R.A. Convergence of ODE approximations and bounds on performance models in the steady-state. In *PASTA*, pages 49–56, 2010.

[50] Hayden, R.A. Mean Field for Performance Models with Deterministically-Timed Transitions. In *QEST*, pages 63–73. IEEE Computer Society, 2012.

[51] Hayden, R.A. and Bradley, J.T. A fluid analysis framework for a Markovian process algebra. *Theoretical Computer Science*, 411(22-24): 2260–2297, 2010.

[52] Hayden, R.A., and Horvàth, I., and Telek, M. Mean Field for Performance Models with Generally-Distributed Timed Transitions. In *QEST*, volume 8657 of *LNCS*, pages 90–105. Springer, 2014.

[53] Heidelberger, P. Fast simulation of rare events in queueing and reliability models. *ACM Transactions on Modeling and Computer Simulation*, 5: 43–85, 1995.

[54] Henzinger, T.A. and Mateescu, M. and Mikeev, L. and Wolf, V. Hybrid Numerical Solution of the Chemical Master Equation. In *CMSB*, pages 55–65. ACM, 2010.

[55] Hermanns, H., Hahn, E.M., Wachter, B., and Zhang, L. Time-bounded model checking of infinite-state continuous-time Markov chains. *Fundamenta Informaticae*, 95(1):129–155, 2009.

[56] Hillston, J. *A compositional approach to performance modelling.* Cambridge University Press, 1996.

[57] Hillston, J. Fluid Flow Approximation of PEPA models. In *QEST*, pages 33–43. IEEE Computer Society, 2005.

[58] Hillston, J. The Benefits of Sometimes Not Being Discrete. In *CONCUR*, volume 8704 of *LNCS*, pages 7–22. Springer, 2014.

[59] Hillston, J. and Tribastone, M. and Gilmore, S. Stochastic Process Algebras: From Individuals to Populations. *The Computer Journal*, 55(7): 866–881, 2011.

[60] Ingber, L. Simulated annealing: Practice versus theory. *Mathematical and Computer Modelling*, 18(11):29 – 57, 1993.

[61] Kaspersky Lab. Kaspersky Lab provides its insights on Stuxnet worm. `http://www.kaspersky.com/news?id=207576183`, retrieved 7-10-2010.

[62] J.P. Katoen and A. Mereacre. Model Checking HML on Piecewise-Constant Inhomogeneous Markov Chains. In *FMATS*, volume 5215 of *LNCS*, pages 203–217. Springer, 2008.

[63] Kleczkowski, A. and Grenfell, B.T. Mean-field-type equations for spread of epidemics: the 'small world' model. *Physica A: Statistical Mechanics and its Applications*, 274(1–2):355 – 360, 1999.

[64] Kolesnichenko, A., de Boer, P. T., Remke, A., Zambon, E., Haverkort, B.R. Is Quantitative Analysis of Stuxnet Possible? In *Fast Abstracts at QEST*, CTIT Workshop Proceedings WP11-03, pages 9–10. CTIT, University of Twente, 2011.

[65] Kolesnichenko, A., Remke, A., de Boer, P.T. and Haverkort, B.R. Comparison of the mean-field approach and simulation in a peer-to-peer botnet case study. In *EPEW*, volume 6977 of *LNCS*, pages 133–147. Springer, 2011.

[66] Kolesnichenko, A., Remke, A., de Boer, P.T. and Haverkort, B.R. A logic for model-checking mean-field models. In *DSN/PDF*, pages 1–12. IEEE CS Press, 2013.

[67] Kolesnichenko, A., Senni, V., Pourranjabar, A., and Remke A. Applying Mean-field Approximation to Continuous Time Markov Chains. *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems*, LNCS 8453:242–280, 2014.

[68] Kriaa, S., Bouissou, M,. and Pietre-Cambacedes, L. Modeling the Stuxnet attack with BDMP: Towards more formal risk assessments. In *CRiSIS*, pages 1–8. IEEE, 2012.

[69] Kurtz, T.G. Solutions of Ordinary Differential Equations as Limits of Pure Jump Markov Processes. *Journal of Applied Probability*, 7(1):49–58, 1970.

[70] Latella, D., Loreti, M. and Massink, M. On-the-fly Fast Mean-Field Model-Checking. In *TGC*, volume 8358 of *LNCS*, pages 297–314. Springer, 2014.

[71] Le Boudec, J.Y. The Stationary Behaviour of Fluid Limits of Reversible Processes is Concentrated on Stationary Points. Technical report, 2010.

