Practical Applications of Performance Modelling of Security Protocols Using PEPA

Thesis by

Yishi Zhao

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

Newcastle University

Newcastle upon Tyne, UK

2011

To my parents

&

To Xin Jing

Acknowledgments

Firstly and foremost, I would like to thank my supervisor, Dr. Nigel Thomas, for his invaluable guidance, supervision, and advice. I also appreciate Prof. Aad van Moorsel and Dr. Nick Cook in my thesis committee for discussing the direction of my research.

Secondly I'd like to thank my examiners: Dr. Jeremy Bradley of Imperial College London and Prof. Aad van Moorsel for the valuable comments in the viva.

Many thanks then go to Allan Clark, Adam Duguid, Stephen Gilmore and Micro Tribastone from University of Edinburgh for clarifying aspects of the PEPA Eclipse Plug-in and the apparent rate.

I also would like to thank my friends Yingjie He, Rachel Zhang, Xiao Chen and Ahmad Salah El Ahmad for making my four years PhD life vigorous.

Finally, I would like to give the credits to my parents and my fiancee for their love, support and encouragement.

Abstract

Trade-off between security and performance has become an intriguing area in recent years in both the security and performance communities. As the security aspects of security protocol research is fully-fledged, this thesis is therefore devoted to conducting a performance study of these protocols. The long term objective is to translate formal definitions of security protocols to formal performance models automatically, then analysing by relevant techniques. In this thesis, we take a preliminary step by studying five typical security protocols, and exploring the methodology of construction and analysis of their models by using the Markovian process algebra PEPA. Through these case studies, an initial framework of performance analysis of security protocol is established.

Firstly, a key distribution centre is investigated. The basic model suffers from the commonly encountered state space explosion problem, and so we apply some efficient solution techniques, which include model reduction techniques and ordinary differential equation based fluid flow analysis. Finally, we evaluate a utility function for this secure key exchange model. Then, we explore two non-repudiation protocols. Mean value analysis has been applied here for a class of PEPA models, and it is compared with an ODE approximation. After that, an optimistic non-repudiation protocol with off-line third trust party is studied. The PEPA model has been formulated using a concept of multi-threaded servers with functional rates. The final case study is a cross-realm Kerberos protocol. A simplified technique of aggregation with an ODE approximation is performed to do efficient analysis. All these modelling and analysis methods are illustrated through numerical examples.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation and Aims

One of the more intriguing areas of system engineering to emerge over recent years has been the study of the performance overhead introduced by making a system secure. According to Stallings [59]:

> *"The more frequently session keys are exchanged, the more secure they are, because the opponent has less cipher text to work with for any given session key. On the other hand, the distribution of session keys delays the start of any exchange and places a burden on network capacity. A security manager must try to balance these competing considerations in determining the lifetime of a particular session key."*

It is clear that as more functionality is added to a system, more execution time is required to complete the additional tasks involved. However, in the case of security, the benefit accrued from any additional overhead is not easy to quantify and so it is very hard for the performance engineer to argue a particular performance target should take precedence over a security goal. One area where alternative secure solutions exist is in cryptography, where there may be a choice of algorithm, or even a choice of key length, which will greatly influence the

performance of the system. In many situations performance also contribute to the security rather than just speeding up or slowing down the system. Under a poor performance, the legitimate protocol actions might lead the system to leak restricted information. If the scalability of system under heavy load, the security protocol is potentially vulnerable by timing attacks, and it is also very likely to expose a functional flaw. Moreover, if we consider a user send or exchange some restrict information under a security protocol which has a very good security but execute slowly. The user is likely to lose his patient to wait, or do not have enough time to wait. Under the situation, the user might avoid the security protocol and send his information by insecure means. Finally recalling our quoted paragraph above, if a client holding a session key to wait a very long time in a key exchange protocol, the hackers may obtain more cipher text and also has more time to work on cracking the key. From the motivation, cryptographic protocols are one of the few areas of security to have received much attention from the both performance and security community. To date this work has been largely limited to measurement and has not addressed the underlying causes of delay which might be understood by modelling or detailed code analysis.

If a trade-off problem is being faced by a security manager, how to choose or how to use a protocol becomes a major issue. Security properties of common security protocols are generally well understood. However, investigation of performance aspects of the security protocols is usually ignored in the development stage. Hence, security managers need to study a new protocol every time a trade-off analysis is required. This motivated us to investigate the idea that whether we can translate a security protocol from a security aspect model to a performance model, and then, manipulate the performance model to fit different environment, and finally choose suitable analysis techniques to analyse the protocol to obtain performance metrics, *e.g.* utilisation, response time, queue length, etc. (see Figure 1.1) As the security protocols are usually modelled by process algebra, obviously, the best type of performance model formalism as the translation is stochastic process algebra. This translation can keep the formality of the security protocols and can also be formally analysed and reasoned. That is also one of the

major reasons for using PEPA (this modelling language introduced in Chapter 2) in this work. O'Shea has been working on the translation of security protocols, and a naive version has been implemented in [49].

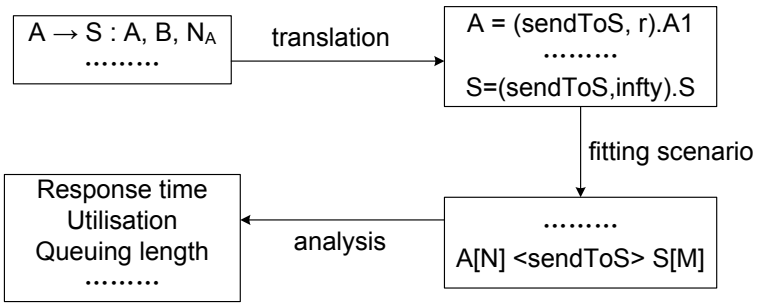

Figure 1.1: An overview of performance analysis of security protocol

The work in this thesis focuses on the later part of our initial idea. After obtaining a translated performance model, which is assumed to use a stochastic process algebra, we then investigate how to manipulate the model to fit different scenarios and what kind of analysis techniques can be employed to different types of protocols. By doing this, a security manager could easily study the performance aspects of security protocols, even if the translation is not automatic. In addition, because the state space explosion becomes a major issue in formal analysis, we primarily focus on efficient and scalable solutions for analysis. In this thesis, four different kinds of security protocols have been investigated. Several modelling styles and analysis techniques have been explored with these protocols, and the advantages and disadvantages of these techniques has been addressed. Through these cases studies, we aim to produce a initial version of framework by which arbitrary security protocols can be analysed for performance and possibly the trade-off between performance and security for security managers. We propose a rough work flow that is followed for these case studies in Figure 1.2.

The first step is modelling the system by PEPA. We then simplify the model to make it easier for analysis. After that, the model is analysed by some techniques. Once the metrics (such as queue length and response time) are obtained, we utilise it in a cost function analysis. By the cost function, one can conduct scalability

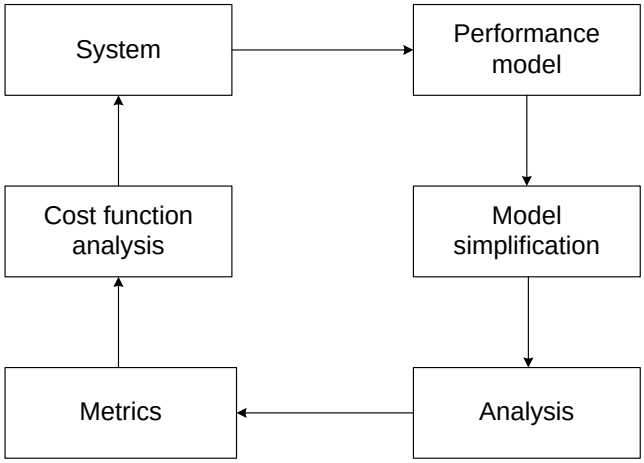

Figure 1.2: The work flow of performance analysis in context

management and capacity planning to the system.

1.2 Outline of thesis

The remainder of this thesis is organised as follows:

Chapter 2 (Literature Review) This chapter generally reviewed the technical background which includes the PEPA language and its steady state analysis techniques, especially, solutions for large scale methodologies, and the case studies about performance analysis of security related applications.

Chapter 3 (Key Distribution Centre) In this chapter we explore performance of a model of a key distribution centre. The basic model suffers from the commonly encountered state space explosion problem, and so we apply some efficient techniques, including model simplification (partial evaluation) and ODE based fluid flow analysis, to solve it. Finally, we evaluate a utility function of this secure key exchange model. The analysis techniques are explored through numerical results. The approximation is compared with discrete event simulation and the ODE results are verified by stochastic simulation.

Chapter 4 (Non-repudiation Protocols ZG1&ZG3) In this chapter we study the overhead introduced by secure functions in considering two of non-repudiation protocols. Again, considering scalable analysis and following the previous study of performance modelling on the KDC, *partial evaluation* and fluid flow approximation based on ordinary differential equations (ODEs) have been chosen. Then, traditional *mean value analysis* (MVA) is applied to PEPA models here. Additionally, *Functional Rates* has been adopted in the second model (ZG3) to avoid unintended system behaviour. The models are analyzed numerically and results derived from mean value analysis are compared with the ODE solution.

Chapter 5 (An Optimistic Fair Exchange Protocol) This chapter studies an optimistic fair exchange e-commerce protocol, which is in fact a non-repudiation protocol with an off-line *third trust party* (TTP). Two PEPA models are formulated in two cases: without TTP (no misbehaviour) and with TTP (misbehavior exists). A multiple threads modelling concept and a new version of functional rates has been adopted here. Following the case studies in the last two chapters, *partial evaluation* and the most efficient ODE approximation are employed, and concept of functional rates for ZG3 is utilised in the model of the TTP involving. The analysis is explored numerically and compared with stochastic simulation.

Chapter 6 (Kerberos Protocol) In this chapter an authentication protocol, Kerberos, is investigated. To consider the scenario of the multi-realms environment, a new type of simplification is applied. The simplified model is equivalent to original model only in terms of the steady state distribution, and it is analysed numerically by our previously studied fluid flow approximation, then, again it is verified by stochastic simulation.

Chapter 7 (Conclusion and Further Work) This chapter gives a conclusion of this thesis and proposes further work.

1.3 Publication History

Journals:

Efficient solutions of a PEPA model of a key distribution centre, Y. Zhao and N. Thomas, in:*Performance Evaluation* **67**(8), pp 740-756, Elsevier B.V., 2010.

Mean value analysis for a class of PEPA models, N. Thomas and Y. Zhao, accepted by *Computer Journal*. doi: 10.1093/comjnl/bxq064 (extended version of EPEW 2009).

Conferences and workshops:

Modelling secure secret key exchange using stochastic process algebra, Y. Zhao and N. Thomas, in:*Proceedings of 23rd UK Performance Engineering Workshop*, Edge Hill University, 2007.

Approximate solution of a PEPA model of a key distribution centre, Y. Zhao and N. Thomas, in:*Performance Evaluation - Metrics, Models and Benchmarks: SPEC International Performance Evaluation Workshop*, LNCS 5119, Springer Verlag, 2008.

Fluid flow analysis of a model of a secure key distribution centre, N. Thomas and Y. Zhao, in:*Proceedings 24th Annual UK Performance Engineering Workshop*, Imperial College London, 2008.

A cost model analysis of a secure key distribution centre, Y. Zhao and N. Thomas, in:*Proceedings of International Symposium on Trusted Computing*, IEEE Computer Society, 2008.

Mean value analysis for a class of PEPA models, N. Thomas and Y. Zhao, in:*6th European Performance Engineering Workshop*, Springer-Verlag, 2009.

Experiences of using the PEPA performance modelling tools with a non-repudiation protocol, Y. Zhao and N. Thomas, in:*European Simulation and Modelling Conference*, EUROSIS-ETI, pp 95-100, 2009.

Comparing methods for the efficient analysis of PEPA models of non-repudiation protocols, Y. Zhao and N. Thomas, in:*15th International Conference on Parallel and Distributed Systems*, December 8-11, Shenzhen, Guangdong, China, pp 821-827, IEEE Computer Society, 2009.

A simplified solution of a PEPA model of kerberos protocol, in : *The 1st International Workshop on Service Oriented QoS Management from Theory to Practice*, IEEE Computer Society, 2011.

1.4 Contributions

The main contributions are summarised following:

- The development of partial evaluation techniques that can be used to permit fluid analysis in Chapter 3. This is a general simplification technique which can reduced the system component in the model, and can be applied by a range of protocols in some scenarios.
- We identify the diverge point (N^*) in fluid flow analysis with PEPA, and discuss engineering approach to tackle engineering approximation of ODE analysis in Chapter 3.
- We identify that in closed queueing network model, the ODE analysis of its equivalent PEPA model is exactly the same to the asymptotic bounds of it.
- The Mean Value Analysis has firstly been applied to a range of PEPA models, and it is detailed compared with fluid flow analysis in Chapter 2 and 4.
- Two version of non-repudiation protocols are analysed and compared in terms of their trade-off between performance and security in Chapter 4.
- We employ functional rates in Chapter 4 in PEPA model to solve an unintended completion between different service actions. It is also can be

considered as a simplification, as it avoids to specify the detailed behaviour of the non-repudiation server.

- We apply multi-threads modelling form for an optimistic fair exchange protocol in Chapter 5.
- The optimistic fair exchange protocol with misbehaviour and without misbehaviour are explored and compared in Chapter 5.
- An aggregation technique has been applied to Keberos protocol for efficient analysis in Chapter 6.
- Utility function is proposed and analysed for all cases studies from Chapter 3 to Chapter 6.

Chapter 2

Literature Review

This chapter reviews several literatures related to this thesis. Section 2.1 introduces the PEPA modelling language used throughout this thesis, and solution methods for large scale systems. Solutions like simulation, fluid flow analysis and mean value analysis are described precisely, because they are applied to the models in the subsequent chapters. Other solution methods are only given a brief description. The analysis techniques described here all give rise to steady state solutions. The mean value analysis is the novel contribution of the work presented here. Furthermore, some related work are surveyed in Section 2.2.

2.1 Specification and Solution of Large PEPA Models

2.1.1 PEPA

In this thesis we model protocols using the Markovian process algebra PEPA. This approach has a number of advantages over a direct approach using Markov chains. As a formal specification, a PEPA model can be derived automatically from, and compared automatically with, formal definitions of the protocol we

are modelling. Functional properties of the model, such as deadlock freeness, can also be checked automatically. These attributes of the model specification are particularly important in the field of security, where correctness is vital if security properties are to be maintained. Furthermore, the analysis of the model we are considering here is based on formulating progressively simplified versions of the model. Because of the formal nature of the specification we can apply formal transformations to the model based on known concepts of equivalence. Therefore we know that the approximate model we derive shares certain properties with the original model. In brief, we know, and can prove, that the approximation is still a valid model of the original protocol. This would not be possible if we simply chose the approximation by some expert intuition or arrived at it by some less formal means. Furthermore, by taking this approach we can apply the same set of techniques to other related protocols, potentially automating the solution.

A formal presentation of PEPA is given in [33], in this section a brief informal summary is presented. PEPA, being a Markovian Process Algebra, only supports actions that occur with rates that are negative exponentially distributed. Specifications written in PEPA represent Markov processes and can be mapped to a continuous time Markov chain (CTMC). Systems are specified in PEPA in terms of *activities* and *components*. An activity (α, r) is described by the type of the activity, α, and the rate of the associated negative exponential distribution, r. This rate may be any positive real number, or given as unspecified using the symbol $\top$.

2.1.1.1 Syntax

The syntax for describing components is given as:

$$(\alpha, r).P \mid P + Q \mid P/L \mid P \underset{\mathcal{L}}{\bowtie} Q \mid A$$

The component $(\alpha, r).P$ performs the activity of type α at rate r and then behaves

like P. The component $P + Q$ behaves either like P or like Q, the resultant behaviour being given by the first activity to complete.

The component P/L behaves exactly like P except that the activities in the set L are concealed, their type is not visible and instead appears as the unknown type τ.

Concurrent components can be synchronised, $P \underset{\mathcal{L}}{\bowtie} Q$, such that activities in the cooperation set $\mathcal{L}$ involve the participation of both components. In PEPA the shared activity occurs at the slowest of the rates of the participants and if a rate is unspecified in a component, the component is passive with respect to activities of that type. $A \stackrel{def}{=} P$ gives the constant A the behaviour of the component P.

In this thesis we consider only models which are cyclic, that is, every derivative of components P and Q are reachable in the model description $P \underset{\mathcal{L}}{\bowtie} Q$. Necessary conditions for a cyclic model may be defined on the component and model definitions without recourse to the entire state space of the model.

2.1.1.2 Apparent rate and Operational semantic

The **Apparent rate** is defined as the externally observed rate of that type of activity. Consider the case where N clients waiting in a queue for K $(K < N)$ servers with single service rate μ, this is the same as that the N clients are served by a single server with the service rate $K\mu$. More formally, the *apparent rate* of action type α in a component P, denoted by $r_\alpha(P)$, is the sum of the rates of all activities of type α which can be performed by P. It is formulated as follows:

1. $r_\alpha((\beta, r).P) = \begin{cases} r & if \beta = \alpha \\ 0 & if \beta \neq \alpha \end{cases}$
2. $r_\alpha(P + Q) = r_\alpha(P) + r_\alpha(Q)$
3. $r_\alpha(P/L) = \begin{cases} r_\alpha(P) & if \alpha \notin L \\ 0 & if \alpha \in L \end{cases}$

4. $r_\alpha(P \underset{L}{\bowtie} Q) = \begin{cases} r_\alpha(P) + r_\alpha(Q) & if \alpha \notin L \\ min(r_\alpha(P), r_\alpha(Q)) & if \alpha \in L \end{cases}$

The **Operational semantic** of PEPA defines the rules of how the system enables. Figure 2.1 formally presents the *operational semantics* of PEPA, in which the operational rules are given and that if the transition above the inference line can be inferred, then we can infer the transition below the line. Following the rules, a *labelled transition system* can be generated which represents an underlying Markov chain for the PEPA model.

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F} (\alpha \notin L) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'} (\alpha \notin L)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'} (\alpha \in L) \quad \text{where } R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L} (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L} (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} (A \stackrel{def}{=} E)$$

Figure 2.1: Operational Semantic of PEPA

2.1.1.3 A simple example

Here is a simple and well known example following [33] (page 25):

$$
\begin{aligned}
Process_0 &\stackrel{def}{=} (use, r_1).Process_1 \\
Process_1 &\stackrel{def}{=} (task, r_2).Process_0 \\
Resource_0 &\stackrel{def}{=} (use, r_3).Resource_1 \\
Resource_1 &\stackrel{def}{=} (update, r_4).Resource_0 \\
System &\stackrel{def}{=} Process_0 \underset{use}{\bowtie} Resource_0
\end{aligned}
$$

The model describes a simple system in which a process executes some task cyclically, and it must utilize a resource firstly to finish the task. Meanwhile, the resource is only not available when being accessed by the process, and then reset the state for other processes. Therefore, the *Process* component will undertake two activities with two states ($Process_0$ and $Process_1$) in the model. $Process_0$ *use* with rate r_1, and $Process_1$ *task* with rate r_2. Similarly, $Resource_0$, represents a state of *Resource*, which carries out action *use*, in cooperation with *Process*, and another state $Resource_1$ *update* at rate r_4. Finally, the system is interpreted by initial states ($Process_0$, $Resource_0$) for each component cooperating over action *use*. This example, referred to as the Process/Resource Model, is utilized to illustrate some of the solutions throughout this chapter.

2.1.2 Notions of equivalence

Concept of equivalence is usually used for investigating the relationship between two models in process algebra, and it is also very useful to simplify the complex model to a neat version for analysis. This Section introduces several notions of equivalence of PEPA that proposed by Hillston in [33].

2.1.2.1 Bisimulations

The bisimuation is based on the observable behaviour. Two components are considered bisimulation if their behaviour appears the same to a external observer, denoted by $\sim$. Therefore, if $P \sim Q$, an action performed by P must match the related action with Q, and also the subsequent actions must also be matched. The formal definition can be referred to [33]. We use a simple example to demonstrate the concept here. Two components P and Q are defined in PEPA as follows.

$$\begin{aligned} P &\stackrel{def}{=} (task1, r_1).P_1 \\ P_1 &\stackrel{def}{=} (task2, r_2).P \end{aligned}$$

$$\begin{aligned} Q &\stackrel{def}{=} (task1, r_1).Q_1 \\ Q_1 &\stackrel{def}{=} (task2, r_2).Q_2 \\ Q_2 &\stackrel{def}{=} (task1, r_1).Q_3 \\ Q_3 &\stackrel{def}{=} (task2, r_2).Q \end{aligned}$$

P perform $task1$ that matches Q, then its derivative P_1 matches Q_1. After that P goes back to perform $task1$ again, however, it matches Q_2 as the subsequent derivative of Q_1. Therefore $P \sim Q$.

There is also a weak bisimulation concept, which allows the internal actions are not in the set of observable actions. Let us define τ as the internal action that can not be seen by a external observer, and redefine P as follows:

$$\begin{aligned} P &\stackrel{def}{=} (task1, r_1).P_1 \\ P_1 &\stackrel{def}{=} (\tau, r).P_2 \\ P_2 &\stackrel{def}{=} (task2, r_2).P \end{aligned}$$

Now, P and Q are considered as weak bisimulation. Although there is not matched action of Q to τ, a external observer can not observe τ of P as well.

2.1.2.2 Isomorphism and weak isomorphism

Different from bisimulation, isomorphism is not observation based, and it is a very strong notion of equivalence, denoted by $=$. Two components are considered as isomorphism if all the derivatives of one component correspond to another, and they can perform exactly the same activities. Therefore, we must compare their derivation graphs to show that two components are isomorphism.

To set a example, let us define component S as:

$$\begin{aligned} P &\stackrel{def}{=} (\alpha, r_1).P_1 \\ P_1 &\stackrel{def}{=} (\beta, r_2).P \end{aligned}$$

$$\begin{aligned} Q &\stackrel{def}{=} (\alpha, r_1).Q_1 \\ Q_1 &\stackrel{def}{=} (\gamma, r_3).Q \end{aligned}$$

$$S \stackrel{def}{=} P \underset{\alpha}{\bowtie} Q$$

and define component R as:

$$\begin{aligned} R &\stackrel{def}{=} (\alpha, r_1).R_1 \\ R_1 &\stackrel{def}{=} (\beta, r_2).(\gamma, r_3).R \\ &\quad +(\gamma, r_3).(\beta, r_2).R \end{aligned}$$

By comparing the derivation graph of component S (Figure 2.2) and R (Figure

2.3), we can find that their derivatives are match and they are able to carry out exactly the same activities. Therefore, S and R are isomorphic ($S = R$).

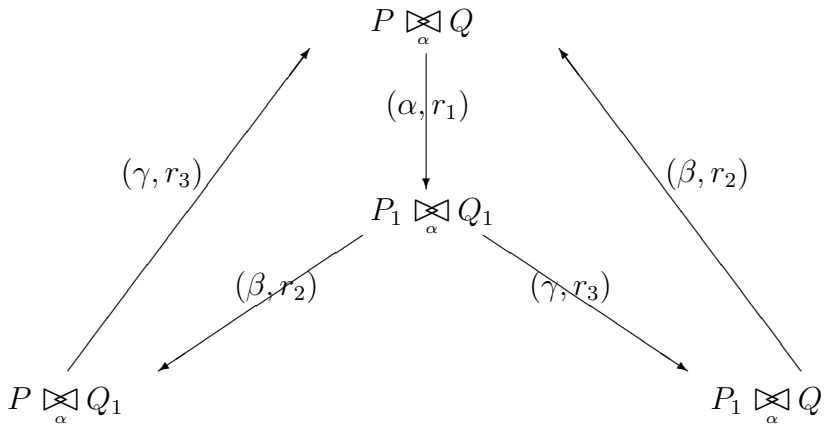

Figure 2.2: The derivation graph of S

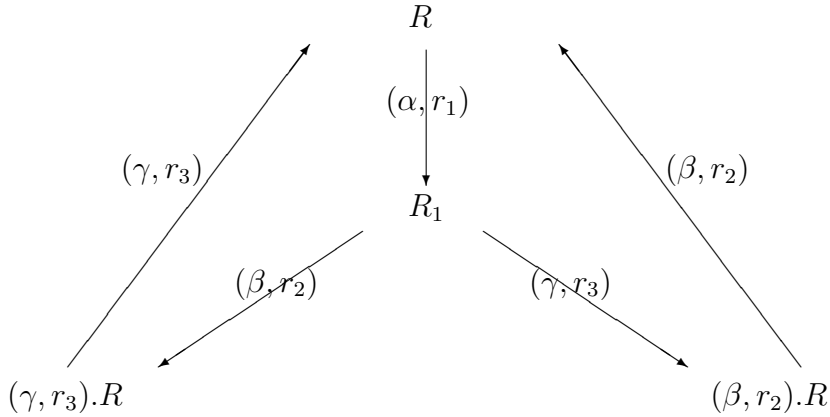

Figure 2.3: The derivation graph of R

Weak isomorphism has the same definition for two components as isomorphism expect for their internal actions, τ, denoted by $\approx$. Therefore, τ type activities can have the different details if two components are weak isomorphism. Let us define component A and B as follows:

$$A \stackrel{def}{=} (\tau, r_1).(\tau, r_2).A$$

$$B \stackrel{def}{=} (\tau, r_3).B$$

A and B are not match for the internal action τ. However, once we set r_3 as $\frac{1}{\frac{1}{r_1}+\frac{1}{r_2}}$, component B now has a single τ activity of equivalent total duration and they (A and B) are have the same visible behaviour. Hence, A and B are now weak isomorphic ($A \approx B$).

2.1.2.3 Strong equivalence

Another equivalence notion of PEPA is strong equivalence. According to [33], "Two PEPA component are strong equivalent if there is an equivalence relation between them such that, for any type α, the total conditional transition rates from those components to any equivalence class, via activities of this type, are the same." This kind of equivalence has not been used in context, one can refer to [33] for details.

2.1.3 CTMC based methods

2.1.3.1 Derivation of CTMC

It is easy to understand from the definition of PEPA that this formalism is a high level modelling language to a Continues-Time Markov Chain. To solve a PEPA model, therefore, it is necessary to derive an equivalent CTMC as a first step. The CTMC is represented by a derivation graph which is more clearly understood by an analyst. A derivation graph consists of nodes, which represent the components and its derivatives, and arcs, which illustrate possible transitions between the corresponding components and labeled by the action type and action rate. According to [33] (page 32-33) and the operational semantics of PEPA, the derivation graph of the Process/Resource Model given in Figure 2.4 can be

drawn. It is essential to note that this only can be done manually with a small CTMC and it has been assumed to have a finite state space.

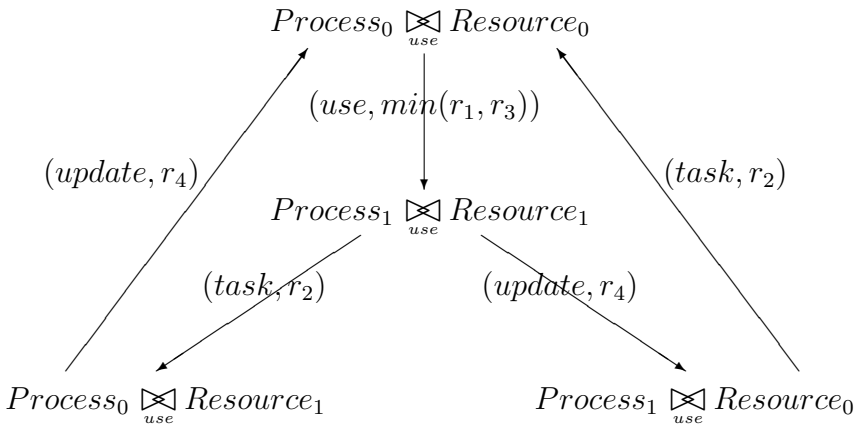

Figure 2.4: The underlying CTMC of Process/Resource Model

2.1.3.2 Matrix solution

Given the underlying CTMC derived from a PEPA model depicted in the derivation graph, we can solve the model in a number of ways. The most direct and obvious way is solving a matrix to give the steady state distribution. [61] This matrix, which is termed the *infinitesimal generator matrix* and denoted by $\mathbf{Q}$, is an $n \times n$ matrix that characterizes the n states of the CTMC. The elements q_{ij} represent the transition rates of the system from state i to j, in the jth column of the ith row of the matrix. The diagonal elements are chosen to ensure that the sum of the elements in every row is zero, hence, $q_{ii} = -\sum_{j \in S, j \neq i} q_{ij}$, where S is state space. Once $\mathbf{Q}$ is obtained, the following global balance equation can be utilised to calculate steady state distribution (π):

$$\pi \mathbf{Q} = 0$$

Where $\sum \pi_i = 1$.

To consider the small example Process/Resource model, firstly, four states can

be identified:

$$X_0 = Process_0 \underset{use}{\bowtie} Resource_0$$
$$X_1 = Process_1 \underset{use}{\bowtie} Resource_1$$
$$X_2 = Process_0 \underset{use}{\bowtie} Resource_1$$
$$X_3 = Process_1 \underset{use}{\bowtie} Resource_0$$

Then, the *infinitesimal generator matrix* Q is:

$$\mathbf{Q} = \begin{pmatrix} -min(r_1, r_3) & min(r_1, r_3) & 0 & 0 \\ 0 & -(r_2 + r_4) & r_2 & r_4 \\ r_4 & 0 & -r_4 & 0 \\ r_2 & 0 & 0 & -r_2 \end{pmatrix}$$

Finally, by solving the global balance equation, the steady state distribution (π) is obtained:

$$\pi(X_0) = \frac{r_2 r_4 (r_2 + r_4)}{(r_2 + r_4) r_2 r_4 + min(r_1, r_3) r_2 r_4 + min(r_1, r_3) r_2^2 + min(r_1, r_3) r_4^2}$$
$$\pi(X_1) = \frac{r_2 r_4 min(r_1, r_3)}{(r_2 + r_4) r_2 r_4 + min(r_1, r_3) r_2 r_4 + min(r_1, r_3) r_2^2 + min(r_1, r_3) r_4^2}$$
$$\pi(X_2) = \frac{min(r_1, r_3) r_2^2}{(r_2 + r_4) r_2 r_4 + min(r_1, r_3) r_2 r_4 + min(r_1, r_3) r_2^2 + min(r_1, r_3) r_4^2}$$
$$\pi(X_3) = \frac{min(r_1, r_3) r_4^2}{(r_2 + r_4) r_2 r_4 + min(r_1, r_3) r_2 r_4 + min(r_1, r_3) r_2^2 + min(r_1, r_3) r_4^2}$$

Now, from π, one can calculate performance metrics, i.e. utilisation, throughput, etc, by corresponding rules.

All the above steps can be done manually, because the example is very small and the *infinitesimal generator matrix* is quite small also. Under the circumstances, it is a simple matter to solve the global balance equation by Gaussian elimination. However, direct Gaussian elimination is infeasible when the *infinitesimal generator matrix* is large. An automated solution can be limited by a lack of memory, caused by having to store a manipulate a large matrix Q. If the matrix solution is infeasible then the iterative method, block method or projection method can be applied to address the state space explosion problem to some extent [62].

2.1.3.3 Aggregation

As the state space explosion becomes the major issue in state space based modelling, several techniques for large PEPA models have been proposed. Aggregation, which has been applied to PEPA by Gilmore, Hillston and Ribaudo [21], is one of these methods, and it is widely used technique for reducing the size of CTMC in performance modelling. The state space of the CTMC can be divided to several classes, each of which is considered as a single state and form a new CTMC that is relatively amenable to a matrix solution. Details of this technique are described in [21], and a simple example is illustrated here with the Process/Resource model:

Let us consider two processes competing for one resource in this model by rewriting the system equation as:

$$System \stackrel{def}{=} (Process_0 || Process_0) \underset{use}{\bowtie} Resource_0$$

From this initial state, there are two one step states can be evolved by first process engaging *use* with resource or second process carrying out *use*:

$$(Process_1||Process_0) \underset{use}{\bowtie} Resource_1$$
$$(Process_0||Process_1) \underset{use}{\bowtie} Resource_1$$

Since only overall behaviour is concerned, these two states has been considered to be two states of exactly the same class of state. After identifying all the classes, therefore, by aggregating all the states in each class, a new CTMC can be obtained by modifying old transition rates. Obviously, the size of new CTMC has been reduced.

2.1.3.4 Decomposition

Decomposition is an alternative solution for state space explosion of PEPA model. PEPA has been formally-defined as a compositional formalism on the underlying Markov chain. This compositional characteristic may take advantage of separately solving corresponding sub-models before combing these results to the whole model when direct solution of large models are infeasible. In such a way, generating the global state space has been avoided.

2.1.3.4.1 Product form If the state space S of a Markov process $X(t)$ is in the form of $S = S_1 \times S_2$, that means there are two pieces of information in each state capturing different aspects of the current state. When the process $X(t)$ exhibits a product form solution, which write as $\pi(s) = \pi_1(s_1) \times \pi_2(s_2)$, it indicates that these different aspects of the state description are independent with respect to their equilibrium distribution.

Product form solution is a traditional technique that has been applied to queueing models and stochastic Petri nets. In terms of application of PEPA, refer to [35] for details of results in the classical style.

More recently Harrison has derived a number of interesting results using proper-

ties of the reversed process, captured in his Reversed Compound Agent Theorem (RCAT) and its extensions [24, 26, 25, 27]. These results are particularly noteworthy for their ability to describe previously distinct product form solutions within the same theoretical framework.

2.1.3.4.2 Component substitution When a *product form* does not exist, the PEPA models can still be decomposed under certain conditions to provide solutions to certain global measures. The property of *behavioural independence*, exploited by Thomas [63], can be applied to derive different types of decomposition. In the class of PEPA model, which exhibit the property of *behavioural control*, a component is amenable to be replaced by one of its simpler derivatives. This simple component has fewer states, nevertheless, gives the same set of interactions. Thomas provides an iterative solution in [66], which can be used to obtain approximate results but is much more scalable.

2.1.3.4.3 Distributed solution Knottenbelt [40, 39] proposed another decomposition solution of large CTMC based on distributed disk-based architecture. This technique does not avoid solving the whole Markov chain, but is solving the entire Markov chain in a decomposed way. The Markov chain is divided in several partitions, then distributed to different machine to process. Finally, those distributed results are combined to a final solution in some way. Bradley [4] applied this technique to PEPA through the IPC tool, by which a PEPA model can be converted to an intermediate format that can be solved in distributed manner.

2.1.3.5 Kronecker

In the case of a large Markov model that consists of N submodels, the generator matrix Q can be expressed as a set of N smaller matrices which are represented by tensor algebra. This compact representation requires less memory for storing and manipulating Q, and it is termed Kronecker representation. This representation has been applied to PEPA model by Hillson and Kloul [36]. Depending on the

functional dependency, the infinitesimal generator matrix of the Markov chain associated with a PEPA model is expressed as:

$$Q = \bigoplus_{i=1}^{N} R_i + \sum_{\alpha \in Z} r_\alpha \left(\bigoplus_{i=1}^{N} P_{i,\alpha} - \bigoplus_{i=1}^{N} \bar{P}_{i,\alpha} \right)$$

where

- N is the total number of components in the PEPA model and Z is the set of cooperating action types.
- r_α is the minimum of the functional rates of action type α over all components $C_i, i = 1 \ldots N$.
- R_i is the transition matrix of component C_i relating solely to its individual activities.
- $P_{i,\alpha}$ is the probability transition matrix of component C_i due to activity of type α.
- $\bar{P}_{i,\alpha}$ is a matrix representing the normalization associated with the shared activity α in component C_i.

One can refer to [37] for functional dependency, and details of the Kronecker representation are illustrated by an example and validated in [36]. In generally Kronecker methods use less memory than direct matrix solution, and so are potentially more scalable, however the solution may be slower.

2.1.4 Simulation

Simulation is usually applied to do performance analysis when the model cannot be solved analytically or numerically, this because the state space cannot be handled. By counting a *simulation time*, events occur following relevant rules,

simulating a real system running. One can carry out a short-run or long-run scheme to obtain different performance measures. From the time aspect, simulation model can be classified broadly as either a *discrete time* system or *continuous time* system. In a discrete time system, only selected moments in time are considered (the time at which specific events occur). In addition, some economic models are recognized as this type, because the required data is only available at particular instants (*e.g.* end of day trading figures). *Continuous time* can be further classified as either *continuous time continuous event* system or *continuous time discrete event* system. The former type studies systems where the state continuously changes with time, and will be used here as an approximate solution that introduced in Section 2.5. *Continuous time discrete event* system, which depicted in first subsection, is our most interesting class, because it describes the same features as our PEPA model. More detailed coverage of simulation can be found in [45, 28].

In the context of this thesis, the discrete time instants would represent stochastic events, such as arrivals or service completion. There are many examples of discrete time models of computer systems, where the time instants correspond to system (clock) cycles. Furthermore, efficient discrete time methods may be used to simulate continuous time systems by imposing fixed instants at which observations can be made.

2.1.4.1 Discrete event simulation

In *continuous time discrete event* systems, the time parameter is continuous and events construct a list in a chronological sequence manner. When simulating, each event occurs at an instant in time and indicates a system state evolution. For example, suppose that a printer is being simulated, and the system is defined only by the number of jobs waiting for printing. Therefore, one job joining or leaving marks a change of state and happens at a discrete time moment. There are a number of programming languages can be utilised to discrete event simulation, *e.g.* C, C++, Java, Simulink(Matlab), etc. One also can use a simulator to

particular class of systems, for example network simulators such as NS2. Details can be found from [2], some important components of discrete event simulation are described as follows:

- **Clock**: The simulation must keep track of current simulation time, therefore, a simulation clock is necessary to be created. In contrast to performance measurement on a real system, this clock hops a time interval given by a completion of an event. The time interval is added to the clock depends on the particular distribution which associated with relevant events. In simulating a PEPA model, all time intervals are negative exponential distributed with a mean which is reciprocal of the rate of relevant events.

- **Event lists**: Every event represents a potential change of state of the system. Events in the current list are all the events that could occur in the next system step. Therefore the event list should be updated in every simulation clock hop. The rate of each event is proportional to the probability of which event will happen next.

- **Random number generation**: To determine which event will happen in the event list, a random number should be generated. As the random number is chosen by a computer, even the best algorithm cannot obtain a real random number, but can only approach real random. Therefore, it is usually termed pseudo-random number. Several algorithms can be directly used for the simulation, these include linear congruential generators, lagged fibonacci generators, linear feedback shift registers, feedback with carry shift registers, Blum Blum Shub, Fortuna, and the Mersenne twister, etc.

- **Termination condition**: Because the simulation usually running in a cycling manner, one should provide an ending condition for the program to stop it running forever. According to different performance measures, common choices are at "any particular time", "after a number of events executed", or "when some performance measures reach some typical value".

One approach to simulating a PEPA model without writing the entire simulation code, is to use the **Möbius** [14] tool. Möbius is a multi-paradigm modelling tool, by which different formalisms can be combined to model one system. In such way, all advantages of each formal method can be utilised in one model. PEPA is included in those formalisms that are supported by Möbius, however, it is not the original version, but an extend version called $PEPA_k$ [12]. Three additional items have been added to original PEPA:

- **Formal Parameters**: a process variable or a component now takes one or more parameters to specify its state in the form of $P[x_1, \cdots, x_n] \stackrel{def}{=} Q$.
- **Guards**: consequent events are enabled only when the condition in the guards has been satisfied. For example, consider an expression $P[x] \stackrel{def}{=} [x < 10] => Q$, which indicates only in the condition of x less than 10, can this local transition happen.
- **Value Passing**: values may now be communicated between sequential components via activities.

Möbius also supports a number of analysis methods, which include discrete event simulation. By transferring the original PEPA to $PEPA_k$ in Möbius, one can easily run a simulation. The KDC model in Chapter 3 has been solved in Möbius by discrete event simulation, but the results are not as reliable as we expected. It would appear that same pitfalls may exist with $PEPA_k$ under Möbius, at least, in some particular cases.

2.1.4.2 Stochastic simulation

Stochastic simulation, proposed by Gillespie [20], is a major technique for large chemical reaction systems. Bradley *et al* applied this traditional technique to PEPA models in [6]. A set of rate equations can be generated by the stoichiometric function from the original PEPA model, and then, solved by the Dizzy

tool [52]. The technique was applied to a voting system modelled in PEPA, showing that at least $O(10^{10000})$ states are able to be solved within only a few seconds, clearly demonstrating that it is far more efficient than CTMC based methods.

2.1.5 Stochastic Model checking

Stochastic model checking is used to check whether a stochastic model meets a given specification. The specification is formally expressed by probabilistic temporal logic, which includes Probabilistic Computation Tree Logic (PTCL) and Continuous Stochastic Logic (CSL) for DTMC and CTMC respectively. Informally, a specification could, for example, be "the probability of a system failure less than 10 hours is 25% ". PEPA model checking is supported by the model checker PRISM [38]. More recently, the CSL for PEPA model checking has been implemented in PEPA eclipse-plugin [68] by Smith [58]. Stochastic model checking is not investigated in this thesis, more background of stochastic model checking can be found in [41].

2.1.6 Fluid flow analysis

ODE based fluid flow analysis is an approximate analysis technique based on the solution of coupled ordinary differential equations (ODEs), first applied to stochastic process algebra by Hillston [34]. In this style of model analysis, the model is expressed as a finite number of replicated components and ODEs which represent the flow between behaviours (PEPA derivatives) of the components. Thus, by solving the ODEs, it is possible to count the number of components behaving as a given derivative at any given time, t. In the absence of oscillations, the limit, $t \longrightarrow \infty$, then tends to a steady state value.

It is important to make two crucial observations about this approach. Firstly, this is a fluid approximation, not a discrete behaviour. Therefore, we observe a continuous evolution of a derivative, so we can, at any given time, see a fraction

of a derivative behaving in some way, and another fraction behaving in another. Secondly the analysis is deterministic. Thus, not only will simulating such a system produce exactly the same results every time, but also if the rate of an action is r, then a component will have completely evolved (or *flowed*) into its derivative in exactly $1/r$ time units. Furthermore, ODE analysis has been shown to only give a good prediction in certain classes of model, when there is at most one active minimum function [65]. Despite these restrictions, the technique is extremely useful when considering very large numbers of components. The accuracy of ODE analysis is discussed in Section 2.1.6.6.

2.1.6.1 ODE derivation

The following general differential equation [34] can be used to generate a set of ODEs for a system:

$$\begin{aligned}\frac{dN(C_{ij},t)}{dt} &= -\sum_{(\alpha,r)\in E_x(C_{ij})} r \times \min_{C_{kl}\in E_x(\alpha,r)}(N(C_{kl},t)) \\ &\quad + \sum_{(\alpha,r)\in E_n(C_{ij})} r \times \min_{C_{kl}\in E_x(\alpha,r)}(N(C_{kl},t))\end{aligned}$$

Where, C_{ij} denotes jth local derivative of component type C_i. $N(C_{ij},t)$ is the number of C_{ij} at time t, $E_x(C_{ij})$ and $E_n(C_{ij})$ is the set of exit activities and entry activities of C_{ij} respectively.

Consider our Process/Resource Model, the set of ODEs of the system can be generated as:

$$\begin{aligned}\frac{d}{dt}Process_0(t) &= r_2 Process_1(t) - min(r_1,r_3)Process_0(t) \\ \frac{d}{dt}Process_1(t) &= min(r_1,r_3)Process_0(t) - r_2 Process_1(t)\end{aligned}$$

$$\frac{d}{dt}Resource_0(t) = r_4 Resource_1(t) - min(r_1, r_3)Resource_0(t)$$
$$\frac{d}{dt}Resource_1(t) = min(r_1, r_3)Resource_0(t) - r_4 Resource_1(t)$$

2.1.6.2 Solution by simulation

The simplest way of solving a set of ODE is simulating over a suitably long time frame until we observe the long run (steady state) behaviour. In doing so we need to be careful that in discretizing time we make the time step sufficiently small so as to not alter the system behaviour. Typically we take the time step, δt, such that, $\delta t \leq 1/(r_{max}N)$.

2.1.6.3 Analytic solution

When the system is in steady state, it is clear that the state of derivatives will not change anymore. Hence, the left part of the set of ODEs become zero as $t \longrightarrow \infty$, if a steady states solution exists. To consider our Process/Resource model, two equations with four variables can be obtained:

$$r_2 \lim_{t\to\infty} Process_1(t) - min(r_1, r_3) \lim_{t\to\infty} Process_0(t) = 0$$
$$min(r_1, r_3) \lim_{t\to\infty} Resource_0(t) - r_4 \lim_{t\to\infty} Resource_1(t) = 0$$

Further more, we know the initial number of each component. For example, we assume there are N process component and M resource component in initial stage. Therefore:

$$\lim_{t\to\infty} Process_0(t) + \lim_{t\to\infty} Process_1(t) = N$$
$$\lim_{t\to\infty} Resource_0(t) + \lim_{t\to\infty} Resource_1(t) = M$$

Consequently, a closed form solution can be obtained by solving four equations with four variables.

$$\begin{aligned}
\lim_{t\to\infty} Process_0(t) &= \frac{r_2}{min(r_1, r_3) + r_2} N \\
\lim_{t\to\infty} Process_1(t) &= \frac{min(r_1, r_3)}{min(r_1, r_3) + r_2} N \\
\lim_{t\to\infty} Resource_0(t) &= \frac{r_4}{min(r_1, r_3) + r_4} M \\
\lim_{t\to\infty} Resource_1(t) &= \frac{min(r_1, r_3)}{min(r_1, r_3) + r_4} M
\end{aligned}$$

2.1.6.4 Stochastic differential equations

A stochastic differential equation (SDE) based fluid flow approximation is proposed by Hayden and Bradley [29, 30]. Noise is introduced to the deterministic ODEs by recourse to a *Brownian motion* or *thinned Poisson random measures*, in the situation that one is willing to model the influence that introduced by random noise in a system. This type of analysis has not been utilised here. Details of the approach, which is mathematically challenging, can be found in [29, 30, 57].