[72] Le Boudec, J.Y., McDonald, D., and Mundinger, J. A generic mean field convergence result for systems of interacting objects. In *QEST*, pages 3–18. IEEE Computer Society, 2007.

[73] Liddle, A. R. Statistical Methods for Cosmological Parameter Selection and Estimation. *Annual Review of Nuclear and Particle Science*, 59(1): 95–114, 2009.

[74] Maler, O. and Nickovic, D. Monitoring temporal properties of continuous signals. In *FORMATS*, volume 3253 of *LNCS*, pages 152–166. Springer, 2004.

[75] Mathworks. Global Optimization Toolbox: User's Guide (r2011b). `http://www.mathworks.nl/help/gads/index.html`, 2011.

[76] Mikeev, L. and Wolf, V. Parameter Estimation for Stochastic Hybrid Models of Biochemical Reaction Networks. In *HSCC*, pages 155–166. ACM, 2012.

[77] Moore, D. and Shannon, C. and Brown, J. Code-Red: a case study on the spread and victims of an Internet worm. In *IMW*, pages 273–284. ACM SIGCOMM/USENIX, 2002.

[78] Myung, I.J. Tutorial on maximum likelihood estimation. *Journal of Mathematical Psychology*, 47(1):90 – 100, 2003.

[79] Nelder, J. A. and Mead, R. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965.

[80] Nickovic, D. and Maler, O. AMT: a property-based monitoring tool for analog systems. In *FORMATS*, volume 4763 of *LNCS*, pages 304–319. Springer, 2007.

[81] University of California. Lawrence Berkeley National Laboratory. `http://www.lbl.gov/`.

[82] Paxson, V. Bro: A System for Detecting Network Intruders in Real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.

[83] Pnueli, A. The temporal logic of programs. In *FOCS*, pages 46 –57. IEEE Computer Society, 1977.

[84] Ramsay, J. O., Hooker, G., Campbell, D. and Cao, J. Parameter estimation for differential equations: a generalized smoothing approach. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 69(5):741–796, 2007.

[85] Remke, A., and Haverkort, B.R. CSL Model Checking Algorithms for Infinite-State Structured Markov Chains. In *FORMATS*, volume 4763 of *LNCS*, pages 336–351. Springer, 2007.

[86] Rizk, A., Batt, G., Fages, F., and Soliman, S. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *CMSB*, volume 5307 of *LNCS*, pages 251–268. Springer, 2008.

[87] Sanders, W.H., and Meyer, J. Stochastic Activity Networks: Formal Definitions and Concepts. *Lectures on Formal Methods and Performance Analysis*, 2090:315–343, 2001.

[88] Silicon Defence. Code Red analysis page. `http://web.archive.org/web/20011031043459/http://www.silicondefense.com/cr/`, 2011.

[89] Silva, M. and Recalde, L. On fluidification of Petri Nets: from discrete to hybrid and continuous models . *Annual Reviews in Control*, 28(2):253 – 266, 2004.

[90] Simon, R.J. *Windows NT Win32 API SuperBible*. Waite Group Press, 1997.

[91] Stefanek, A., Hayden, R.A., and Bradley, J.T. Mean-field analysis of hybrid Markov population models with time-inhomogeneous rates. *Annals of Operations Research*, pages 1–27, 2014.

[92] Stefanek, A., Hayden, R.A., Gonagle, M.M., and Bradley, J.T. Mean-field analysis of Markov models with reward feedback. In *ASMTA*, volume 7314 of *LNCS*, pages 193–211, 2012.

[93] Storn, R. and Price, K. Differential Evolution - a Simple and Efficient Heuristic for Global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11:341–359, 1997.

[94] Symantec. Norton Cybercrime Report: The Human Impact. `http://www.symantec.com/content/en/us/home_homeoffice/media/pdf/cybercrime_report/Norton_USA-Human%20Impact-A4_Aug4-2.pdf`, 2010.

[95] The Cooperative Association for Internet Data Analysis. The UCSD Network Telescope. `http://www.eEye.com`.

[96] The Cooperative Association for Internet Data Analysis. The caida dataset on the code-red worms - july and august 2001. `http://www.caida.org/data/passive/codered_worms_dataset.xml`, 2001.

[97] Tribastone, M. and Gilmore, S. and Hillston, J. Scalable Differential Analysis of Process Algebra Models. *IEEE Transactions on Software Engineering*, 38(1):205–219, 2012.

[98] van Kampen, N.G. *Stochastic Processes in Physics and Chemistry*. North-Holland Personal Library. Elsevier Science, 2011.

[99] van Ruitenbeek, E. and Sanders, W.H. Modeling peer-to-peer botnets. In *QEST*, pages 307–316. IEEE Computer Society, 2008.