2.1.6.5 Higher moment from ODEs

Later on, Hayden and Bradley explored general-order moment analysis of fluid approximation in [32, 31]. This approach can analyse more significant aspects

of the models, such as variance of second moment, rather than mean value of first moment. It also provides a path to evaluate the accuracy of the first order analysis. Furthermore, this type of analysis can be easily conducted by the tool GPA, which developed by Stefanek, Hayden and Bradley [60].

2.1.6.6 Accuracy of fluid approximation

As only ODE based fluid flow analysis is utilised in this thesis, we illustrate the accuracy of ODE approximation in this section by two typical examples. Firstly, we plot one of the results from the KDC case study in Chapter 3 following.

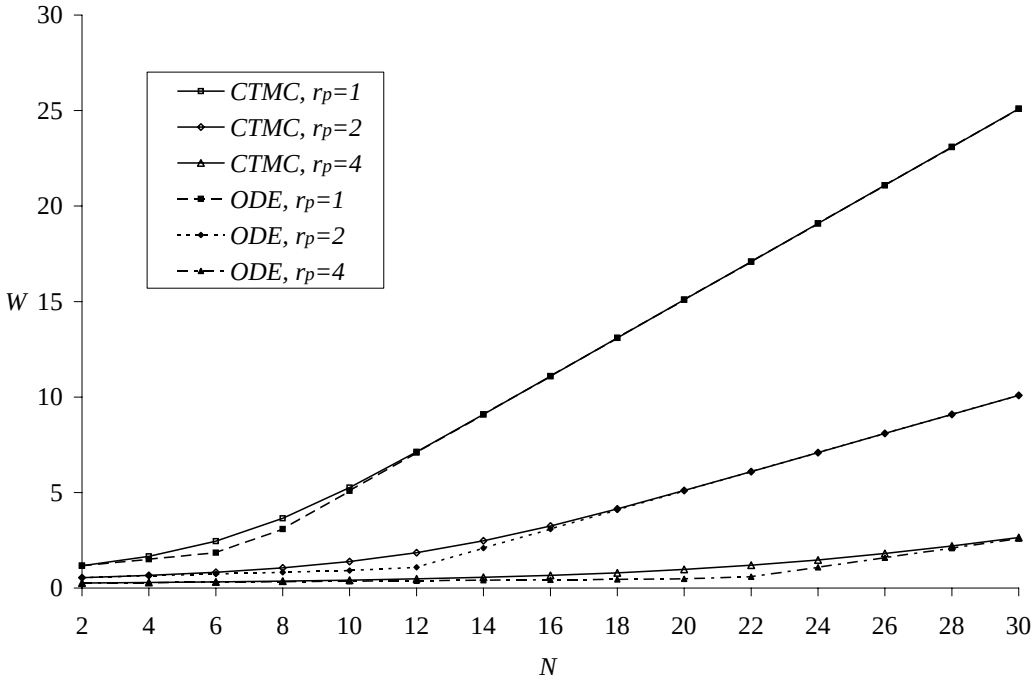

Figure 2.5: Average response time calculated by the ODEs and CTMC, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

One can refer to Section 3.4 for the KDC model and Section 3.5 for the ODEs. Figure 2.5 shows the average response time of KDC server against the scale (number of clients, N)) of the system. The results from ODE approximation are very close to CTMC results when N is extremely small, and the ODE approximation gives accurate results when the system scale is very large. However, it is not difficult to notice that there are divergence areas for each set of results in the middle of

the curves. The inaccurate areas can be identified, and the details are in Section 3.5. Furthermore, we can find that the small scale and large scale are determined by the parameters in particular cases. In practice, this ODE approximation is not applicable if one is looking for exact solution in the divergence area. However, some of the optimistic job and trend analysis are fit for the approximation.

The KDC model in Section 3.4 only have one service type. Further inaccurate issue has been found by Thomas in [65] with two kind of services in a model. We use the example from [65] to shown the situation as follow.

$$
\begin{aligned}
Proc &\stackrel{def}{=} (service, \mu).Proc \\
Disk &\stackrel{def}{=} (request, \eta).Disk
\end{aligned}
$$

$$
\begin{aligned}
User_1 &\stackrel{def}{=} (think, \xi).User_2 \\
User_2 &\stackrel{def}{=} (service, p\mu).User1 + (service, (1-p)\mu).User3 \\
User_3 &\stackrel{def}{=} (write, \eta).User_1
\end{aligned}
$$

$$
System \stackrel{def}{=} (Proc||Disk[K]) \underset{\{service,write\}}{\bowtie} User_1[N]
$$

The system describes a processor and an array of K independent disks. Users request a service from the processor.After this they either think for a while, before making another request, or their results requires writing to a disk before thinking and then another request.

Following derivation equations in Section 2.1.6.1, the ODEs can be derived as:

$$
\begin{aligned}
\frac{d}{dt}User_1(t) &= p\mu min(1, User_2(t)) + \eta min(K, User_3) - \xi User_1(t) \\
\frac{d}{dt}User_2(t) &= \xi User_1(t) - \mu min(1, User_2(t))
\end{aligned}
$$

$$
\begin{aligned}
\frac{d}{dt}User_3(t) &= (1-p)\mu min(1, User_2(t)) - \eta min(K, User_3) \\
\frac{d}{dt}Proc(t) &= 0 \\
\frac{d}{dt}Disk(t) &= 0
\end{aligned}
$$

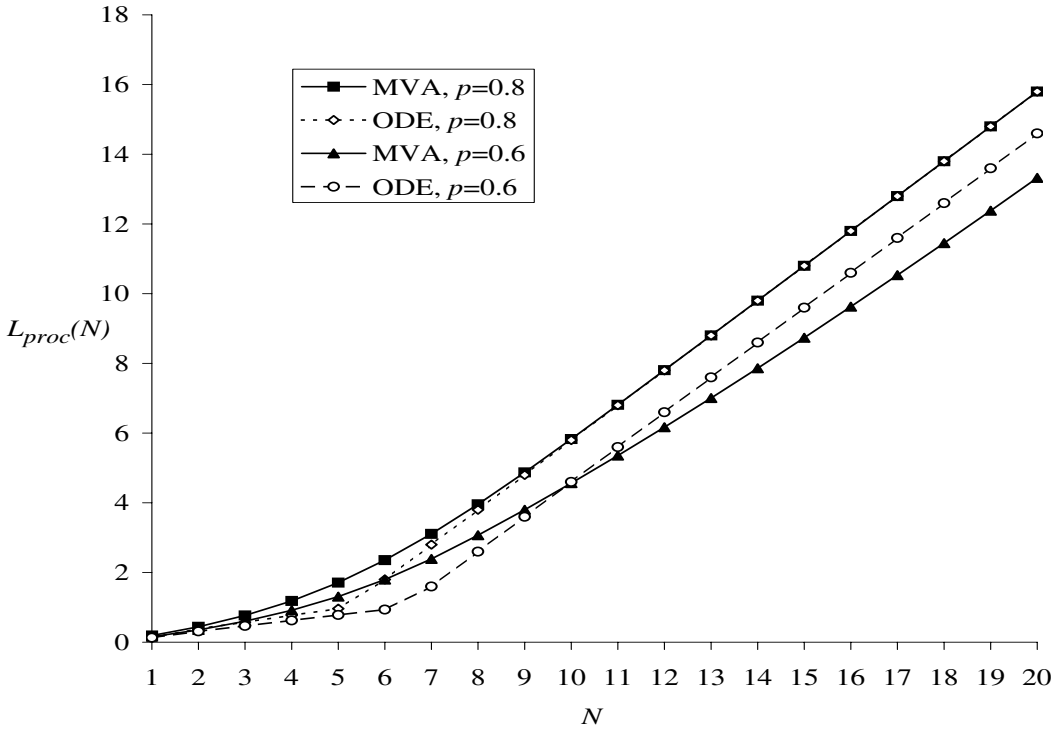

Figure 2.6: Average queue length at processor varied with population size, $\xi = 10, \mu = 30, \eta = 5, K = 3$

Clearly, there are two type of minimum functions in the ODEs. Figure 2.6 compares the ODE analysis with a exact solution, mean vale analysis in terms of the queue length at processor. When p is large (0.8), ODEs give our expected results that it is accurate after the divergence area with large number of clients. Under this value of p, only a few Users go to the *write* service at Disk, and this means one of the queue is heavy loaded and another one is not busy. However, if we assign the similar workloads to each server by setting p to 0.6, ODEs results do not converge with the exact solution even when number of clients is very large.

From the investigation of ODE analysis with applications, we can initially concludes that ODE method perform very well with very small scale and large scale

situation in cycled PEPA models with only one minimum function. If there are more than one minimum functions in system ODEs, the method still could give acceptable results if one of the server services most of the clients. However, the ODE analysis is inaccurate when the servers are equally loaded.

2.1.7 Mean Value Analysis

Mean value analysis (MVA) is a method for deriving performance metrics based on steady state averages directly from the queueing network specification, without the need to derive any of the underlying Markov chain. As such it is relatively computationally efficient as long as the population size is not excessively large.

2.1.7.1 A class of closed queueing networks in PEPA

Consider a model of a closed queueing network of N jobs circulating around a network of M service stations, denoted $1, \ldots, M$; each station is either a queueing station or an infinite server station. There are M_q queueing stations. Let $\mathcal{M}$ be the set of all queueing stations. At each queueing station, i, there is an associated queue (bounded at N) with K_i servers. The servers are able to serve jobs of only one type; each job type, j, is served at rate r_j. At each infinite server station, i, jobs of type i experience a random delay with mean $1/r_i$. All services times are negative exponentially distributed.

There are J job types. Each job type can be served at most one station. When a job of type j completes a service at a given station, it will proceed to service at a station (possibly the same station) as a job of type k according to some routing probability p_{jk}.

In PEPA a queue station can be modelled as

$$QStation_i \stackrel{def}{=} (service_i, r_i).QStation_i \ , \ \forall i \in \mathcal{M}$$

Note that r_i is always specified as finite, and not $\top$. This is because passive

actions are subject to the *apparent rate* in PEPA. The infinite server stations are not represented explicitly.

Each job will receive service from a sequence of stations determined by a set of routing probabilities,

$$Job_i \stackrel{def}{=} \sum_{k=1}^{J} (service_j, p_{jk} r_j).Job_k \ , \ 1 \leq i, j \leq J$$

Where, $0 \leq p_{jk} \leq 1$ and

$$\sum_{k=1}^{J} p_{jk} = 1 \ , \ 1 \leq j \leq J$$

Denote $\mathcal{S}_i$ to be the set of all job types which perform $service_i$ actions, i.e. $\mathcal{S}_i = j$ if $service_i \in \mathcal{A}(Job_j)$.

The entire system can then be represented as follows:

$$\left(\prod_{\forall i \in \mathcal{M}} (QStation_i[K_i]) \right) \underset{\mathcal{L}}{\bowtie} Job_1[N] \tag{2.1}$$

Where

$$\mathcal{L} = \bigcup_{\forall i \in \mathcal{M}} \{service_i\}$$

2.1.7.2 Solution

We now consider the *arrival theorem*, first derived independently by Sevcik and Mitrani [55] and Lavenberg and Reiser [43], applied to this class of PEPA model.

Theorem 1 ***Arrival Theorem****. Consider a component Job_i evolving into its successor derivative, Job_j in a system given by (2.1). The steady state distribution of the number of components behaving as any component Job_k at that moment is equal to the steady state distribution of the number of components behaving as Job_k in a system without the evolving job.*

The arrival theorem is as profound as it is simple and seemingly intuitive. It consequently gives rise to the well known *mean value analysis*, whereby the average behaviour of a system of N components may be derived from the average behaviour of a system of $N-1$ components. Therefore it is never necessary to derive a solution to the full CTMC if we are only concerned with the average behaviour of systems of this kind. This follows from the following set of relationships, derived following the pattern of Haverkort [28] pp. 241-245.

Theorem 1 implies that the average time a component spends in behaviour Job_j, denoted $W_j(N)$, where $\mathcal{A}(Job_j) = service_i$ and $i \in \mathcal{M}$, is given by the average number of Job_k ($\forall k \in \mathcal{S}_i$) components in a system with one fewer Job_l, $\forall i$, components in total. Denote $L_j(N)$ to be the steady state average number of components behaving as Job_j in a system with N jobs in total. If $\sum_{\forall i \in \mathcal{S}_j} L_i(N-1) \leq K_j - 1$ and $j \in \mathcal{M}$ then

$$W_i = \frac{1}{r_j} \ , \ \forall i \in \mathcal{S}_j \tag{2.2}$$

Otherwise, if $\sum_{\forall i \in \mathcal{S}_j} L_i(N-1) > K_j - 1$ and $j \in \mathcal{M}$ then

$$W_i \approx \frac{1 + \sum_{\forall i \in \mathcal{S}_j} L_i(N-1)}{K_j r_i} \tag{2.3}$$

Clearly, if $j \notin \bigcup_{\forall i \in \mathcal{M}} \mathcal{S}_\rangle$, then $W_j(N)$ is a constant, given as $W_j(N) = 1/r_j$. The exact solution of equation (2.3) can be found in [67].

We now need to compute a quantity generally referred to as the *visit count*, denoted V_i. The visit count is the number of times derivative Job_i is visited, relative to the number of times some reference derivative Job_I is visited, where $1 \leq I \leq J$. The actual value of V_i is not crucial, rather its value relative to the value of V_I. As such the choice of I is strictly arbitrary.

We can compute the visit count from the routing probabilities p_{ij}. Define the probability that a component will evolve from Job_i to Job_j, without revisiting

Job_i, as follows:

$$P_{ij}(\sigma) = p_{ij} + \sum_{\forall k \notin \sigma} p_{ik} P_{kj}(\sigma)$$

The set σ here contains only the starting and ending behaviours of interest, in this case i and j, i.e. it is used to tell us if we reach Job_j or first return to Job_i. For convenience define the shorthand,

$$P_{ij} = P_{ij}(\{i, j\})$$

By definition, $P_{ii} = 1$. Clearly the system is irreducible if

$$P_{ij} > 0 \; \forall i, j \; , \; i \neq j$$

Now we choose some reference point I, such that,

$$V_i = \frac{P_{Ii}}{P_{iI}} \; , \; \forall i \neq I$$

and $V_I = 1$. Thus, V_i gives the number of times a component assumes the behaviour Job_i, relative to the number times it assumes the behaviour Job_I.

Given the quantity V_j, we can now compute the average response time per passage for a component behaving as Job_j.

$$\hat{W}_j(N) = V_j W_j(N) \tag{2.4}$$

From Little's theorem we know that

$$L_j(N) = X_j(N) W_j(N) = X(N) V_j W_j(N) = X(N) \hat{W}_j(N) \tag{2.5}$$

Where $X_j(N)$ is the observed rate of activity $service_j$ when the population size is N, and $X(N)$ is the sum of all possible $X_j(N)'s$.

Summing (2.5) over all behaviours Job_i, $i = 1, 2, \ldots, J$ gives,

$$\sum_{j=1}^{J} L_j(N) = X(N) \sum_{j=1}^{J} \hat{W}_j(N) = X(N)\hat{W}(N) = N$$

where $\hat{W}(N) = \sum_{j=1}^{J} \hat{W}_j(N)$. Thus,

$$X(N) = \frac{N}{\hat{W}(N)}$$

Hence, with Little's law applied for a given behaviour Job_j,

$$L_j(N) = X_j(N)W_j(N) = X(N)V_jW_j(N) = \frac{N}{\hat{W}(N)}\hat{W}_j(N) \tag{2.6}$$

We are now in a position to calculate $L_j(N)$ for any value of N if we can calculate $L_j(1)$. A solitary Job_i component will never compete for cooperation over the actions in $\mathcal{L}$, and so will experience a delay of $1/r_i$ in each derivative Job_i. Hence, the average number of components behaving as Job_j when $N = 1$, $L_j(1)$ is given by the proportion of time a component spends in that behaviour.

$$L_j(1) = \frac{V_j}{r_j \sum_{i=1}^{J} \frac{V_i}{r_i}} \tag{2.7}$$

We now apply the following iterative solution.

1. Calculate $L_j(1)$ for $j = 1, 2, \ldots J$, using (2.7).
2. $n = 2$
3. Compute $\hat{W}_j(n)$ for $j = 1, 2, \ldots J$, using (2.2), (2.3) and (2.4) and $L_j(n-1)$ from 1 above.
4. Compute $\hat{W}(n) = \sum_{j=1}^{J} \hat{W}_j(n)$.
5. Compute $L_j(n)$ for $j = 1, 2, \ldots J$, using (2.6) and $\hat{W}(n)$ from 4 above.
6. Increment n.

7. If $n \leq N$ return to step 3 else end.

2.1.7.3 Limitation

Clearly, the approach is limited in both the metrics that can be derived and also the class of model that is considered. The former limitation is a feature of mean value analysis (hence the name). However, the class of model could be extended in a number of ways. Mean value analysis applies to multiple classes of jobs in closed queueing network. Therefore it should be straightforward to define a class of model with different groups of components, each with potentially different action rates and routing probabilities. This would be a relatively simple extension of the current class, but would involve careful use of notation to distinguish classes in a meaningful way.

Moreover, it should be noted that there can only be one service action type at a station and that must be given the same rate in any job type where it is enabled. Although intuitively it is possible to model the case where there are more job types enabled at a queueing station, doing so potential introduces race conditions and therefore distorts the effective service rate. We are still considering how such a situation might be best specified and it may be more feasible to consider approximate solutions in this scenario.

2.2 Related work

In this section, various related papers has been reviewed. All these papers explore the performance of security mechanisms, or timing properties of security systems more generally. According to the analysis methods, we classify those papers to informal, which includes performance measurement and computer based simulation, and formal modelling, which are mathematical based modelling techniques. We focus on the latter part.

2.2.1 Informal analysis

Argyroudis *et al* [1] established a test bed upon an HP mobile device to measure three commonly used security protocols: $SSL, S/MIME$ and $IPsec$. By employing different key length and different scenarios, the results show that it is not a obstacle to apply strong cryptographic protocols on handheld devices. This paper only studied the three types of security protocols individually, still stick on SSL and $S/MIME$, Gutmann [22] conducts a comparison between these two protocols and those with similar functions protocols: PGP and SSH. SSL and PGP are both secure messaging protocols, and $S/MIME$ amd SSH are both secure communication protocols. In contrast to Argyroudis *et al* [1], Gutmann [22] measured the performance the protocols upon a normal PC and investigated more thoroughly and detailed in terms of the overhead analysis. In addition, these protocols also have been compared in low-powered devices. The results form a reference for performance and security engineers to choose one of these protocols in the relevant environment. In [16], Dick and Thomas only focus on the PGP protocol. Based on four different hardwares, two versions of PGP have been explored. They studied the overhead which is introduced by different encryption/decryption algorithms and different key lengths.

Potlapally *et al* [50] investigated another element of performance, energy consumption, over the SSL protocol. The experiments have been set on a client-server environment which consists of a mobile device as a client equipped a wireless access point to the LAN and a PC as a server that is wired to the LAN. Different ciphers and hash algorithms with SSL have been tested and compared for the energy loss. Further more, the authors analysed the trade-off between energy consumption and security for this protocol.

Rodeh *et al* [54] studied security protocols that have been applied to *ensemble group communication system*. They measured the latency of an *ensemble stack* which is defined as being the period from message arrival to message departure, and this delay also has been compared in two machines with different CPUs. Moreover, timing property of rekey actions has been measured as this action

potentially affects the trade-off between security and performance.

Moralis *et al* [47] compared X.509 standard and the Kerberos protocol in a web services environment. By setting up a test bed, delay of requests in different stages have been measured in the scenario of heavy load with different message size and fix message size with different loads. The results show that either mechanism could perform better in appropriate conditions. Still considering the security of web services, Lamprecht *et al* [42] measured encryption/decryption time of different implementations of different encryption algorithms in the scenario of secure transaction in web services environment. Chromiak and Lojewshi studied similar research issue in [10], in which they only focus on JCE/JCA with Bouncy Castle provider but covered more algorithms. They then measured the time taking of every stage in encryption/decryption process. These commonly used cryptographic algorithms have also been investigated and measured by Freeman and Miller with different processors in [19]. Furthermore, to consider performance-critical systems, the authors analysed the security-performance trade-off in case of best security/worst performance, good performance/good security and good security/fast performance.

Zhao evaluated the BGP protocol in her PHD thesis [70] by using a Java-based discrete event simulator SSFNet. She analysed detailed performance issues of BGP and its extension Secure-BGP, and those performance issues have been illustrated numerically. Additionally, Public Key Infrastructure (PKI) has been evaluated, and analysed how this infrastructure affect security of BGP. In [56] Sha *et al* simulated an access control protocol ALOHA and its extension $ALOHA/PCD$. By comparing the packet delay and channel throughput under the same conditions, the results show that ALOHA/PCD performs much better than its older version.

Although informal analysis could be applied to any type of system in any situation, all the experiment results do not show the underlying reason of the latency and that could be realised by formal analysis.

2.2.2 Formal analysis

Cho *et al* [9] investigated the potential attack in *Dynamic Group Communication system* (DGCs) to explore how the different rekeying methods affect both security and performance of the whole system, and optimised those methods with appropriate parameters. Based on Stochastic Petri-Net (SPN), the model of three rekeying protocols in DGCs have been established in the environment of Mobile Ad Hoc Networks, and the results clearly show the optimisation point. However, the largest number of requests in the experiments is eight, and that is far less than a real case. The authors also have not provided the analysis techniques or tools for readers to establish more confidence of the results.

Wang *et al* [69] formulated a queueing model for three types of attack on email systems. To address the trade-off of security and performance, not only the performance metric, average queue length, is obtained, but also metrics of dependability, system availability and information leakage probability, are calculated. In the context, the queueing model represents a quasi-birth-death Markovian process, and can be analytically solved efficiently. Nevertheless, the computation effort apparently becomes very significant if a large number of agents are involved in the system.

El-Hadidi *et al* [17] evaluated performance of the Kerberos protocol in an distributed environment. A basic scenario of Kerberos protocol has been modeled by queueing network, and the queueing model has been analytically solved to obtain average time for transferring a message between a client and a server. Later on, the authors extended the paper by comparing the performance of the Kerberos protocol to two other authentication protocols: authenticated Diffie-Hellman exchange protocol and HAH protocol in [18]. Again, the authors formulated queueing models for all three authentication protocol and solved them analytically. By comparing the average messaging time, the results show that the Kerberos protocol has the best performance in terms of the speed, nevertheless, the HAH protocol is the best protocol by considering the trade-off between security and performance. There are two drawbacks in these two papers: firstly, the authors

did not investigate the case of multi-realms in the Kerberos protocol, which could cause more traffic issues; secondly, the analytical solution may be frustrated if the system is large enough. Those issues have been considered in [23] by Harbitter and Menasce. the Kerberos protocol has been investigated more thoroughly with three inner-mechanism in multi-realms scenario. Again, queueing network techniques have been employed to model the systems, however, rather than analytical solution, the authors applied *mean value analysis* (MVA). MVA is only able to obtain mean values, but avoids generating the entire state space, and so is a very efficient analysis technique. The results of MVA have been compared with those obtained from measurement.

Liu *et al* [44] studied an authentication protocol, and derived a queueing model, which represents a quasi-birth-death Markovian process. By applying RG factorization, the authors efficiently solved the queueing model analytically to obtain the state space distribution. Performance metrics then have been calculated and analysed. To some extent, this analysis method is efficient, however, it is not difficult to find that the equations of the analytical solution may take a long time to solve, or even terminated in some point because of limited memory.

Bodei *et al* [3] proposed a new method for performance evaluation of security protocols based on LySa. LySa is a process algebra used for analysing security properties of security protocols, however, authors assigned rates to enhanced labels of the transitions of the system to transform to a Markov chain, which has been utilised for performance analysis. Under this way, a security protocol is able to be investigated for both security and performance property to realise trade-off analysis, and the method has been illustrated by an example, the Otway-Rees protocol. The transformed Markov chain of the Otway-Rees protocol has been analytical solved to obtain the steady state distribution. However, it is difficult to fit a particular scenario from this performance aspect and is infeasible to analyse a security protocol under a large scale environment. A way of scalable trade-off analysis has been proposed in [7]. Based on a study of timing attack, Buchholtz *et al* studied both security and performance aspects of a security protocol, the Wide-Mouthed Frog (WMF). Rather than transform the security model

to a Markov chain, the authors extracted both security and performance model from a UML diagram. Again, the security model is specified by $LySa$, however, the performance model is specified by PEPA (Performance Evaluation Process Algebra). One could modify this PEPA model to fit any scenario to conduct performance analysis. The authors recommended a tool chain $IPC/DNAmaca$ as a PEPA analyser, that not only can be used for steady state analysis, but also can obtain passage-time distribution. Further more, $IPC/DNAmaca$ is able to solve large scale PEPA models.

Thomas [64] conducted a peformability study of a secure e-voting system. The model is formulated by a stochastic process algebra PEPA. Along with more distributed voters coming to the system, the model encountered the state space explosion problem. Two simplification methods were proposed. Firstly, the author aggregated some internal actions to reduce the state space. Secondly, a queue-based approximation is derived. By comparing the state space of the original model and both approximations, the queue-based approximation model was more scalable than aggregation, however, the aggregation method was more accurate.

Bradley *et al* [5] explored three types of Internet worm attack based on a PEPA model. To consider scalable analysis, fluid flow approximation (based on ordinary differential equations) is employed to analyse the PEPA models. This kind of analysis approximates the original discrete state space into continuous states, and it is able to cope with model of 10^{10000} states and beyond. The authors investigated number of infected machines, network connections, susceptible machines and changes over time with different network capacity and patch rate under these three types of Internet worm attack. However, these results have not been verified by observation.

2.3 Summary

This chapter describes the PEPA language and some of the current efficient solutions for PEPA models that we employed or attempted to employ in this work, and some related case studies.

Chapter 3

Key Distribution Centre

3.1 Introduction

This chapter studies a secure symmetric key exchange protocol, which utilises a trusted third party known as a key distribution centre (KDC), first proposed by Needham and Schroeder [48]. Firstly, three versions of the model have been specified and analyzed numerically, but, in doing so, we encountered the state space explosion problem. The remainder of this chapter explores possible approaches to solving and evaluating a utility function, based on those techniques, to better understand the behaviour of this system. The contributions of this Chapter includes the development of partial evaluation, identification of diverge point (N^*) in ODE approximation, and identification of ODE analysis is the same as asymptotic bounds of the equivalent closed queueing model.

This chapter is organised as follows. In the next section (Section 3.2) we introduce the system to be modelled, the key distribution centre and a utility function used to evaluate its behaviour. This is followed by three proposed modelling choices with preliminary numerical results. Section 3.4 introduces the chosen modelling form, followed by a simplified (equivalent) version and an approximation. Some numerical results of the approximation are presented, including comparison of the approximation results with discrete event simulation. After that, a simple

fluid analysis based on ordinary differential equations is introduced in Section 3.5, along with numerical results which are compared with both the earlier approximation and stochastic simulation. The utility function and its numerical results analysis are illustrated in Section 3.6. Finally we will end with a brief summary of this chapter in Section 3.7.

3.2 Protocol specification

We now describe the specific problem we seek to model. The protocol is illustrated in Figure 3.1 below, following the descriptions in [59] and [11].

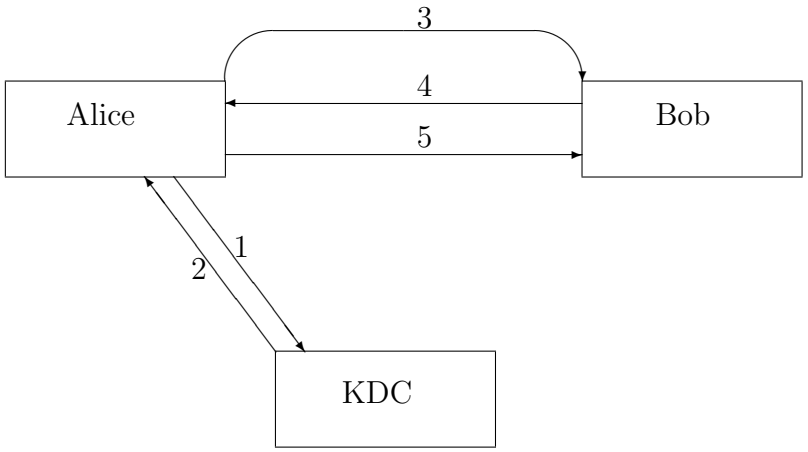

Figure 3.1: Key Distribution Scenario.

Additionally:

- Alice and KDC share a key K_A
- Bob and KDC share a key K_B

Informally we can describe the protocol as follows.

1. Alice sends a request to the KDC.
2. If Alice is known and trusted by the KDC, it responds with an encrypted message only Alice can read, which includes a session key for Alice and Bob to use and a further encrypted message only Bob can read.

3. Alice will then forward part of the message from the KDC directly to Bob.

4. Bob will decrypt this message (as he knows the decryption key) and responds to Alice with some random data (called a nonce) encrypted with the session key.

5. Alice will confirm the use of the session key by replying to Bob with a modification of the nonce, encrypted with the session key.

The key features of this protocol are that only Alice can read the message sent by the KDC (step 2) as only Alice and the KDC know the key K_A. Included in this message is another message further encrypted with K_B, the key shared by Bob and the KDC. Alice cannot read this message, but instead forwards it to Bob (step 3). This message tells Bob that Alice is genuine (i.e. has communicated with the KDC and displays a correct ID) and informs Bob of the session key; only Bob can read this message. Alice and Bob now both know the session key K_S and the remainder of the protocol ensures that Bob trusts Alice and the session key (and Alice trusts Bob).

More formally we can define the protocol as follows [11].

1. $Alice \longrightarrow KDC \ : \ A, B, N_1$

2. $KDC \longrightarrow Alice \ : \ \{K_S, A, B, N_1\{K_S, ID_A\}_{K_B}\}_{K_A}$

3. $Alice \longrightarrow Bob \ : \ \{K_S, ID_A\}_{K_B}$

4. $Bob \longrightarrow Alice \ : \ \{N_2\}_{K_S}$

5. $Alice \longrightarrow Bob \ : \ \{f(N_2)\}_{K_S}$

where,

- $X \longrightarrow Y$ denotes a communication from X to Y.
- $x_1, \ldots x_n$ denotes a tuple of n values.

- $\{x_1, \ldots x_n\}_K$ denotes a tuple of n values encrypted with the key K.
- N_1 and N_2 are nonces (random items of data).
- ID_A is a unique identifier for Alice.
- $f(N)$ denotes a predefined function applied to the nonce N.

The performance of the protocol use is dominated by competition for resources at the KDC. Thus, when a client (Alice) contacts the KDC to obtain a session key, the request will be queued awaiting service and processed along with other waiting requests according to some scheduling strategy. We assume that this strategy is FCFS, and hence the service of any incoming request will be dependent on the service of all other requests already present in the system, the rate at which requests can be serviced, r_p, and the number of servers available, K.

We now wish to explore two closely related questions: "how many clients can a given KDC configuration support?" and "how much service capacity must we provide at a KDC to satisfy a given number of clients?" In the first instance we would fix K and r_p and find the largest value of the population size, N, before the performance begins to significantly degrade. In the latter case we would fix N and r_p and find an optimal value of K. In addition to these two cases, we may also ask, given a demand (from N client pairs with use rate r_u and request rate r_q) on a given system (of K servers running at rate r_p), what is the maximum rate at which keys can be refreshed before the KDC performance begins to degrade?

In answering these questions we need to consider what we mean by the performance of the KDC. In the solution of this model we will introduce a number of performance measures, including utilisation, throughput, average queue length and average response time. All of these measures are important, but considering all of them at once will not lead to a clear picture of optimal performance. Instead we introduce a utility function to be optimised. This function is based on the assumption that there is a cost in keeping customers waiting (as the longer they wait, the less they will be satisfied) and a competing cost in providing resources at the KDC (e.g. purchasing and maintaining or leasing servers).

This gives rise to the following simple utility function.

$$C = c_1 L + c_2 K r_p \ \ , \ c_1, c_2 \geq 0 \tag{3.1}$$

Where L is the mean queue length at the KDC. The cost rates c_1 and c_2 are dependent on the particular system in question and may further depend on the type of quality of service contract that is in place with customers. If we wish to improve the responsiveness of the system, we would increase c_1, whereas if we want to minimise running costs we would increase c_2.

3.3 Modelling Choices

There are three approaches that we have developed to model this secure key distribution scenario for multiple clients. First, we consider Alice and Bob as a pair of clients, and repeated this pair in the model to communicate with key distribution centre to specify multiple clients, as illustrated in Figure 3.2. In this model, a response from the KDC must succeed the request behaviour of each corresponding Alice, and precede any other interaction.

In PEPA this model can be modelled as follows:

$$KDC \stackrel{def}{=} (request, \top).(response, r_p).KDC;$$

$$\begin{aligned} Alice \stackrel{def}{=} & (request, r_q).(response, \top).(sendBob, r_B). \\ & (sendAlice, \top).(confirm, r_c).(usekey, r_u).Alice; \end{aligned}$$

$$Bob \stackrel{def}{=} (sendBob, \top).(sendAlice, r_A).(confirm, \top).(usekey, \top).Bob;$$

$$System \stackrel{def}{=} KDC \underset{\mathcal{L}}{\bowtie} (Alice \underset{\mathcal{K}}{\bowtie} Bob)[N]$$

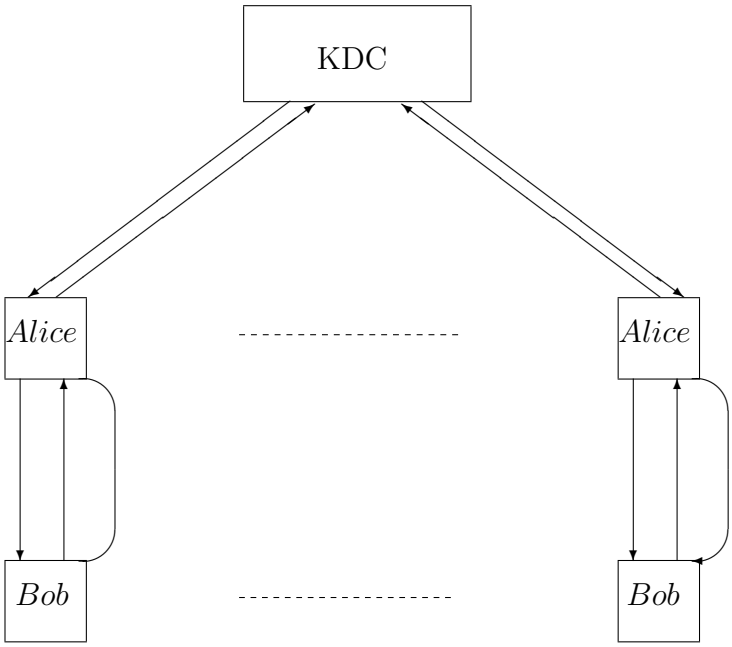

Figure 3.2: Initial model of key distribution centre.

where $\mathcal{L} = \{request, reponse\}, \mathcal{K} = \{sendBob, sendAlice, confirm, usekey\}$, N is number of pairs of clients.

The second model has been approached in a different way. In this approach, multiple clients were manually added by different names and parallel request and response are allowed here, this means that the KDC can receive (and queue) several request before responding to them, as shown in Figure 3.3.

This approach can be modelled in PEPA as follows (ϕ in this model is the number of pairs of clients):

$$\begin{aligned} KDC &\stackrel{def}{=} (requestA, \top).KDC_1 \\ &+ (requestC, \top).KDC_2 \\ &+ \cdots \\ &+ (requestA_{(2\phi-2)}, \top).KDC_\phi; \end{aligned}$$

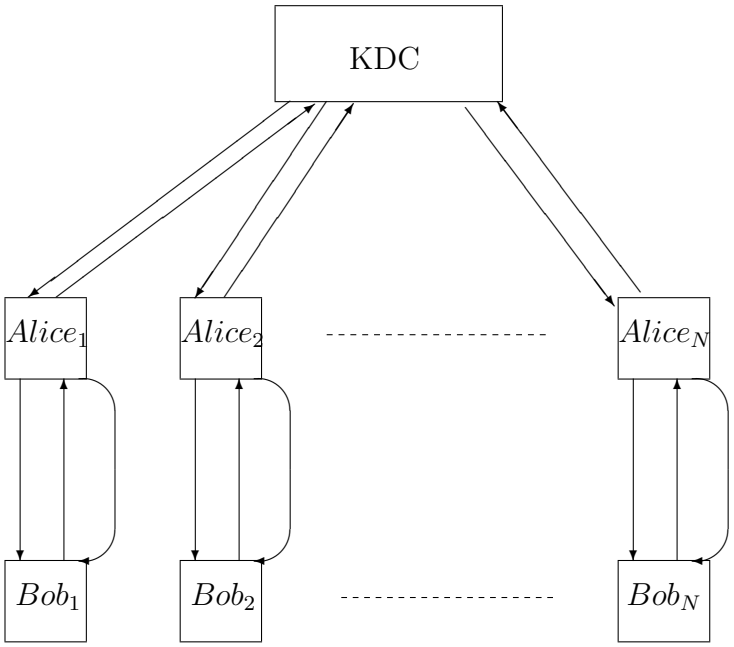

Figure 3.3: Alternative model of key distribution centre.

$$
\begin{aligned}
KDC_1 \stackrel{def}{=}\ & (responseA, r_p).KDC + (requestC, \top).KDC_{(\phi+1)} \\
+\ & (requestE, \top).KDC_{(\phi+2)} \\
+\ & \cdots \\
+\ & (requestA_{(2\phi-2)}, \top).KDC_{(2\phi-1)};
\end{aligned}
$$

$\cdots$

$$
\begin{aligned}
A \stackrel{def}{=}\ & (requestA, r_q).(responseA, \top).(sendB, r_B).(sendA, \top). \\
& (confirmA, r_c).(usekeyA, r_u).A;
\end{aligned}
$$

$$
B \stackrel{def}{=}\ (sendB, \top).(sendA, r_A).(confirmA, \top).(usekeyA, \top).B;
$$

$\cdots$

$$
\begin{aligned}
A_{(2\phi-2)} \stackrel{def}{=}\ & (requestA_{(2\phi-2)}, r_q).(responseA_{(2\phi-2)}, \top). \\
& (sendA_{(2\phi-1)}, rA_{(2\phi-1)}).(sendA_{(2\phi-2)}, \top).
\end{aligned}
$$

$$(confirm, r_c).(usekey, r_u).A_{(2\phi-2)};$$

$$A_{(2\phi-1)} \stackrel{def}{=} (sendA_{(2\phi-1)}, \top).(sendA_{(2\phi-2)}, rA_{(2\phi-2)}).$$
$$(confirm, \top).(usekey, \top).A_{(2\phi-1)};$$

$$System \stackrel{def}{=} KDC \underset{\mathcal{L}}{\bowtie} ((A \underset{\mathcal{K}}{\bowtie} B)|| \cdots ||(A_{(2\phi-2)} \underset{\mathcal{Z}}{\bowtie} A_{(2\phi-1)})$$

where $\mathcal{L} = \{requestA, responseA, \cdots, requestA_{(2\phi-2)}, responseA_{(2\phi-2)}\}$, $\mathcal{K} = \{sendB, sendA, confirmA, usekeyA\}$, $\mathcal{Z} = \{sendA_{(2\phi-1)}, sendA_{(2\phi-2)}, confirm_{(\phi-1)}, usekey_{(\phi-1)}\}$.

The third approach uses the same infrastructure as model one (Figure 3.2). The two main differences are that model three makes requests and responses in parallel, so the KDC can hold several requests in a queue, as in model two; and an anonymous response mechanism is introduced here, that means KDC does not explicitly distinguish between requests.

PEPA model for third approach as follows: (ϕ in this model means number of pair of clients)

$$KDC \stackrel{def}{=} (request, \top).KDC_1;$$

$$KDC_1 \stackrel{def}{=} (response, r_p).KDC + (request.\top).KDC_2;$$

$$KDC_2 \stackrel{def}{=} (response, r_p).KDC1 + (request, \top).KDC_3;$$

$$KDC_3 \stackrel{def}{=} (response, r_p).KDC2 + (request, \top).KDC_4;$$

$\cdots$

$$KDC_{\phi} \stackrel{def}{=} (response, r_p).KDC_{(\phi-1)};$$

$$Alice \stackrel{def}{=} (request, r_q).(response, \top).(sendBob, r_B).(sendAlice, \top).$$
$$(confirm, r_c).(usekey, r_u).Alice;$$

$$Bob \stackrel{def}{=} (sendBob, \top).(sendAlice, r_A).(confirm, \top).(usekey, \top).Bob;$$

$$System \stackrel{def}{=} KDC \underset{\mathcal{L}}{\bowtie} (Alice \underset{\mathcal{K}}{\bowtie} Bob)[N]$$

where $\mathcal{L} = \{request, reponse\}, \mathcal{K} = \{sendBob, sendAlice, confirm, usekey\}$, N is number of pair of clients.

3.3.1 Preliminary results

The three models of key distribution are now compared numerically using the PEPA Workbench [13]. In all cases the parameters are set to 1.0 (except ru=1.1 for numerical computation reasons) and other parameters are varied as shown.

Three experiments have been set up for each model to test the utilisation of the KDC, i.e. the state of the KDC holding at least one request. First, we increased number of clients to the limit of the PEPA Workbench. Then we varied the rate of *usekey* (r_u) in case of three pair of clients in second experiment. Finally, rate of *request* (r_q) has been varied in case of three pair of clients for testing.

In the first experiment, a "run out of memory java heap space" occurs when the number of pairs exceeds five. Therefore, five data points were acquired for each model in the first trial. Figure 3.4 shows the KDC utilisation as the number of client pairs is increased. We see that for all three models, utilisation increases when adding more clients to the model, as expected. The reason is clearly that as

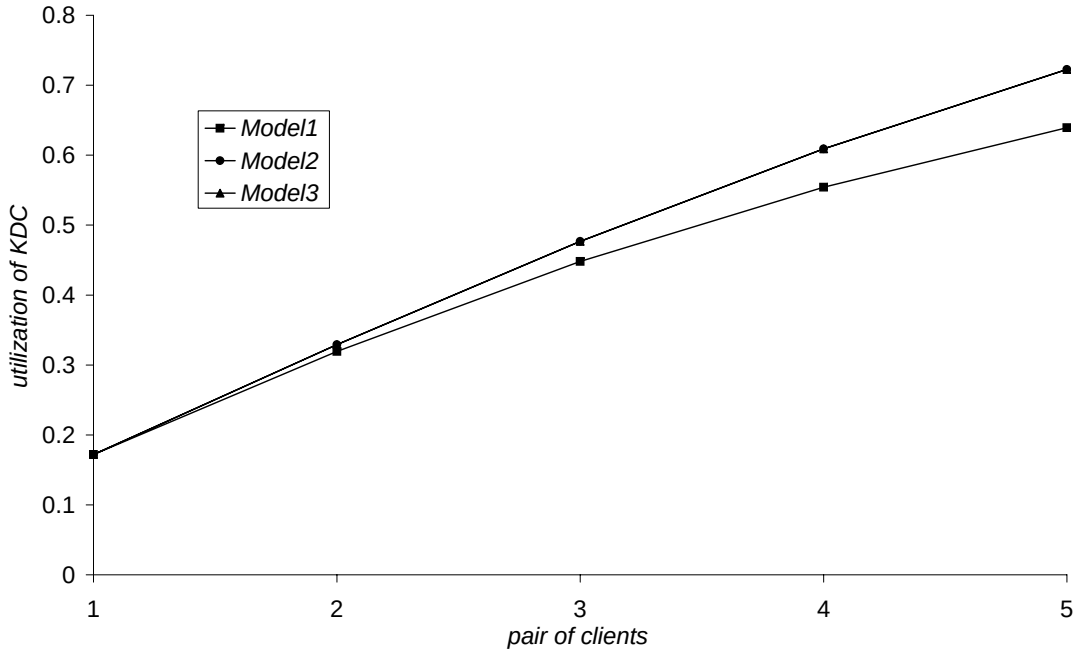

Figure 3.4: Utilisation of the KDC varied with number of client pairs.

more clients are involved, more requests will be made, thus, the KDC has more work to do.

The second feature that can be observed in Figure 3.4 is that the utilisation of the KDC in model 1 increases slower and is smaller at any point (except the start) than for model 2 and model 3. In the case of one pair of clients, the three models gave exactly the same result, as we would expect. In model 1, the response of the KDC must succeed a request behaviour of each corresponding Alice. As such, the KDC cannot hold multiple requests at the same time. Thus, subsequent requests are blocked until the KDC is idle when the request can be made. In the case of models 2 and 3, requests are queued so that once one request has been processed, another service may begin immediately. Thus, the system described by model 1 is therefore clearly less efficient.

Another aspect shown in Figure 3.4 is that model 2 and model 3 have exactly the same results. The only difference between model 2 and model 3 is in distinguishing which client pair are being responded to. In model 3 the KDC component merely keeps track of the number of waiting clients, whereas in model 2 each client is distinguished by name and action. This means that more information

is potentially available in model 2, although in practice this does not change the amount of work the KDC has to undertake, so the utilisation is the same in each case.

Figure 3.5 shows the result of the second experiment.

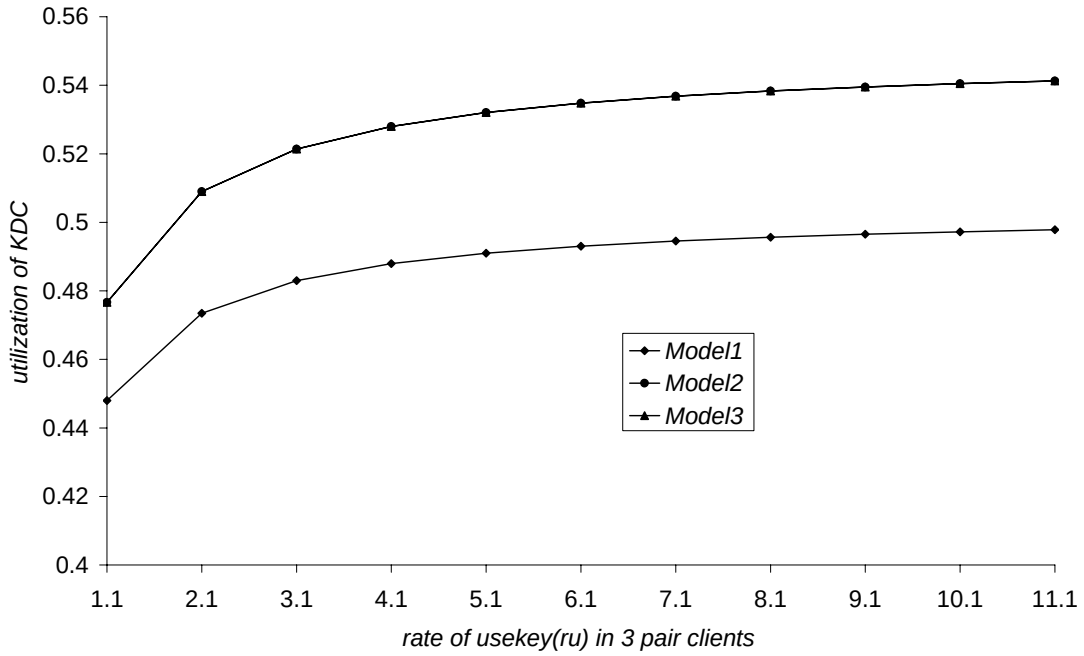

Figure 3.5: KDC utilisation varied with rate of use of the key.

For the reason discussed above, model 2 and model 3 have the same result in this experiment as well. The utilisation of the KDC in model 1 is again smaller than in model 2 and model 3 in any same rate of key use. There are some new features here in experiment two: first, the utilisation of the KDC increases when r_u is increased; secondly, the utilisation of the KDC in all models increases more slowly as r_u gets larger; finally, all three models keep the same increasing rate. For the first feature, the reason is that increasing r_u leads to clients sending requests to the KDC more frequently. Therefore, the KDC has more requests to process. The profile of the plots is a direct result of the variation of r_u, which is the reciprocal of the duration of the key use. When r_u is small, a small increase has a large effect (a large decrease in *usekey* duration), however obviously the same increase has a much smaller effect when r_u is large (duration is very short, and the decrease is minimal).

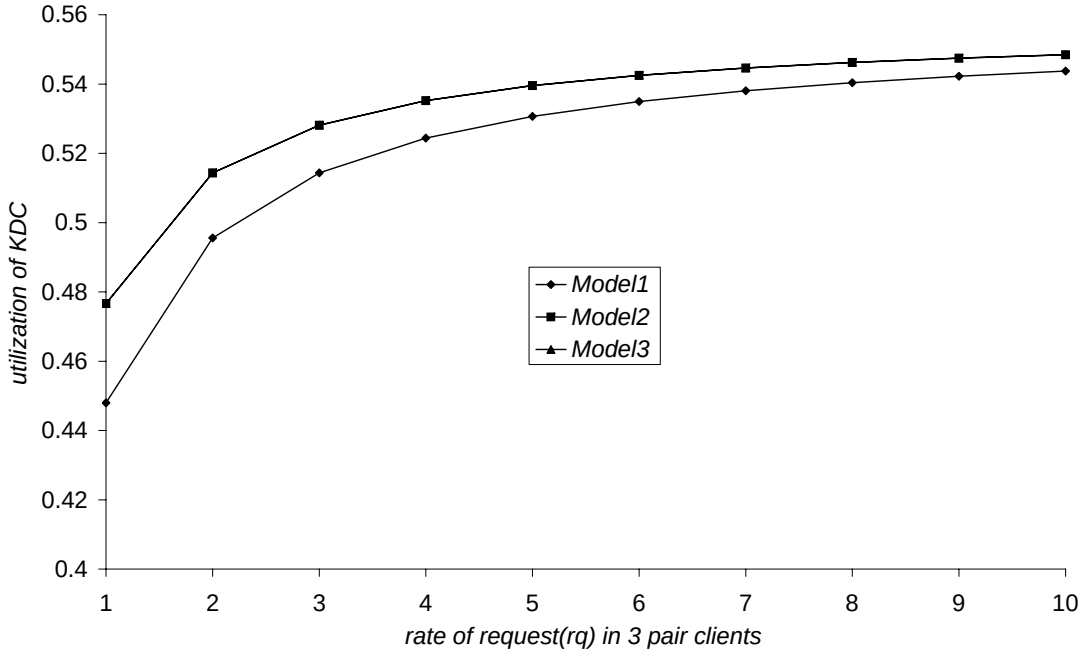

Figure 3.6: KDC utilisation varied against the rate of request.

Finally, we come to the third experiment show in Figure 3.6. We again found two features are the same as discussed in experiments one and two, namely, that model 2 and model 3 have the same result in this experiment, and the utilisation of the KDC in model 1 is smaller than in model 2 and model 3 for any given rate of request. The differences from other experimental results are that utilisation of the KDC increases as r_q increases. Clearly, the faster the clients request, more likely that the KDC is busy. Another characteristic of these results is that the utilisation of the KDC in all models increases more slowly as r_q gets larger. The reason is similar as changing r_u in experiment two. The time for a client pair to cycle through their states is given as $T_k + \frac{1}{r_q} + \frac{1}{r_u}$, thus increasing of r_q makes $\frac{1}{r_q}$ smaller every time, although the increasing part is getting smaller every time as well. Finally, we found that utilisation of the KDC in model 1 is getting close to that in model 2 and model 3. With r_q increasing, the time that the KDC has to wait between requests in model 1 is getting smaller, which makes model 1 closer to model 2 and model 3 where there is no such waiting as the requests are queued. Clearly as $r_u \to \infty$ the different calculations of utilisation will converge.

As well as the utilisation of the key distribution centre, we would also wish to

measure the performance perceived by the user. To do this we calculate the average response time, which we define as the time from when the previous session key has finished being used, to the time when the new session key was started to be used. The average response time, W, is calculated as follows:

$$W = \frac{1 - p_{use}}{p_{use} r_{use}} \tag{3.2}$$

Where p_{use} is the steady state probability that a given key is being used by a given communicating pair. Because all the models are symmetric with respect to communicating pairs, it does not matter which component we choose to measure to find p_{use}. The reason that response time is defined in this way, and not as more conventionally to be just the time taken by the server, is that model 1 includes blocking of requests when busy. This is a clear performance difference between model 1 and the other two approaches and therefore needs to be incorporated in the metric to get a consistent comparison.

The same three experiments we conducted as above, and calculated average response time by (3.2). Figure 3.7 shows the response time varied as number of clients pair is increased.

Again, we found Model 2 and Model 3 have exactly the same results and all three models become the same in the case of one pair of client, for the same reasons that have been discussed above. Here, for all three models, response time increases when adding more clients to the system. It is clear that more clients involved in, the system takes more time to respond in average. Another feature is that average response time in Model 1 is larger than in Model 2 and Model 3 in any case of same number of pairs of clients (except one). There are two parts involved in our defined response time: request time and service time. All models have the same request time in this experiment. But, for Model 2 and Model 3, system need less time to process all jobs as requests are queued at the KDC rather than requests being blocked until the previous one has finished being processed in Model 1. Therefore, Model 1 shows less efficient results.

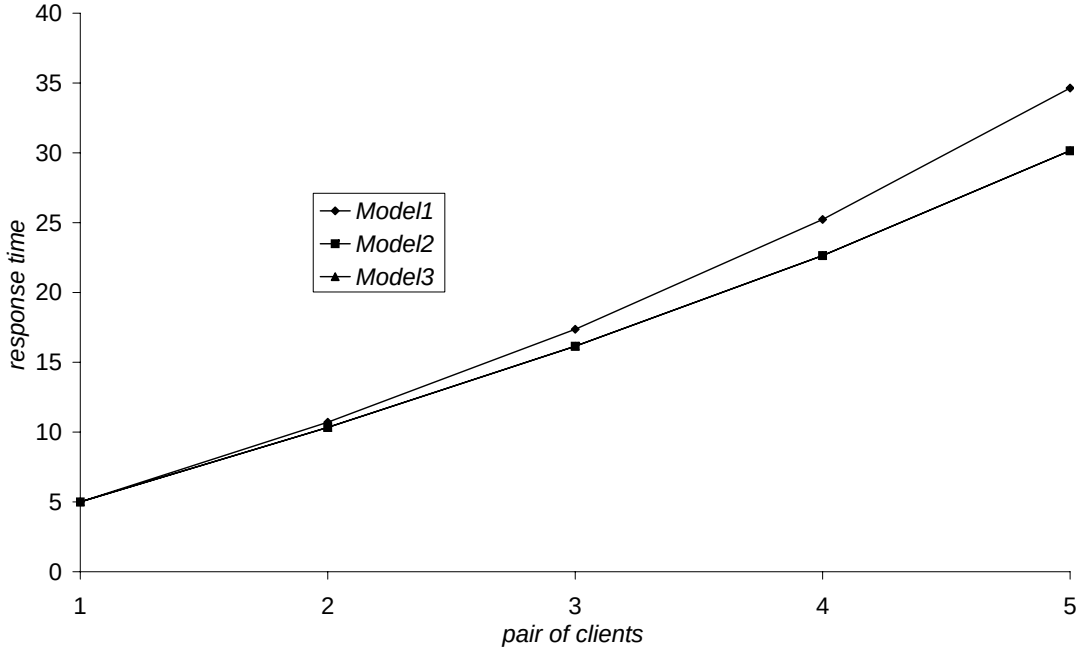

Figure 3.7: Response time varied with number of client pairs.

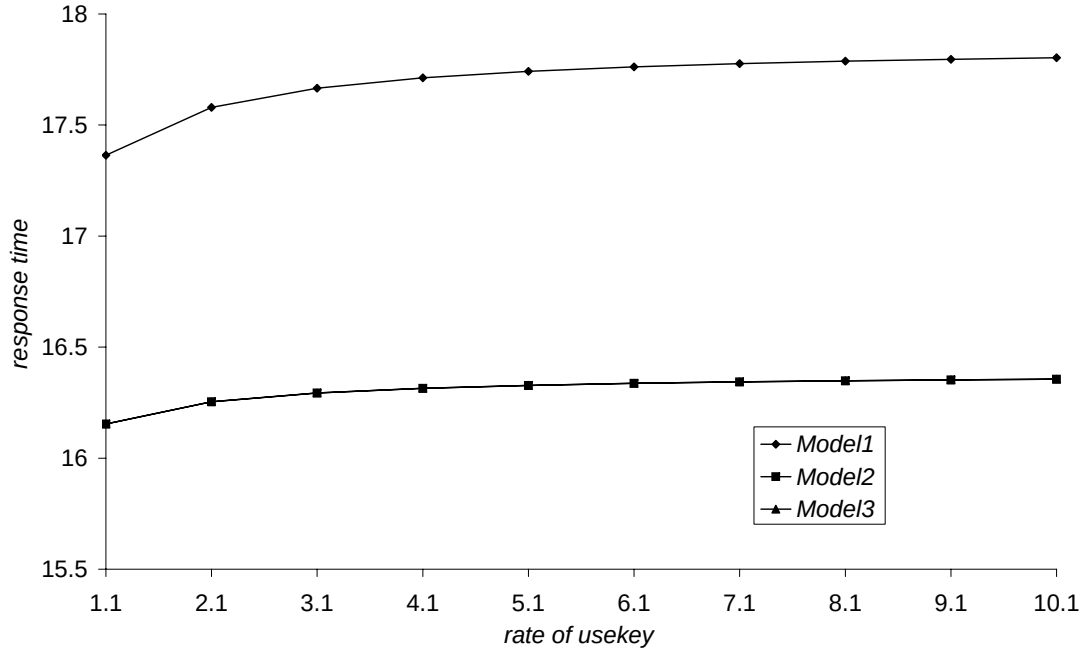

Figure 3.8: Response time varied with rate of use of the key.

Figure 3.8 shows the results of varying the rate of *usekey* (r_u) in case of three pair of clients. Response time becomes larger as the rate of *usekey* is increased; then, average response time in all models increased more slowly as r_u gets larger; finally, all three models almost keep the same increasing rate. Two parts are

involved in our defined response time: request time and service time. Request time is stable in this experiment. Increasing rate of *usekey* leads to requests to the KDC more frequently, this increasing waiting time. Thus, increasing average waiting time is the reason for average response time getting larger here. For the second feature, the average waiting time for one in n requests is that $\frac{n-1}{nr_s}$ (r_s is service rate of KDC), which equal to $\frac{1}{r_s} - \frac{1}{nr_s}$. The increasing part is getting smaller when more requests waiting. There is no changing influence fundamental differences among these three models, the results keep almost the same increasing rate consequently.

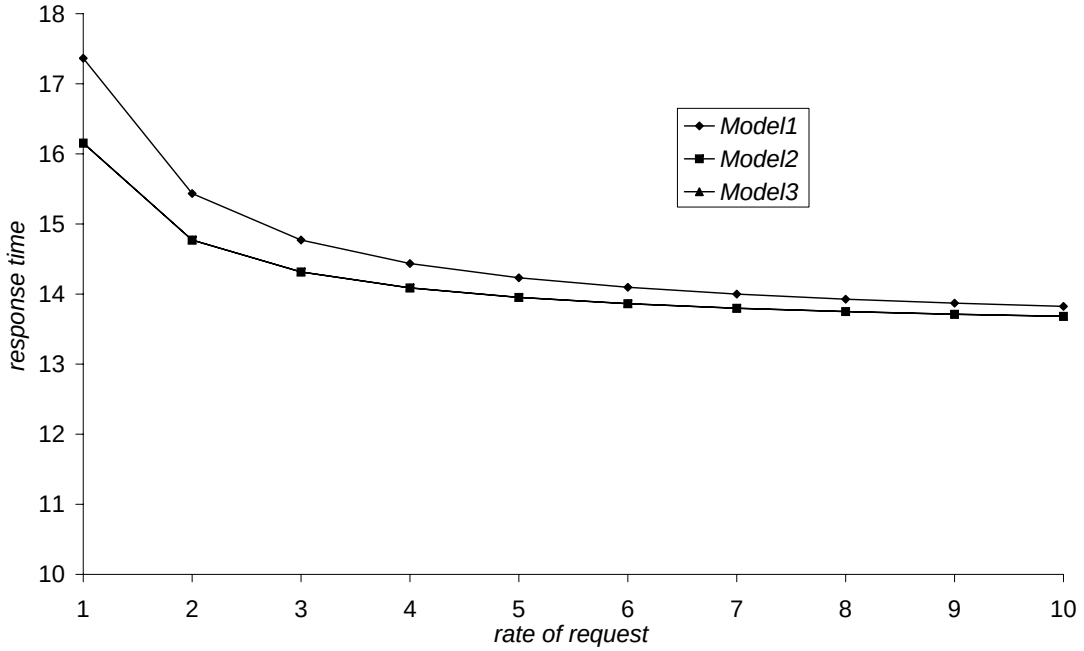

Figure 3.9: Response time varied against the rate of request.

The results of last experiment were showed in Figure 3.9. The difference here is that average response time for all three models decreased when r_q increases. Again, request time and service time are the two parts which influence response time. Request time ($\frac{1}{r_q}$) decreasing here has more effect than increasing of service time, for which influences the decreasing average response time. Another characteristic of these curves is that average response time in all models decreases more slowly when r_q is larger.

3.4 The Model

To consider the three modelling approaches, Model 1 is obviously less desirable according to the results. In addition, the *request* action in the KDC only allows customers start to enter the queue after the previous client has been served, which does not capture reality and is not efficient. In Model 2 we introduced the possibility that the KDC is serving multiple requests from multiple Alices. Each Alice still only makes one request at a time and each request is served by the KDC (we are not overly concerned here about the order of service). Note that due to the semantics of the specification events occur sequentially and not simultaneously. In addition we do not allow batched requests. It is worth observing here that Model 2 is cumbersome to specify; if we want to consider an extra client then we not only need to specify new *Alice* and *Bob* components, but also the KDC component needs to be modified to incorporate the additional behaviours. Model 2 also suffers from the commonly encountered state space explosion problem. For each Alice (and corresponding Bob) the state space is multiplied by another 6 behaviours, hence the state space is 6^N, where N is the number of client pairs (Alice+Bob).With $N = 9$ the state space has already grown to over 1 million states; if N is only 5, the solution still involves (very sparce) matrices with over 60 million elements. Even the best distributed Markov chain solvers generally only tackle state spaces of a few million states at most. Therefore, Model 3 is the most efficient modelling form of these three approaches.

3.4.1 Model simplification and approximation

Although Model 3 is relatively efficient, the state space explosion still exists. Although not as bad as Model 2, the KDC state in Model 3 still needs to be modified when one wants to add a new client to the system. To counter this, and to make the model easier to specify and understand, some simplification techniques have been applied to derive a form of the model which gives the same results for the key steady state metrics. This approach is based on the concept

known as bisimulation; whereby two models may be said to be equivalent if any sequence of actions that is possible in one model, has an equivalent sequence of actions (at the same rate) in the other model (*strong* bisimulation requires that equivalent actions have the same name, which is not the case here). This leads us to an alternative representation of the model as follows (*Model 4*).

$$KDC \stackrel{def}{=} (request, \top).KDC + (response, r_p).KDC$$

$$Alice \stackrel{def}{=} (request, r_q).(response, \top).Alice'$$

$$Alice' \stackrel{def}{=} (sendBob, r_B).(sendAlice, \top).(confirm, r_c).Alice''$$

$$Alice'' \stackrel{def}{=} (usekey, r_u).Alice$$

$$Bob \stackrel{def}{=} (sendBob, \top).(sendAlice, r_A).(confirm, \top).Bob'$$

$$Bob' \stackrel{def}{=} (usekey, \top).Bob$$

$$System \stackrel{def}{=} KDC \underset{\mathcal{L}}{\bowtie} \Big(Alice \underset{\mathcal{K}}{\bowtie} Bob\Big)\,[N]$$

Where, $\mathcal{L} = \{request, response\}$, $\mathcal{K} = \{sendBob, sendAlice, confirm, usekey\}$, N is number of pair of clients.

Clearly the component *Bob* is almost redundant, and the sharing of the action *request* and its enabling in the KDC component has no effect on the behaviour of the model. Hence an even simpler equivalent specification would be (*Model 5*): (this process has been termed *partial evaluation* in [11])

$$KDC \stackrel{def}{=} (response, r_p).KDC$$

$$Alice \stackrel{def}{=} (request, r_q).(response, \top).Alice'$$

$$Alice' \stackrel{def}{=} (sendBob, r_B).(sendAlice, r_A).(confirm, r_c).Alice''$$

$$Alice'' \stackrel{def}{=} (usekey, r_u).Alice$$

$$System \stackrel{def}{=} KDC \underset{response}{\bowtie} Alice[N]$$

Where N is number of Alices.

This model and the preceding one are clearly isomorphic, i.e. they have equivalent CTMCs with a one-to-one mapping between states and transitions. We can now apply the well known approximation technique of combining successive internal actions into a single action with a modified rate. This is equivalent to lumping states in the underlying Markov chain (Hillston [33] introduced the *weak isomorphism* equivalence for exactly this purpose). Thus we obtain the following simple form of the model (*Model 6*).

$$KDC \stackrel{def}{=} (response, r_p).KDC$$

$$Alice \stackrel{def}{=} (response, \top).(\tau, r_x).Alice$$

$$System \stackrel{def}{=} KDC \underset{response}{\bowtie} (Alice || \ldots || Alice)$$

Where r_x is given by

$$r_x = \left(\frac{1}{r_q} + \frac{1}{r_B} + \frac{1}{r_A} + \frac{1}{r_c} + \frac{1}{r_u} \right)^{-1}$$

Model 6 is equivalent to a simple closed queueing system with one queueing station (the KDC) and an exponential delay after service before returning to the queue. It is a simple matter to write down the balance equations for such a system.

$$r_p \pi_i = (N + 1 - i) r_x \pi_{i-1} \ , \ 1 \leq i \leq N$$

where π_i is the steady state probability that there are exactly i jobs waiting for a response from the KDC and N is the number of pairs of clients (the number of instances of *Alice* in the above PEPA model specification). Thus it is possible to derive expressions for the average utilisation of the KDC and the average number of requests waiting for a response.

$$U = 1 - \left[N! \sum_{i=0}^{N} \frac{\rho^i}{(N-i)!} \right]^{-1}$$

and,

$$L = N!(1 - U) \sum_{i=1}^{N} \frac{\rho^i i}{(N-i)!}$$

where $\rho = r_x / r_p$.

This approximation is, in fact, an *M/M/1/./N* queue and the throughput and average response time are easily computed from the above expressions (see Mitrani [46] pages 195-197).

$$T = (N - L) r_x$$

and

$$W = \frac{N}{T} - \frac{1}{r_x}$$

3.4.2 Numerical results

This approximation is now compared with simulation results for the full model. The simulation was written in Java using the roulette wheel approach. The simulation has been verified numerically against the PEPA model using the PEPA Workbench Eclipse Plug-in [68] for small numbers of clients ($N \leq 6$). The PEPA Workbench will not give results for larger models due to problems with performing computations on the large matrices involved, hence the need for the simulation. Initially in the experiments which follow, the parameters are set to 1.0 (except r_u=1.1 for numerical computation reasons in the PEPA Workbench) and other parameters are varied as shown.

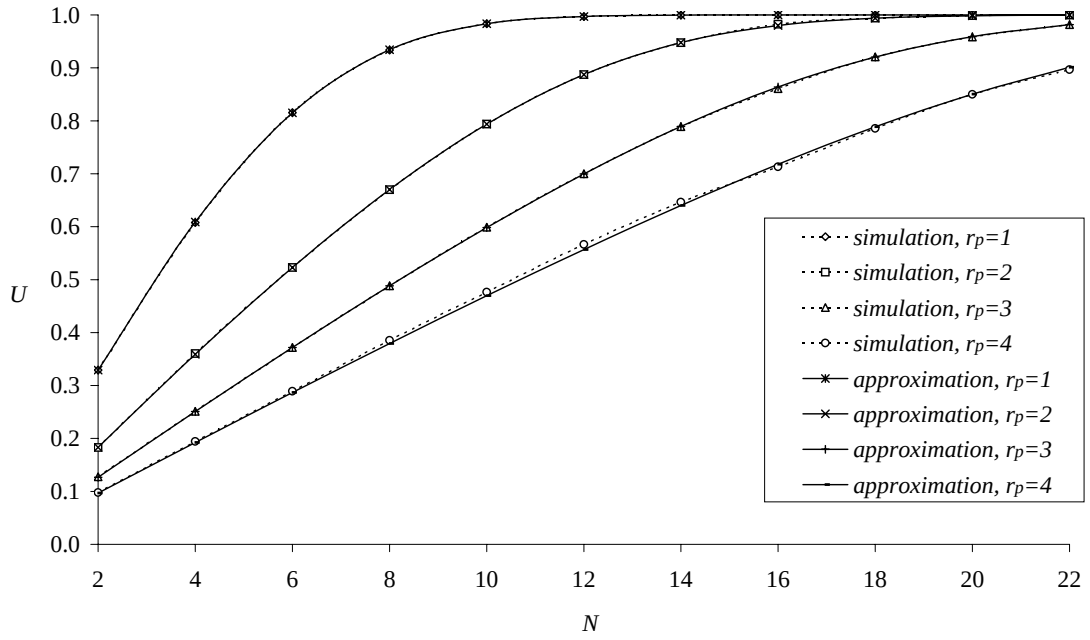

Figure 3.10: Average utilisation varied against the number of client pairs. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

In Figure 3.10 we show the utilisation (of the KDC) varied against the numbers of client pairs for both the simulation and the approximation for various values of r_p. Increasing the value of r_p in this way is equivalent to replacing the KDC with a faster server. In Figure 3.11 we show the average response time (average waiting time plus average service time) of the KDC for the same systems. Clearly, for both metrics, there is a very close match between the simulation and the

approximation. Hence, in Figures 3.12 and 3.13, we show the percentage error, given as (approximation-simulation)/simulation, for both metrics to provide a greater insight into the accuracy of the approximation. This shows that the approximation and simulation agree to within 2% for the utilisation and within 4% for average response time. In all cases the simulation is run to a terminating condition of a 95% confidence interval. Not surprisingly this becomes increasingly more difficult to attain as N increases, hence the run-time increases with N.

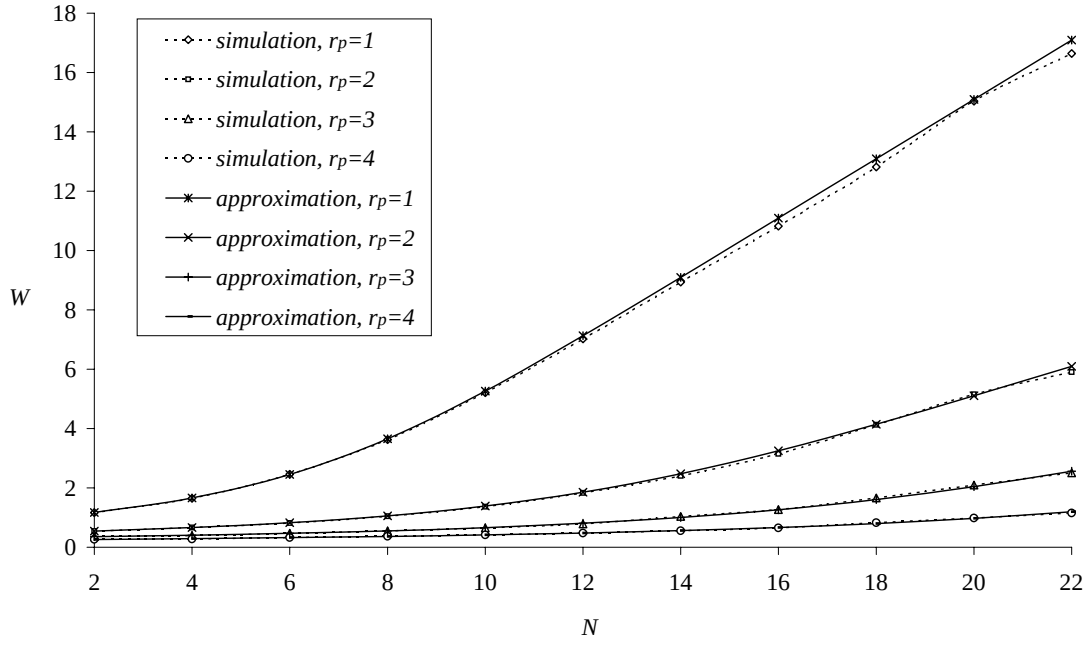

Figure 3.11: Average response time varied against the number of client pairs. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

The most significant difference between the simulation and the approximation is the time it takes to derive results. The simulation took several weeks to code and each run takes in excess of 10 hours (we are not claiming this to be the most efficient simulation possible) whereas the approximation was coded into MS Excel in less than half an hour and results are almost instantaneous. It is worth noting that these metrics are based on long run averages, which we would expect the approximation to be fairly accurate in predicting, particularly utilisation. If the measure of interest was a transient measure then the lumping of states might not give such an accurate picture. Furthermore, if we wished to predict the end to end performance of the protocol, i.e. from *request* to *confirm*, then we would need

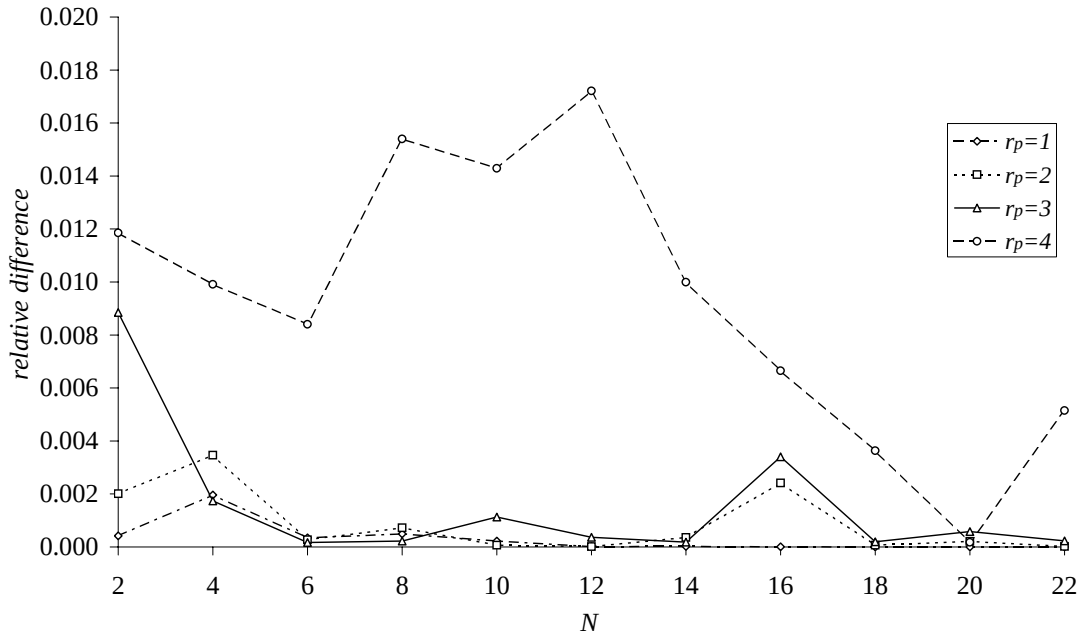

Figure 3.12: Relative error in utilisation of approximation compared to simulation. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

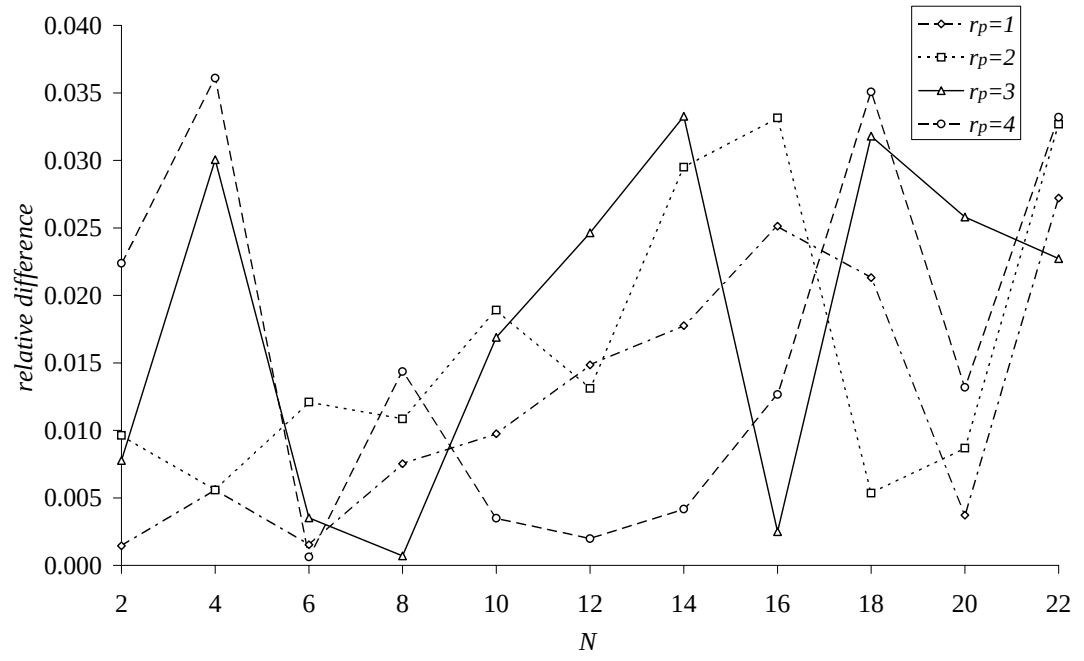

Figure 3.13: Relative error in average response time of approximation compared to simulation. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

to perform a slightly different approximation which separates the *usekey* action from the other lumped actions.

The results show that there is obviously a benefit from increasing the server speed at the KDC, but the increase in server speed is not necessarily exactly proportional to the increase in capacity. For example, if we have a target maximum utilisation of 0.65, then with $r_p = 1$ the KDC can cope with at most 4 client pairs. If we increase the server rate to $r_p = 3$ then the capacity is 10 client pairs, not 12 as we might intuitively expect. However, if we specify the maximum average response time to be 2, then $r_p = 1$ gives the capacity as 4, $r_p = 2$ gives 12, and $r_p = 3$ gives the capacity as 18 client pairs. Clearly in this case the increase in server speed from $r_p = 1$ to $r_p = 2$, or $r_p = 3$, has a significantly greater impact on the client capacity than we might expect. Note also that, whilst intuitively we may consider that it is possible that a greater impact could be made by considering multiple KDC severs, we know (from well established queueing theory results) that for a simple *M/M/k* queue it is preferable to have one fast server than two of half the speed. Clearly therefore we would rather double the speed of the processor, than double the number of processors at the KDC (although doing both would clearly be beneficial).

In the above experiments the duration for which the session key is used is set to be approximately the same as the durations for any other action. We have done this so that we can explore the behaviour of the KDC when it is heavily loaded, despite only having a small number of client pairs. Clearly this is not a practical scenario and having established the accuracy of the approximation we can now go on to consider larger systems with a greater duration of the use of the session key.

Figures 3.14 and 3.15 show the utilisation and average response time for various values of r_u and r_p when $N = 150$. When the use rate is low ($r_u = 0.01$) the performance is good for $r_p > 2$ (in fact the response time for $r_p = 2$ is more than five times that of $r_p = 5$, although this is not clear in the graph). However, increasing the use rate has a dramatic effect on both the utilisation and the average response time. The systems rapidly become saturated, except $r_p = 5$ (and to a lesser extent $r_p = 4$) which grows more gently. At $r_u = 0.05$ all the systems are saturated (100% utilisation). A similar picture is evident

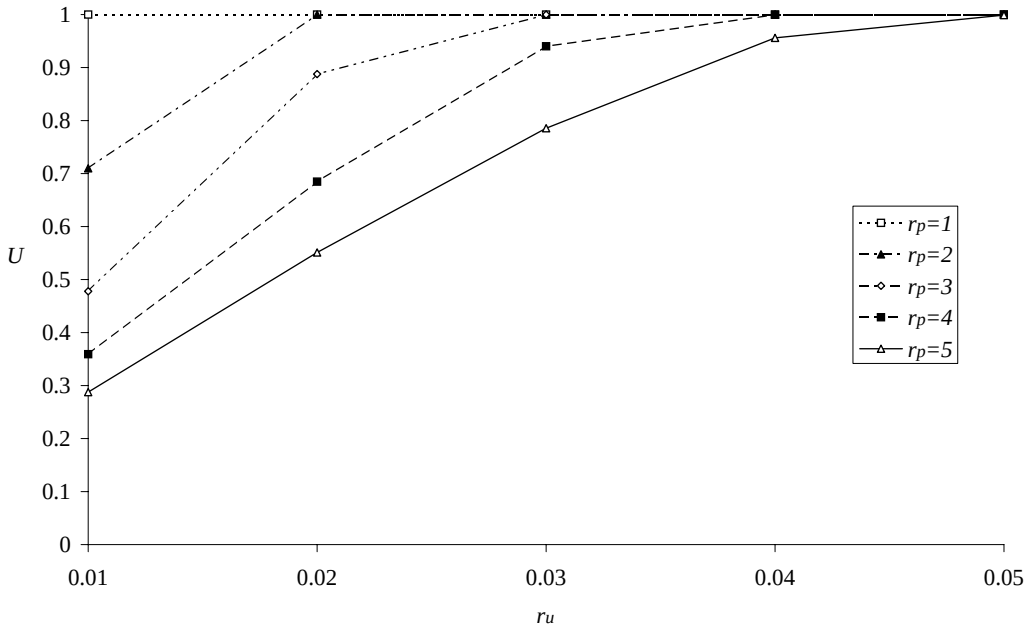

Figure 3.14: Average utilisation varied against the rate of session key use, r_u. $r_q = r_A = r_B = r_c = 1$, $N = 150$

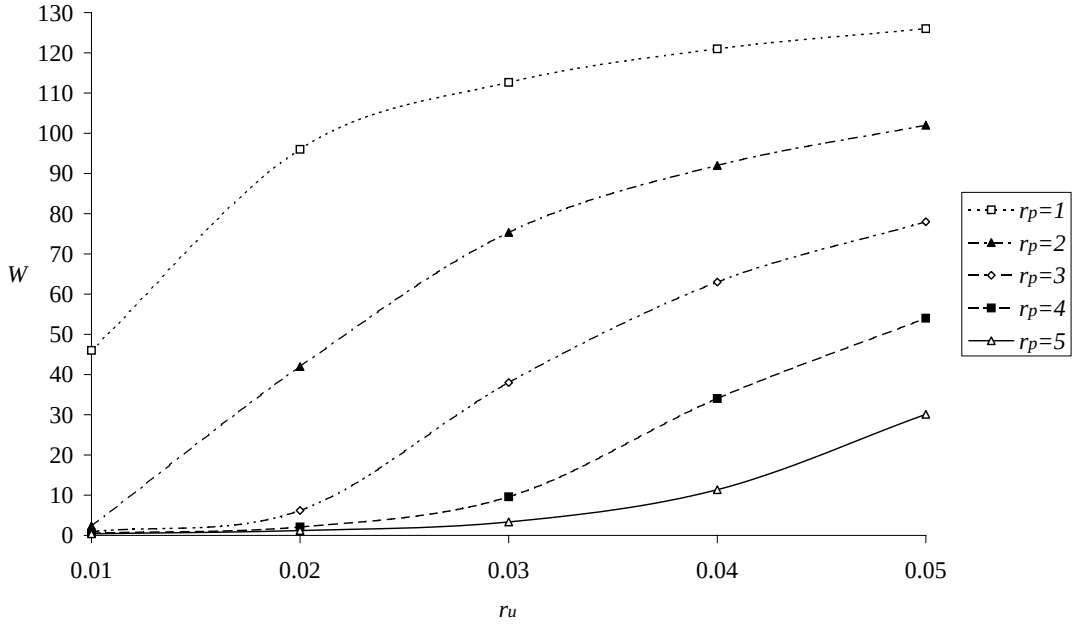

Figure 3.15: Average Response time varied against the rate of session key use, r_u. $r_q = r_A = r_B = r_c = 1$, $N = 150$.

for the average response time. For $r_p = 5$ the average response time increases exponentially. However, for $r_p = 1$, where the response time is obviously much greater, the increase is inversely exponential, i.e. the rate of increase decreases as

r_u increases. This is because $r_p = 1$ is already saturated at $r_u = 0.01$ and so a large number of clients are already spending a long time in the queue awaiting a response from the KDC. Hence, decreasing the time they use the session key does not greatly change their overall behaviour (which is already dominated by queueing). The other cases of $1 < r_u < 5$ fall between these extremes, with the saturation point being clearly evident in the plot of the average response time.

3.5 Fluid analysis

Thus far we have considered a traditional approach to modelling and analysis. In this section we consider an alternative approach proposed by Hillston [34], based on the solution of ordinary differential equations. In this style of analysis, the model is expressed as a number of replicated components and the ODEs represent the flow between behaviours (PEPA derivatives) of the components. Thus, by solving the ODEs, it is possible to 'count' the number of components behaving as a given derivative at any given time, t. In the absence of oscillations, the limit, $t \longrightarrow \infty$, then gives a steady state value.

Rewriting model slightly, removing redundancy and naming each derivative of *Alice* (for clarity) we get:

$$KDC \stackrel{def}{=} (response, r_p).KDC$$

$$\begin{aligned}
Alice &\stackrel{def}{=} (request, r_q).Alice_1 \\
Alice_1 &\stackrel{def}{=} (response, r_p).Alice_2 \\
Alice_2 &\stackrel{def}{=} (sendBob, r_B).Alice_3 \\
Alice_3 &\stackrel{def}{=} (sendAlice, r_A).Alice_4 \\
Alice_4 &\stackrel{def}{=} (confirm, r_c).Alice_5 \\
Alice_5 &\stackrel{def}{=} (usekey, r_u).Alice
\end{aligned}$$

The system is then defined as:

$$KDC[K] \underset{response}{\bowtie} Alice[N]$$

Where, K is the number of *KDC*'s (hitherto $K = 1$) and N is the number of client pairs (*Alices*'s). It is then a simple matter to write down the ODEs for this system as follows.

$$\begin{aligned}
\frac{d}{dt}Alice(t) &= r_u Alice_5(t) - r_q Alice(t) \\
\frac{d}{dt}Alice_1(t) &= r_q Alice(t) - r_p min(KDC(t), Alice_1(t)) \\
\frac{d}{dt}Alice_2(t) &= r_p min(KDC(t), Alice_1(t)) - r_B Alice_2(t) \\
\frac{d}{dt}Alice_3(t) &= r_B Alice_2(t) - r_A Alice_3(t) \\
\frac{d}{dt}Alice_4(t) &= r_A Alice_3(t) - r_c Alice_4(t) \\
\frac{d}{dt}Alice_5(t) &= r_c Alice_4(t) - r_u Alice_5(t) \\
\frac{d}{dt}KDC(t) &= 0
\end{aligned}$$

In our analysis we are interested primarily in the number of client pairs awaiting a response from the *KDC* (or *KDC*'s). This is represented in the model by the number of $Alice_1$'s; $L(N) = Alice_1(t \longrightarrow \infty)$ when there are N client pairs (*Alice*'s) in the population. From this we can derive the average response time which can be compared with that derived from the queueing network approximation. We compute the average response time for a system of N client pairs and one KDC server ($K = 1$), W(N), as follows;

$$W(N) = \frac{L(N-1)+1}{r_p}$$

This computation is based on the queueing theory result of an arrival as random observer, see Mitrani [46] page 141 for example. For $K > 1$ the computation is only slightly more complex. If the random observer sees a free server, then the average response time will be the average service time. However, if the random

observer sees all the servers busy, then the average response time will be the average service time plus the time it takes for one server to become available (including scheduling the other jobs waiting ahead of the random observer).

$$\begin{aligned} W(N) &= \frac{1}{r_p}\ ,\ L(N-1)+1 \leq K \\ W(N) &= \frac{1}{r_p} + \frac{L(N-1)+1-K}{Kr_p} = \frac{L(N-1)+1}{Kr_p}\ ,\ L(N-1)+1 > K \end{aligned}$$

It is a feature of the fluid flow approximation that (for $t > 0$) the KDC will never be idle, but instead will always have some fluid flowing through it. As such we are unable to compute the utilisation of the KDC directly. This is clearly a limitation of this form of analysis.

3.5.1 Numerical results of ODEs

Figure 3.16 shows the evolution over time of the number of clients awaiting a response as derived from the ODE analysis. Initially all the clients are behaving as *Alice*, hence $Alice_1(0) = 0$. Shortly after the start there is a large influx of fluid into $Alice_1$ before the system settles into a stable flow. Interestingly this initial surge is much more pronounced when $r_p = 4$ than $r_p = 1$. This is due to the fact that the flow out of $Alice_1$ is much greater when $r_p = 4$.

Figure 3.17 shows the average response time calculated by the ODE method, compared with the queueing approximation described earlier. This approximation has previously been compared with simulation and shown to be accurate to within the 95% confidence interval of the simulation in Section 3.4.2.