[100] Wikia. CodeRed. `http://malware.wikia.com/wiki/CodeRed#`, 2006.

[101] Winkel, B. Parameter estimates in differential equation models for chemical kinetics. *International Journal of Mathematical Education in Science and Technology*, 42(1):37 — 51, 2010.

[102] Witten, I.H, Frank, E., Hall, M.A. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers, 3rd edition, 2011.

[103] Wolfram Research, Inc. Mathematica tutorial. `http://reference.wolfram.com/mathematica/tutorial/IntroductionToManipulate.html`, 2010.

[104] Wolfram Research, Inc. NMinimize. `http://reference.wolfram.com/mathematica/ref/NMinimize.html`, 2014.

[105] Wolfram Research, Inc. Optimization. Mathematica Guide. `http://reference.wolfram.com/mathematica/guide/Optimization.html`, 2014.

[106] Zhang, Z. Parameter estimation techniques: a tutorial with application to conic fitting. *Image and Vision Computing*, 15(1):59 – 76, 1997.

[107] Zhigljavsky, A.A. *Theory of Global Random Search*. Mathematics and its Applications. Springer, 1991.

[108] Zou, C.C., Gong, W., and Towsley, D. Code Red Worm Propagation Modeling and Analysis. In *CCS*, pages 138–147. ACM, 2002.

Acknowledgements

As many other activities in my life, completing this thesis would not be possible without help and support of many people. In this part of the thesis I would like to express my gratitude to all of you.

I want to start with my supervisors, *prof. dr. ir. Boudewijn R. Haverkort, prof. dr. Anne Remke*, and *dr. ir. Pieter-Tjerk de Boer*. I consider myself very lucky because I had an opportunity to work with them. I am thankful to *Boudewijn* for always being inspirational, very professional, extremely helpful, and understanding. Many thanks go to *Anne* for number of discussions (both professional and personal), fresh ideas, and guidance through the challenging academical world. I am grateful to *Pieter-Tjerk* for sharing his broad knowledge on topics related to nearly anything, for many fruitful discussions, for always being precise and careful with the details.

I would like to thank the members of my graduation committee for their comments on thesis thesis. Special thanks to Joost-Pieter Katoen and Hans van den Berg for their detailed comments, and Peter Buchholz for the fruitful discussion.

I want to express my gratitude to all my *teachers* and *mentors*, who inspired me on my way to this PhD project. First of all I would like to mention the person, who laid the foundation of my interest in math and computer science, *Yuriy Vasilievich Lepechin*, National Teacher of Russia, Hero of Labour of the Russian Federation, and my math teacher. He tough me to work hard, to be patient, and to never give up on what I want to do, and I am grateful for each extra hour I spent behind the computer or math books. In addition I would like to thank *prof. dr. Ljudmila A. Bordag* whose experience, understanding and kindness helped me to finish master program in Sweden.

I would like to acknowledge my *DACS colleagues* for making UT a joyous place, where both professional and personal help can be easily found. Thanks to *Hamed* for our long and interesting discussions and for being my yes and hands in the office when I was absent, to *Idilio*, *Rafael*, and *Giovane* for all

the laughs we had in our old office. Many thanks to all of you, *Aiko*, *Hans*, *Geert*, *Anna*, *Martijn*, *Ramin*, *M&M (Mozhdeh and Morteza)*, *Sarwar*, *Bjorn*, *Luuk*, *Jair*, *Jessica*, *Roland*, *Freek*, *Rick*, *Ricardo*, *Karol*, *Wouter*, *Georgios*, *Mataijs*, *Daniel*, *Martijn*, *Tiago*, *Desi*, and *Stephan* for your help, for sharing your knowledge and experience, for every smile during DACS uitjes and retreats, and for making every day at DACS exciting. I would like to thank our secretary *Jeanette Rebel-de Boer* for her professionalism and ability to help in any situation, and for practically being our Dutch mammy.

I would also like to thank *Jair* and *Victoria* for agreeing to be my paranymphs. You both are amazing people and very good friends. I am thankful for having you in my life.

Talking about amazing people, I want to mention my *old* and *new friends*. I would like to thank *Galochka*, *Natashechka*, *Chukochka*, *Kasya*, *Stramilochka* and *Sanek* for being with me through more then 10 (or even 15) years of my life, for sharing many-many funny, interesting, sometimes crazy or even dangerous stories, for always being supportive and understanding. I am thankful to *Zhenechka*, *Katjushka*, *Lorik* and *Zmaray* for the happy times in Groningen and outside it. I would like to thank all the nice people I met in Enschede during these happy 4,5 years, to mention few names: *Sanjka*, *Ksenja*, *Olja*, *Anton*, *Nuno*, *Zambon*, *Oscar*, *Tanja*, *Andrey*, *Regina*, *Aydar* and many-many more. Thank you for all the smiles you gave me. Additionally I would like to thank Angel, who helped me to keep my body and mind balanced for the last two years..