We expect the ODE method to be accurate when N is large. Figure 3.17 shows that it is possible to generate accurate results even when N is quite small. However, there is a clear difference between the two methods where the gradient changes. This is shown more explicitly in Figure 3.18, where the evolution of the ODEs is compared with the stochastic simulation of the PEPA model [6] derived directly using the PEPA Eclipse Plug-in. When N is sufficiently far

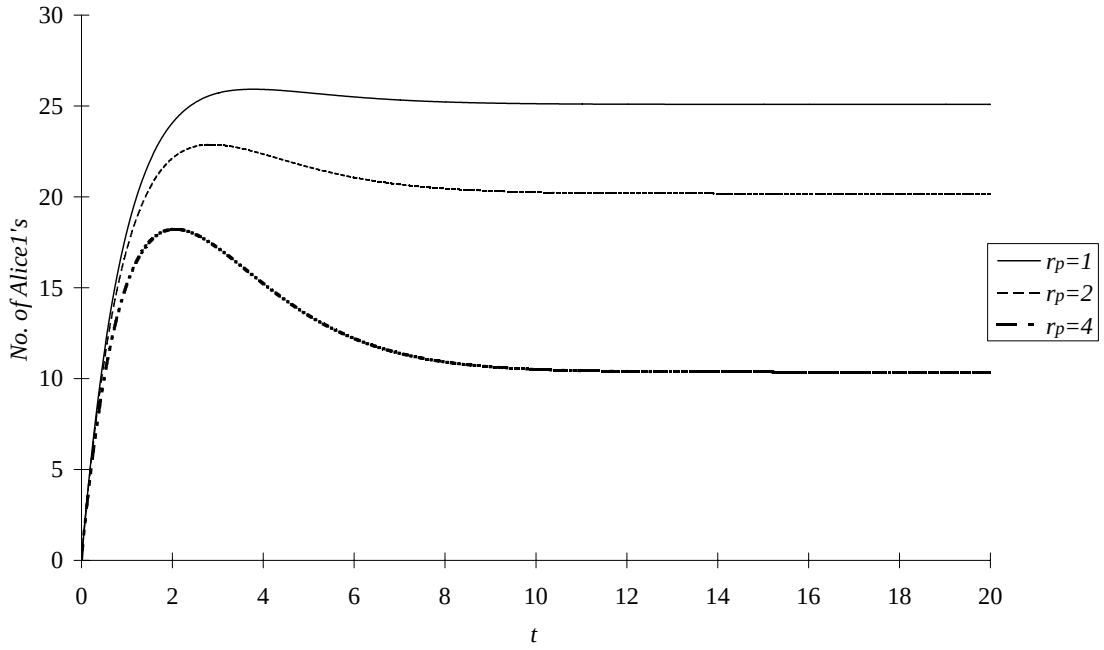

Figure 3.16: Number of waiting clients over time, $N = 30$, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

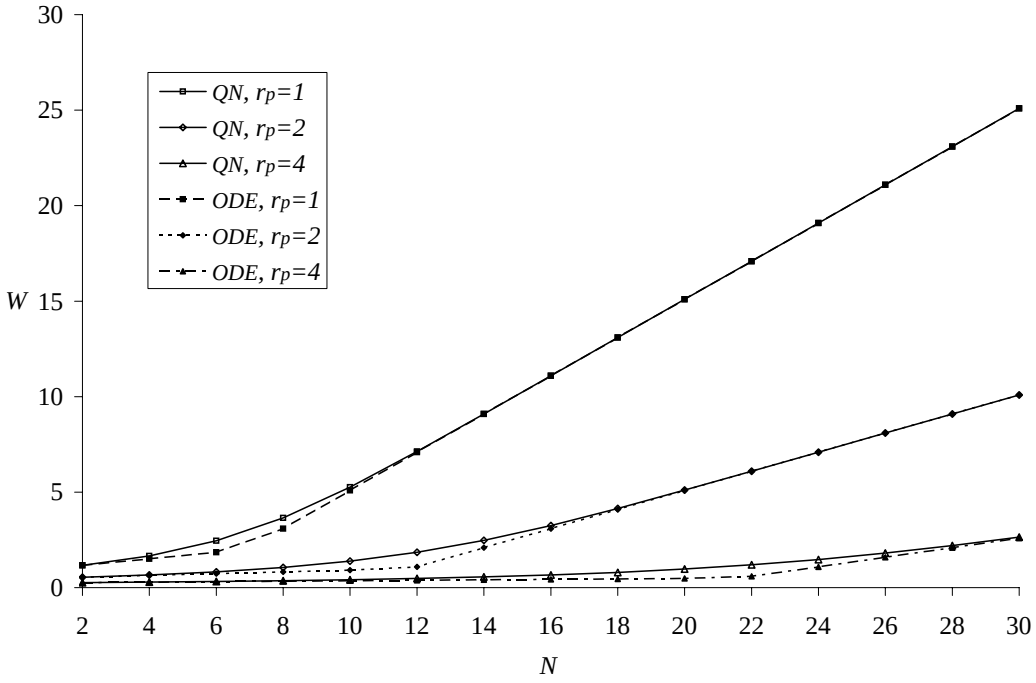

Figure 3.17: Average response time calculated by the ODE method and QN approximation, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

from the gradient change there is good agreement between the ODE solution and the stochastic simulation. However, at $N = 6$ the divergence is significant;

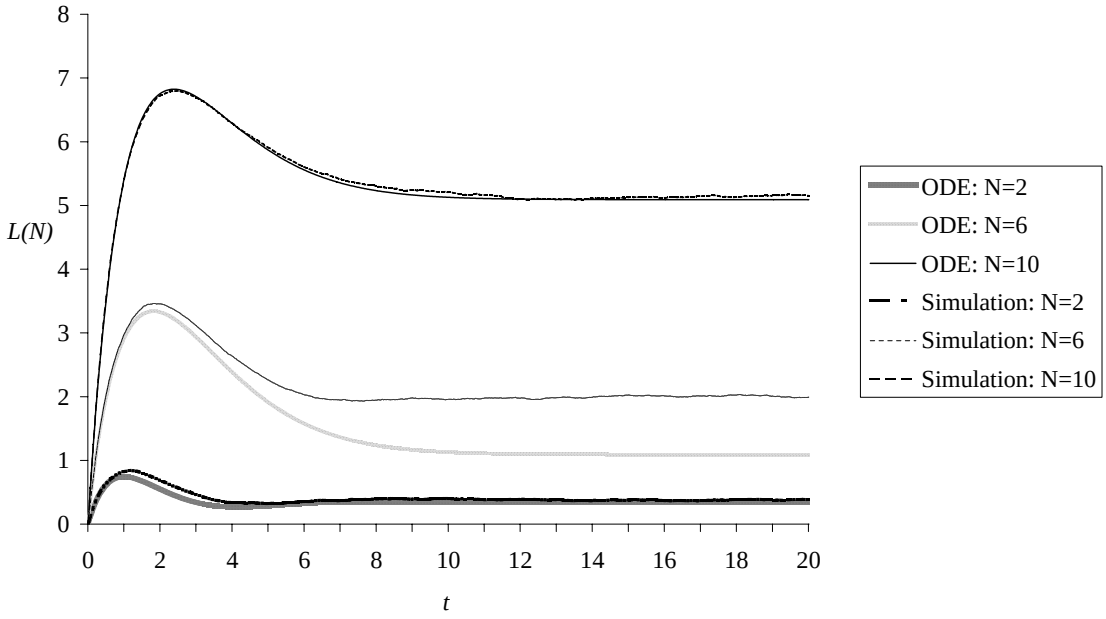

Figure 3.18: Number of waiting clients over time, $r_p = r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

the stochastic simulation never achieves the lower queue length predicted by the ODEs.

It is of clear practical importance to be able to predict the divergence. This point, N^*, can be estimated using the method of asymptotic bounds of closed queueing networks (see Haverkort [28] pages 245-246 for example).

$$N^* = K + \frac{Kr_p}{r_x} = K + Kr_p\left(\frac{1}{r_q} + \frac{1}{r_B} + \frac{1}{r_A} + \frac{1}{r_c} + \frac{1}{r_u}\right) \tag{3.3}$$

Below N^* the asymptotic bound is given as

$$L(N) = \frac{Nr_x}{r_x + r_p} \tag{3.4}$$

Above N^* the asymptotic bound is given as

$$L(N) = \frac{Nr_x - Kr_p}{r_x} \tag{3.5}$$

These bounds can also easily be found by solving the ODEs analytically in the limit $t \longrightarrow \infty$, where the $min(KDC(t), Alice_1(t))$ term is replaced with $Alice_1(t)$ and $KDC(t)$ respectively. Thus, in this instance at least, the ODE solution is giving an alternative means for calculating known asymptotic results for closed queueing networks (see [65]). Note that $W(N)$ is computed from $L(N-1)$, and so in Figure 10, the divergence occurs at approximately 6.91 ($r_p = 1$), 11.82 ($r_p = 2$) and 21.64 ($r_p = 4$), i.e. $N^* + 1$.

We have also compared the two methods for larger values of N and have found there to be almost no difference for $N > 40$ for the parameters used here. It is important to note that there are numerical issues with computing the queueing approximation due to the difficulty of handling large factorials (and their reciprocals) and these problems do not occur with the ODE solutions or asymptotic results. Thus, as long as we avoid the region around N^*, the ODE solution is giving accurate results without problems with scalability.

3.5.2 Multiple KDC servers

We now turn our attention to the consideration of multiple servers at the KDC. In particular we would wish to know if it is more beneficial to increase the number of servers or increase the speed of the server. It is well known that for an M/M/K queue, it is preferable to have 1 server serving at rate μ than K servers serving at rate μ/K. This is because if there are less than K jobs in the queue then some of the K servers will be idle, thus reducing the overall service rate. In the ODEs above this is evident in $r_p min(KDC(t), Alice_1(t))$. If $Alice_1(t) > K$ then all K servers are in use and the flow rate from the KDC would be Kr_p. However, if $Alice_1(t) < K$ then fewer servers would be in use and the rate would be $r_p Alice_1(t)$.

There is an issue with specifying the interactions between multiple components in PEPA that we need to be aware of here. We can easily increase the number

of servers at the KDC in the PEPA specification.

$$System \stackrel{def}{=} (KDC|| \ldots ||KDC) \underset{response}{\bowtie} (Alice|| \ldots ||Alice)$$

However, we must give the *response* action in *Alice* the rate r_p, rather than being passive.

$$Alice \stackrel{def}{=} (response, r_p).(\tau, r_x).Alice$$

This is because of the way in which a passive action would be subject to the *apparent rate* in PEPA. Hence, K KDCs and 1 Alice would give rise to *response* occurring at rate Kr_p; whereas if the rate is r_p in both KDC and *Alice*, then this problem does not arise.

Thus the approximation becomes an *M/M/K/./N* queue, where K is the number of instances of the KDC component (i.e. servers at the KDC). Hence the balance equations become,

$$\begin{aligned} (N-i)r_x\pi_i &= (i+1)r_p\pi_{i+1}\ , 0 \leq i < K \\ (N-i)r_x\pi_i &= Kr_p\pi_{i+1}\ , K \leq i < N \end{aligned}$$

Thus we can calculate π_0

$$\pi_0 = \left[N! \sum_{i=0}^{K-1} \frac{\rho^i}{(N-i)!i!} + N! \sum_{i=K}^{N} \frac{\rho^i}{(N-i)!K!K^{i-K}} \right]^{-1}$$

The average queue length can be then calculated by

$$Ł = N!\pi_0 \left[\sum_{i=1}^{K-1} \frac{\rho^i i}{(N-i)!i!} + \sum_{i=K}^{N} \frac{\rho^i i}{(N-i)!K!K^{i-K}} \right]$$

The average response time and throughput can then be computed as before.

Figure 3.19 shows the proportion of Alices waiting at the KDC (i.e. $L(N)/N$) for $K = 1$ with $r_p = 4$ and $K = 4$ with $r_p = 1$ for both the queue approximation and the ODE solution. When N is large (in this case $N \geq 25$) the ODE values are the

same, however for smaller N the single faster server is seen to perform better (for the reason discussed above). The reason the ODE values are identical for large N is simply that the fluid level of Alices waiting at the KDC will never fall below K in the ODE solution. The values for the QN approximation differ slightly from each other, even when $N = 40$. This is because even at this load there is still the chance that the queue will fall below 4 requests for short periods. Clearly, as N increases the probability that this happens will become increasingly insignificant and hence the values will converge.

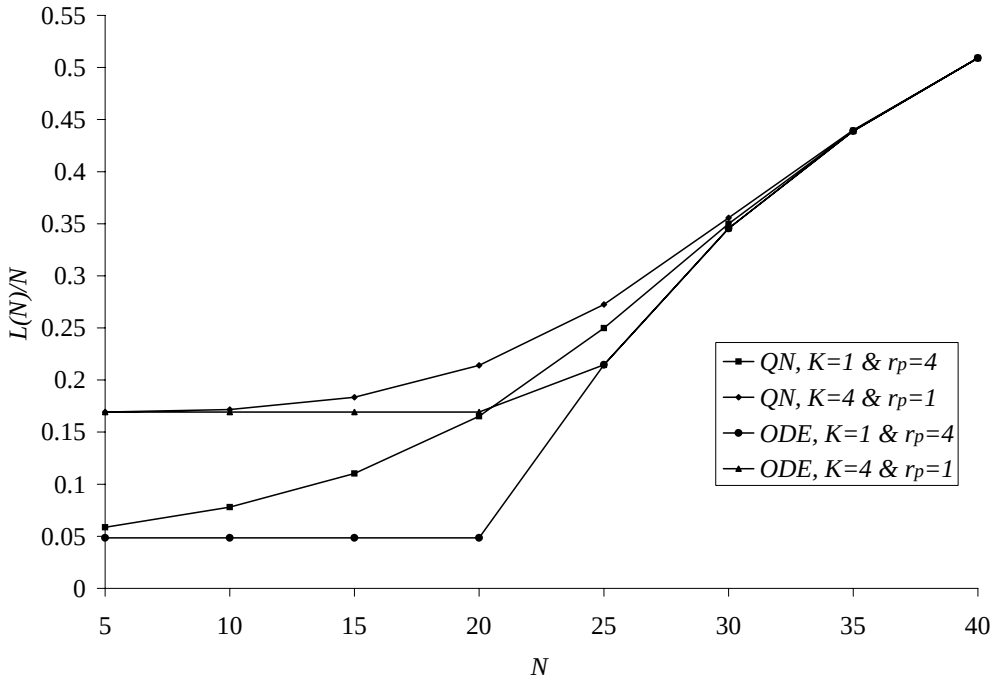

Figure 3.19: Proportion of $Alice_1$ components, calculated by ODE solution and QN model, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

There is a clear divergence between the ODE and QN results around the change in gradient as we have already observed in Figure 3.17. Figure 3.20 shows this in more detail for the average queue size. Note that although the two ODE solutions converge at $N = 25$, there is still a significant difference with the queueing network (QN) solutions at this point, near to N^*.

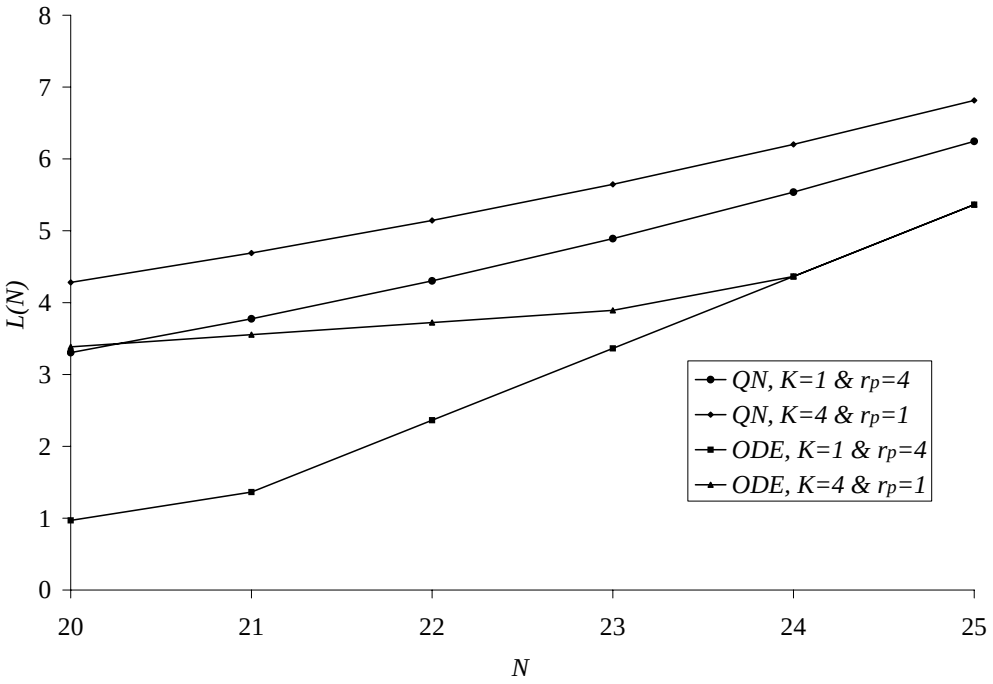

Figure 3.20: Average queue length calculated by ODE solution and QN model, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

3.6 Utility function

We now return to our proposed utility function.

$$C = c_1 L + c_2 K r_p \ \ , \ c_1, c_2 \geq 0 \tag{3.6}$$

Taking the asymptotic results (3.4) and (3.5) for L gives a simple exposition of this utility function. If $N < N^*$ then

$$C = \frac{c_1 N r_x}{r_x + r_p} + c_2 K r_p \tag{3.7}$$

This function is always increasing with K. If $N > N^*$ then

$$C = c_1 \frac{N r_x - K r_p}{r_x} + c_2 K r_p \tag{3.8}$$

If we seek to find the largest value of N before the performance begins to signif-

icantly degrade, then (3.7) and (3.8) will increase linearly with N (as all other parameters are fixed). Clearly, the rate of cost increase after N^* will always be greater than the rate below N^*. Thus, N^* represents the point at which the cost begins to grow significantly, especially if r_p is large. Similarly, if we seek to minimise C with respect to K for a given N then $c_1 N r_x$ is fixed and so (3.8) is increasing with K if $c_1 < r_x c_2$.

Now, recall that (3.3) gives

$$N^* = K\left(\frac{r_x + r_p}{r_x}\right)$$

So, for small K, N^* will also be small and as K increases, so will N^*. If N is fixed, then we can define K^* such that:

$$K^* = N\left(\frac{r_x}{r_x + r_p}\right)$$

Clearly, if $K < K^*$ then $N > N^*$ and if $K > K^*$ then $N < N^*$. Thus, (3.7) and (3.8) predict the optimal value of K to be $K_{opt} = 1$ if $c_1 < r_x c_2$ and $K_{opt} = INT(K^* + 0.5)$ if $c_1 > r_x c_2$ However, as these asymptotic results will always underestimate L in the region around N^* the relationship is more accurately described as $K_{opt} = 1$ if $c_1 \leq r_x c_2$ and $K_{opt} \approx K^*$ if $c_1 > r_x c_2$.

3.6.1 Numerical results of utility function

We now illustrate the scenarios described above through numerical examples. Figures 3.21 and 3.22 show the cost varied against the number of clients, calculated by the queueing network model and ODEs respectively. Clearly, under these parameter values with $r_p = 1$, the cost rises rapidly at around 80 clients, which is the approximate maximum capability that the KDC can handle before performance starts to significantly degrade. In a small system ($N < 60$), the utility function is dominated by $c_2 K r_p$, hence the cost is greater for a faster KDC. The reason for this is the server will often be idle, and so the system is

not making efficient use of computational resources. In the cases $r_p = 2, 3$ and 4, when $N = 120$ clients the exact computation of C using the queueing network model becomes more costly, and so we adopt the use of the ODE solution. Figure 3.22 shows that the maximum number of clients which can be supported by the KDC in case of $r_p = 2, 3$ and 4 are around 210, 310 and 410, respectively with these parameter values. Thus, doubling the service rate from $r_p = 2$ to $r_p = 4$ effectively doubles the capacity of the system in this case.

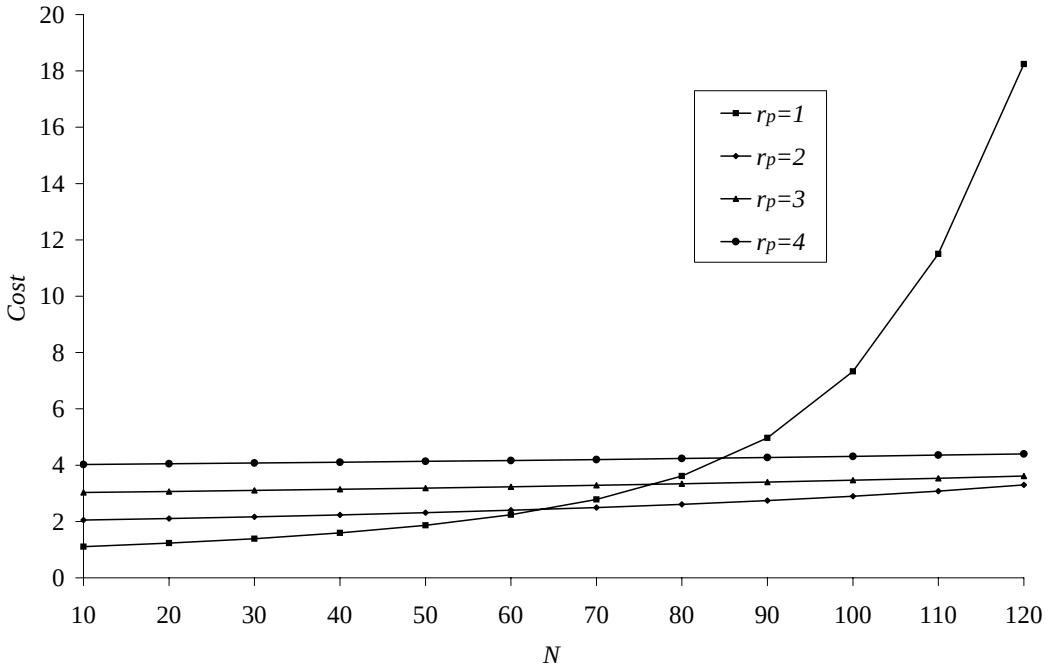

Figure 3.21: Cost varied against the number of clients calculated by the queueing network model, $K = 1$, $c1 = c2 = 1, r_q = r_A = r_B = r_c = 1$, $r_u = 0.01$

We now start to seek to find the optimal value of K in order to minimize the cost of the system. According to the analysis in the previous section, K_{opt} always equals 1 when $c_1 \leq r_x c_2$. Hence, investigation of K_{opt} when $c_1 > r_x c_2$ is more interesting and valuable. We employ the ODE solution as the more efficient approach in optimisation, particularly as we wish to explore larger population sizes ($N = 500, 1000, 2000$ clients). In addition, we consider the running cost c_2 to be either 1/10 or 10, to explore the case where we are more interested in minimising running costs or queue length (hence response time).

Figure 3.23 shows the results of cost varied with number of KDCs. Generally,

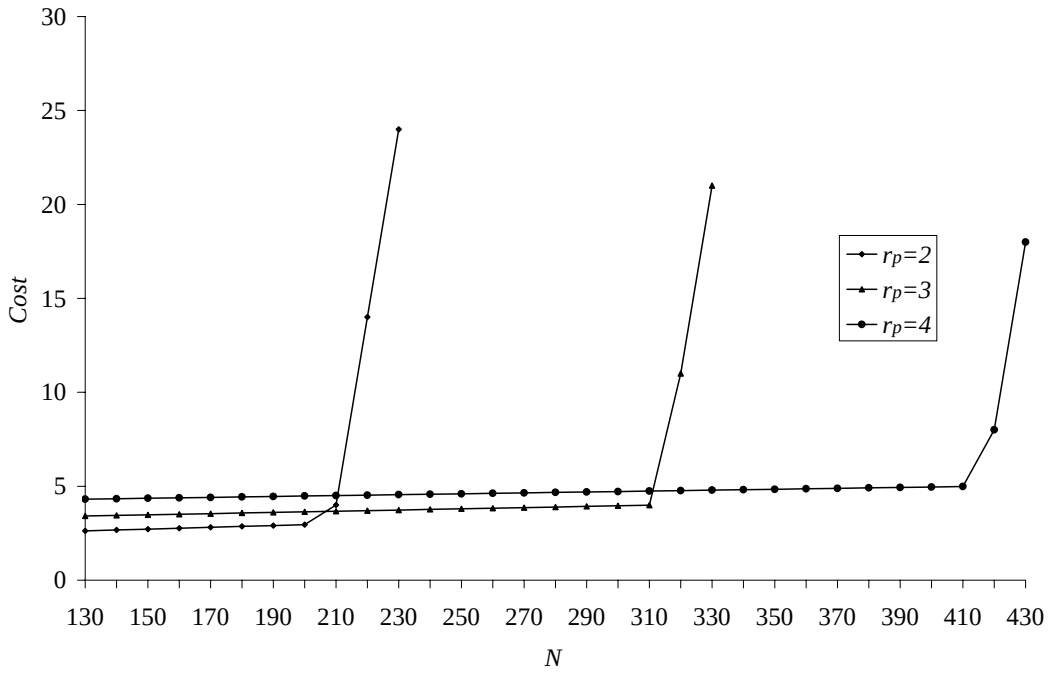

Figure 3.22: Cost varied against the number of clients calculated by ODEs, $K = 1$, $c1 = c2 = 1, r_q = r_A = r_B = r_c = 1$, $r_u = 0.01$

more clients need more KDCs to support them, in order to reduce the optimal cost. In each of the three cases ($N = 500$, $N = 1000$ and $N = 2000$ clients), the larger c_2 results in a decreasing cost before the optimal point and an increasing rate after that. For any given number of clients, the optimal value of K in Figure 14 is shown to be independent of c_2. K_{opt} =5, 10, 20 for $N = 500, 1000, 2000$ although results from the asymptotic solution are not entirely accurate around N*. The real optimal value of K should be around these values (K^*). Consequently, we calculated a range of K around K^* by the queueing network model to show how the true value of K_{opt} can vary from K^*. Figure 3.24 compares the cost around $K*$ in case of 1000 clients with $c_2 = 0.1$ and 10 calculated by ODEs and the queueing network model. In the case of $c_2 = 10$, K_{opt} is 10 which is the same as the ODE result but with a slightly different value of cost. In the case of $c_2 = 0.1$, 15 KDCs gives the minimal cost with value 11.144 which is slightly smaller than case of $K = 14$ and 16 with value 11.184 and 11.178, respectively. The cost at K^* is calculated by ODEs which is clearly close to the optimal value. As such we propose that the asymptotic bound, whilst not giving the exact optimal solution in every case, can often be used to calculate a near

optimal cost extremely efficiently.

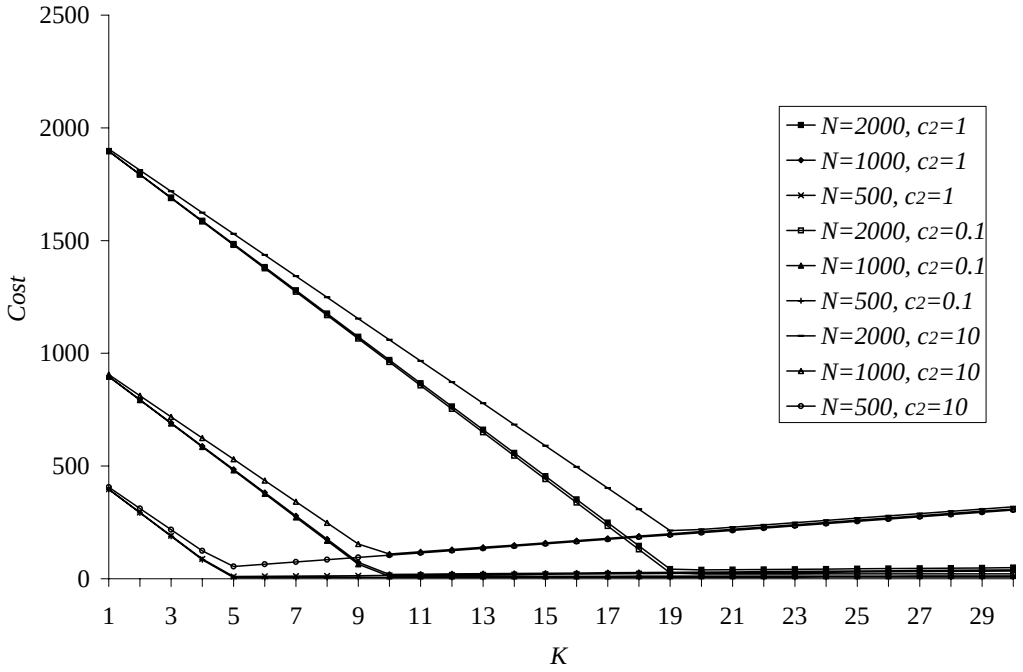

Figure 3.23: Cost varied with number of KDCs calculated by ODEs, $c1 = 1$, $r_q = r_p = r_A = r_B = r_c = 1.0$, $r_u = 0.01$

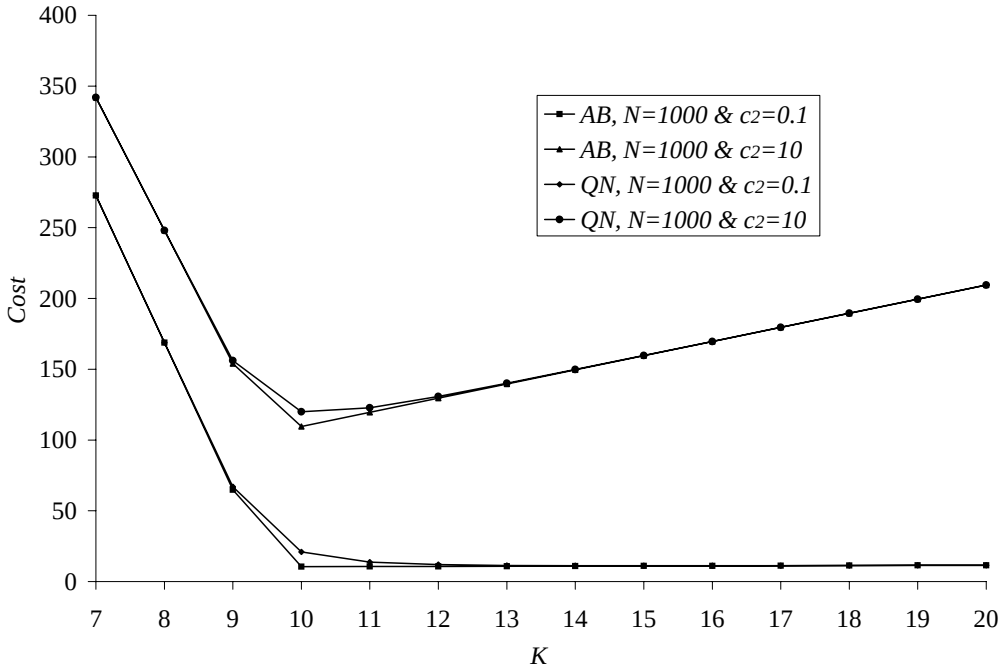

Figure 3.24: Cost varied with number of KDCs calculated by ODEs and QN model, $c1 = 1$, $r_q = r_p = r_A = r_B = r_c = 1.0$, $r_u = 0.01$

Figure 3.25 shows the cost varied with the rate of key refresh, calculated by ODEs.

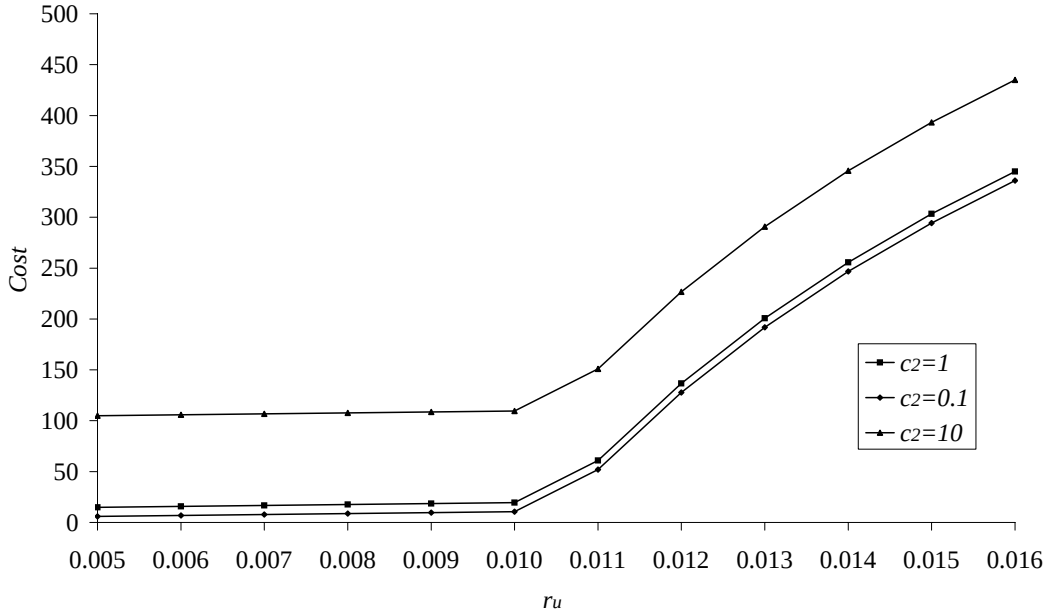

Figure 3.25: Cost varied with rate of *usekey* calculated by ODEs, $c1 = 1$, $r_q = r_p = r_A = r_B = r_c = 1.0$, $N = 1000$, $K = 10$

In this experiment, the number of servers and clients has been fixed. Generally, large r_u results in a greater cost, caused by more frequently refreshing the key and hence more workload has been added to the KDC servers, and consequently more waiting jobs. As noted by Stallings [59], although the cost is increased, the system becomes more secure as the eavesdropper has less cipher text to crack any given session key. A security manager must try to make a balance between security and performance. The balance point here is the value of r_u where the cost starts to increase rapidly. It is a simple matter to find that $r_u = 0.01$ is the approximate optimal refresh rate for all three cases considered ($c_2 = 0.1, 1$ and 10). Thus it is clear that the simple ODE solution is sufficient to find the measure require and in practical circumstance, the security manager could choose a value for $r_u \leq 0.01$.

3.7 Summary

In this chapter we have shown how a key distribution centre can be modelled and analysed using the Markovian process algebra PEPA. The intuitive means

of modelling the protocol is cumbersome and suffers from state space explosion, preventing meaningful analysis with significant numbers of clients. We have taken three main approaches to coping with this problem; first we have implemented a simulation of the model, secondly we have attempted to approximate the system behaviour with a much simpler model, and finally ODE analysis has been applied as a deterministic fluid flow analysis. The approximation shows good accuracy of prediction when compared with simulation, scales exceptionally well and is fast to compute. ODE results have been compared with those derived from the approximation. This (ODE) approach has two main limitations. Firstly, it is not always as accurate as the queueing approximation and secondly, we have not been able to obtain all our desired metrics. However, the ODE approach does not suffer the same numerical problems as the queueing approximation, is very efficient to solve and is shown to be very accurate when the number of clients is large. By using the asymptotic results, it is possible to compute the metrics of interest very efficiently. These techniques have been applied to a utility function to explore the capacity planning for a system associated with this protocol. Three proposed questions have been answered through numerical results which have been acquired by a very efficient approach of combining techniques.

We can improve our proposed work flow by this case study. In the modelling stage, one can model the clients as the same replicated component (first and third modelling choice in this Chapter) or assign different name to it (second modelling choice in this Chapter). If the performance model is simple and small scaled, one can directly analyse it, otherwise, the model should be simplified. Partial evaluation is able to be employed to reduced the components of clients, if the clients in security protocols are tightly coupled. In the analysis stage, CTMC method can be applied if the simplified model is small enough. If the PEPA model can be transformed to a simple closed queueing network model, one only need to calculated the balance equation to solve the model. Furthermore, one can resort to ODE analysis to very large scale models. This is an approximation which only gives accurate results when system is very large. Therefore, it is a good complement to other exact solutions.

Chapter 4

Non-repudiation Protocols ZG1&ZG3

4.1 Introduction

In this chapter two non-repudiation protocols are studied that are used to extend the kind of modelling and analysis introduced in Chapter 3. The behaviour of the first protocol is similar to the KDC model in Chapter 3. The second protocol behaves more complicatedly as the non-repudiation server is able to serve multiple classes of job. For the analysis, the ODE solution has been employed for reasons of efficiency. The second protocol highlights a PEPA modelling problem, which is the server capturing an unintended race. This problem has been solved by using *functional rates*. In addition, *mean value analysis* (MVA) techniques has been applied. MVA is not as computationally efficient as the ODE solution, however, it is also not complicated and might therefore be used as an alternative solution in situations when the ODE solution is not accurate, i.e. around N^*. Finally, a utility function of second protocol is introduced to make a cost analysis. The novel contributions of this Chapter includes applying MVA to a class of PEPA model, the functional rates specification to solve an unintended resource competition, and detailed comparison between ODE approximation and MVA.

The remainder of this chapter is organised as follows. In the next section we introduce the two non-repudiation protocols to be modelled. Then, the PEPA models of two non-repudiation protocols are introduced, and this is followed by numerical results. Section 4.5 describes some of the problems with the current mean value analysis approach in PEPA and presents some extended numerical results. After that, *functional rates* are utilized to solve the problem of an unintended race and illustrated by some numerical results in Section 4.6. In Section 4.7, a cost function of ZG3 and its numerical results are presented. There is a summary in final section.

4.2 Non-repudiation protocols specification

A non-repudiation service will prevent either of the principals involved from denying the contract after the agreement. The two protocols depicted here were first proposed by Zhou and Gollmann [71, 72] and use a non-repudiation server, known as a *Trusted Third Party* (TTP). We denote these two protocols by ZG1 and ZG3, respectively.

4.2.1 ZG1 specification

- A: originator of the non-repudiation exchange
- B: recipient of the non-repudiation exchange
- TTP: on-line trusted third party provide network services accessible to the public
- M: message sent from A to B
- C: ciphertext for message M
- K: message key defined by A
- $NRO = sS_A(f_{NRO}, B, L, C)$: Non-repudiation of origin for M

- $NRR = sS_B(f_{NRR}, A, L, C)$: Non-repudiation of receipt for M
- $sub\_K = sS_A(f_{SUB}, B, L, K)$: proof of submission of K
- $con\_K = sS_T(f_{CON}, A, B, L, K)$: confirmation of K issued by TTP

First, A sends the ciphertext (C) and a non-repudiation origin (NRO) for message M to B, and then B replies back with a non-repudiation receipt (NRR) to A. Now B possesses the ciphertext, but cannot read it as he still hasn't got the key to decrypt C. According to the non-repudiation requirement, B is not a trusted agency to A for sending the key directly to B, they only can resort to a trusted third party (TTP). After receiving the key and proof of submission (sub_K), the TTP will generate a confirmation of K (con_K) and publish in a read only public area. Finally, B can get the key from this public area to decrypt ciphertext (C) and A fetches the confirmation of submission as non-repudiation evidence.

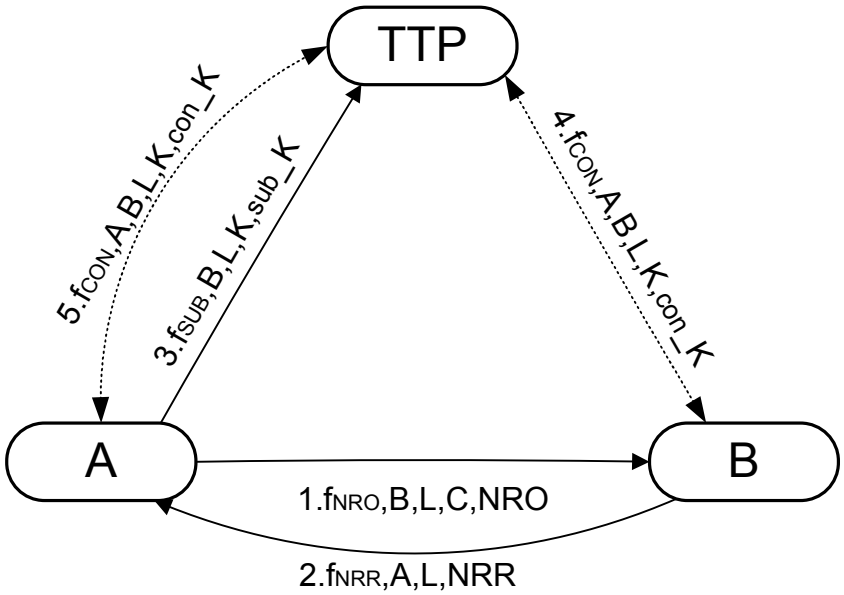

Figure 4.1: A non-repudiation protocol invented by Zhou and Gollmann (ZG1)

$$
\begin{aligned}
(sendB) \quad & 1.A \rightarrow B: \ f_{NRO}, B, L, C, NRO \\
(sendA) \quad & 2.B \rightarrow A: \ f_{NRR}, A, L, NRR \\
(sendTTP) \quad & 3.A \rightarrow TTP: \ f_{SUB}, B, L, K, sub\_K \\
(publish\& & \\
getByB) \quad & 4.B \leftrightarrow TTP: \ f_{CON}, A, B, L, K, con\_K \\
(publish\& & \\
getByA) \quad & 5.A \leftrightarrow TTP: \ f_{CON}, A, B, L, K, con\_K
\end{aligned}
$$

4.2.2 ZG3 specification

- L : a unique label chosen by TTP to identify the message M
- T_s : the time that TTP received A's submission
- T_d : the time that TTP delivered and available to B
- $NRO = sS_A(f_{NRO}, TTP, B, M)$: non-repudiation of origin for M
- $NRS = sS_D(f_{NRS}, A, B, T_s, L, NRO)$: non-repudiation of submission of M
- $NRR = sS_B(f_{NRR}, TTP, A, L, NRO)$: non-repudiation of receiving a message labelled L
- $NRD = sS_D(f_{NRD}, A, B, T_d, L, NRR)$: non-repudiation of delivery of M

ZG1 describes a non-repudiation protocol with minimized involvement of a trusted third party, acting as a "low weight notary". However, timing evidence of sending and receiving is required in some applications; hence ZG3 can be adopted in this situation. A sends the plaintext (M) and a non-repudiation origin (NRO) to the trusted third part (TTP), and then fetches the time of receiving (T_s) and non-repudiation of submission (NRS) from a public area, after TTP has published this information. The TTP tells B it received M from A by sending the NRO. B generates a non-repudiation of receiving for TTP following. Finally, B and A can fetch M and the time of delivery (T_d), with other non-repudiation evidence, from the public area, after the TTP has published.

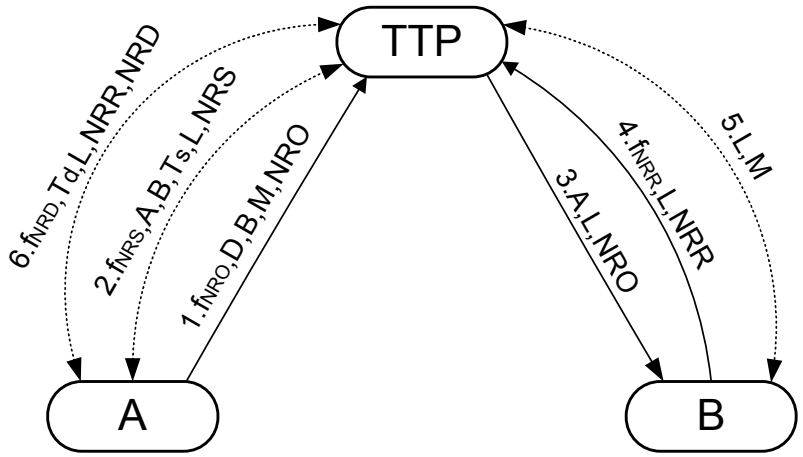

Figure 4.2: Another non-repudiation protocol invented by Zhou and Gollmann (ZG3)

$$
\begin{aligned}
(request)\quad & 1.A \rightarrow TTP:\ f_{NRO}, TTP, B, M, NRO \\
(response\& & \\
getByA1)\quad & 2.A \leftrightarrow TTP:\ f_{NRS}, A, B, T_s, L, NRS \\
(response)\quad & 3.TTP \rightarrow B:\ A, L, NRO \\
(sendTTP)\quad & 4.B \rightarrow TTP:\ f_{NRR}, L, NRR \\
(response\& & \\
getByB)\quad & 5.B \leftrightarrow TTP:\ L, M \\
(response\& & \\
getByA2)\quad & 6.A \leftrightarrow TTP:\ f_{NRD}, T_d, L, NRR, NRD
\end{aligned}
$$

As ZG1 is very similar to KDC, we now only propose three similar performance questions for ZG3: "how many clients can a given TTP configuration support?", "how much service capacity must we provide at a TTP to satisfy a given number of clients?" and "what is the maximum rate at which keys can be refreshed before the TTP performance begins to degrade?" These questions are answered through numerical results in section 4.7.

4.3 PEPA models of non-repudiation

Following the approach established in the analysis of KDC protocols in Chapter 3, we form a translation from the protocol specification into a PEPA model and extend this to the multiple client case.

4.3.1 ZG1 PEPA Model

We begin by forming components of a pair of principals A and B.

$$TTP \stackrel{def}{=} (publish, r_p).TTP$$

$$\begin{aligned}
A0 &\stackrel{def}{=} (sendB, r_b).A1 \\
A1 &\stackrel{def}{=} (sendA, r_a).A2 \\
A2 &\stackrel{def}{=} (sendTTP, r_t).A3 \\
A3 &\stackrel{def}{=} (publish, r_p).A4 \\
A4 &\stackrel{def}{=} (geyByA, r_{ga}).A5 \\
A5 &\stackrel{def}{=} (work, r_w).A0
\end{aligned}$$

$$\begin{aligned}
B0 &\stackrel{def}{=} (sendB, r_b).B1 \\
B1 &\stackrel{def}{=} (sendA, r_a).B2 \\
B2 &\stackrel{def}{=} (publish, r_p).B3 \\
B3 &\stackrel{def}{=} (getByB, r_{gb}).B4 \\
B4 &\stackrel{def}{=} (work, r_w).B0
\end{aligned}$$

$$SystemZG1 \stackrel{def}{=} TTP[K] \underset{publish}{\bowtie} (A0 \underset{\mathcal{L}}{\bowtie} B0)[N]$$

Where, $\mathcal{L} = \{sendB, sendA, work\}$.

In order to simplify the model specification and analysis, we combine A and B

into a new component called AB, using a process referred to as *partial evaluation* [11]. This gives rise to the following description for the complete system when there are N pairs of principals.

$$\begin{aligned}
TTP &\stackrel{def}{=} (publish, r_p).TTP \\
AB_0 &\stackrel{def}{=} (sendB, r_b).AB_1 \\
AB_1 &\stackrel{def}{=} (sendA, r_a).AB_2 \\
AB_2 &\stackrel{def}{=} (sendTTP, r_t).AB_3 \\
AB_3 &\stackrel{def}{=} (publish, r_p).AB_4 \\
AB_4 &\stackrel{def}{=} (getByA, r_{ga}).AB_5 \\
&\quad\ +(getByB, r_{gb}).AB_6 \\
AB_5 &\stackrel{def}{=} (getByB, r_{gb}).AB_7 \\
AB_6 &\stackrel{def}{=} (getByA, r_{ga}).AB_7 \\
AB_7 &\stackrel{def}{=} (work, r_w).AB_0 \\
SystemZG1 &\stackrel{def}{=} TTP[K] \underset{publish}{\bowtie} AB_0[N]
\end{aligned}$$

AB_0 to AB_7 in the above ZG1 PEPA model denote the different behaviours of the AB component, and its evolution along the sequence of prescribed actions in the protocol. The choice from AB_4 to AB_5 and AB_6 means step 4 and step 5 in ZG1 can happen in any order. The *work* action is used to define that B can do something with the key and ciphertext after he has obtained these, before returning to the state AB_0 to make a new request again, which forms a working cycle to investigate the steady state.

4.3.2 ZG3 PEPA Model

Once again we begin by defining the behaviour of a pair of principals.

$$TTP \stackrel{def}{=} (response, r_p).TTP$$

$$
\begin{aligned}
A0 &\stackrel{def}{=} (request, r_{t1}).A1 \\
A1 &\stackrel{def}{=} (response, r_p).A2 \\
A2 &\stackrel{def}{=} (getByA1, r_{ga1}).A3 \\
A3 &\stackrel{def}{=} (response, r_p).A4 \\
A4 &\stackrel{def}{=} (sendTTP, r_{t2}).A5 \\
A5 &\stackrel{def}{=} (response, r_p).A6 \\
A6 &\stackrel{def}{=} (getByA2, r_{ga2}).A7 \\
A7 &\stackrel{def}{=} (work, r_w).A0
\end{aligned}
$$

$$
\begin{aligned}
B0 &\stackrel{def}{=} (response, r_p).B1 \\
B1 &\stackrel{def}{=} (getByA1, r_{ga1}).B2 \\
B2 &\stackrel{def}{=} (response, r_p).B3 \\
B3 &\stackrel{def}{=} (sendTTP, r_{t2}).B4 \\
B4 &\stackrel{def}{=} (response, r_p).B5 \\
B5 &\stackrel{def}{=} (getByB, r_{gb}).B6 \\
B6 &\stackrel{def}{=} (work, r_w).B0
\end{aligned}
$$

$$
SystemZG3 \stackrel{def}{=} TTP[K] \underset{response}{\bowtie} (A0 \underset{\mathcal{L}}{\bowtie} B0)[N]
$$

Where, $\mathcal{L} = \{getByA1, sendTTP, work\}$.

As before, these are combined to form the merged component AB in the description of the complete system.

$$
\begin{aligned}
TTP &\stackrel{def}{=} (response, r_p).TTP \\
AB_0 &\stackrel{def}{=} (request, r_{t1}).AB_1 \\
AB_1 &\stackrel{def}{=} (response, r_p).AB_2 \\
AB_2 &\stackrel{def}{=} (getByA1, r_{ga1}).AB_3 \\
AB_3 &\stackrel{def}{=} (response, r_p).AB_4
\end{aligned}
$$

$$
\begin{aligned}
AB_4 &\stackrel{def}{=} (sendTTP, r_{t2}).AB_5 \\
AB_5 &\stackrel{def}{=} (response, r_p).AB_6 \\
AB_6 &\stackrel{def}{=} (getByB, r_{gb}).AB_7 \\
&\quad +(getByA2, r_{ga2}).AB_8 \\
AB_7 &\stackrel{def}{=} (getByA2, r_{ga2}).AB_9 \\
AB_8 &\stackrel{def}{=} (getByB, r_{gb}).AB_9 \\
AB_9 &\stackrel{def}{=} (work, r_w).AB_0 \\
SystemZG3 &\stackrel{def}{=} TTP[K] \underset{response}{\bowtie} AB_0[N]
\end{aligned}
$$

The PEPA model of ZG3 has a similar structure to that for ZG1. The main difference is the TTP component in ZG3 should respond three times for different requests in one cycle, which increases the difficulty of modelling and analysis, as discussed in Section 4.5.

4.3.3 Mean value analysis

Mean value analysis, which has been described in Chapter 2, is a traditional efficient solution for stochastic models. MVA is efficient as it avoids generation of the entire Markov chain. We defined a class of PEPA models, which can be applied with MVA. ZG1 and ZG3, which are the first two PEPA models that have been applied with MVA, belong to this class of PEPA model. Because of the computational effort of the exact solution, an approximated version has been applied here. The approximation also shows acceptable results in a case study in Chapter 3, and it is good enough for most purposes.

Three key points should be addressed for the models studied here. First, in order to apply this characteristic model amenable to mean value analysis to ZG1 and ZG3, the branching actions have been modified to have the same name in each model (see the equations below for each model respectively). This modification does not affect the analysis and is merely a consequence of the way in which the

class has been specified.

$$AB_4 \stackrel{def}{=} (get, r_{ga}).AB_5 + (get, r_{gb}).AB_6 \ (ZG1)$$

$$AB_6 \stackrel{def}{=} (get, r_{gb}).AB_7 + (get, r_{ga2}).AB_8 \ (ZG3)$$

The second factor is the calculation of a quantity referred to as the visit count (V_i). This quantity specifies the number of times a derivative AB_i is encountered, relative to some reference derivative AB_I. For both models here, all V_i is 1, except the branching points. For ZG1:

$$\begin{aligned} V_5 &= \frac{r_{ga}}{r_{ga} + r_{gb}} \\ V_6 &= \frac{r_{gb}}{r_{ga} + r_{gb}} \end{aligned}$$

For ZG3:

$$\begin{aligned} V_7 &= \frac{r_{gb}}{r_{ga2} + r_{gb}} \\ V_8 &= \frac{r_{ga2}}{r_{ga2} + r_{gb}} \end{aligned}$$

The final observation is that the rates of the *response* actions in ZG3 are the same in AB_1, AB_3 and AB_5. This restriction is necessary in order to restrict the overall service rate at the TTP. Whilst this can affect the performance estimation of individual parts of the protocol, this restriction has negligible effect of on overall mean system performance.

4.3.4 ODE analysis

ODE analysis is the most efficient solution of the techniques that were introduced in the last chapter. Although, the approximation is inaccurate around a location (N^*), it gives a very good trend and is very accurate when population is large. The area around N^* also can be solved by other exact solutions (i.e. MVA) as a combined solution if necessary. In addition the location of N^* can be predicted by deriving the point at which the two sides of the minimum function coincide. A more detailed description can be found in [34] and also a brief description in Chapter 2.

In experiments we have performed with different models, we have observed that the ODEs give good predictions of the steady state behaviour only when there is at most one active minimum function [65]. This condition holds for the models considered here as there is only one type of Trusted Third Party.

The ODEs for ZG1 and ZG3 can be derived following the approach of Hillston [34].

ODEs of ZG1:

$$
\begin{aligned}
\frac{d}{dt}AB_0(t) &= r_w AB_7(t) - r_b AB_0(t) \\
\frac{d}{dt}AB_1(t) &= r_b AB_0(t) - r_a AB_1(t) \\
\frac{d}{dt}AB_2(t) &= r_a AB_1(t) - r_t AB_2(t) \\
\frac{d}{dt}AB_3(t) &= r_t AB_2(t) - r_p min(AB_3(t), TTP(t)) \\
\frac{d}{dt}AB_4(t) &= r_p min(AB_3(t), TTP(t)) \\
&\quad -r_{ga} AB_4(t) - r_{gb} AB_4(t) \\
\frac{d}{dt}AB_5(t) &= r_{ga} AB_4(t) - r_{gb} AB_5(t) \\
\frac{d}{dt}AB_6(t) &= r_{gb} AB_4(t) - r_{ga} AB_6(t) \\
\frac{d}{dt}AB_7(t) &= r_{gb} AB_5(t) + r_{ga} AB_6(t) - r_w AB_7(t)
\end{aligned}
$$

$$\frac{d}{dt}TTP(t) = 0$$

ODEs of ZG3:

$$\begin{aligned}
\frac{d}{dt}AB_0(t) &= r_w AB_9(t) - r_{t1} AB_0(t) \\
\frac{d}{dt}AB_1(t) &= r_{t1} AB_0(t) - [r_p \frac{AB_1(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_2(t) &= -r_{ga1} AB_2(t) + [r_p \frac{AB_1(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_3(t) &= r_{ga1} AB_2(t) - [r_p \frac{AB_3(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_4(t) &= -r_{t2} AB_4(t) + [r_p \frac{AB_3(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_5(t) &= r_{t2} AB_4(t) - [r_p \frac{AB_5(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_6(t) &= -r_{gb} AB_6(t) \\
&\quad -r_{ga2} AB_6(t) + [r_p \frac{AB_5(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_7(t) &= r_{gb} AB_6(t) - r_{ga2} AB_7(t) \\
\frac{d}{dt}AB_8(t) &= r_{ga2} AB_6(t) - r_{gb} AB_8(t) \\
\frac{d}{dt}AB_9(t) &= r_{ga2} AB_7(t) + r_{gb} AB_8(t) - r_w AB_9(t) \\
\frac{d}{dt}TTP(t) &= 0
\end{aligned}$$

Our analysis is interested primarily in the number of clients waiting for a *publish* (or *response* in ZG3) action from the TTP, as the clients can then fetch what they need from the public area or obtain a service results. This is represented

in the model by the number of AB_3 in ZG1, AB_1, AB_3 and AB_5 in ZG3. The average queuing length $L(N)$ is the number of requests awaiting a response from the TTP. It is the number of the AB_3 (in ZG1), or AB_1, AB_3 and AB_5 (in ZG3), derivatives when $t \longrightarrow \infty$ when there are N customers in the population.

The average response time is another interesting metric for us. Again, the same as KDC application to obtain this we apply the arrival theorem. It can be approximately calculated as follows.

$$\begin{aligned} W(N) &= \frac{1}{r_p}\ ,\ L(N-1)+1 \leq K \\ W(N) &= \frac{1}{r_p} + \frac{L(N-1)+1-K}{Kr_p} \\ &= \frac{L(N-1)+1}{Kr_p}\ ,\ L(N-1)+1 > K \end{aligned}$$

4.4 Numerical results

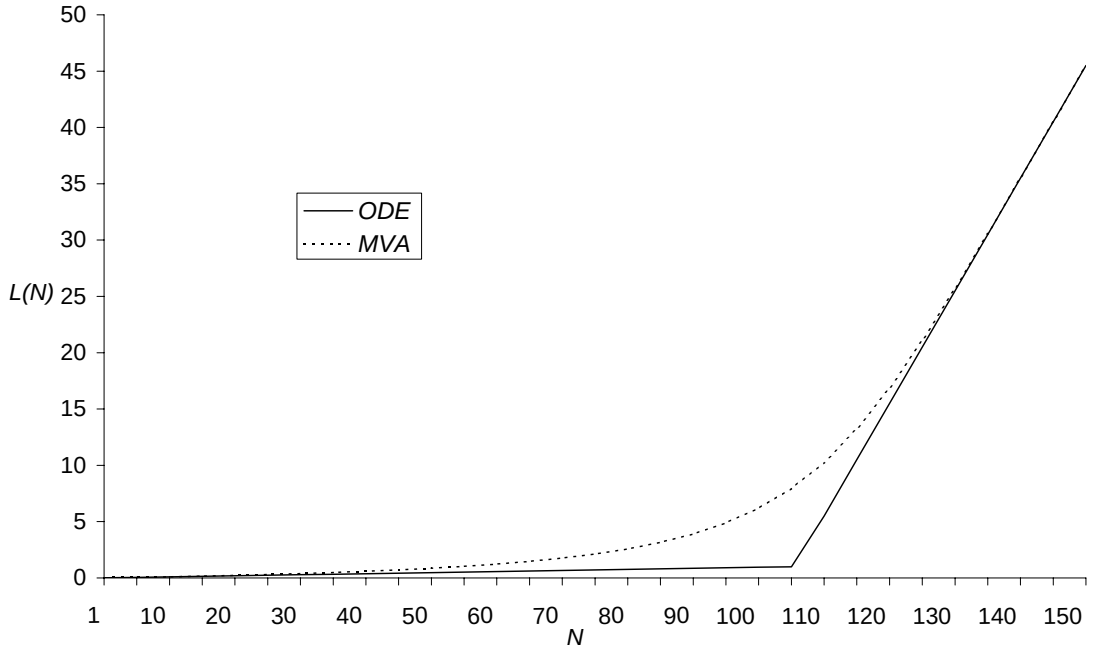

Figure 4.3: Average number of waiting jobs of ZG1 calculated by the ODE and MVA, $r_p = r_b = r_a = r_t = r_{ga} = r_{gb} = 1, r_w = 0.01$

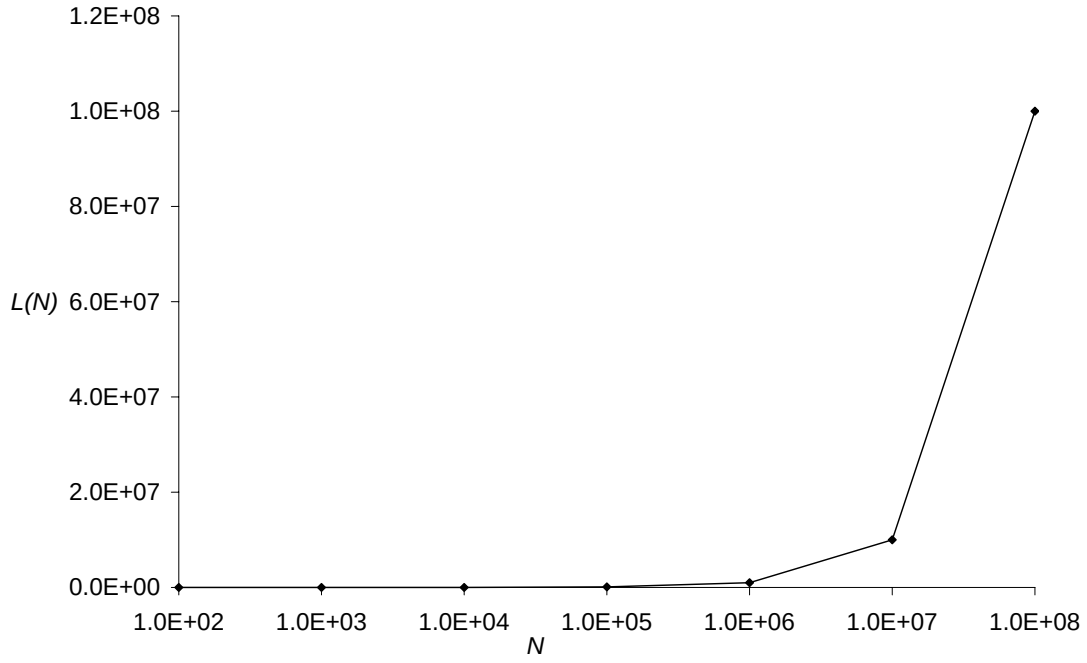

Figure 4.4: Average number of waiting jobs of ZG1 calculated by the ODE, $r_p = r_b = r_a = r_t = r_{ga} = r_{gb} = 1, r_w = 0.01$

Figure 4.3 shows the average number of waiting client requests calculated by MVA compared with ODE results. Obviously, the average number of waiting clients increases as the population size increases. The rate of increase is initially slow, but is far greater when N is large. As we would expect, the results from ODE analysis are acceptable when N is very small or very large. The maximum divergence point emerges when $N = 110$, the further we go beyond this point, the more accurate the results become. Hence, if we set this case in a real situation, ODE analysis becomes the best choice as its efficiency and accuracy when N larger than about 150, as MVA tends to compute more slowly for larger N. The results of ODE analysis for larger numbers of clients are presented in Figure 4.4. The ODEs are solved in Matlab within less than one minute for largest case ($N = 10^8$) in our graph, rather than MVA which takes more than 20 minutes when N is 10^5.

Figure 4.5 shows the average number of waiting jobs at the TTP, for different values of service (*publish*) rate, r_p, varied with population size, N. The comparison is between a single server and multiple servers with the same total service

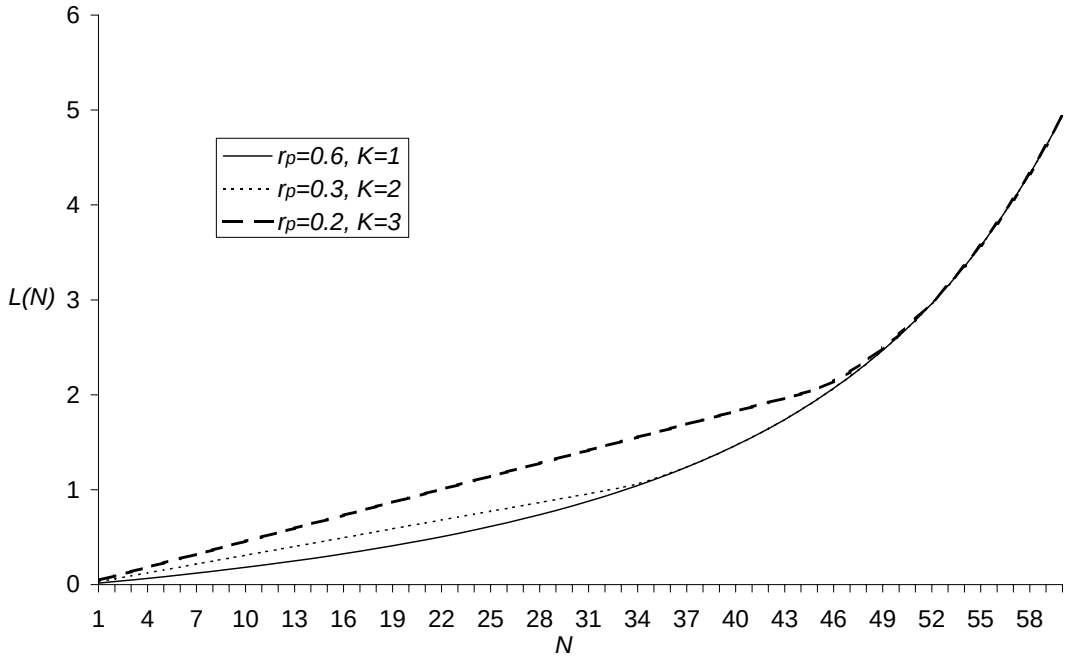

Figure 4.5: Average number of waiting jobs of ZG1 varied with population size calculated by the MVA, $r_b = r_a = r_t = r_{ga} = r_{gb} = 1, r_w = 0.01$

capacity. In initial stages (small N and small average queue length), the case of $r_p = 0.2$ and $K = 3$ shows the worst performance(longer queueing size), as not all serves have been utilised and the working servers have lower capability. Therefore, we can see a linear growth of queueing length when $K = 3$ and $N < 50$. However, once the population grows sufficiently for all servers to be highly utilised, then the three cases will show identical performance. Obviously, when demand is low, it is better to have one fast server than several slow ones, but as demand increases, only the overall service capacity matters.

This situation is clearer when we plot the average response time varied with number of clients in Figure 4.6. The average response time does not change when there are a small number of clients involved in the system, but the performance converges once there are sufficiently large populations.

Under the assumption of the same rate of *response* for derivatives of AB_1, AB_3 and AB_5, ZG3 performs very similarly to ZG1, as illustrated in Figure 4.7, Figure 4.8 and Figure 4.9 with the same parameters as the ZG1 model. According to the assumption, AB_1, AB_3 and AB_5 always have the same value. Therefore, we

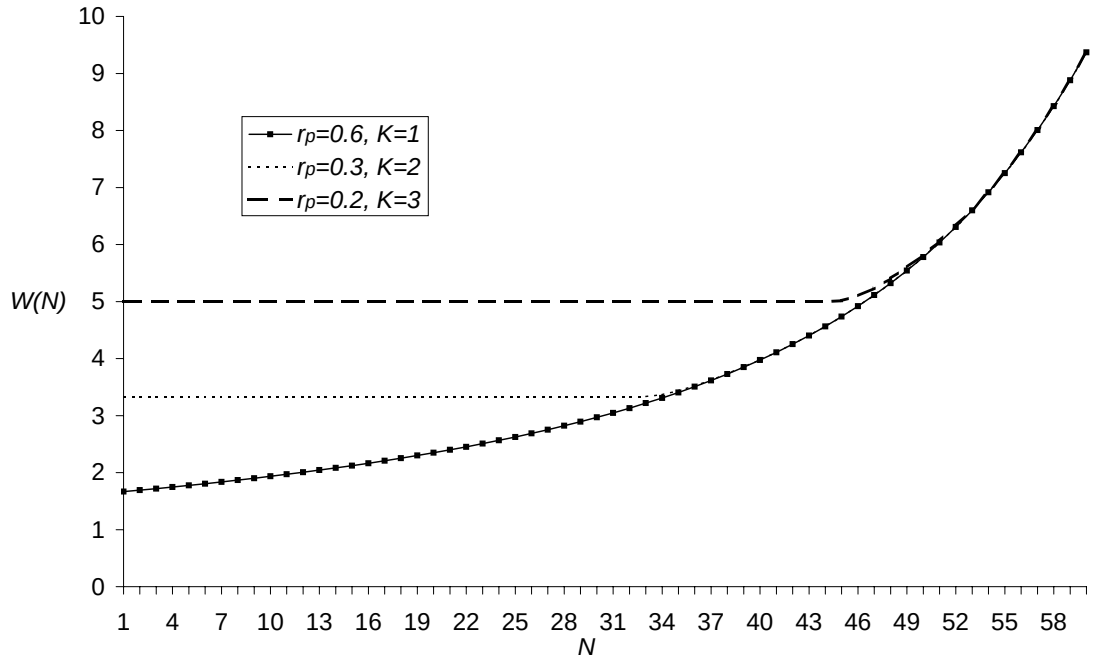

Figure 4.6: Average response time of ZG1 varied with population size calculated by the MVA, $r_b = r_a = r_t = r_{ga} = r_{gb} = 1, r_w = 0.01$

choose one of them to be presented in these three figures.

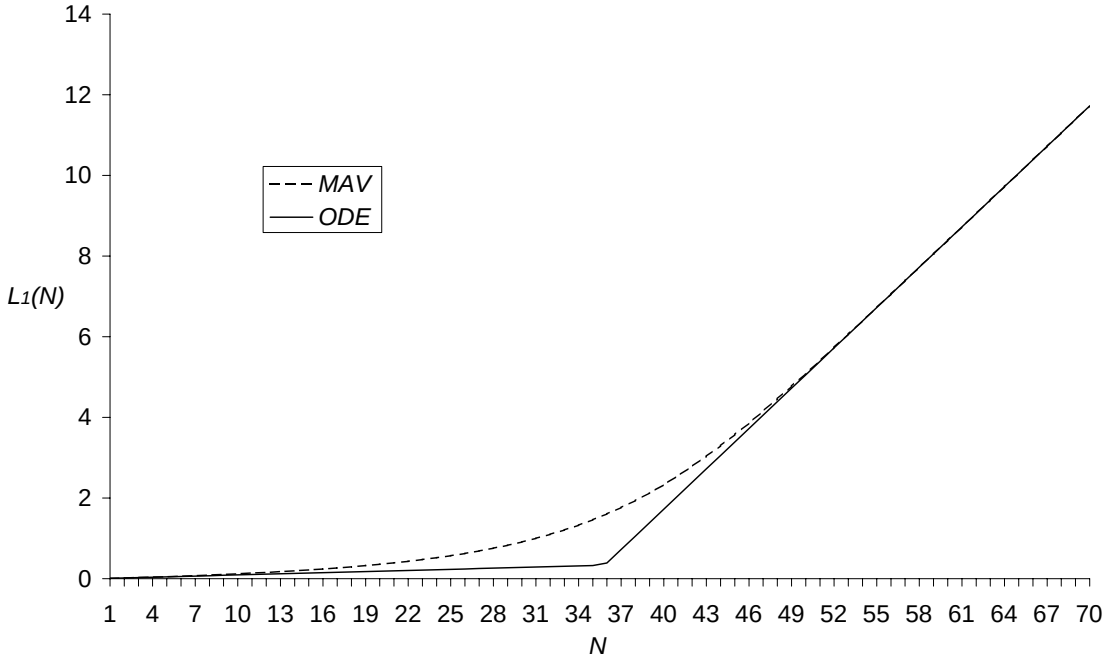

Figure 4.7: Average number of AB1 component of ZG3 calculated by the ODE and MVA, $r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

Figure 4.10 compares the performance between ZG1 and ZG3. Obviously, as the

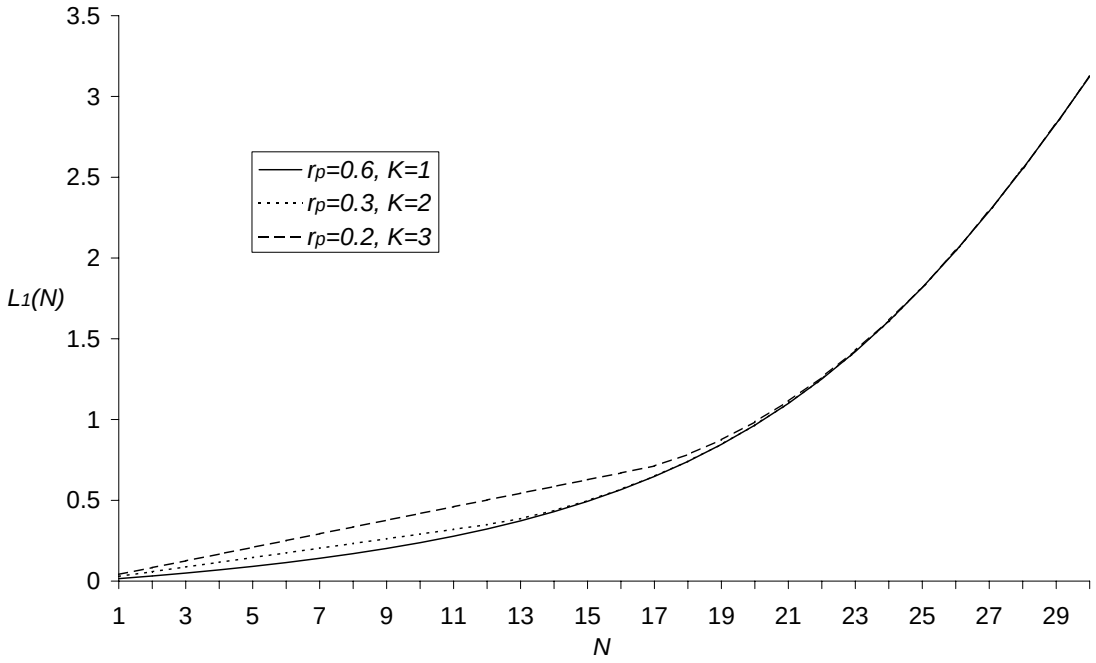

Figure 4.8: Average number of AB1 component of ZG3 varied with population size calculated by the MVA,$r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

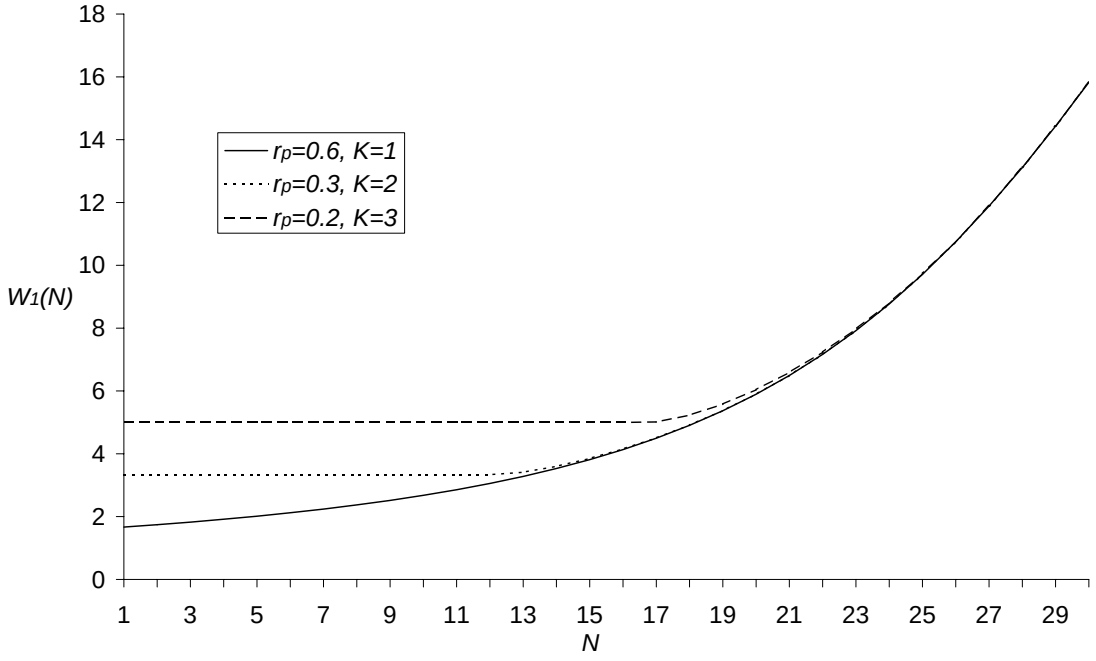

Figure 4.9: Average response time for AB1 component of ZG3 varied with population size calculated by the MVA, $r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

TTP in ZG1 is designed as a "low weight notary", the number of waiting requests at the TTP of ZG1 is always smaller than that in ZG3 with the same parameters.

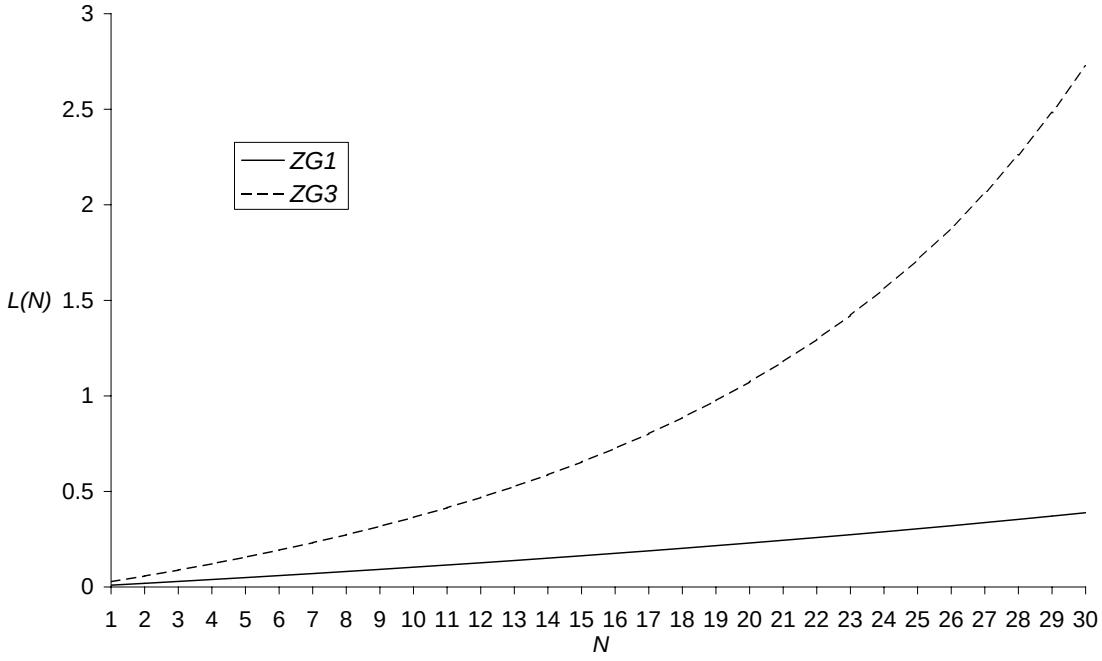

Figure 4.10: Average number of waiting jobs with ZG1 and ZG3 calculated by the MVA, $r_p = r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

However, a system engineer should clearly be very careful to choose either of these two protocols based on the trade off between performance and the need for added security functionality.

4.5 Limitations and Extended results

We have specified the ZG3 model using the same service ($response$) rate, as there are two main difficulties if publishing with different rates for derivative AB_1, AB_3 and AB_5. Firstly, if the TTP serves the three jobs with different rates, then the PEPA model of ZG3 is intuitively written like

$$\begin{aligned} TTP \overset{def}{=} & (response1, r_{p1}).TTP + (response2, r_{p2}).TTP \\ & +(response3, r_{p3}).TTP \end{aligned}$$

However, this expression gives rise to a competition between $response1$, $response2$

and *response*3 in PEPA, which does not capture the intended behaviour of the actual system. This problem can be solved by using *functional rates* (see details in next section).

Another problem is the average response time calculation using MVA with different service rates at the TTP and multiple servers. In this case, in order to obtain the response time, the time it takes for one TTP server to become available should be calculated first. However, in FCFS queueing this requires us to know the queued order of the requests, which is clearly infeasible. We can only obtain the response time for three responding rates when there is a single TTP server. Thus, the waiting time for an arriving request is the time for a single TTP server to respond to all the requests in the queue, which does not require any knowledge about the order in which requests are queued.

This situation has been illustrated in Figure 4.11, which we include here as an indication of the kind of scenario we could investigate. Here $W(1)$, $W(3)$ and $W(5)$ denotes the response times for the three responding actions by the TTP in the ZG3 protocol, with the rates r_{p1},r_{p2} and r_{p3} respectively. These are equivalent to the derivatives AB_1, AB_3 and AB_5 in the PEPA model. Clearly, the average response time for the second job type is slightly larger than first one and smaller than third job, because of the reciprocal ratio between response time and responding rate. However, average response time of all three job types grow at the same rate. The reason is obviously that the time for processing all the requests already within the queue is the same, only the time to process the arriving request differs. Thus, the difference between the response times of these three response actions is a constant.

It is also interesting to note the differences that occur as we alter the rate of the second and third response action. This difference between the two sets of curves is quite significant, far more so than we might naively expect. The initial stage ($N = 1 \sim 5$) of the average response time of the second type of jobs ($W(3)$) becomes larger as response rate decreases. Nevertheless, all three job types tends to respond quicker than the first set as N increases, because the average service

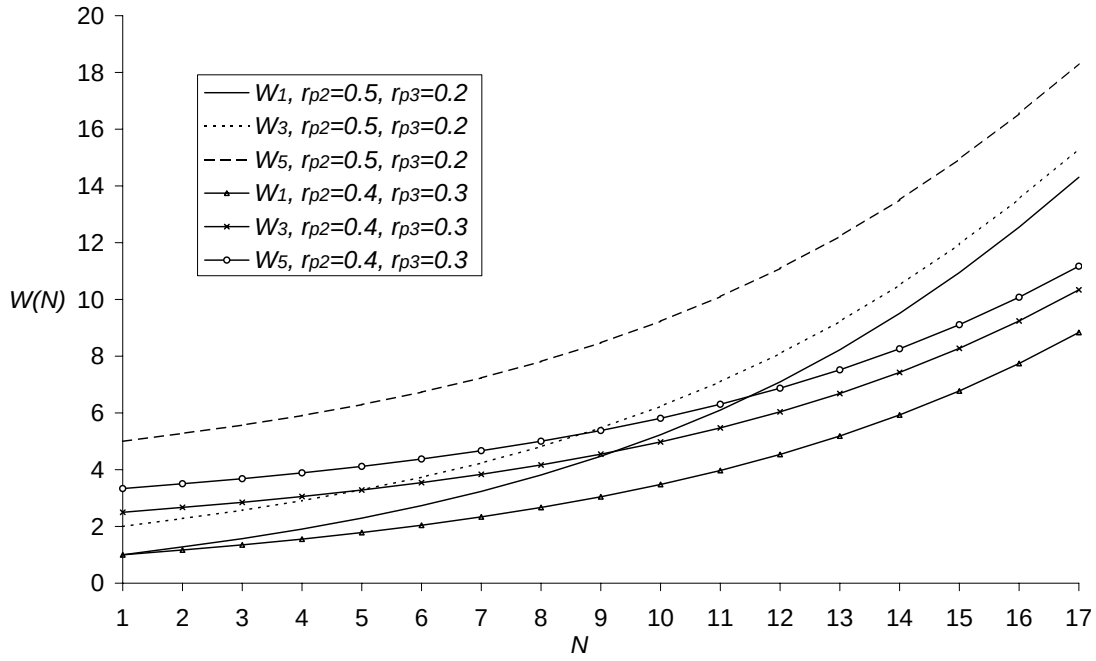

Figure 4.11: Average response time of ZG3 varied with population size with different service rates on second job type calculated by the MVA, $r_{p1} = r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

time $(1/r_{p1}+1/r_{p2}+1/r_{p3})$ is decreased, and the proportion of this type of request waiting at the TTP is smaller.

4.6 Functional Rates Specification

As we mentioned above, an intended race gives over capacity of the servers when different publish actions are modelled. Each type of services should only stick on this type of customer. Therefore, the actual service rate in our model becomes a function of number of each type of customers waiting for the service. The model with functional rates can be specified as follow:

$$AB_0 \stackrel{def}{=} (request, r_{t1}).AB_1$$
$$AB_1 \stackrel{def}{=} (publish1, r_{x1}).AB_2$$
$$AB_2 \stackrel{def}{=} (getByA1, r_{ga1}).AB_3$$

$$
\begin{aligned}
AB_3 &\stackrel{def}{=} (sendB, r_{x3}).AB_4 \\
AB_4 &\stackrel{def}{=} (sendTTP, r_{t2}).AB_5 \\
AB_5 &\stackrel{def}{=} (publish2, r_{x2}).AB_6 \\
AB_6 &\stackrel{def}{=} (getByB, r_{gb}).AB_7 \\
&\quad +(getByA2, r_{ga2}).AB_8 \\
AB_7 &\stackrel{def}{=} (getByA2, r_{ga2}).AB_9 \\
AB_8 &\stackrel{def}{=} (getByB, r_{gb}).AB_9 \\
AB_9 &\stackrel{def}{=} (work, r_w).AB_0 \\
TTP &\stackrel{def}{=} (publish1, r_{x1}).TTP \\
&\quad +(publish2, r_{x2}).TTP \\
&\quad +(sendB, r_{x3}).TTP \\
System &\stackrel{def}{=} TTP[K] \underset{publish1, publish2, sendB}{\bowtie} AB_0[N]
\end{aligned}
$$

Where,

$r_{x1} = r_{p1}\frac{AB_1(t)}{AB_1(t)+AB_3(t)+AB_5(t)} min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))$,

$r_{x2} = r_{p2}\frac{AB_5(t)}{AB_1(t)+AB_3(t)+AB_5(t)} min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))$.

$r_{x3} = r_b\frac{AB_3(t)}{AB_1(t)+AB_3(t)+AB_5(t)} min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))$.

$publish1, publish2, sendB$ are the different action names corresponding to *response* in ZG3 model in section 4.2.

In this ZG3 PEPA model, we specified the functional rates for each cooperation under action $publish1$, $publish2$ and $sendB$ by r_{x1}, r_{x2} and r_{x3}, respectively, instead of r_{p1}, r_{p2} and r_b. Each of these function describes a product of the actual service rate for one job in the system(r_{p1}, r_{p2} or r_b), the proportion of the number of waiting jobs of each type $(ABi * ((AB1 + AB3 + AB5)^{-1}), i = 1, 3, 5)$ and the times of service $(min(TTP, AB1+AB3+AB5))$, which stick each service only on its job type to eliminate the potential race.

There are several ways to solve this model. The most convenient and direct way is loading the PEPA in IPC (International PEPA Compiler) [73] tool and solve it. The PEPA eclipse plug-in [68, 74] is usable, but does not support PEPA models

with functional rates. However, it is possible to derive an equivalent model in the *CMDL (Chemical Model Definition Language)* format directly from above PEPA model using a version of the PEPA eclipse plug-in, which can contain rate functions and is able to be analyzed by the fluid flow approach (based on ODE) or stochastic simulation, supported by the tool. The following *CMDL* model is generated by eclipse plug-in and modified with *functional rates*.

$$
\begin{array}{ll}
//Rates & //Population\ sizes \\
r_b = 1.0; & AB_0 = N; \\
r_{ga1} = 1.0; & AB_1 = 0; \\
r_{ga2} = 1.0; & AB_2 = 0; \\
r_{gb} = 1.0; & AB_3 = 0; \\
r_{p1} = 1.0; & AB_4 = 0; \\
r_{p2} = 1.0; & AB_5 = 0; \\
r_{t1} = 1.0; & AB_6 = 0; \\
r_{t2} = 1.0; & AB_7 = 0; \\
r_w = 0.01; & AB_8 = 0; \\
 & AB_9 = 0; \\
 & TTP = K;
\end{array}
$$

$//Reactions$

$getByA1, AB_2 \rightarrow AB_3, r_{ga1};$

$getByA21, AB_6 \rightarrow AB_7, r_{ga2};$

$getByA22, AB_8 \rightarrow AB_9, r_{ga2};$

$getByB1, AB_6 \rightarrow AB_8, r_{gb};$

$getByB2, AB_7 \rightarrow AB_9, r_{gb};$

$publish1, TTP + AB_1 \rightarrow TTP + AB_2, r_{x1};$

$publish2, TTP + AB_5 \rightarrow TTP + AB_6, r_{x2};$

$request, AB_0 \rightarrow AB_1, r_{t1};$

$sendB, TTP + AB_3 \rightarrow TTP + AB_4, r_{x3};$

$sendTTP, AB_4 \rightarrow AB_5, r_{t2};$

$work, AB_9 \rightarrow AB_0, r_w;$

Where,

$r_{x1} = [rp1 * AB1 * ((AB1 + AB3 + AB5)^{-1}) * min(TTP, AB1 + AB3 + AB5)]$

$r_{x2} = [rp2 * AB5 * ((AB1 + AB3 + AB5)^{-1}) * min(TTP, AB1 + AB3 + AB5)]$

$r_{x3} = [rb * AB3 * ((AB1 + AB3 + AB5)^{-1}) * min(TTP, AB1 + AB3 + AB5)]$

This *CMDL* format model is formed by *Rates*, *Population sizes* and *Reactions*. The *Rates* section is exactly the same as specified in the PEPA model. *Population sizes* contains the initial population of all derivatives and components. In our scenario, there are N pair clients which have not started any behaviours at the initial stage, that are represented by $AB_0 = N$ and other derivatives have no population. K is the population of the TTP all the time as there are no derivatives associated with it. The most important and main section of *CMDL* definition is *Reactions*, in which system behaviours defined as all actions name, individual state transitions and their rates.

The final solution is derived manually by generating a set of ODEs which represent the PEPA Model, then solving these ODEs by any mathematical tools, *e.g.* MatLab. This kind of method is not convenient in terms of personal effort(writing ODEs and scripting code), however, it is more flexible. The set of ODEs with functional rates can be derived as follows:

$$
\begin{aligned}
\frac{d}{dt}AB_0(t) &= r_w AB_9(t) - r_{t1} AB_0(t) \\
\frac{d}{dt}AB_1(t) &= r_{t1} AB_0(t) - [r_{p1} \frac{AB_1(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_2(t) &= -r_{ga1} AB_2(t) + [r_{p1} \frac{AB_1(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_3(t) &= r_{ga1} AB_2(t) - [r_b \frac{AB_3(t)}{AB_1(t) + AB_3 t + AB_5(t)} \\
&\quad \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_4(t) &= -r_{t2} AB_4(t) + [r_b \frac{AB_3(t)}{AB_1(t) + AB_3 t + AB_5(t)}
\end{aligned}
$$

$$
\begin{aligned}
& \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_5(t) &= r_{t2}AB_4(t) - [r_{p2}\frac{AB_5(t)}{AB_1(t) + AB_3t + AB_5(t)} \\
& \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_6(t) &= -r_{gb}AB_6(t) \\
& -r_{ga2}AB_6(t) + [r_{p2}\frac{AB_5(t)}{AB_1(t) + AB_3t + AB_5(t)} \\
& \times min(AB_1(t) + AB_3(t) + AB_5(t), TTP(t))] \\
\frac{d}{dt}AB_7(t) &= r_{gb}AB_6(t) - r_{ga2}AB_7(t) \\
\frac{d}{dt}AB_8(t) &= r_{ga2}AB_6(t) - r_{gb}AB_8(t) \\
\frac{d}{dt}AB_9(t) &= r_{ga2}AB_7(t) + r_{gb}AB_8(t) - r_wAB_9(t) \\
\frac{d}{dt}TTP(t) &= 0
\end{aligned}
$$

4.6.1 Numerial results

Previously in this chapter, an assumption of the same action name and the same rates has been made for $publish1$, $publish2$ and $sendB$. With functional rates, now more general scenario can be investigated. e.g. any differences between these three TTP services.

Figure 4.12 shows the average queue length varied with the number of customers involved in this non-repudiation system, solved by the ODE solution supported by the tool. The divergence point (N^*) is at $N = 14$ and 16 respectively for the two curves. According to the results derived in the previous chapter, it is known that the further one goes from this area, the more accurate the ODE analysis becomes. However, this does not really affect our analysis, and the divergence point can be exactly predicted. The queuing length increases when more customers join the system for both cases. In the case where $r_{p1} = 0.5$ and $r_{p2} = 0.2$, the number of waiting jobs is always larger than when $r_{p1} = 0.4$ and $r_{p2} = 0.3$, due to the lower average service rate. In addition, the queue length of this set of parameters increases faster, because the slower server gets more load.