I would like to express my biggest love and gratitude to my *family*. To my mum *Irina* and dad *Victor* for always being supportive and loving parents, for trusting me and giving me all the opportunities they could to fulfil my dreams. I would like to thank my granny *Tonja* for sharing her love and wisdom, for being my role-model since I was a little girl. To my parents in law *Valentina* and *Nikolay*, my sister *Nastja*, and my big extended family for their love and support. And last but not least, my husband *Vitechka*, thank you for your patience, for your enormous help, and for making me a happy loved wife!

Благодарности

Как и многое другое в моей жизни, завершение этой диссертации было бы невозможно без помощи и поддержки многих людей. В этой части диссертации я бы хотела выразить мою благодарность всем этим людям.

Я хочу начать со слов благодарности в адрес моих научных руководителей: *проф. Баудевайн Хаферкорт*, *проф. Аннэ Ремке*, и *др. Питер-Тьерк де Бур*. Я очень счастлива, что имела возможность работать с вами. Я благодарна *Баудевайну* за то, что он всегда был вдохновляющим, профессиональным и понимающим руководителем. Спасибо *Аннэ* за ее прекрасные идеи, за ее помощь и поддержку в профессиональных и личных вопросах. Я благодарна *Питеру-Тьерку* за то, что он всегда был готов поделиться своими обширными знаниями во всех областях, за множество интересных и плодотворных дискуссий и за его внимание к деталям.

Я хотела бы поблагодарить членов комиссии за их комментарии к данной диссертации. Особая благодарность Юсту-Питеру Катуну и Хансу ван ден Бергу за очень подробные и интересные комментарии и Питеру Буххолцу за плодотворную беседу.

Я хочу поблагодарить всех моих учителей и руководителей, которые помогли мне на моем пути к докторской. Мой первоначальный интерес к серьёзной математике и информатике был заложен Юрием Васильевичем Лепехиным, Народным учителем России, Героем Труда Российской Федерации. Он научил меня быть настойчивой в достижении цели, много трудиться и никогда не сдаваться, и я бесконечно благодарна ему за каждый дополнительный час, что я провела за компьютером или задачкой. Также я хотела бы поблагодсарить *проф. Людмилу Алексеевну Бордаг*, чьи знания и опыт, понимание и доброта помогли мне закончить магистратуру в Швеции.

Я бы хотела поблагодарить моих *коллег* за то, что они сделали университет местом, где всегда можно рассчитывать на профессиональную помощь и личную поддержку. Спасибо *Хамеду* за наши долгие и интересные

разговоры, *Идилио, Рафаелю* и *Джоване* за все смешные ситуации, которые происходили в нашем старом офисе. Огромное спасибо *Айко, Хансу, Херту, Анне, Мартайну, Рамину, M&M (Можде и Мортезе), Сарвару, Бъёрну, Люку, Жаиру, Джессике, Роланду, Фрейку, Рику, Рикардо, Каролу, Ваутеру, Джорджиусу, Матайсу, Даниелю, Мартайну, Тъяго, Дези* и *Стэфану* за их помощь, за то,что делились своими знаниями и за каждую улыбку во время групповых выездов. Я также хочу поблагодарить секретаря группы *Жанет Ребел-де Бур* за ее профессионализм и умение помочь в любой ситуации и за то, что она заботится о нас, как наша общая голландская мама.

Мне также очень хочется поблагодарить *Викторию* и *Жаира* за то, что они согласились быть моими паранимфами. Вы оба просто потрясающие люди и очень хорошие друзья. Я очень благодарна вам за то, что вы есть в моей жизни.

И если уж я начала говорить о потрясающих людях, я хочу сказать несколько слов о моих старых и новых друзьях. Я хочу поблагодарить *Галочку, Наташечку, Чукочку, Касю, Страмилочку* и *Санька* за то, что они были со мной долгие 10 (а некоторые и 15) лет, за огромное количество смешных, интересных, иногда сумасшедших или даже опасных историй, связывающих нас, за их поддержку и понимание. Я благодарна *Женечке, Катюшке, Лорочке* и *Змараю* за счастливые моменты, которые мы вместе пережили в Гронингене и за его пределами. Я бы хотела поблагодарить замечательных людей, которых я встретила в Энсхеде за эти 4,5 года: *Саньку, Ксеню, Олю, Антона, Нуно, Замбона, Оскара, Таню, Андрея, Регину, Айдара* и многих-многих других. Спасибо за все те улыбки, что вы мне подарили.