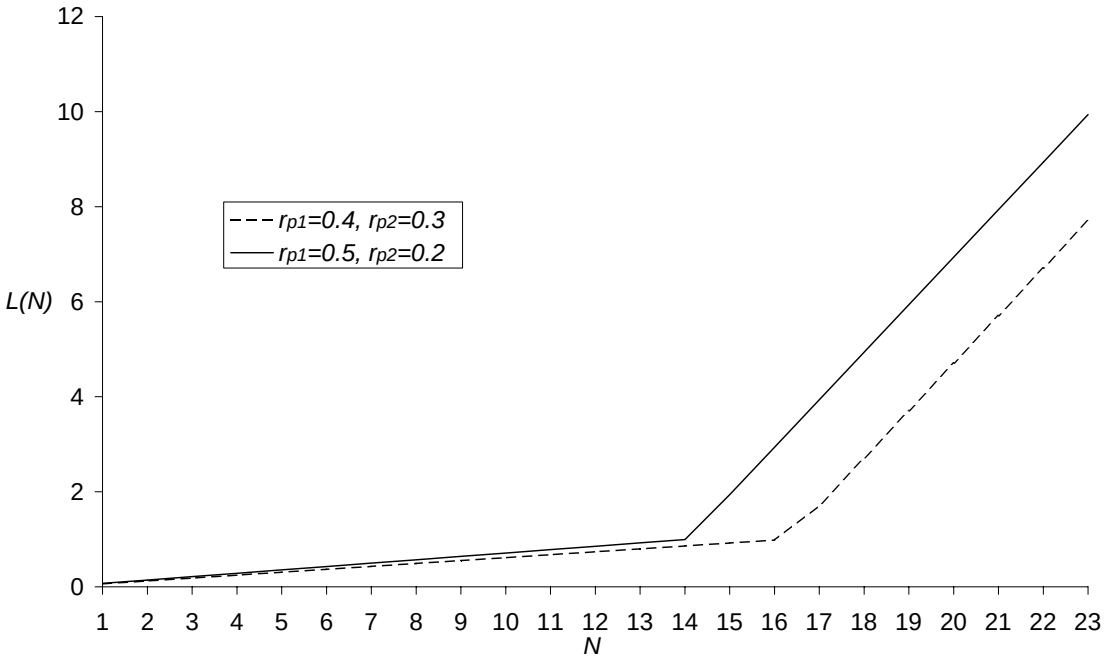

Figure 4.12: Average queue length varied with population size calculated by the ODE, $r_b = r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01, K = 1$

To calculate the average response time, the equations which were proposed in Section 4.4 can no longer be adopted. As the second limitation we described in Section 4.5, response time can not be calculated easily in the case of multiple servers and multi-type services in one station. However, if we assume there is just one sever in a queueing station, then the average response time is the average service time of all jobs ahead of a randomly observed job, plus the time to serve the randomly observed job [46]. In this case(ZG3), the equation can be written as follows:

$$W = \frac{L(1)}{r_{p1}} + \frac{L(3)}{r_b} + \frac{L(5)}{r_{p2}} + \frac{1}{r_i}\ ,\ r_i = r_{p1}, r_b, r_{p2} \tag{4.1}$$

Where, the service rate r_i depends on the job type of the random observer, and L(1),L(3) and L(5) are the number of different types of waiting jobs.

Figure 4.13 shows the average response time varied with system capacity by individual service behaviours. Here $W(1)$, $W(3)$ and $W(5)$ denote the response

times for the three responding actions by the TTP in the protocol, with the rates r_{p1},r_b and r_{p2} respectively. These are equivalent to the derivatives AB_1, AB_3 and AB_5 in the PEPA model. Clearly, the average response time for the first job type is slightly larger than third one (AB_5) and smaller than second job (AB_3), because of the reciprocal ratio between response time and responding rate. However, average response time of all three job types grow at the same rate. The reason is obviously that the average time for processing all the requests already within the queue is the same, only the time to process the arriving request differs. Thus, the difference between the response times of these three response actions is a constant.

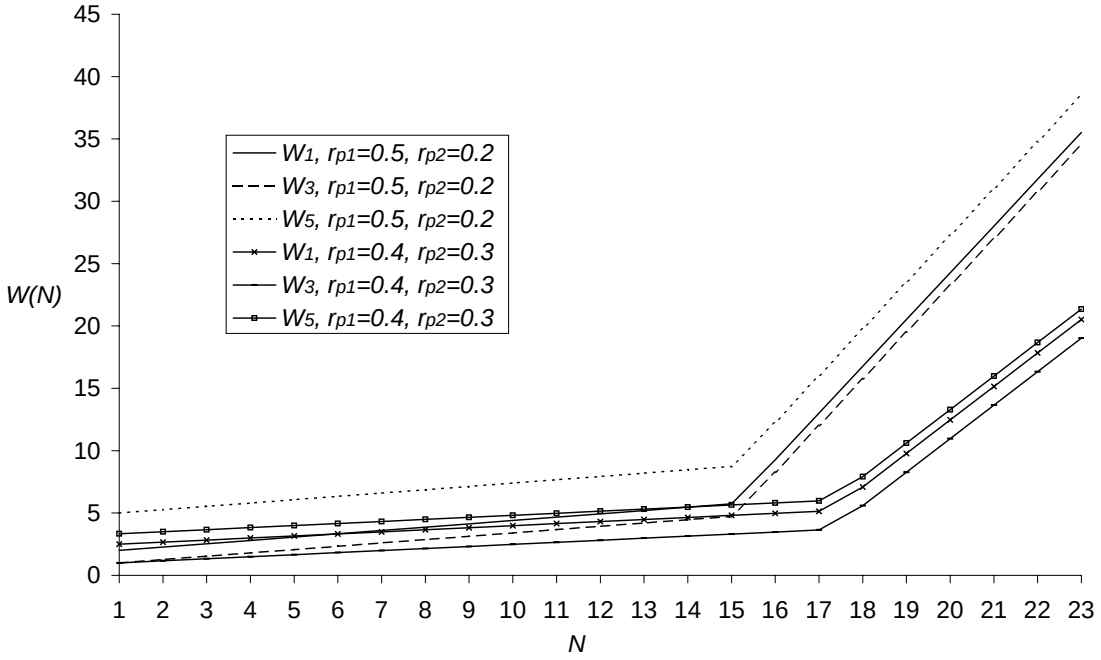

Figure 4.13: Average response time varied with population size calculated by the ODE, $r_b = r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01, K = 1$

When we alter the rate of the second and third response actions, the difference between the two sets of curves is quite significant, far more so than we might naively expect.The initial stage ($N = 1 \sim 6$) of the average response time of the first type of jobs ($W(1)$) becomes larger as response rate decreases. Nevertheless, all three job types tend to respond quicker than the first set as N increases, because the average service time ($1/r_{p1} + 1/r_b + 1/r_{p2}$) is decreased, and the proportion of this type of request waiting at the TTP is smaller.

Moreover, multiple servers can be analyzed, as illustrated in Figure 4.14. Here, $L(1)$, $L(3)$ and $L(5)$ denote the queuing lengths for the three responding actions by the TTP in the protocol, corresponding to AB_1, AB_3 and AB_5 in the PEPA model. In each set of curves, larger service rate results in a smaller number of waiting customers. Generally, there are fewer jobs waiting if more servers are provided. Nevertheless, the number of the first type waiting jobs (L1) with four TTP servers reaches the number of second type jobs with two TTP servers when $N = 145$, as they are fastest and slowest one in each set respectively.

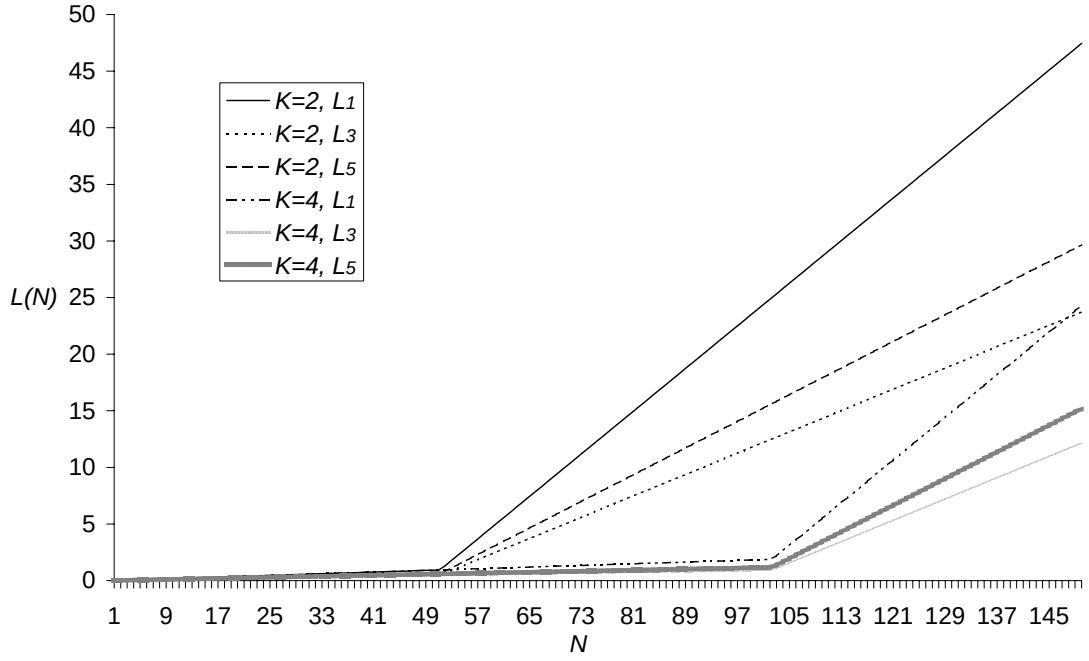

Figure 4.14: Average queue length varied with population size with different number of servers calculated by ODEs, $r_b = r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_{p1} = 0.5, rp2 = 0.8, r_w = 0.01$

4.7 Utility function

We now introduce the utility function to answer our proposed performance questions for ZG3.

$$C = c_1 L + c_2 K r_p \ \ , \ c_1, c_2 \geq 0 \tag{4.2}$$

This utility function keeps the same form as which we defined in Chapter 3 for KDC to make a consistent investigation. In this function, L denotes the average waiting customers at the non-repudiation server (TTP), and K is the number of servers. r_p is the response rate of the TTP server. We assume the TTP server responds any type of jobs in the same rate here. C_1 and C_2 are cost rates, and they many depend on the type of system or quality of service agreement with customers.

4.7.1 Numerical results

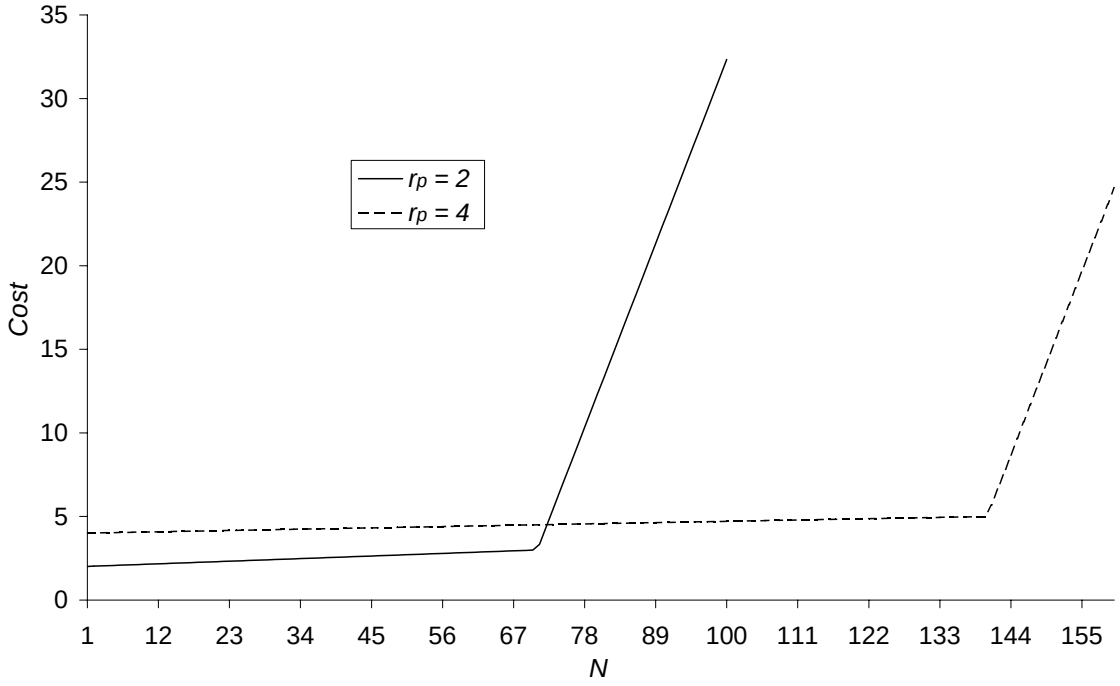

Figure 4.15: Cost varied against the number of clients calculated by ODEs, $K = 1$, $c1 = c2 = 1, r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = 1, r_w = 0.01$

Figure 4.15 presents the cost varied against the number of clients, calculated by ODEs. Generally, the more clients come to the system, the longer the average queue length is. As the service capacity does not change in this experiment, therefore, the cost increases along with the number of clients according to the utility function. Moreover, it is not difficult to find that the performance start to degrades significantly since $N = 70$ when $r_p = 2$ and $N = 140$ when $r_p = 4$

under these parameters.

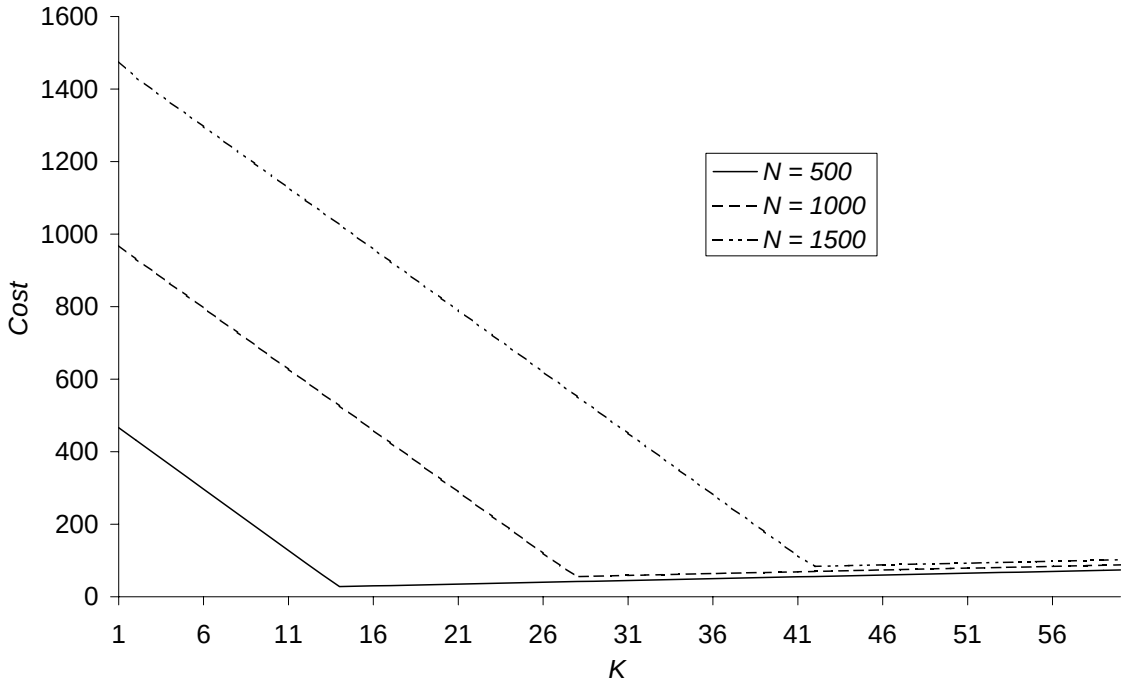

Figure 4.16: Cost varied with number of TTP servers calculated by ODEs, $c1 = c2 = 1, r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = r_p = 1, r_w = 0.01$

Figure 4.16 shows the cost varied with the number of TTP servers, calculated by ODEs. As we expected, the system needs more TTP server to support more clients to keep a reasonable cost. At the early stage of increasing number of servers, cost of service is very small. Thus, waiting customers dominate the total costs. With more servers are introduced to the system, cost of service is increasing. Meanwhile, average queue length is reducing under the same total number of client. Therefore, cost of service will dominate the total cost finally. However, there should be a optimal point between the two thresholds. The optimal points in our experiment are $K = 14$, $K = 28$, and $K = 42$ for $N = 500$, $N = 1000$, and $N = 1500$ respectively.

The relationship between cost and rate of key fresh is illustrated in Figure 4.17. The more frequently the users request a new session key, the more workload has been introduced in the system at TTP server. Consequently, cost of customers waiting is increased. However, the high frequency of refreshing key gives a hacker less time and cipher text to creak the session key. In practical situation, the

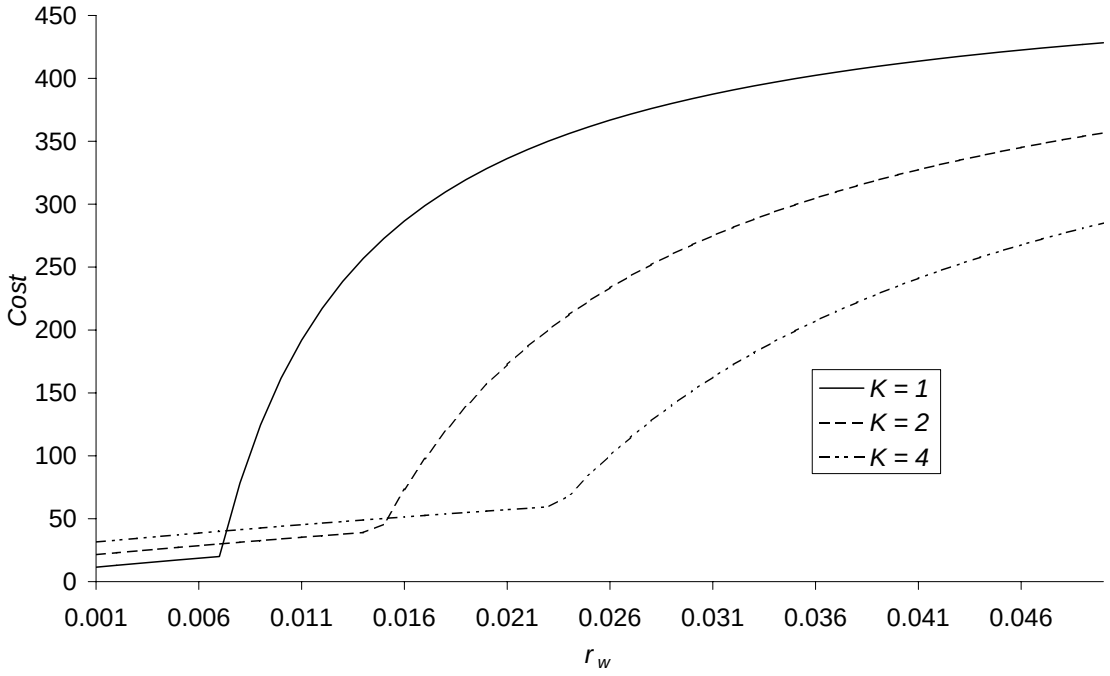

Figure 4.17: Cost varied with rate of $work$ calculated by ODEs, $c1 = c2 = 1, r_{t1} = r_{ga1} = r_b = r_{t2} = r_{gb} = r_{ga2} = r_p = 1, N = 500$

balance need to achieve by a engineer to realise both acceptable security and performance. In this case, we assume the balance point is the value of r_w where the cost start to increase rapidly. Hence, it is easy to find that the optimal point here is $r_w = 0.008$, $r_w = 0.014$ and $r_w = 0.024$ for one server, two servers and four servers respectively.

4.8 Summary

In this chapter we have applied both mean value analysis and fluid approximations to solve PEPA models of two non-repudiation protocols. Both approaches enable systems with extremely large numbers of components to be solved efficiently. When the population size, N, is very large, there is almost no difference between the values obtained by either method, although the fluid approximation is slightly quicker. However, when N is small the fluid approximation diverges in the region around a known point, hence in such cases, mean value analysis is preferable. We also analysed a utility function for ZG3 to further understand the system from a

practical aspect.

In addition, this study has highlighted some limitations with the initial approach used. The unintended competition behaviour has been solved by applying *functional rates* and deriving ODEs. For the average response time calculation, although we can solve this scenario when there is only one server (at the trusted third party), we have not been able to apply mean value analysis (even by using ODEs solution, we only can obtain average queue length) to the case where there are multiple servers.

We can add two aspects to our proposed work flow by through this case study. In simplification step, functional rates specification can be utilised to avoid to write full detailed behaviour of the server when there is an unintended competition between different service actions. MVA is able to utilised in the analysis stage as an exact solution, however, it is only can be applied in our defined class of PEPA model.

Chapter 5

An Optimistic Fair Exchange Protocol

5.1 Introduction

The case study in this chapter is another type of non-repudiation protocol, different from *ZG1* and *ZG3* introduced in the last chapter. This is an optimistic non-repudiation protocol, which only utilises the *third trust party* when accidents happen. This leads us to model the protocol in two ways: with misbehaviour and without. Moreover, we employ a different modelling form, in which a server has been considered as several threads, with each thread associated with a customer. Hence, the service rates of the server becomes a function of the number of threads. The analysis technique used in this case is ODEs as it is scalable and efficient. The contribution of this Chapter are employment of a multi-threads modelling concept and the comparison between the protocol with and without misbehaviour.

In the next section a specification of the basic version (no misbehaviour) of the e-commerce protocol is given. The subsequent section then introduces the PEPA model of this basic version of the protocol, followed by numerical results. After that, an extended version (with misbehaviour) of the e-commerce protocol is

described. Section 5.6 gives the PEPA model and a set of ODEs for this case, then some numerical results. We then conduct a utility function analysis in Section 5.8. Finally this chapter has been summarised in Section 5.9.

5.2 A e-commerce Protocol (basic)

This e-commerce protocol is an optimistic non-repudiation protocol, which adopts an offline TTP (*Third Tust Party*) not only to ensure fair exchange, but also to minimize the workloads from TTP server. Following the formal description in [51], the basic protocol (without misbehaviour of any principals) is illustrated below:

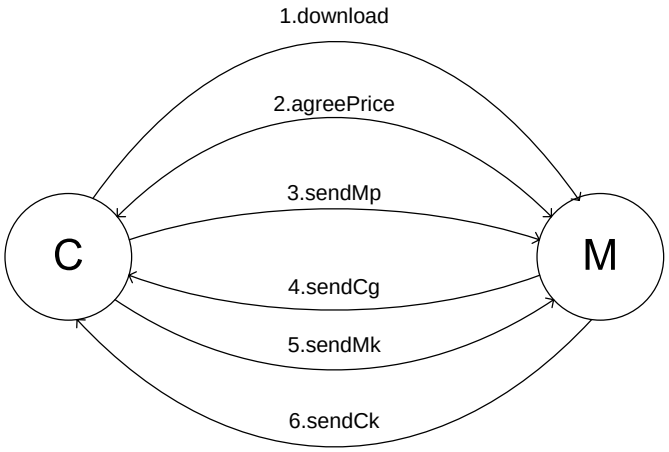

Figure 5.1: The basic protocol

There is a set environment before protocol operates, in which C (*Customer*) opens an account with B (*Bank*) and M (*Merchant*) registers with the TTP. The protocol is then covered in six steps:

1. **C selects a product to purchase.(*download*)** The customer chooses a product, and downloads it from the Internet merchant. However, this e-product has been encrypted, and so the customer cannot acquire the product without a decryption key. This product can be used for validation later.

2. **C and M agree upon a price for the product.(*agreePrice*)** Several messages may be exchanged between C and M in this step.

3. **C sends PO (*purchase order*) to M.(*sendMp*)** The customer sends three elements to the merchant:*(a)*the purchase order; *(b)*a digitally signed cryptographic checksum of the PO; and *(c)* the PT (*Payment Token*).

4. **M sends encrypted product to C or abort the transaction.(*sendCg or sendCabort*)** the merchant checks the purchase order which was received at the last step: if the merchant is not satisfied, then an abort message is sent to C; otherwise, the following is sent to C: *(a)* a sgined cryptographic checksum of the purchase order; *(b)* encrypted product; *(c)* signed cryptographic checksum of the encrypted product; *(d)* encrypted random number; and *(e)* signed cryptographic checksum of the encrypted random number.

5. **C sends payment token decryption key to M or abort the transaction.(*sendMk or sendMabort*)** C checks the message from M, if it is an abort, then abort the transaction. C validates the product, and sends M a signed abort message if the product has failed to be validated; otherwise, sends the payment token decryption key and a signed cryptographic checksum of the encrypted product decryption key.

6. **M sends product decryption key to C or terminates the transaction.(*sendCk*)** If M receives an abort message from C, it terminates the transaction. Otherwise, if the received PT decryption key works, M sends the following to C:*(a)* the product decryption key; *(b)* signed cryptographic checksum of the encryption product decryption key; *(c)* the multiplicative inverse of a random number; *(d)* signed cryptographic checksum of the encrypted multiplicative inverse of the random number.

To address the performance aspects of this protocol, this illustration focuses on the behaviour. Therefore the security contents have been eliminated from the description above. The original paper [51] gives a more detailed version. Moreover,

the protocol presented here is the basic version without TTP involved, as a "no dispute" assumption has been made. An extended model with misbehaviour of participants will be illustrated in Section 6.

5.3 PEPA Model of the basic protocol

In this work, we consider a scenario of a number of customers buying an e-product from a merchant under this fair exchange protocol. This merchant has been divided to number of threads (T) to serve each customer, and the number of threads has been allocated in dynamic way which means the threads number depend on the number of active customers. Therefore, all the rates of active actions of the merchant depend on the active customers numbers. In other words it depends on the state of the system, through a *functional rate*. The more threads that have been allocated, the slower the individual rate of each thread. Consequently, these functional rates and the number of threads are in inverse ratio. The model is formulated as:

$$
\begin{aligned}
C_0 &\stackrel{def}{=} (download, r_d).C_1 \\
C_1 &\stackrel{def}{=} (agreePrice, r_a).C_2 \\
C_2 &\stackrel{def}{=} (sendMp, r_{smp}).C_3 \\
C_3 &\stackrel{def}{=} (sendCg, f_1).C_4 + (sendCabort, f_2).C_7 \\
C_4 &\stackrel{def}{=} (sendMk, r_{smk}).C_5 + (sendMabort, r_{sma}).C_6 \\
C_5 &\stackrel{def}{=} (sendCk, f_3).C_6 \\
C_6 &\stackrel{def}{=} (work, r_w).C_0 \\
C_7 &\stackrel{def}{=} (sendMabort, r_{sma}).C_6
\end{aligned}
$$

$$
\begin{aligned}
T_0 &\stackrel{def}{=} (download, r_d).T_1 \\
T_1 &\stackrel{def}{=} (agreePrice, r_a).T_2
\end{aligned}
$$

$$
\begin{aligned}
T_2 &\stackrel{def}{=} (sendMp, r_{smp}).T_3 \\
T_3 &\stackrel{def}{=} (sendCg, f_1).T_4 + (sendCabort, f_2).T_7 \\
T_4 &\stackrel{def}{=} (sendMk, r_{smk}).CT_5 + (sendMabort, r_{sma}).T_6 \\
T_5 &\stackrel{def}{=} (sendCk, f_3).T_6 \\
T_6 &\stackrel{def}{=} (work, r_w).T_0 \\
T_7 &\stackrel{def}{=} (sendMabort, r_{sma}).T_6
\end{aligned}
$$

$$
System \stackrel{def}{=} C_0[N] \underset{\mathcal{L}}{\bowtie} T_0[N]
$$

Where, $\mathcal{L} = \{download, agreePrice, sendMp, sendCg, sendCabort, sendMk, sendMabort, sendCk, work\}$

$f_1 = r_{scg}$, $f_2 = r_{sca}$, $f_3 = r_{sck}$, (if $N = 1$);

$f_1 = \frac{r_{scg}}{C_3+1}$, $f_2 = \frac{r_{sca}}{C_3+1}$, $f_3 = \frac{r_{sck}}{C_5+1}$, (if $N > 1$).

Again, as each customer and merchant thread are tightly coupled, the *partial evaluation* [11] approach has been employed and the model as follows:

$$
\begin{aligned}
CT_0 &\stackrel{def}{=} (download, r_d).CT_1 \\
CT_1 &\stackrel{def}{=} (agreePrice, r_a).CT_2 \\
CT_2 &\stackrel{def}{=} (sendMp, r_{smp}).CT_3 \\
CT_3 &\stackrel{def}{=} (sendCg, f_1).CT_4 + (sendCabort, f_2).CT_7 \\
CT_4 &\stackrel{def}{=} (sendMk, r_{smk}).CT_5 + (sendMabort, r_{sma}).CT_6 \\
CT_5 &\stackrel{def}{=} (sendCk, f_3).CT_6 \\
CT_6 &\stackrel{def}{=} (work, r_w).CT_0 \\
CT_7 &\stackrel{def}{=} (sendMabort, r_{sma}).CT_6 \\
System &\stackrel{def}{=} \underbrace{CT_0||CT_0||\cdots||CT_0}_{N}
\end{aligned}
$$

Where, $f_1 = r_{scg}$, $f_2 = r_{sca}$, $f_3 = r_{sck}$, (if $N = 1$);

$f_1 = \frac{r_{scg}}{CT_3+1}$, $f_2 = \frac{r_{sca}}{CT_3+1}$, $f_3 = \frac{r_{sck}}{CT_5+1}$, (if $N > 1$).

CT_0 to CT_7 in the above model denote the different states and the evolution of the CT component. The *work* action is utilized to define that customers can use the product to do something before returning to state CT_0 to buy another product, which forms a working cycle to investigate the steady state. The system consists of a number of parallel independent CT components that are initialized with state CT_0. Each customer has been partially evaluated with a merchant thread, hence no cooperation between the new combined components CT. In addition, the functional rate we mention above has been roughly defined as "the rate of merchant active actions when only one customer involved in the system" divided by "the number of active customers associated with there relevant action plus one" if more than one customer in the system, following the rule that they are in inverse ratio. Those functional rates have been expressed as f_1, f_2 and f_3 in our model. This assumption is probably not acceptable from engineering aspect. However, it's not possible to get a accurate functional rate expression without practical experiment. Hence, the main purpose of this work is to show the feasibility of this kind of modelling and analysis approach rather than practical results. Additionally, the reason to add one in the denominator of the functional rate is to avoid the numerical fault in ODE analysis by using Matlab in next section. However, the addend could be ignored with increasing number of customers.

5.3.1 ODE analysis

Once again, because of the scalability and efficiency properties, ODEs have been employed here. The set of ODEs for this fair exchange protocol can be derived following the approach of Hillston [34].

$$
\begin{aligned}
\frac{d}{dt}CT_0(t) &= r_w CT_6(t) - r_d CT_0(t) \\
\frac{d}{dt}CT_1(t) &= r_d CT_0(t) - r_a CT_1(t) \\
\frac{d}{dt}CT_2(t) &= r_a CT_1(t) - r_{smp} CT_2(t) \\
\frac{d}{dt}CT_3(t) &= r_{smp} CT_2(t) - \frac{r_{scg}}{CT_3+1} CT_3(t) - \frac{r_{sca}}{CT_3+1} CT_3(t) \\
\frac{d}{dt}CT_4(t) &= \frac{r_{scg}}{CT_3+1} CT_3(t) - r_{smk} CT_4(t) - r_{sma} CT_4(t) \\
\frac{d}{dt}CT_5(t) &= r_{smk} CT_4(t) - \frac{r_{sck}}{CT_5+1} CT_5(t) \\
\frac{d}{dt}CT_6(t) &= \frac{r_{sck}}{CT_5+1} CT_5(t) + r_{sma} CT_7(t) + r_{sma} CT_4(t) - r_w CT_6(t) \\
\frac{d}{dt}CT_7(t) &= \frac{r_{sca}}{CT_3+1} CT_3(t) - r_{sma} CT_7(t)
\end{aligned}
$$

In our analysis we are interesting in the average number of waiting clients that are represented by the sum of number of CT_3 and CT_5 in steady state($t \rightarrow \infty$). Following this thread modelling concept, it a simple matter to calculate the average response time for each service action of a merchant thread as:

$$
\begin{aligned}
W_{sendCg} &= \frac{1}{f_1}(t \rightarrow \infty), \\
W_{sendCabort} &= \frac{1}{f_2}(t \rightarrow \infty), \\
W_{sendCk} &= \frac{1}{f_3}(t \rightarrow \infty).
\end{aligned}
$$

5.4 Numerical results of the basic protocol

Figure 5.2 shows the average number of waiting customers in *sendCg* and *sendCabort* against total number of customers involved in the system. As we would expect, the average number of waiting customers increases when the population

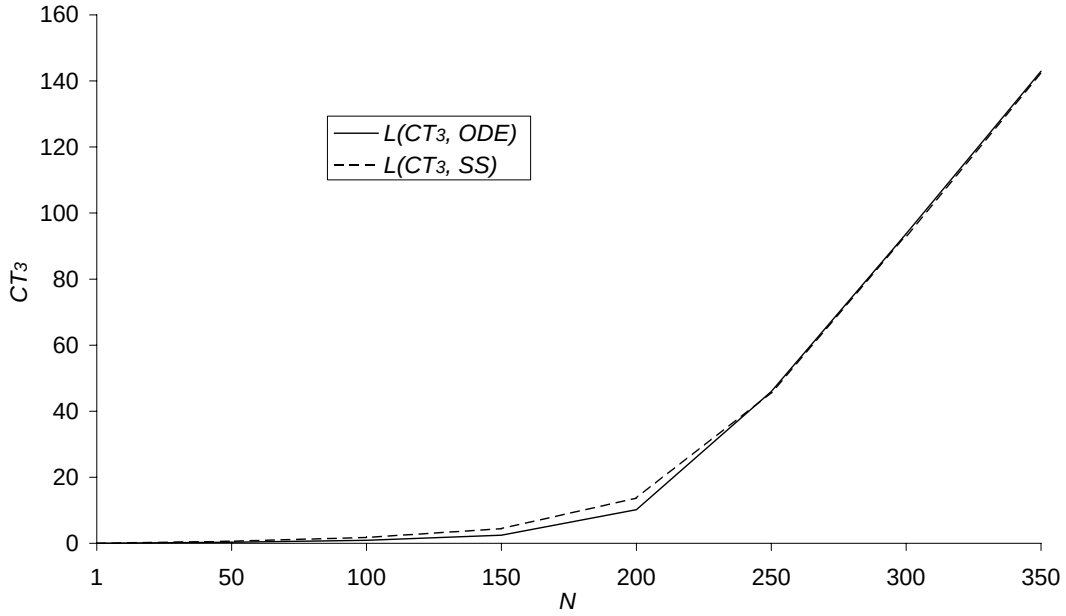

Figure 5.2: Average number of waiting customers in *sendCg* and *sendCabort* varied with population size calculated by ODEs and stochastic simulation,$r_d = r_a = r_b = r_{smp} = r_{smk} = r_{scg} = r_{sck} = r_{sca} = r_{sma} = 1, r_w = 0.01$

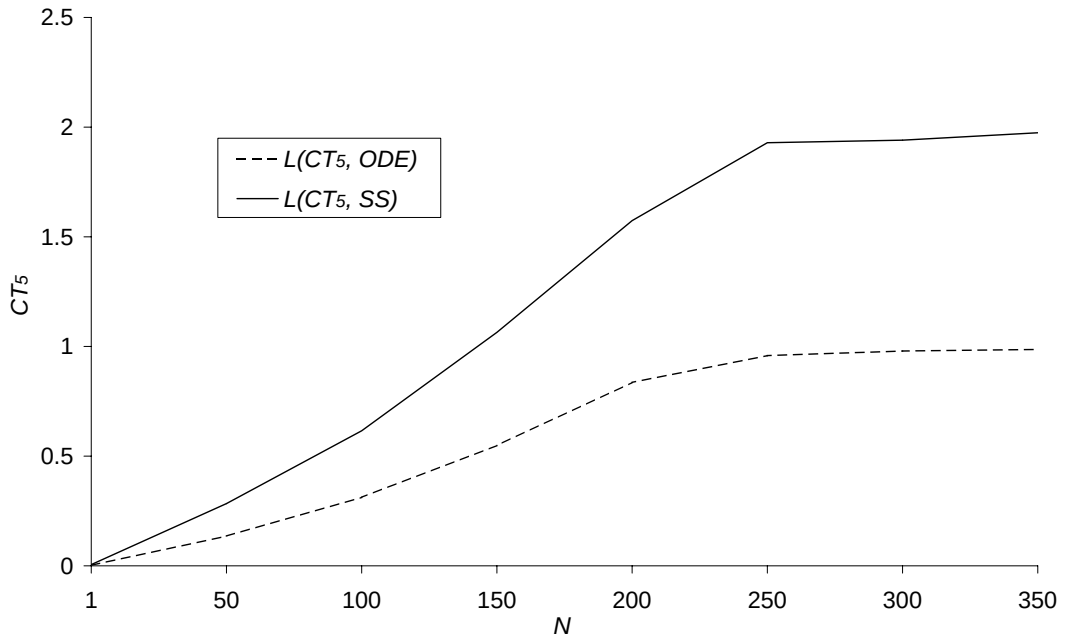

Figure 5.3: Average number of waiting customers in *sendCk* varied with population size calculated by ODEs and stochastic simulation,$r_d = r_a = r_b = r_{smp} = r_{smk} = r_{scg} = r_{sck} = r_{sca} = r_{sma} = 1, r_w = 0.01$

size increases. In the initial stage, the increase rate is slow, and it become grater when N is large. Obviously, this results follows the common consensus that ODE gives very accurate results when the system is large. In this situation, the large

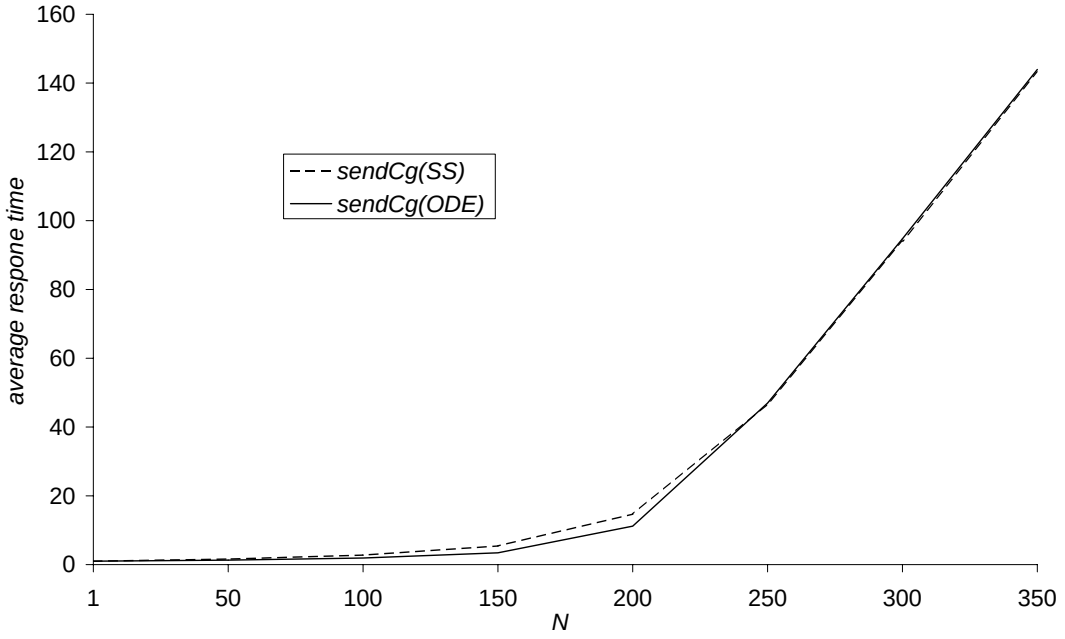

Figure 5.4: Average response time of action "*sendCg*(*sendCabort*)" varied with population size calculated by ODEs and stochastic simulation,$r_d = r_a = r_b = r_{smp} = r_{smk} = r_{scg} = r_{sck} = r_{sca} = r_{sma} = 1, r_w = 0.01$

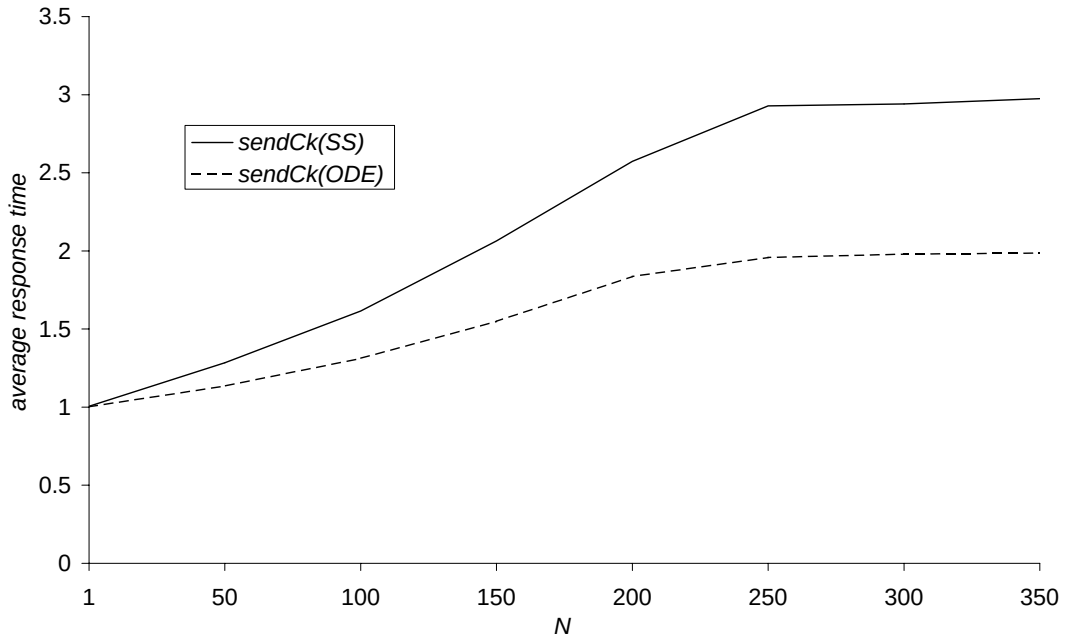

Figure 5.5: Average response time of action "*sendCk*" varied with population size calculated by ODEs and stochastic simulation,$r_d = r_a = r_b = r_{smp} = r_{smk} = r_{scg} = r_{sck} = r_{sca} = r_{sma} = 1, r_w = 0.01$

scale has been defined as $N > 250$, as ODE and simulation results are converge from this area. However, this is not reflected in another part of total queue length (average number of waiting customers in *sendCk*, CT_5), and which is show

in Figure 5.3. It not difficult to identify two issues from this graph: firstly, the number of CT_5 becomes almost stable and increases extremely slowly in both ODE and stochastic simulation analysis; then, ODE results and SS results do not coverage for CT_5 even when $N = 350$ and beyond, tough they converge for CT_3 if $N > 250$. So, we go back to the PEPA model, and address the rates that flow into CT_3 (r_{smp}), flow out from CT_3 (f_1 and f_2), flow into CT_5 (r_{smk}) and flow out from CT_5 (f_3). The rates r_{smp} and r_{smk} are fixed, hence, f_1, f_2 and f_3 are the keys to affect the results. As we know, f_1, f_2 and number of CT_3 are in inverse ratio, and f_3 and CT_5 are in inverse ratio, therefore, f_3 is much larger (up to 100 times) than f_1 and f_2 as more customers are involved in the system, according to the results of number of CT_3 and CT_5 in steady state from Figure 5.2 and Figure 5.3, especially when $N > 200$. Hence, more customers have been blocked in the CT_3 state as more customers arrive, and so the change of number of customers in state CT_5 becomes very small. This is also the reason for divergence between the ODE and SS results. From the number of CT_5 in Figure 5.3, obviously, this is still a very small scale for CT_5 even when $N = 350$, because the very slow rate f_1 causes a very few customers flowing into state CT_5 when N is large. Once the average number of waiting customers has been acquired, we can easily calculate average response time using the formulas given in last chapter, shown in Figure 5.4 and Figure 5.5. The profiles of the curves of average response time for action *sendCg*(*sendCabort*) and *sendCk* are quite similar to those of their queue lengths.

5.5 Extended protocol

The protocol illustrated in Section 5.2 is a basic version, that operates without misbehaviour of any participants. As this is an optimistic fair exchange protocol, it is necessary to investigate the performance of the TTP. Following [51], to get TTP to operate, several misbehaviours have been introduced as follows:

M behaves improperly:

- M receives the payment token decryption key in step 5, but does not send

the correct product decryption key in step 6.

1. C send all he got in the exchange to TTP. (*sendTPall*)
2. TTP asks M to send the correct decryption key and start a timer. (*notifyM1*)
3. M send the correct key to TTP or has no response. (*sendTPk1* or *tiemout1*)
4. if M sends the correct key, TTP forwards the key to C; if not, TTP sends a decryption key which preserved before this exchange to C and take appropriate action against M. (*sendCkbyTP1* or *sendCkbyTP2, takeactionM*)

- M receives the payment token decryption key in step 5, but disappears without sending the product decryption key.

  1. C's timer expires. (*noresponsedelay*)
  2. C send all he got in the exchange to TTP. (*sendTPall*)
  3. TTP asks M to send the correct decryption key and start a timer. (*notifyM2*)
  4. M has no response. (*timeout2*)
  5. TTP sends a decryption key which preserved before this exchange to C and take appropriate action against M. (*sendCkbyTP2, takeactionM*)

- M claims that it did not send the correct decryption key because it has not received payment.

  1. M sends the reason that he did not receive proper payment. (*sendTPreason*)
  2. M still need to send product decryption key to TTP. (*sendTPk2*)
  3. Once TTP got the product decryption key from M, he sends appropriate decryption key to M and C. (*sendMkbyTP1, sendCkbyTP3*)

C behaves improperly: M got the payment decryption key from TTP again after he claims the wrong key in first instance. However, he still can not decrypt the payment by the key again:

1. Notify TTP that the fail of using the payment decryption key again. (*sendTP-noti*)
2. TTP gets in touch with Bank to obtain a new key. (*getkfromB*)
3. Sends the new key to M. (*sendMkbyTP2*)

Again, the description above is mainly about behavior, in order address performance. More detailed security content has been described in [51]. The terms in the brackets after each item with bold font are the action name we used in the PEPA model. Moreover, we would like to propose three performance questions for this extended protocol as well as our previous case studies: "how many clients can a given TTP configuration support?", "how much service capacity must we provide at a TTP to satisfy a given number of clients?" and "what is the maximum rate at which keys can be refreshed before the TTP performance begins to degrade?" These questions are answered through numerical results in section 5.8.

5.6 PEPA Model of extended protocol

According to the same scenario of basic protocol, extended version can be modelled following:

$$
\begin{aligned}
CT_0 &\stackrel{def}{=} (download, r_d).CT_1 \\
CT_1 &\stackrel{def}{=} (agreePrice, r_a).CT_2 \\
CT_2 &\stackrel{def}{=} (sendMp, r_{smp}).CT_3 \\
CT_3 &\stackrel{def}{=} (sendCg, f_1).CT_4 + (sendCabort, f_2).CT_7
\end{aligned}
$$

$$
\begin{aligned}
CT_4 &\stackrel{def}{=} (sendMk, r_{smk}).CT_5 + (sendMabort, r_{sma}).CT_8 \\
CT_5 &\stackrel{def}{=} (sendCk, f_3).CT_6 + (noresponsedelay, r_n).CT_{14} \\
CT_6 &\stackrel{def}{=} (work, r_w).CT_0 + (sendTPall, r_{stp}).CT_9 \\
CT_7 &\stackrel{def}{=} (sendMabort, r_{sma}).CT_8 \\
CT_8 &\stackrel{def}{=} (work, r_w).CT_0 \\
CT_9 &\stackrel{def}{=} (notifyM1, r_1).CT_{10} \\
CT_{10} &\stackrel{def}{=} (sendTPk1, f_7).CT_{11} + (timeout1, r_{10}).CT_{12} \\
CT_{11} &\stackrel{def}{=} (sendCkbyTP1, r_3).CT_8 \\
CT_{12} &\stackrel{def}{=} (sendCkbyTP2, r_4).CT_{13} \\
CT_{13} &\stackrel{def}{=} (takeactionM, r_6).CT_8 \\
CT_{14} &\stackrel{def}{=} (sendTPall, r_{stp}).CT_{15} \\
CT_{15} &\stackrel{def}{=} (notifyM2, r_2).CT_{16} \\
CT_{16} &\stackrel{def}{=} (sendTPreason, f_4).CT_{17} + (timeout2, r_{11}).CT_{12} \\
CT_{17} &\stackrel{def}{=} (sendTPk2, f_5).CT_{18} \\
CT_{18} &\stackrel{def}{=} (sendMkbyTP1, r_7).CT_{19} \\
CT_{19} &\stackrel{def}{=} (sendCkbyTP3, p * r_5).CT_{20} + (sendCkbyTP3, (1-p) * r_5).CT_8 \\
CT_{20} &\stackrel{def}{=} (sendTPnoti, f_6).CT_{21} \\
CT_{21} &\stackrel{def}{=} (getkfromB, r_9).CT_{22} \\
CT_{22} &\stackrel{def}{=} (sendMkbyTP2, r_8).CT_8 \\
TP &\stackrel{def}{=} (notifyM1, r_1).TP + (notifyM2, r_2).TP + (sendCkbyTP1, r_3).TP \\
&\quad +(sendCkbyTP2, r_4).TP + (sendCkbyTP3, r_5).TP \\
&\quad +(takeactionM, r_6).TP + (sendMkbyTP1, r_7).TP \\
&\quad +(sendMbyTP2, r_8).TP + (getKfromB, r_9).TP \\
&\quad +(timeout1, r_{10}).TP + (timeout2, r_{11}).TP \\
System &\stackrel{def}{=} TP[K] \underset{\mathcal{L}}{\bowtie} CT_0[N]
\end{aligned}
$$

Where, $\mathcal{L} = \{notifyM1, notifyM2, sendCkbyTP1, sendCkbyTP2,$
$sendCkbyTP3, takeactionM, sendMkbyTP1, sendMkTP2, getKfromB,$

$timeout1, timeout2\}$,

$r_1 = r_{nm1}\frac{CT_9}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_2 = r_{nm2}\frac{CT_{15}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_3 = r_{scktp1}\frac{CT_{11}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_4 = r_{scktp2}\frac{CT_{12}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_5 = r_{scktp3}\frac{CT_{19}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_6 = r_{ta}\frac{CT_{13}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_7 = r_{smktp1}\frac{CT_{18}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_8 = r_{smktp2}\frac{CT_{22}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_9 = r_{kb}\frac{CT_{21}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_{10} = r_{t1}\frac{CT_{10}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$r_{11} = r_{t2}\frac{CT_{16}}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP)$,

$\sum waitingJobs_{TP} = \sum_{\forall i} CT_i(t), i \in \{9, 15, 11, 12, 19, 13, 18, 22, 21, 10, 16\}$.

if $N = 1$:

$f_1 = r_{scg}$, $f_2 = r_{sca}$, $f_3 = r_{sck}$, $f_4 = r_{stpr}$, $f_5 = r_{stpk}$, $f_6 = r_{stpno}$, $f_7 = r_{stpk}$,

if $N \neq 1$:

$f_1 = \frac{r_{scg}}{CT_3+1}$, $f_2 = \frac{r_{sca}}{CT_3+1}$, $f_3 = \frac{r_{sck}}{CT_5+1}$, $f_4 = \frac{r_{stpr}}{CT_{16}+1}$, $f_5 = \frac{r_{stpk}}{CT_{17}+1}$, $f_6 = \frac{r_{stpno}}{CT_{20}+1}$, $f_7 = \frac{r_{stpk}}{CT_{10}+1}$.

Compared to the PEPA model of basic protocol, a number of actions have been added. Thus, the number of local states of the CT component increases to 23, elearly enlarging the state space. Furthermore, the number of *functional rate* expressions has been extended to 7, each following the same formula as in the model of basic protocol. Let f_1 to f_7 denote these functional rates. Following a previous study in Chapter 4, a new kind of functional rates have been applied to avoid over estimating the value of rates of cooperation actions, which are denoted by $r_i, i = 1, 2, \cdots, 11$. Each of these functions describes the actual service rate if there is one job in the system(r_{nm1}, r_{nm2}, r_{scktp1}, r_{scktp2}, r_{scktp3}, r_{ta}, r_{smktp1}, r_{smktp2}, r_{kb}, r_{t1} and r_{t2}), or as a proportion of the number of waiting jobs (at TTP) of each type ($CT_i/\sum waitingJobs_{TP}, i = 9, 15, 11, 12, 19, 13, 18, 22, 21, 10, 16$) and the times of service ($min(TP, \sum waitingJobs_{TP})$, which allocates each service with respect to its job type to eliminates the potential race.

5.6.1 ODE analysis

Again, a set of ODEs, which represents the PEPA model of extended protocol, can be generated as follows:

$$
\begin{aligned}
\frac{d}{dt}CT_0 &= r_w CT_6(t) + r_w CT_8(t) - r_d CT_0(t) \\
\frac{d}{dt}CT_1 &= r_d CT_0(t) - r_a CT_1(t) \\
\frac{d}{dt}CT_2 &= r_a CT_1(t) - r_{smp} CT_2(t) \\
\frac{d}{dt}CT_3 &= r_{smp} CT_2(t) - \frac{r_{scg}}{CT_3(t)+1} CT_3(t) - \frac{r_{sca}}{CT_3(t)+1} CT3(t) \\
\frac{d}{dt}CT_4 &= \frac{r_{scg}}{CT_3(t)+1} CT_3(t) - r_{smk} CT_4(t) - r_{sma} CT_4(t) \\
\frac{d}{dt}CT_5 &= r_{smk} CT_4(t) - \frac{r_{sck}}{CT_5(t)+1} CT_5(t) - r_n CT_5(t) \\
\frac{d}{dt}CT_6 &= \frac{r_{sck}}{CT_5(t)+1} CT_5(t) - r_w CT_6(t) - r_{stp} CT_6(t) \\
\frac{d}{dt}CT_7 &= \frac{r_{sca}}{CT_3(t)+1} CT_3(t) - r_{sma} CT_7(t) \\
\frac{d}{dt}CT_8 &= r_{sma} CT_4(t) + r_{sma} CT_7(t) \\
&\quad + r_{scktp1} \frac{CT_{11}(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
&\quad + r_{ta} \frac{CT_{13}(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
&\quad + (1-p) r_{scktp3} \frac{CT_{19}(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
&\quad + r_{smktp2} \frac{CT_{22}(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) - r_w CT8(t) \\
\frac{d}{dt}CT_9 &= r_{stp} CT_6(t) - r_{nm1} \frac{CT_9(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{10} &= r_{nm1} \frac{CT_9(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
&\quad - \frac{r_{stpk}}{CT_{10}(t)+1} CT_{10}(t) \\
&\quad - r_{t1} \frac{CT_{10}(t)}{\sum waitingJobs_{TP}} min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{11} &= \frac{r_{stpk}}{CT_{10}(t)+1} CT_{10}(t)
\end{aligned}
$$

$$
\begin{aligned}
& -r_{scktp1}\frac{CT_{11}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{12} &= r_{t1}\frac{CT_{10}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& +r_{t2}\frac{CT_{16}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -r_{scktp2}\frac{CT_{12}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{13} &= r_{scktp2}\frac{CT_{12}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -r_{ta}\frac{CT_{13}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{14} &= r_n CT_5(t) - r_{stp}CT_{14}(t) \\
\frac{d}{dt}CT_{15} &= r_{stp}CT_{14} - r_{nm2}\frac{CT_{15}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{16} &= r_{nm2}\frac{CT_{15}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -\frac{r_{stpr}}{CT_{16}(t)+1}CT_{16}(t) \\
& -r_{t2}\frac{CT_{16}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{17} &= \frac{r_{stpr}}{CT_{16}(t)+1}CT_{16}(t) - \frac{r_{stpk}}{CT_{17}(t)+1}CT_{17}(t) \\
\frac{d}{dt}CT_{18} &= \frac{r_{stpk}}{CT_{17}(t)+1}CT_(17)(t) \\
& -r_{smktp1}\frac{CT_{18}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{19} &= r_{smktp1}\frac{CT_{18}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -r_{scktp3}\frac{CT_{19}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{20} &= pr_{scktp3}\frac{CT_{19}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -\frac{r_{stpno}}{CT_{20}(t)+1}CT_{20}(t) \\
\frac{d}{dt}CT_{21} &= \frac{r_{stpno}}{CT_{20}(t)+1}CT_{20}(t) \\
& -r_{kb}\frac{CT_{21}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
\frac{d}{dt}CT_{22} &= r_{kb}\frac{CT_{21}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP) \\
& -r_{smktp2}\frac{CT_{22}(t)}{\sum waitingJobs_{TP}}min(\sum waitingJobs_{TP}, TP)
\end{aligned}
$$

$$\frac{d}{dt}TP = 0$$

Where, $\sum waitingJobs_{TP} = \sum_{\forall i} CT_i(t), i \in \{9, 15, 11, 12, 19, 13, 18, 22, 21, 10, 16\}$.

In this analysis, we are primarily interesting in average number of waiting customers and average response time. The total waiting customers consist of customers waiting at Merchant (L(merchant)) and TTP (L(ttp)), and they are calculated as:

$$\begin{aligned} L(merchant) &= \sum_{\forall i} CT_i + CT_{10} * (\frac{f_7}{f_7 + r_{10}}) + CT_{16} * (\frac{f_4}{f_4 + r_{11}}), i \in \{3, 5, 17, 20\}. \\ L(ttp) &= \sum_{\forall i} CT_i + CT_{10} * (\frac{r_{10}}{f_7 + r_{10}}) + CT_{16} * (\frac{r_{11}}{f_4 + r_{11}}), \\ & \quad i \in \{9, 11, 12, 13, 15, 18, 19, 21, 22\}. \end{aligned}$$

It is easy to find that, in state CT_{10} and CT_{16}, customers have been forked into either the waiting service from the merchant or TTP. Hence, the average waiting customers in CT_{10} and CT_{16} has been divided into two parts according to the relevant rate of racing in above equations. As with the queue length, the average response time also contains two types. A merchant's mean response time is calculated as:

$$W_{merchant-i} = \frac{1}{f_i}, i \in \{1, 2, 3, 4, 5, 6, 7\}.$$

And following the previous study in Chapter 4, we can only evaluate the model when the number of TTP (K) is 1. To apply the arrival theorem, the average response time at the TTP can be approximately calculated as:

$$W_{ttp-i}(N) = \sum_{\forall i} \frac{L_i(N-1)}{r_i} + \frac{1}{r_i}, i \in \{1, \cdots, 11\}.$$

Where, $L_1 = CT_9, L_2 = CT_{15}, L_3 = CT_{11}, L_4 = CT_{12}, L_5 = CT_{19}, L_6 = CT_{13}, L_7 = CT_{18}, L_8 = CT_{22}, L_9 = CT_{21}, L_{10} = CT_{10} * (\frac{r_{10}}{f_7+r_{10}}), L_{11} = CT_{16} * (\frac{r_{11}}{f_4+r_{11}})$.

Furthermore, we intend to investigate the proportion of satisfied customers (been served), which defined as:

$$P_s(N) = \frac{CT_8}{N}.$$

5.7 Numerical results of extended protocol

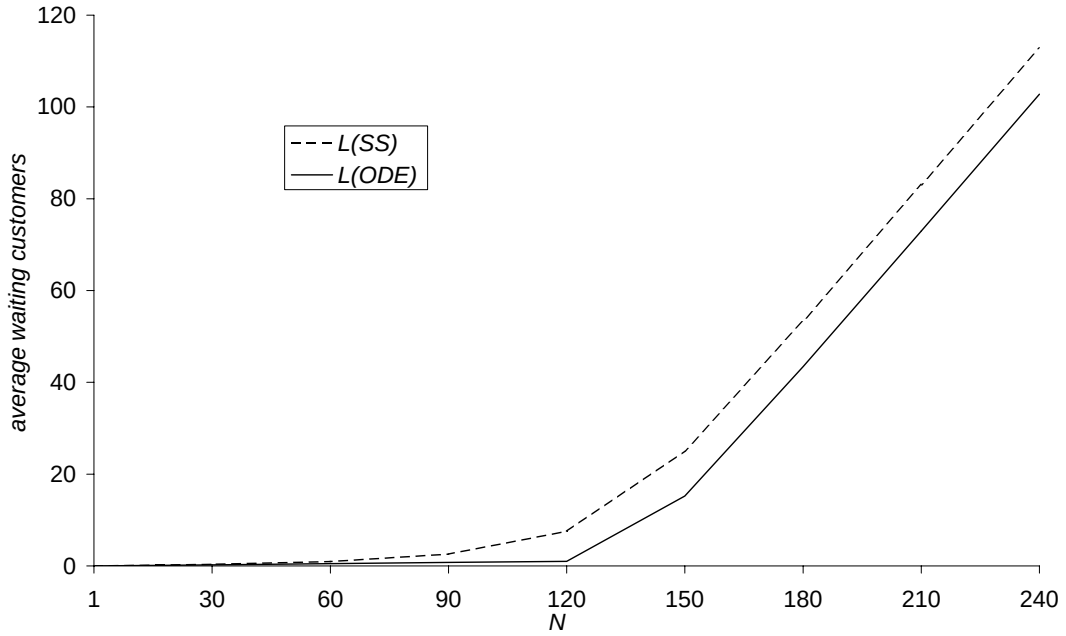

Figure 5.6: Average number of waiting customers at TTP varied with population size calculated by ODEs and stochastic simulation, $p = 0.5$, $r_w = 0.01$ and all other rates are 1.

Figure 5.6 compares the average number of waiting customers at TTP against initial population of customers calculated by ODEs and stochastic simulation. The queue length increases when more clients are involved in the system. However, it

is not difficult to spot that the two curves seems to keep a constant error when N is larger than 120. This phenomenon does not follow the ODE's normal excellent accuracy when N is very large. To investigate more deeply, we find that the population of behaviours after CT_{12} is actually very small, due to the race that between action *sendTPk* and *timeout1* in component CT_{10}, and also between action *sendTPreason* and *timeout2*. In the case where $N = 240$, it is a simple matter to calculate the functional rate of *sendTPk* $f_7(N = 240) \approx 0.85397$, and the functional rate of *timeout1* $r_{10}(N = 240) \approx 0.0020397$. The large difference also exists between *sendTPreason* and *timeout2*. About 400 times difference causes just a few components evolving to CT_{12} and its further (evolving) behaviours. Thus, $N = 240$ still can not be considered as a large scale system with the current set of rates. That explains why the two methods do not converge when $N = 240$. Nevertheless, the two curves will converge eventually in some value of N. To take a further experiment, we set r_{t1} and r_{t2}, the original rates of *timeout1* and *timeout2*, to 200, and keep all other rates unchanged. This is in order to switch more clients to the behaviours after CT_{12}. Still in case of $N = 240$, $L(ODE) \approx 99.9595$ and $L(SS) \approx 100.0637$, illustrating the argument above.

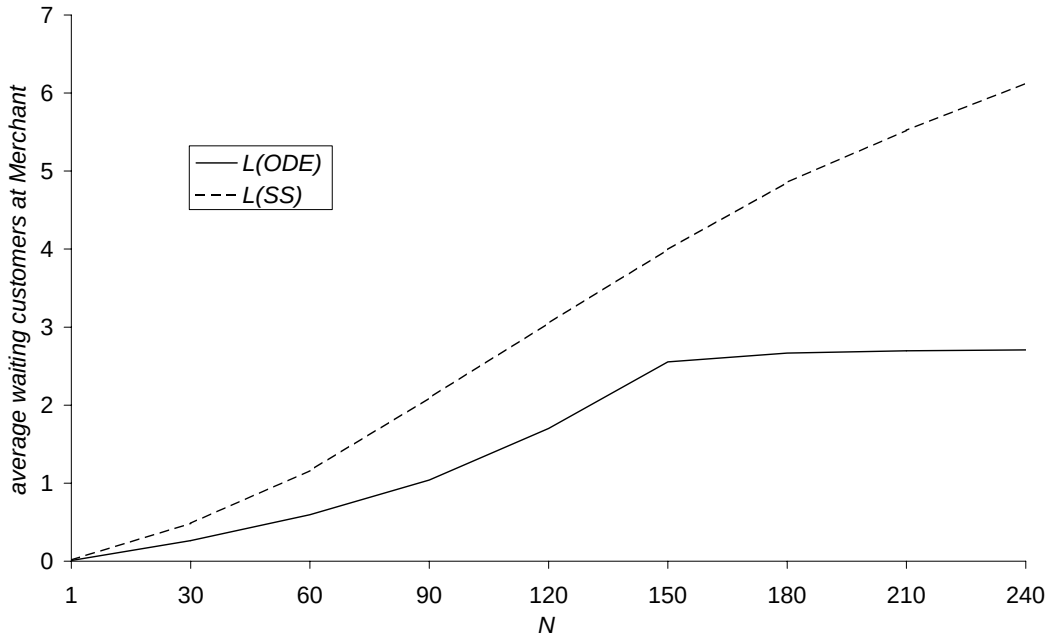

Figure 5.7: Average number of waiting customers at merchant varied with population size calculated by ODEs and stochastic simulation, $p = 0.5$, $r_w = 0.01$ and all other rates are 1.

The average number of waiting customers at the merchant is presented in Figure 5.7. Generally, the more customers involved, the more customers that will be waiting at the merchant. However, the results calculated by ODEs and stochastic simulation do not converged. This is caused by the same reason as discussed above. When the TTP is working for misbehaviour cases, it is become very busy (if there is only one TTP server, as in our model) and most of the customers are waiting for the TTP. Therefore, the scale of the queue length at the merchant remains very small. This is why results of ODE and stochastic simulation did not converge here.

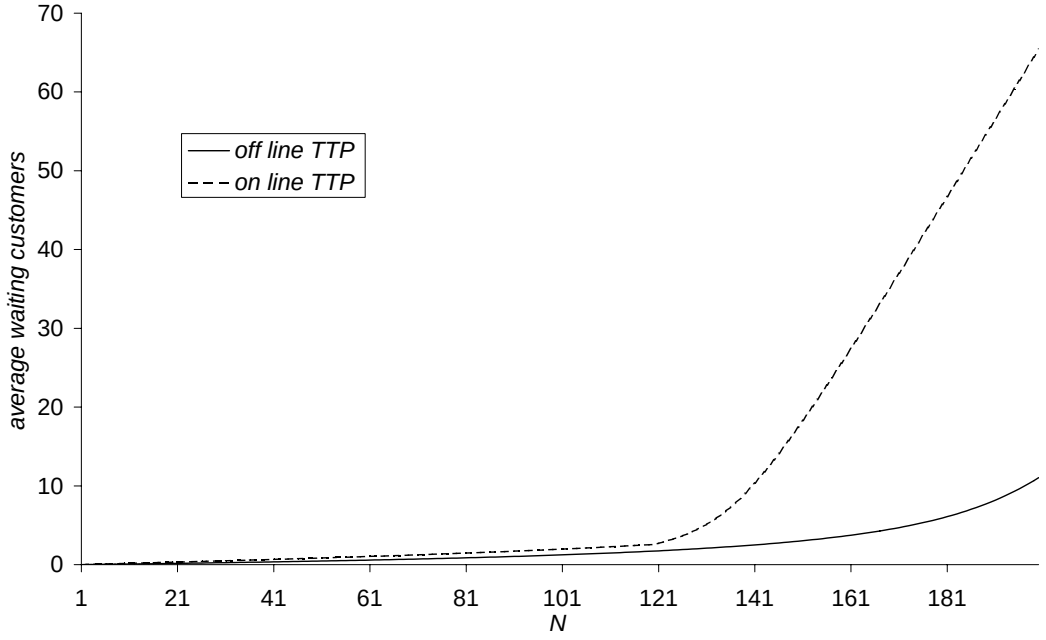

Figure 5.8: Average number of waiting customers with and without TTP varied with population size calculated by ODEs, $p = 0.5$, $r_w = 0.01$ and all other rates are 1.

The total average waiting customers with and without misbehaviour have been compared in Figure 5.8. Under the same rates for each relevant actions and the same involved number of customers, far more customers are waiting in a situation of misbehaviour, especially, when N is very large. That is an intuitive and expected result, because customers who encounter misbehaviour have recourse to the TTP for help, and then wait at the TTP. Hence, it is clear that misbehaviours reduce the performance of the whole system, and also demonstrate that this kind (optimistic) non-repudiation protocol could perform much better than those that

always employ an on-line TTP.

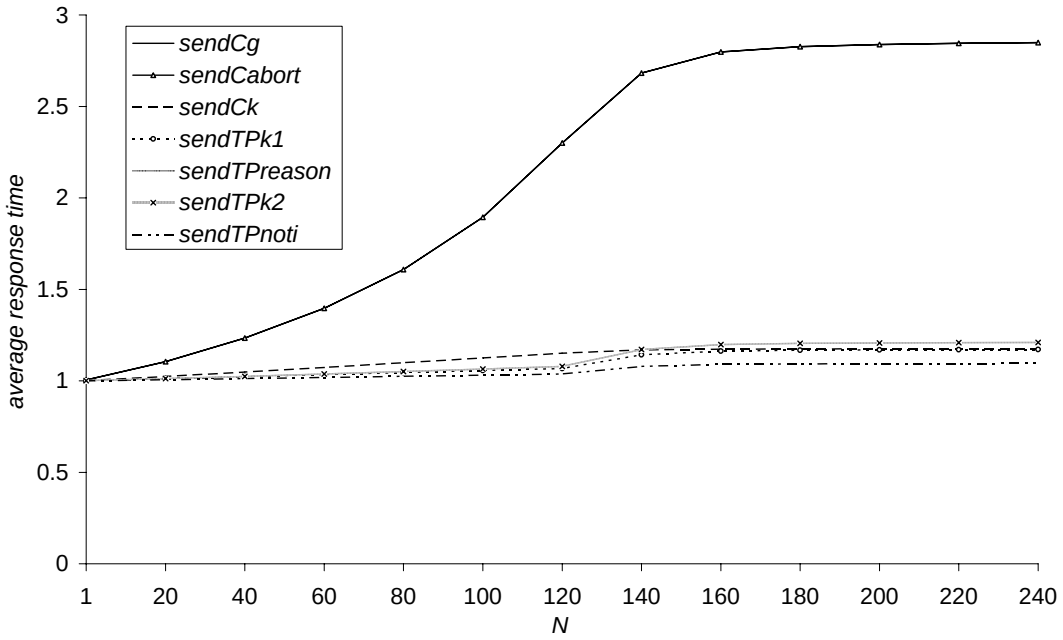

Figure 5.9: Average response time at merchant varied with population size calculated by ODEs, $p = 0.5$, $r_w = 0.01$ and all other rates are 1.

Figure 5.9 shows the average response time for the merchant at different actions. Overall, if we increase total number of clients in the system, the merchant takes longer to process each individual request. However, the response time increases slowly, and that is caused by the queue length which has been shown in Figure 5.7. Following our functional rates definition for the merchant (f_i), it is intuitively understood that queue length and response time should have the same increasing ratio. Moreover, more customers waiting for action *sendCg* and *sendCabort* than others, this gives longer a response time for these two actions.

Then, we experiment to increase the capacity of the TTP to twice that shown before (2), and plot the results for average response time for the TTP in all actions and the merchant in action *sendCg* in Figure 5.10 and Figure 5.11. From Figure 5.10, it is clear to spot that the response time for customers waiting at TTP is smaller if the TTP is more powerful. Nevertheless, the average response time for customers waiting at the merchant for action *sendCg* increases if we double the TTP's capacity. A quicker response from the TTP means that the number of customers waiting at misbehaviour stage decreases. Under the same total

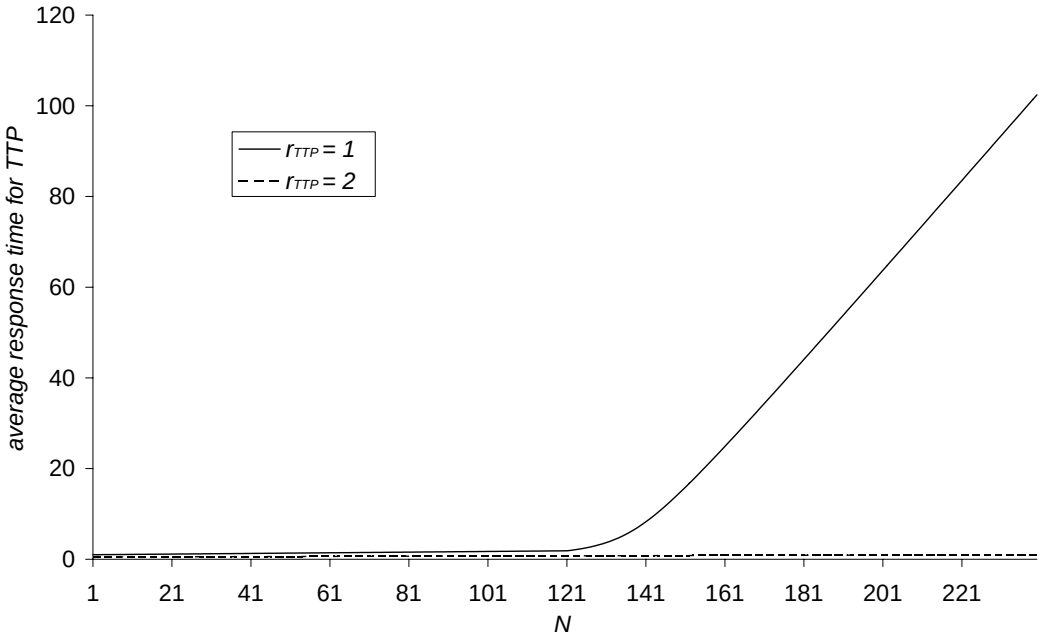

Figure 5.10: Average response time for TTP varied with population size calculated by ODEs, $p = 0.5$, $r_w = 0.01$ and all other rates are 1 except for r_{TTP}, where $r_{TTP} \in \{r_{nm1}, r_{nm2}, r_{scktp1}, r_{scktp2}, r_{scktp3}, r_{ta}, r_{smktp1}, r_{smktp2}, r_{kb}, r_{t1}, r_{t2}\}$.

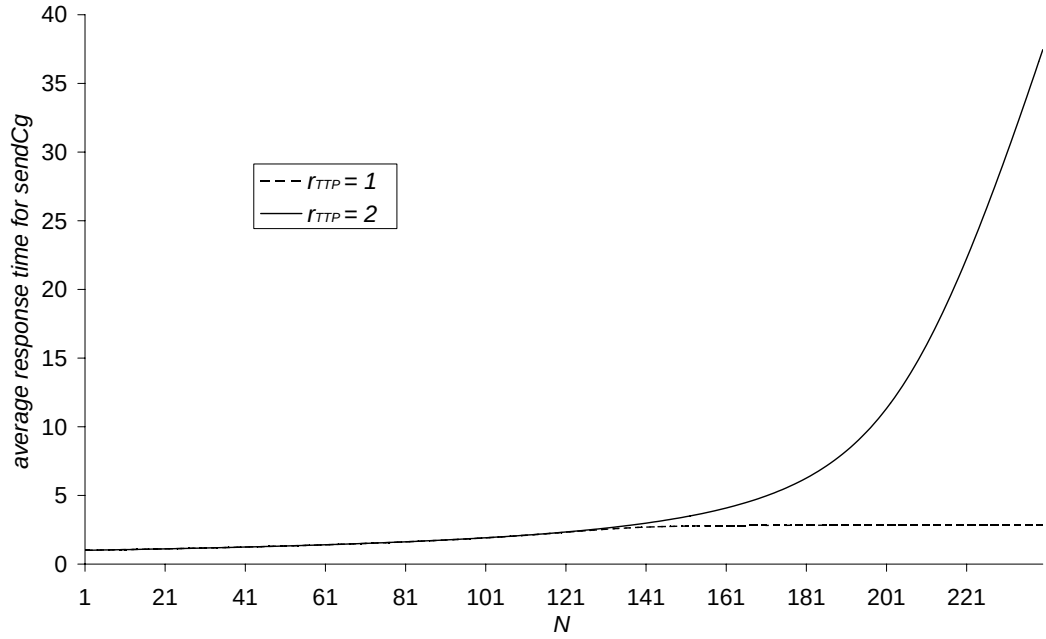

Figure 5.11: Average response time for sendCg varied with population size calculated by ODEs, $p = 0.5$, $r_w = 0.01$ and all other rates are 1 except for r_{TTP}, where $r_{TTP} \in \{r_{nm1}, r_{nm2}, r_{scktp1}, r_{scktp2}, r_{scktp3}, r_{ta}, r_{smktp1}, r_{smktp2}, r_{kb}, r_{t1}, r_{t2}\}$.

number of clients, more customers go to the normal stage without misbehaviour. Consequently, the number of customers (CT_3) waiting for action *sendCg* increases, and the average response time for these customers takes longer.

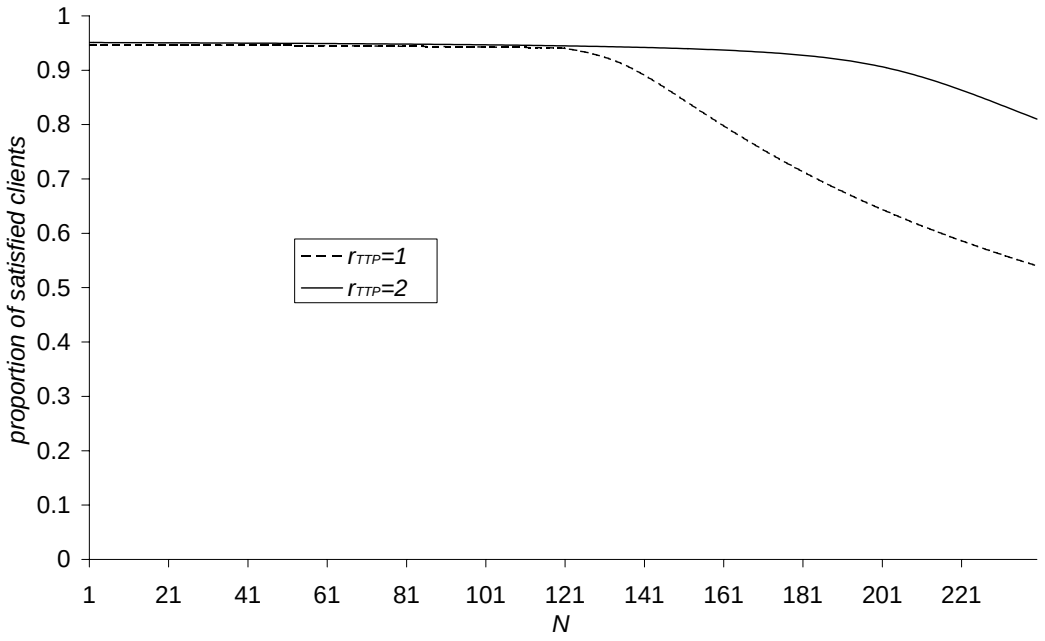

Figure 5.12: Proportion of satisfied customers varied with population size calculated by ODEs, $p = 0.5$, $r_w = 0.01$ and all other rates are 1 except for r_{TTP}, where $r_{TTP} \in \{r_{nm1}, r_{nm2}, r_{scktp1}, r_{scktp2}, r_{scktp3}, r_{ta}, r_{smktp1}, r_{smktp2}, r_{kb}, r_{t1}, r_{t2}\}$.

Finally, we plot the proportion of satisfied customers (been served) in Figure 5.12. Generally, the proportion decreases for both case ($r_{TTP} = 1$ and $r_{TTP} = 2$) if more customers come to the system. The two curves are very close before the point, $N = 120$, and both keep a very high percentage of satisfied customers in this area. After that point, those percentages start to go down clearly. However, the proportion for $r_{TTP} = 1$ drops more quickly than the other, and it becomes 50% when $N = 240$, while the percentage for $r_{TTP} = 2$ is still above 80%.

5.8 Utility function of extended protocol

Again, we apply the similar utility function to answer our proposed performance questions for extended protocol.

$$C = c_1 L + c_2 K r_p \ \ , \ c_1, c_2 \geq 0 \tag{5.1}$$

The same as the cost function in Chapter 3 and 4, L denotes the average waiting

customers at the non-repudiation sever (TTP), and K is number of servers. r_p is the response rate of the TTP. We assume the TTP server responds any type of jobs in the same rate here. C_1 and C_2 are cost rates, and they many depend on the type of system or quality of service agreement with customers.

5.8.1 Numerical results

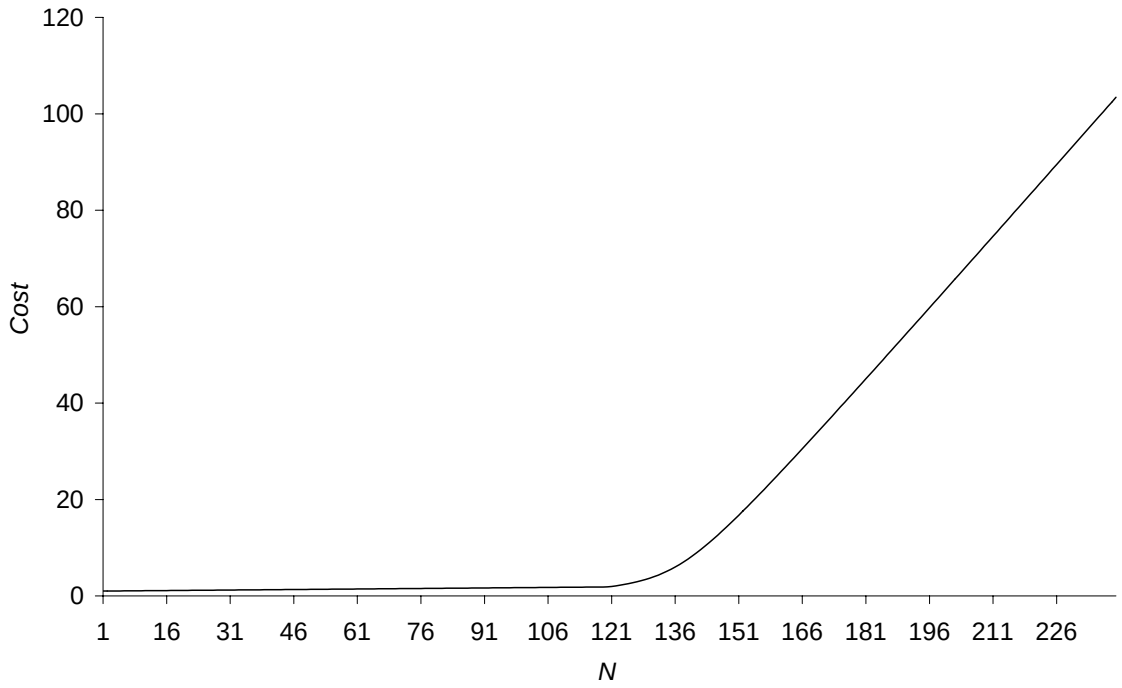

Figure 5.13: Cost varied against the number of clients calculated by ODEs, $p = 0.5$, $K = 1$, $c1 = c2 = 1$, $r_w = 0.01$ and all other rates are 1

Figure 5.13 shows the cost varied against the number of clients calculated by ODEs. Similar to the results of cost function in Chapter 3 and 4, more clients results in more waiting customers with fixed service capacity. Therefore, the total cost increases along with the cost of customer waiting goes up. Furthermore, it is a simple matter to find that the cost rises rapidly when N is around 130, and this is the maximum capacity that the TTP server can handle before performance start to significantly degrade.

Figure 5.14 presents the cost varied with number of TTP servers calculated by ODEs when total number of clients is 500. Again, customer waiting costs more in

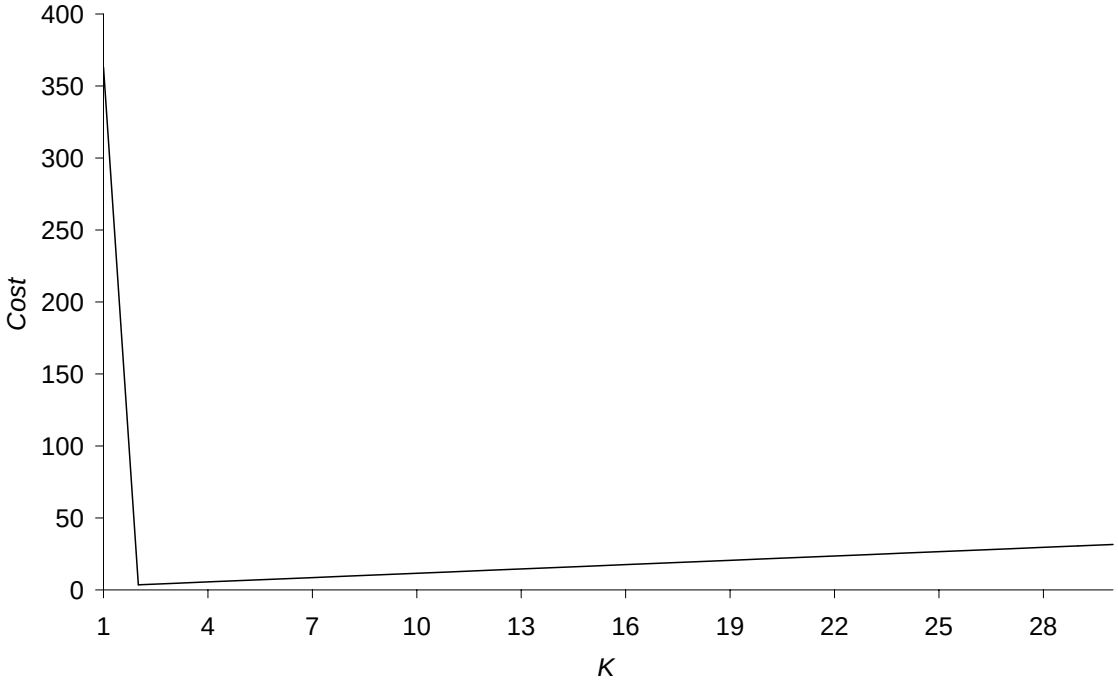

Figure 5.14: Cost varied with number of TTP servers calculated by ODEs, $p = 0.5$, $N = 500$, $c1 = c2 = 1$, $r_w = 0.01$ and all other rates are 1

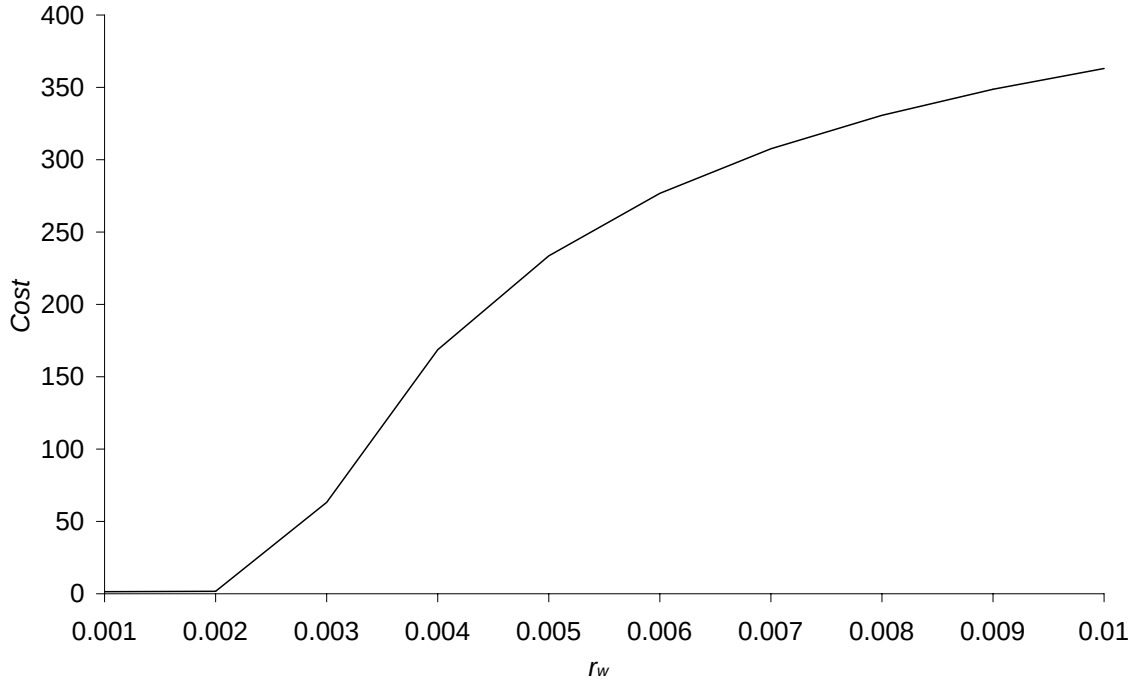

Figure 5.15: Cost varied with rate of *work* calculated by ODEs, $p = 0.5$, $N = 300$, $c1 = c2 = 1$, all other rates are 1 except for r_w

initial stage. Along with the system being given more servers, number of waiting clients is reduced. However, the cost of service dominate the total cost. The optimal point is around 2 in this case.

Figure 5.15 shows the cost varied with the rate of refresh key, r_w, calculated by ODEs. With fixed service capacity and total number of clients, more frequently refresh the session key results in more workload has been added in the system. Therefore, the cost of customer waiting increases. Similarly, we can easily find that the balance point between performance and security is around $r_w = 0.002$.

5.9 Summary

This chapter investigates an optimistic fair exchange protocol in an e-commerce environment. According to the optimistic characteristic, we model the protocol when the *third trust party* is offline and online respectively. Additionally, in this work, we consider the merchant server consists of several threads; PEPA works well in this style of modelling. The ODE solution do not converge with stochastic simulation when N is very large. However, in this context, this large N only gives large scale for part of the derivatives, and they are still may converge under other rates. Finally, a utility function has been analysed to better understand the system.

This case study learnt a new modelling form that can be added to our proposed work flow. In the modelling stage, the server (resource) can be modelled as serval threads with associated functional rates.

Chapter 6

Kerberos Protocol

6.1 Introduction

This chapter explores a common authentication protocol, known as Kerberos. The first PEPA model of multi-realms scenario is cumbersome to manage, because we need to change every customer's behaviour if one more realm is added to the whole environment (this situation has not been studied in previous chapters). Hence, a simplified model, which is bisimilar to the original one, has been proposed. To cope with state space explosion, following previous studies, an ODE based fluid flow analysis is employed to solve the models. The results are compared with stochastic simulation, and also compared with the original model. The novelty of this Chapter is the aggregation technique we applied to reduce the model components.

In the next section, a specification of the multi-realm Kerberos environment has been described. Then, the scenario is modeled in PEPA. After that, the original PEPA model has been simplified in Section 6.4, followed by fluid flow analysis of the simplified model. Numerical results are presented in the subsequent section. In Section 6.7, a utility function has been adapted to evaluate the protocol. Finally, we draw conclusions in Section 6.8.

6.2 Protocol specification

Figure 6.1 illustrates a full-service Kerberos environment, which is termed a *realm*. A realm usually consists of a Kerberos server, several clients, and some application servers. A Kerberos server can work as both an *Authentication server* (AS) and a *Ticket granting server* (TGS). Several realms form a network that we want to investigate. As any clients in a realm can request service from local application servers or remote application servers, following the Kerberos protocol (version 4) specification in [59], two work flows have been defined:

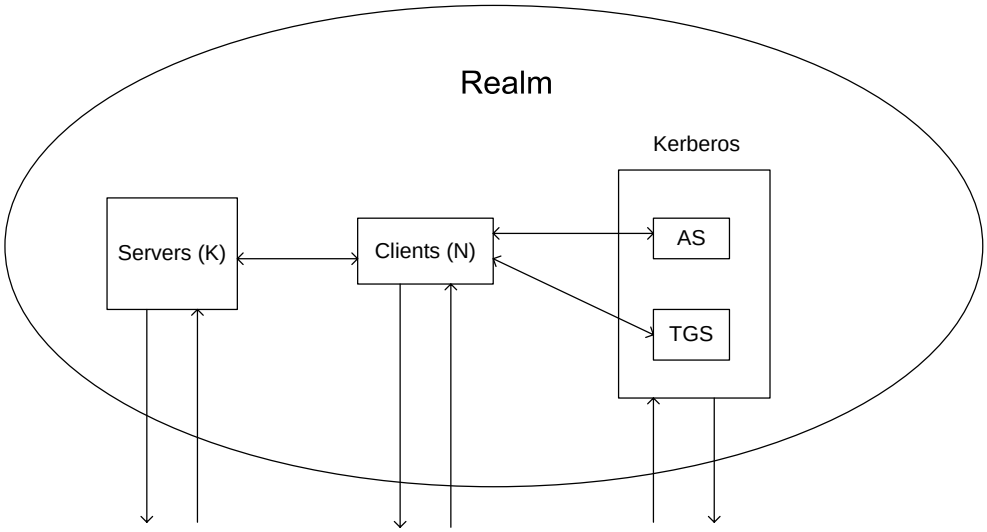

Figure 6.1: One realm in multi-realms Kerberos protocol

- a client requests service from local application server

  1. A client (U) requests a ticket-granting ticket (TGT) from local AS(a function of Kerberos server, denoted by KDC in context). (*requestTGT*)
  2. Local AS sends TGT to U. (*responseTGT*)
  3. U requests a service-granting ticket (SGT) from local TGS(a function of Kerberos server, denoted by KDC in context). (*requstSGTl*)
  4. Local TGS sends SGT to U. (*responseSGTl*)
  5. U request a service from a local application server (S) using SGT that received from TGS in last step. (*requestSl*)

6. Local S delivers the service to U. (*responseSl*)

- a client requests service from remote application server

1. U requests a TGT from local AS. (*requestTGT*)
2. Local AS sends TGT to U.(*responseTGT*)
3. U requests a ticket for remote TGS from local TGS. (*requestTGTr*)
4. Local TGS sends the ticket for remote TGS to U. (*responseTGTr*)
5. U requests SGT for remote application server from remote TGS. (*requestSGTrr*)
6. Remote TGS sends U the SGT for the remote application server. (*responseSGTrr*)
7. U requests a service to remote application server. (*requestSrr*)
8. The remote application server delivers the service to U. (*responseSrr*)

These two work flows concurrently exist with a probability (P) which will be introduced in sections of models.