Я бы хотела выразить свою огромную любовь и благодарность моей *семье*. Моим родителям *Ирине* и *Виктору* я благодарна за их любовь и поддержку, за их доверие и огромную помощь во всех моих начинаниях. Я бы хотела поблагодарить мою бабушку *Тоню* за то, что она делилась со мной своей мудростью, дарила мне свою любовь и за то, что она до сих пор является примером для подражания для меня. Моим вторым родителям *Валентине* и *Николаю*, моей сестре *Насте* и всей наше большой *семье* я благодарна за их любовь и поддержку. И последним по списку, но не по значению, я бы хотела поблагодарить моего мужа *Витечку* за его терпение, поддержку, огромную помощь, за его любовь и за то, что он делает меня счастливой женой.

About the author

Анна Викторовна (Аня) Колесниченко

Figure 1: Photo was taken by Jacob Staley for the spacial eddition of the UT news (December 2012). Style and article: Nissrin Kissi.

Anna (Anja) Kolesnichenko was born in Volgograd, USSR, on the 13th of May 1985. She finished high-school with the specialization in mathematics and computer science in 2002. Afterwards she started her hight education in faculty of Mathematics, in Volgograd State University (VolSU). She graduated from VolSU with the speciality in Applied Mathematics and Computer Science in 2007. She spent one year in Halmstad, Sweden, where she obtained a master degree in Financial mathematics in 2008. She started her PhD research at Centre of Telematics and information Technology (CTIT), University of Twente, as a member of the Design and Analysis of Communication Systems (DACS) group on 1^{st} of April 2010. During her PhD work she was involved in MATMaM and ROCKS projects, and has authored several international publications and served as a reviewer for international conferences and workshops. Below is her Curriculum Vitae and a list of publications.

Experience

Vocational

2010–2014 **PhD researcher** at Design and Analysis of Communication Systems, University of Twente, the Netherlands

2008–2010 **PhD researcher** at Groningen Bioinformatics Center, University of Groningen, the Netherlands

2008 **Assistant specialist** at Department of documentary operations and exchange control, Foreign Trade Bank (VTB), Russia

2007–2008 **Intern** at Group Risk Management (RC market Risk and P&L Unit), HSH Nordbank AG, Germany

Miscellaneous

2002 - 2005 **Instructor** at Summer school for gifted kids "Integral", Volgograd, Russia.

2002 - 2006 **Intern** at Social Opinion Survey companies, Volgograd, Russia.

Education

2006–2007 **Master of Financial Mathematics** at Halmstad University, Sweden.

2007 **University Diploma with Specialization in Mathematics (pre Master)** at Halmstad University, Sweden.

2002–2007 **Specialist (eq. Master) of Applied Mathematics and Computer Science** at Volgograd State University, Russia.

1997–2002 **Math-specialized graduate** at Secondary School 78, Volgograd, Russia.

Publications

2014 Kolesnichenko, A., Senni, V., Pourranjabar, A., and Remke, A. **Applying Mean-Field Approximation to Continuous Time Markov Chains.** *Stochastic Model Checking. Rigorous Dependability Analysis Using Model Checking Techniques for Stochastic Systems.* LNCS 8453:242-280.

2013 Kolesnichenko, A. and de Boer, P.T. and Remke, A. and Haverkort, B.R. **A logic for model-checking mean-field models.** In: 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks, pp. 1-12. IEEE.

2012 Kolesnichenko, A. and Remke, A. and de Boer, P.T. and Haverkort, B.R. **A logic for model-checking of mean-field models.** Technical report. Accepted as presentation report for Tenth Workshop on Quantitative Aspects of Programming Languages.

2011 Kolesnichenko, A. and de Boer, P.T. and Remke, A.K.I. and Zambon, E. and Haverkort, B.R. **Is Quantitative Analysis of Stuxnet Possible?** In: 8th International Conference on Quantitative Evaluation of SysTems: Fast Abstracts, pp. 9-10. CTIT Workshop Proceedings WP11-03. Centre for Telematics and Information Technology University of Twente.

2011 Kolesnichenko, A. and Remke, A. and de Boer, P.T. and Haverkort, B.R. **Comparison of the mean-field approach and simulation in a peer-to-peer botnet case study.** In: 8th European Performance Engineering Workshop, pp. 133-147. LNCS 6977. Springer.