6.3 The model

Following the protocol depiction in the previous section, our Kerberos scenario can be modelled through PEPA as:

$$
\begin{aligned}
&\forall i : \\
&U_{i0} \stackrel{def}{=} (requestTGT_i, r_{qtgt}).U_{i1} \\
&U_{i1} \stackrel{def}{=} (responseTGT_i, p * r_{xi1}).U_{i2l} \\
&\qquad + \sum_{\forall j \neq i} (responseTGT_i, \frac{1-p}{M-1} * r_{xi1}).U_{i2rj} \\
&U_{i2l} \stackrel{def}{=} (requestSGTl_i, r_{qsgtl}).U_{i3}
\end{aligned}
$$

$$
\begin{aligned}
U_{i3} &\stackrel{def}{=} (responseSGTl_i, r_{xi2}).U_{i4} \\
U_{i4} &\stackrel{def}{=} (requestSl_i, r_{qsl}).U_{i5} \\
U_{i5} &\stackrel{def}{=} (responseSl_i, r_{yi1}).U_{i6} \\
U_{i6} &\stackrel{def}{=} (work_i, r_w).U_{i0}
\end{aligned}
$$

$\forall j \neq i, f, g, k :$

$$
\begin{aligned}
U_{i2rj} &\stackrel{def}{=} (requestTGTr_{ij}, r_{qtgtr}).U_{i(7+q*5)} \\
U_{i(7+q*5)} &\stackrel{def}{=} (responseTGTr_{ij}, r_{xif}).U_{i(8+q*5)} \\
U_{i(8+q*5)} &\stackrel{def}{=} (requestSGTrr_{ij}, r_{qsgtrr}).U_{i(9+q*5)} \\
U_{i(9+q*5)} &\stackrel{def}{=} (responseSGTr_{ij}, r_{xig}).U_{i(10+q*5)} \\
U_{i(10+q*5)} &\stackrel{def}{=} (requestSrr_{ij}, r_{qsrr}).U_{i(11+q*5)} \\
U_{i(11+q*5)} &\stackrel{def}{=} (responseSr_{ij}, r_{yik}).U_{i6}
\end{aligned}
$$

$$
\begin{aligned}
KDC_i &\stackrel{def}{=} (responseTGT_i, r_{xi1}).KDC_i + (responseSGTl_i, r_{xi2}).KDC_i \\
&\quad + \sum_{\forall j \neq i and \forall f} (responseTGTr_{ij}, r_{xif}).KDC_i \\
&\quad + \sum_{\forall j \neq i and \forall g} (responseSGTr_{ji}, r_{xig}).KDC_i \\
S_i &\stackrel{def}{=} (responseSl_i, r_{yi1}).S_i + \sum_{\forall j \neq i and \forall k} (responseSr_{ji}, r_{yik}).S_i
\end{aligned}
$$

$$
System \stackrel{def}{=} \prod_{\forall i}(U_{i0}[N]) \underset{\mathcal{L}}{\bowtie} (\prod_{\forall i}(KDC_i) || \prod_{\forall i}(S_i[K]))
$$

Where $i, j \in 1, 2, \cdots, M$ (M is the number of realms).

$q = j - 1, j < i; q = j - 2, j > i.$

$f \in \{a_n | a_n = a_1 + (n-1)d\}, a_1 = 3, d = 1, n = M - 1.$

$g \in \{b_n | b_n = b_1 + (n-1)d\}, b_1 = a_1 + M - 1, d = 1, n = M - 1.$

$k \in \{c_n | c_n = c_1 + (n-1)d\}, c_1 = 2, d = 1, n = M - 1.$

$\mathcal{L} = \{responseTGT_i, responseSGTl_i, responseSl_i, responseTGTr_{ij},$

$responseSGTr_{ji}, responseSr_{ji}\}(\forall j \neq i).$

$r_{xi1} = r_{ptgt} \frac{U_{i1}(t)}{U_{i1}(t) + U_{i3}(t) + \sum_{\forall q} U_{i(7+q*5)(t)} + \sum_{\forall q} U_{i(9+q*5)}(t)}$

$$
\begin{aligned}
& * min(U_{i1}(t) + U_{i3}(t) + \textstyle\sum_{\forall q} U_{i(7+q*5)(t)} + \sum_{\forall q} U_{i(9+q*5)}(t), KDC_i) \\
r_{xi2} = r_{psgtl} & \tfrac{U_{i3}(t)}{U_{i1}(t)+U_{i3}(t)+\sum_{\forall q} U_{i(7+q*5)}(t)+\sum_{\forall q} U_{i(9+q*5)}(t)} \\
& * min(U_{i1}(t) + U_{i3}(t) + \textstyle\sum_{\forall q} U_{i(7+q*5)(t)} + \sum_{\forall q} U_{i(9+q*5)}(t), KDC_i)
\end{aligned}
$$

$\forall j \neq i, f, g$

$$
\begin{aligned}
r_{xif} = r_{ptgtr} & \tfrac{U_{i(7+q*5)}(t)}{U_{i1}(t)+U_{i3}(t)+\sum_{\forall q} U_{i(7+q*5)}(t)+\sum_{\forall q} U_{i(9+q*5)}(t)} \\
& * min(U_{i1}(t) + U_{i3}(t) + \textstyle\sum_{\forall q} U_{i(7+q*5)(t)} + \sum_{\forall q} U_{i(9+q*5)}(t), KDC_i) \\
r_{xig} = r_{psgtr} & \tfrac{U_{i(9+q*5)(t)}}{U_{i1}(t)+U_{i3}(t)+\sum_{\forall q} U_{i(7+q*5)}(t)+\sum_{\forall q} U_{i(9+q*5)}(t)} \\
& * min(U_{i1}(t) + U_{i3}(t) + \textstyle\sum_{\forall q} U_{i(7+q*5)(t)} + \sum_{\forall q} U_{i(9+q*5)}(t), KDC_i)
\end{aligned}
$$

$$
r_{yi1} = r_{psl} \tfrac{U_{i5}(t)}{U_{i5}(t)+\sum_{\forall q} U_{i(11+q*5)}(t)} * min(U_{i5}(t) + \textstyle\sum_{\forall q} U_{i(11+q*5)}(t), S_i)
$$

$\forall j \neq i, k$

$$
r_{yik} = r_{psr} \tfrac{U_{i(11+q*5)}(t)}{U_{i5}(t)+\sum_{\forall q} U_{i(11+q*5)}(t)} * min(U_{i5}(t) + \textstyle\sum_{\forall q} U_{i(11+q*5)}(t), S_i)
$$

In the formal definition of a PEPA model of the Kerberos protocol above, i denotes the i^{th} realm, and therefore the all components of the model is replicated for every realm (M realms in total). A branch is shown in U_{i1}: clients which would like to get service from local servers will consequently behaviour as U_{i2l} with probability p, and those who want to request remote servers will reach state U_{i2rj} (sum of all $j \neq i$) with probability $1 - p$. Hence, U_{i2l} to U_{i5} denote states for the local actions of realm i, and U_{i2j} to $U_{i(11+q*5)}$ are the states in a remote requesting phase. Finally, they all come to state U_{i6}, and we then add a *work* action before the clients starting another request from the initial stage U_{i0} to make it as cycle to investigate the steady state. This *work* action means that the clients do something else or just a simple delay after getting service from servers, and is utilised to control the requesting frequency.

In this model, the initial phase of remote requesting is represented by

$$
\sum_{\forall j \neq i} (responseTGT_i, \frac{1-p}{M-1} * r_{xi1}).U_{i2rj}
$$

j denotes all the realms remote to realm i, therefore, this sum contains $M-1$ items. Each of these items has its own sequential actions for the remote requesting stage. Hence, U_{i2rj} to U_{i6} in the above model is replicated $M-1$ times (all remote realms; $\forall j \neq i$) for each local realm (i). Subscript q, utilised for the client component in the remote requesting stage, is used to denote different names of derivatives for each remote realms ($\forall j \neq i$). As those realms are assumed homogeneous, they have the same probability to be requested. This probability is calculated by the total probability for remote requesting $(1-p)$ dividing by the number of remote realms $(M-1)$.

The servers described in the model only execute their service actions. For the KDC server, $responseTGT_i$, $responseSGl_i$ and $responseTGTr_{ij}(\forall j \neq i)$ are local actions in realm i, and the subscript in $responseTGTr_{ij}$ denotes that the local KDC in realm i responds a TGT to local clients. However, this TGT is utilised to request a remote KDC in realm j. The remote actions of a KDC are $responseSGTr_{ji}(\forall j \neq i)$, and this means that the local KDC in realm i responds a SGT to remote clients from one of the remote realms j. Similarly, an application server (S) responds to local clients (in realm i) by action $responseSl_i$, and to remote clients through actions $responseSr_{ji}(\forall j \neq i)$, which indicates a response to remote clients from realm j by application server in local realm i. Following previous study in Chapter 4, *functional rates* have been employed to avoid the race between different responding services in the same server. The *functional rates* are defined as the product of the actual service rate if there is only one job in the system, the proportion of the number of waiting jobs of each type and the number of instances of service; represented by $r_{xi1}, r_{xi2}, r_{xif}(\forall f), r_{xig}(\forall g)$ for KDC, and $r_{yi1}, r_{yik}(\forall k)$ for S. The subscript x indicates the rates of KDC services, and y implies the rates of S. Let i denotes the number (or name) of the realms where actions are executed; f, g and k are employed to assign different names to those rates. Finally, the system is defined as clients in all realms (M realms and N clients in each realm) shared all service actions with all KDC (one in each realm) and all S (K in each realm). In this model, we assume that there is only one type of application server in each realm.

To understand clearly, a two reals model is presented as follows:

$$
\begin{aligned}
U1_0 &\overset{def}{=} (requestTGT_1, r_{qtgt}).U1_1 \\
U1_1 &\overset{def}{=} (responseTGT_1, p * r_{x11}).U1_{2l} + (responseTGT_1, (1-p) * r_{x11}).U1_{2r} \\
U1_{2l} &\overset{def}{=} (requestSGTl_1, r_{qsgtl}).U1_3 \\
U1_3 &\overset{def}{=} (responseSGTl_1, r_{x12}).U1_4 \\
U1_4 &\overset{def}{=} (requestSl_1, r_{qsl}).U1_5 \\
U1_5 &\overset{def}{=} (responseSl_1, r_{y11}).U1_6 \\
Ul_6 &\overset{def}{=} (work_1, r_w).U1_0
\end{aligned}
$$

$$
\begin{aligned}
U1_{2r} &\overset{def}{=} (requestSGTr_{12}, r_{qtgtr}).U1_7 \\
U1_7 &\overset{def}{=} (responseTGTr_{12}, r_{x13}).U1_8 \\
U1_8 &\overset{def}{=} (requestSGTrr_{12}, r_{qsgtrr}).U1_9 \\
U1_9 &\overset{def}{=} (responseSGTr_{12}, r_{x24}).U1_{10} \\
U1_{10} &\overset{def}{=} (requestSrr_{12}, r_{qsrr}).U1_{11} \\
U1_{11} &\overset{def}{=} (responseSr_{12}, r_{y22}).U1_6
\end{aligned}
$$

$$
\begin{aligned}
U2_0 &\overset{def}{=} (requestTGT_2, r_{qtgt}).U2_1 \\
U2_1 &\overset{def}{=} (responseTGT_2, p * r_{x21}).U2_{2l} + (responseTGT_2, (1-p) * r_{x21}).U2_{2r} \\
U2_{2l} &\overset{def}{=} (requestSGTl_2, r_{qsgtl}).U2_3 \\
U2_3 &\overset{def}{=} (responseSGTl_2, r_{x22}).U2_4 \\
U2_4 &\overset{def}{=} (requestSl_2, r_{qsl}).U2_5 \\
U2_5 &\overset{def}{=} (responseSl_2, r_{y21}).U2_6 \\
U2_6 &\overset{def}{=} (work_2, r_w).U2_0
\end{aligned}
$$

$$
U2_{2r} \overset{def}{=} (requestSGTr_{21}, r_{qtgtr}).U2_7
$$

$$
\begin{aligned}
U2_7 &\stackrel{def}{=} (responseTGTr_{21}, r_{x23}).U2_8 \\
U2_8 &\stackrel{def}{=} (requestSGTrr_{21}, r_{qsgtrr}).U2_9 \\
U2_9 &\stackrel{def}{=} (responseSGTr_{21}, r_{x14}).U2_{10} \\
U2_{10} &\stackrel{def}{=} (requestSrr_{21}, r_{qsrr}).U2_{11} \\
U2_{11} &\stackrel{def}{=} (responseSr_{21}, r_{y12}).U2_6
\end{aligned}
$$

$$
\begin{aligned}
KDC_1 \stackrel{def}{=}\ & (responseTGT_1, r_{x11}).KDC_1 \\
& +(responseSGTl_1, r_{x12}).KDC_1 \\
& +(responseTGTr_{12}, r_{x13}).KDC_1 \\
& +(responseSGTr_{21}, r_{x14}).KDC_1
\end{aligned}
$$

$$
\begin{aligned}
KDC_2 \stackrel{def}{=}\ & (responseTGT_2, r_{x21}).KDC_2 \\
& +(responseSGTl_2, r_{x22}).KDC_2 \\
& +(responseTGTr_{21}, r_{x23}).KDC_2 \\
& +(responseSGTr_{12}, r_{x24}).KDC_2
\end{aligned}
$$

$$
S_1 \stackrel{def}{=} (responseSl_1, r_{y11}) + (responseSr_{21}, r_{y12})
$$

$$
S_2 \stackrel{def}{=} (responseSl_2, r_{y21}) + (responseSr_{12}, r_{y22})
$$

$$
System \stackrel{def}{=} (U1_0[N]||U2_0[N]) \underset{\mathcal{L}}{\bowtie} (KDC_1||KDC_2||S_1||S_2)
$$

Where $\mathcal{L} = \{responseTGT_1, responseSGTl_1, responseTGTr_{12}, responseSGTr_{21}, responseTGT_2, responseSGTl_2, responseTGTr_{21}, responseSGTr_{12}, responseSl_1, responseSr_{21}, responseSl_2, responseSr_{12}\}$

$r_{x11} = r_{ptgt} * \frac{U1_1}{U1_1+U1_3+U1_7+U2_9} * min(U1_1 + U1_3 + U1_7 + U2_9, KDC_1)$

$r_{x12} = r_{psgtl} * \frac{U1_3}{U1_1+U1_3+U1_7+U2_9} * min(U1_1 + U1_3 + U1_7 + U2_9, KDC_1)$

$r_{x13} = r_{ptgtr} * \frac{U1_7}{U1_1+U1_3+U1_7+U2_9} * min(U1_1 + U1_3 + U1_7 + U2_9, KDC_1)$

$r_{x14} = r_{psgtr} * \frac{U2_9}{U1_1+U1_3+U1_7+U2_9} * min(U1_1 + U1_3 + U1_7 + U2_9, KDC_1)$

$$r_{x21} = r_{ptgt} * \frac{U2_1}{U2_1+U2_3+U2_7+U1_9} * min(U2_1 + U2_3 + U2_7 + U1_9, KDC_2)$$
$$r_{x22} = r_{psgtl} * \frac{U2_3}{U2_1+U2_3+U2_7+U1_9} * min(U2_1 + U2_3 + U2_7 + U1_9, KDC_2)$$
$$r_{x23} = r_{ptgtr} * \frac{U2_7}{U2_1+U2_3+U2_7+U1_9} * min(U2_1 + U2_3 + U2_7 + U1_9, KDC_2)$$
$$r_{x24} = r_{psgtr} * \frac{U1_9}{U2_1+U2_3+U2_7+U1_9} * min(U2_1 + U2_3 + U2_7 + U1_9, KDC_2)$$
$$r_{y11} = r_{psl} * \frac{U1_5}{U1_5+U2_11} * min(U1_5 + U2_11, S_1)$$
$$r_{y12} = r_{psr} * \frac{U2_11}{U1_5+U2_11} * min(U1_5 + U2_11, S_1)$$
$$r_{y21} = r_{psl} * \frac{U2_5}{U2_5+U1_11} * min(U2_5 + U1_11, S_2)$$
$$r_{y22} = r_{psr} * \frac{U1_11}{U2_5+U1_11} * min(U2_5 + U1_11, S_2)$$

6.4 Simplification

Clearly, the above model not only suffers from the state space explosion problem with a large population, but is also problematic to specify. Once additional realms are added in the scenario, further remote actions are needed for clients in the remote requesting stage, KDC servers and application servers in the model. Consequently, the number of functional rates is increased which cause further complexity. Therefore, it is necessary to simplify this model, especially, the remote requesting stage. A homogeneous assumption leads us to consider the possibility of aggregating all realms, however, the cross realm actions obstruct a simple combing of all relevant derivatives. That is because the cross realm actions can no longer determine which remote realm to request after aggregation. These actions are $requestSGTrr_{ij}, responseSGTr_{ij}, requestSrr_{ij}$ and $responseSr_{ij}$. The actions $requestSGTrr_{ij}$ and $requestSrr_{ij}$, are active actions of local clients, and these actions are not shared with any other component. Hence, they can be considered a simple delay, and so $responseSGTr_{ij}$ and $responseSr_{ij}$ become the key issue. Following the homogeneous assumption, these realms are exactly the same. Therefore, each KDC_i is the same ($KDC_1 \equiv \cdots \equiv KDC_i \equiv \cdots \equiv KDC_M$). We then can infer that all $responseSGTr_{ji}$ are the same. Hence, $responseSGTr_{ji} \equiv responseSGTr_{ij}$. From an overall performance view, a KDC server responding to a remote client is equivalent to one responding a relevant local client. Similarly,

$responseSr_{ji} \equiv responseSr_{ij}$. Now, the original model is able to be aggregated and simplified as follows:

$$
\begin{aligned}
U_0 &\stackrel{def}{=} (requestTGT, r_{qtgt}).U_1 \\
U_1 &\stackrel{def}{=} (responseTGT, p * r_{x1}).U_{2l} + (responseTGT, (1-p) * r_{x1}).U_{2r} \\
U_{2l} &\stackrel{def}{=} (requestSGTl, r_{qsgtl}).U_3 \\
U_3 &\stackrel{def}{=} (responseSGTl, r_{x2}).U_4 \\
U_4 &\stackrel{def}{=} (requestSl, r_{qsl}).U_5 \\
U_5 &\stackrel{def}{=} (responseSl, r_{y1}).U_6 \\
U_6 &\stackrel{def}{=} (work, r_w).U_0 \\
U_{2r} &\stackrel{def}{=} (requestTGTr, r_{qtgtr}).U_7 \\
U_7 &\stackrel{def}{=} (responseTGTr, r_{x3}).U_8 \\
U_8 &\stackrel{def}{=} (requestSGTrr, r_{qsgtrr}).U_9 \\
U_9 &\stackrel{def}{=} (responseSGTr, r_{x4}).U_{10} \\
U_{10} &\stackrel{def}{=} (requestSrr, r_{qsrr}).U_{11} \\
U_{11} &\stackrel{def}{=} (responseSr, r_{y2}).U_6
\end{aligned}
$$

$$
\begin{aligned}
KDC &\stackrel{def}{=} (responseTGT, r_{x1}).KDC + (responseSTGl, r_{x2}).KDC \\
&\quad +(responseTGTr, r_{x3}).KDC + (responseSGTr, r_{x4}).KDC \\
S &\stackrel{def}{=} (responeSl, r_{y1}).S + (responseSr, r_{y2}).S
\end{aligned}
$$

$$
System \stackrel{def}{=} U_0[M * N] \underset{\mathcal{L}}{\bowtie} (KDC[M] || S[M * K])
$$

where $\mathcal{L} = \{responseTGT, responseSGTl, responseTGTr, responseSGTrr, responseSl, responseSr\}$.

$r_{x1} = r_{ptgt} \frac{U_1(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)} min(U_1(t) + U_3(t) + U_7(t) + U_9(t), KDC)$

$r_{x2} = r_{psgtl} \frac{U_3(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)} min(U_1(t) + U_3(t) + U_7(t) + U_9(t), KDC)$

$r_{x3} = r_{ptgtr} \frac{U_7(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)} min(U_1(t) + U_3(t) + U_7(t) + U_9(t), KDC)$

$r_{x4} = r_{psgtr} \frac{U_9(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)} min(U_1(t) + U_3(t) + U_7(t) + U_9(t), KDC)$

$r_{y1} = r_{psl} \frac{U_5(t)}{U_5(t)+U_{11}(t)} * min(U_5(t) + U_{11}(t), S)$

$r_{y2} = r_{psr} \frac{U_{11}(t)}{U5(t)+U_{11}(t)} * min(U_5(t) + U_{11}(t), S)$

M is number of realms, N is number of clients in one realm, K is number of application servers in one realm.

Original Model	Simplified Model	Original Model	Simplified Model
$\sum_{\forall i} U_{i0}$	U_0	$\sum_{\forall i} \sum_{\forall j \neq i} U_{i2rj}$	U_{2r}
$\sum_{\forall i} U_{i1}$	U_1	$\sum_{\forall i} \sum_{\forall q} U_{i(7+q*5)}$	U_7
$\sum_{\forall i} U_{i2l}$	U_{2l}	$\sum_{\forall i} \sum_{\forall q} U_{i(8+q*5)}$	U_8
$\sum_{\forall i} U_{i3}$	U_3	$\sum_{\forall i} \sum_{\forall q} U_{i(9+q*5)}$	U_9
$\sum_{\forall i} U_{i4}$	U_4	$\sum_{\forall i} \sum_{\forall q} U_{i(10+q*5)}$	U_{10}
$\sum_{\forall i} U_{i5}$	U_5	$\sum_{\forall i} \sum_{\forall q} U_{i(11+q*5)}$	U_{11}
$\sum_{\forall i} U_{i6}$	U_6	$\sum_{\forall i} KDC_i$	KDC
		$\sum_{\forall i} S_i$	S

Figure 6.2: Comparison of relevant derivatives between original and simplified model

Figure 6.2 presents all derivatives from the aggregated model and its relevant states in the original model. Obviously, the state space is reduced to some extent, and it is easier to write it down for different populations by only changing number of realms (M) and clients (N) rather than changing system behaviour, as in original model. However, transient behaviours of this simplification are different from the original model, because some individual behaviours have been eliminated, and those relevant behaviour are combined to one action.

6.5 ODE analysis

From the lessons we have learned from previous studies, fluid flow approximation (based on ODE) has been chosen as the scalable analysis technique here. Following the approach in [34], the set of ODEs of aggregated model is derived below:

$$
\begin{aligned}
\frac{d}{dt}U_0(t) &= r_w U_7(t) - r_{qtgt} U_0(t) \\
\frac{d}{dt}U_1(t) &= r_{qtgt} U_0(t) - r_{x1} \\
\frac{d}{dt}U_{2l}(t) &= p * r_{x1} - r_{qsgtl} U_{2l}(t) \\
\frac{d}{dt}U_3(t) &= r_{qsgtl} U_{2l}(t) - r_{x2} \\
\frac{d}{dt}U_4(t) &= r_{x2} - r_{qsl} U_4(t) \\
\frac{d}{dt}U_5(t) &= r_{qsl} U_4(t) - r_{y1} \\
\frac{d}{dt}U_6(t) &= r_{y1} + r_{y2} - r_w U_6(t) \\
\frac{d}{dt}U_{2r}(t) &= (1-p) * r_{x1} - r_{qtgtr} U_{2r}(t) \\
\frac{d}{dt}U_7(t) &= r_{qtgtr} U_{2r}(t) - r_{x3} \\
\frac{d}{dt}U_8(t) &= r_{x3} - r_{qsgtrr} U_8(t) \\
\frac{d}{dt}U_9(t) &= r_{qsgtrr} U_8(t) - r_{x4} \\
\frac{d}{dt}U_{10}(t) &= r_{x4} - r_{qsrr} U_{10}(t) \\
\frac{d}{dt}U_{11}(t) &= r_{qsrr} U_{10}(t) - r_{y2}
\end{aligned}
$$

$$
\begin{aligned}
\frac{d}{dt}KDC(t) &= 0 \\
\frac{d}{dt}S(t) &= 0
\end{aligned}
$$

Where:

$$r_{x1} = r_{ptgt}\frac{U_1(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)}min(U_1(t)+U_3(t)+U_7(t)+U_9(t), KDC)$$
$$r_{x2} = r_{psgtl}\frac{U_3(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)}min(U_1(t)+U_3(t)+U_7(t)+U_9(t), KDC)$$
$$r_{x3} = r_{ptgtr}\frac{U_7(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)}min(U_1(t)+U_3(t)+U_7(t)+U_9(t), KDC)$$
$$r_{x4} = r_{psgtr}\frac{U_9(t)}{U_1(t)+U_3(t)+U_7(t)+U_9(t)}min(U_1(t)+U_3(t)+U_7(t)+U_9(t), KDC)$$
$$r_{y1} = r_{psl}\frac{U_5(t)}{U_5(t)+U_{11}(t)} * min(U_5(t)+U_{11}(t), S)$$
$$r_{y2} = r_{psr}\frac{U_{11}(t)}{U5(t)+U_{11}(t)} * min(U_5(t)+U_{11}(t), S)$$

To explore the performance of servers, we investigate the average queue length and average response time for the KDC and the application server in this analysis. In the aggregated model, the number of average waiting clients in the KDC is represent by the sum of $U_1(t), U_3(t), U_7(t)$ and $U_9(t)$ when $t \to \infty$. Following the homogeneous assumption, all waiting clients are equally distributed in each realm. Hence, the average queue length for each KDC is calculated as:

$$L(KDC) = \frac{U_1(t)+U_3(t)+U_7(t)+U_9(t)}{M}. \quad (t \to \infty)$$

Similarly, total number of waiting clients in S is $U_5(t) + U_{11}(t)(t \to \infty)$, which can be divided by the number of realms to obtain the average queue length for S in one realm as:

$$L(S) = \frac{U_5(t)+U_{11}(t)}{M}. \quad (t \to \infty)$$

In terms of the average response time, the same problem has been faced as previous study. If we apply *Arrival Theorem* to calculate average response time for multiple servers with multiple jobs, the most essential item need to known is the time takes for one server to become available, and the prerequisite is understanding the order of different jobs in the queue. However, the order is not possible (or not easy) to be obtained. Therefore, firstly, the average response time is defined as waiting time here, which just means the average time to process all the jobs in the queue. We then approximated it as the sum of the time of each type of

jobs being processed by all servers as follows:

$$W(KDC) = \frac{U_1(t)}{M * r_{ptgt}} + \frac{U_3(t)}{M * r_{psgtl}} + \frac{U_7(t)}{M * r_{ptgtr}} + \frac{U_9(t)}{M * r_{psgtr}}. \quad (t \to \infty)$$

$$W(S) = \frac{U_5(t)}{M * K * r_{psl}} + \frac{U_{11}(t)}{M * K * r_{psr}}. \quad (t \to \infty)$$

This approximation assumes each type of jobs are all equally served by all servers. Apparently, closer for the time of processing those types of jobs, more accurate for this approximated results. One can imagine that if a extremely large job is processed firstly, and it will block other small jobs for a long time. This is a huge difference to move it to the end of the queue.

In addition, we would like to propose similar performance questions for Kerberos protocol: "how many clients can a given Kerberos server configuration support?" and "what is the maximum rate at which keys can be refreshed before the Kerberos performance begins to degrade?" As there is only one Kerberos server is each realm, we get ride of the question about varying the number of servers in this case study. These questions are answered through numerical results in section 5.8.

6.6 Numerical results

Figure 6.3 shows the average queue length calculated by ODEs against the number of client in each realm, and verified by stochastic simulation with two realms. Overall, more involved customers cause more clients waiting at the KDC server. The diverged part between ODEs and stochastic simulation occurs when N is in interval of 20 to 50. For $N > 50$, ODE coincides with results from stochastic simulation, which follows the rule that fluid approximation approaching exact results with large population, and the large population can be defined as 50

clients at the KDC in this case. The same comparison is presented in Figure 6.4 for application server (S). More clients waiting at S with increasing number of clients in each realm (N), however, the curves flatten out as N increases. For the whole system, the busier server KDC is, the more customers are waiting at KDC than at S. Therefore, more clients of new arrivals are blocked at the KDC rather than S, and this then leads the increasing rate reduced along with N. This is also the reason that results of ODE and stochastic simulation do not converge for S. Reviewing the value of queue length in Figure 6.3 and Figure 6.4, the number of clients waiting at KDC are 100 times more then S. The scale of number of customers waiting at S is still very small, and apparently results from the ODEs cannot converge to stochastic simulation at this scale.

Figure 6.5 and Figure 6.6 compare the average queue length calculated from the original model and simplified model by ODEs, in the case of two realms and three realms, respectively. As the original model is hard to specify with four realms and more, it is only possible to conduct this comparison manually for a maximum of three realms. Those comparisons show that these results from

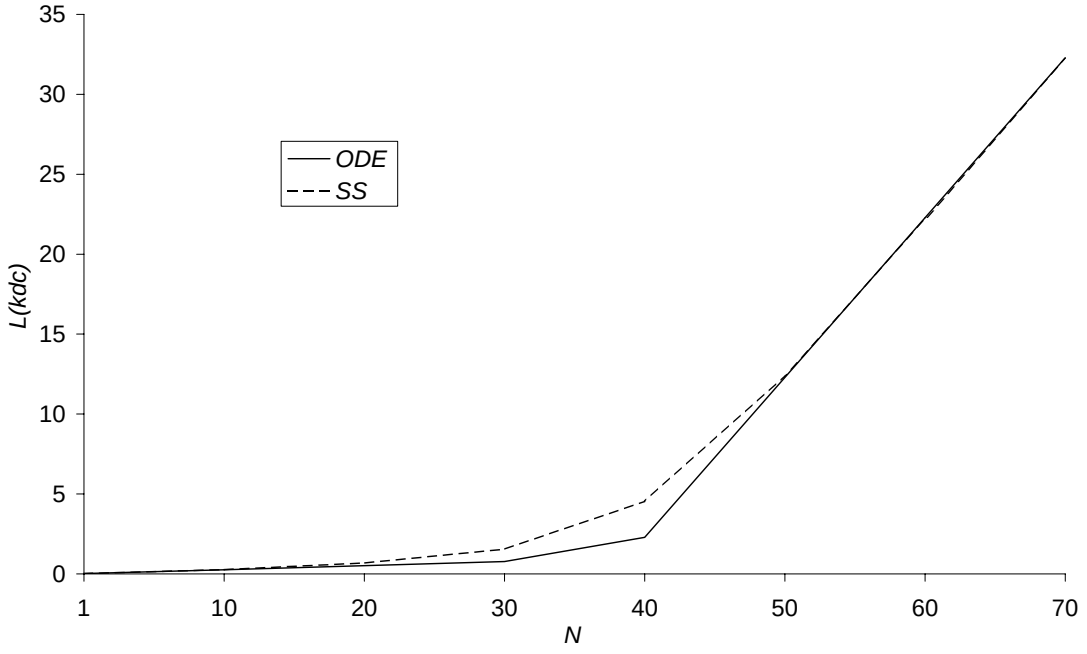

Figure 6.3: Average waiting customer for KDC varied with population size calculated by ODEs and stochastic simulation, $r_{qtgt} = r_{ptgt} = r_{qsgtl} = r_{qtgtr} = r_{psgtl} = r_{qsl} = r_{psl} = r_{ptgtr} = 1, r_{qsgtrr} = r_{psgtr} = r_{qsrr} = r_{psr} = 0.5, r_w = 0.01, p = 0.6, K = 1, M = 2$

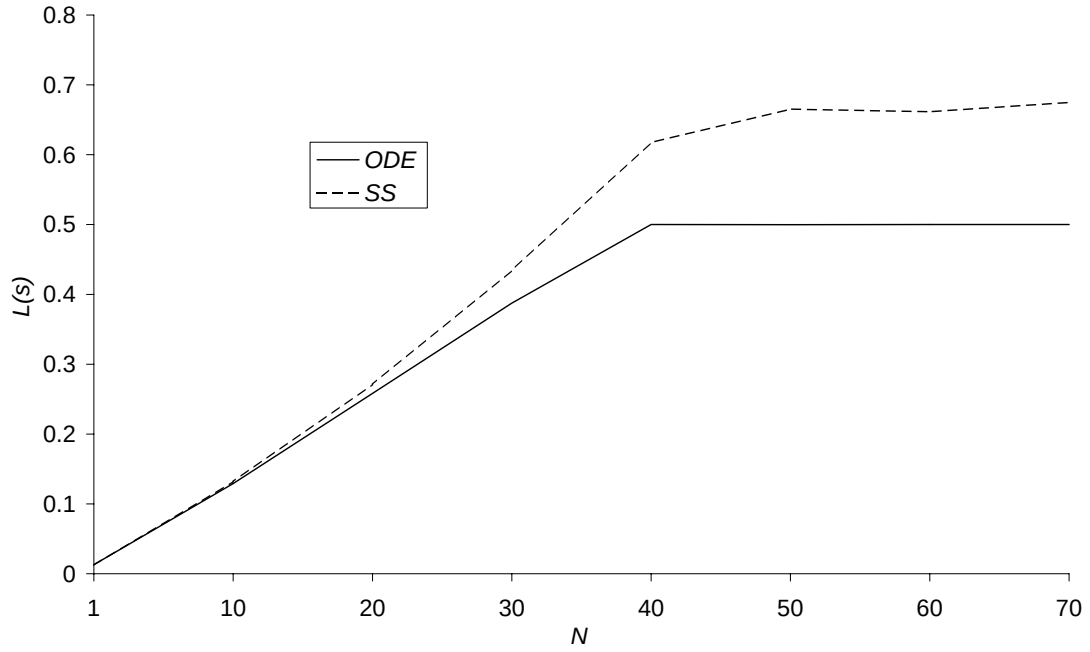

Figure 6.4: Average waiting customer for S varied with population size calculated by ODEs and stochastic simulation, $r_{qtgt} = r_{ptgt} = r_{qsgtl} = r_{qtgtr} = r_{psgtl} = r_{qsl} = r_{psl} = r_{ptgtr} = 1, r_{qsgtrr} = r_{psgtr} = r_{qsrr} = r_{psr} = 0.5, r_w = 0.01, p = 0.6, K = 1, M = 2$

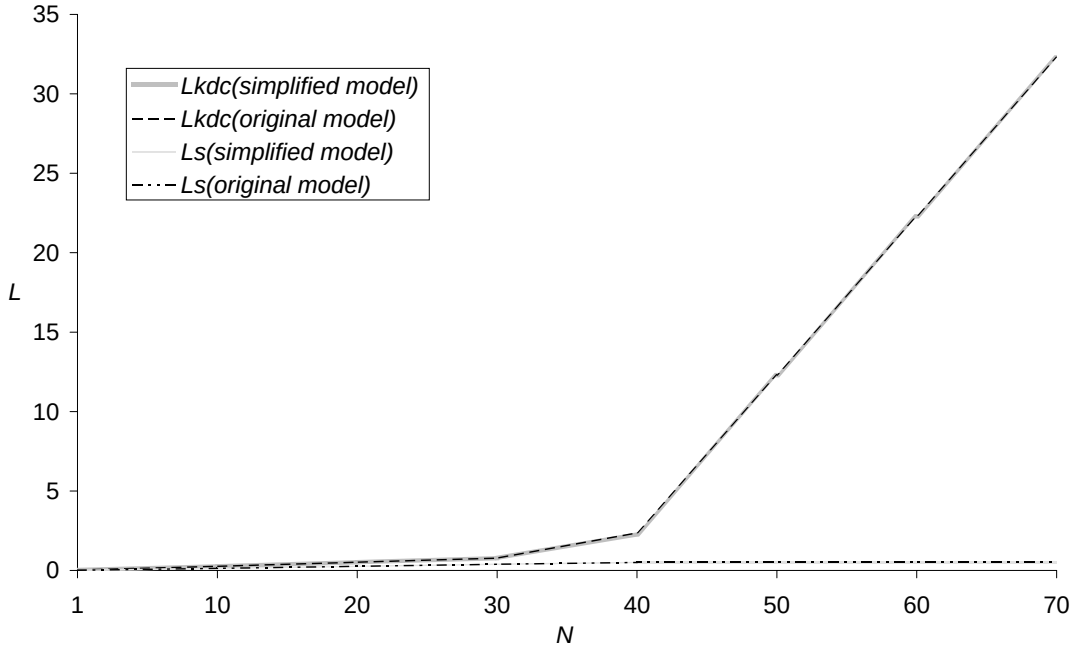

Figure 6.5: Average waiting customer for KDC and S varied with population size for two realms simplified model and original model calculated by ODEs, $r_{qtgt} = r_{ptgt} = r_{qsgtl} = r_{qtgtr} = r_{psgtl} = r_{qsl} = r_{psl} = r_{ptgtr} = 1, r_{qsgtrr} = r_{psgtr} = r_{qsrr} = r_{psr} = 0.5, r_w = 0.01, p = 0.6, K = 1, M = 2$

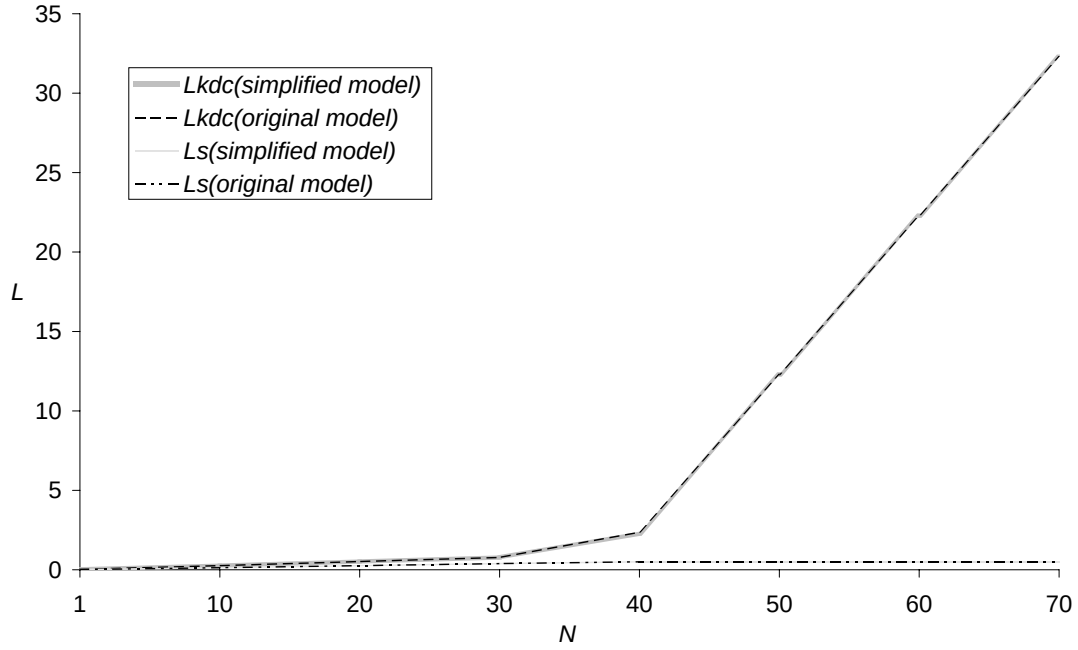

Figure 6.6: Average waiting customer for KDC and S varied with population size for three realms simplified model and original model calculated by ODEs, $r_{qtgt} = r_{ptgt} = r_{qsgtl} = r_{qtgtr} = r_{psgtl} = r_{qsl} = r_{psl} = r_{ptgtr} = 1, r_{qsgtrr} = r_{psgtr} = r_{qsrr} = r_{psr} = 0.5, r_w = 0.01, p = 0.6, K = 1, M = 3$

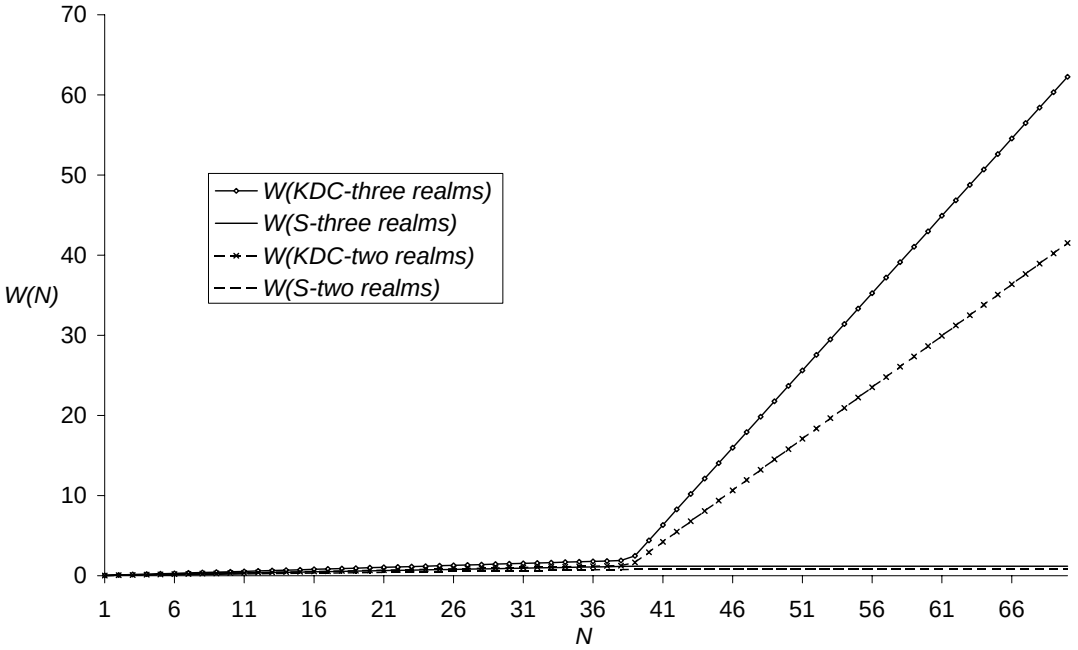

Figure 6.7: Average waiting time for customers at KDC and S varied with population size for three realms and two realms of simplified model calculated by ODEs, $r_{qtgt} = r_{ptgt} = r_{qsgtl} = r_{qtgtr} = r_{psgtl} = r_{qsl} = r_{psl} = r_{ptgtr} = 1, r_{qsgtrr} = r_{psgtr} = r_{qsrr} = r_{psr} = 0.5, r_w = 0.01, p = 0.6, K = 1$

simplified model are exactly the same as original model, and it needs less time to write the model down and less time to analyse that is considered efficient. Additionally, it is noticed that the queue length for each KDC and application server S for two realms is the same as those for three realms. This demonstrates that adding a extra realms does not affect the overall performance of each server. In this scenario, the remote requests will be shared amongst the remote realms. Once a realm is added, more clients are coming the system, however, more servers coming along too, and this could hold the balance between servers and consumers. Nevertheless, for the whole system (network), there are far more waiting clients in three realms than two realms. The situation is different for average waiting time that illustrated in Figure 6.7. In Figure 6.7, we compared the average waiting time for the customers waiting at KDC and S in case of three realms and two realms. Generally, more clients involved in the system leads each customer waiting longer, and the time for waiting at KDC is longer than waiting at S which caused by the queue length (see Figure 6.5 and 6.6). However, different from same queue length for each KDC and S for two and three realms, the average waiting time for two realms are not equal to three realms, and it is longer in three realms. The reason is that even the total queue length for each KDC does not change, proportion of remote requests increasing if additional realm is added. As we set that remote actions are slower then local actions, servers responds those additional new remote requests slower. Hence, it is increasing the average waiting time once adding additional realms.

6.7 Utility function

Once again, we apply the similar utility function to answer our proposed performance questions for Kerberos protocol.

$$C = c_1 L + c_2 r_p \ \ , \ c_1, c_2 \geq 0 \tag{6.1}$$

Slightly different from the cost functions in previous Chapters, there is no number of servers (K) here. This is because we intent to evaluate one individual realm in this case, and there is only one Kerberos server in each realm under the scenario. L denotes the average waiting customers at the non-repudiation server (TTP) in one realm, and r_p is the sum of the rates of local response (r_{ptgt}, r_{psgtl}, r_{ptgtr}) and remote response (r_{psgtr}). C_1 and C_2 are cost rates, and they many depend on the type of system or quality of service agreement with customers.

6.7.1 Numerical results

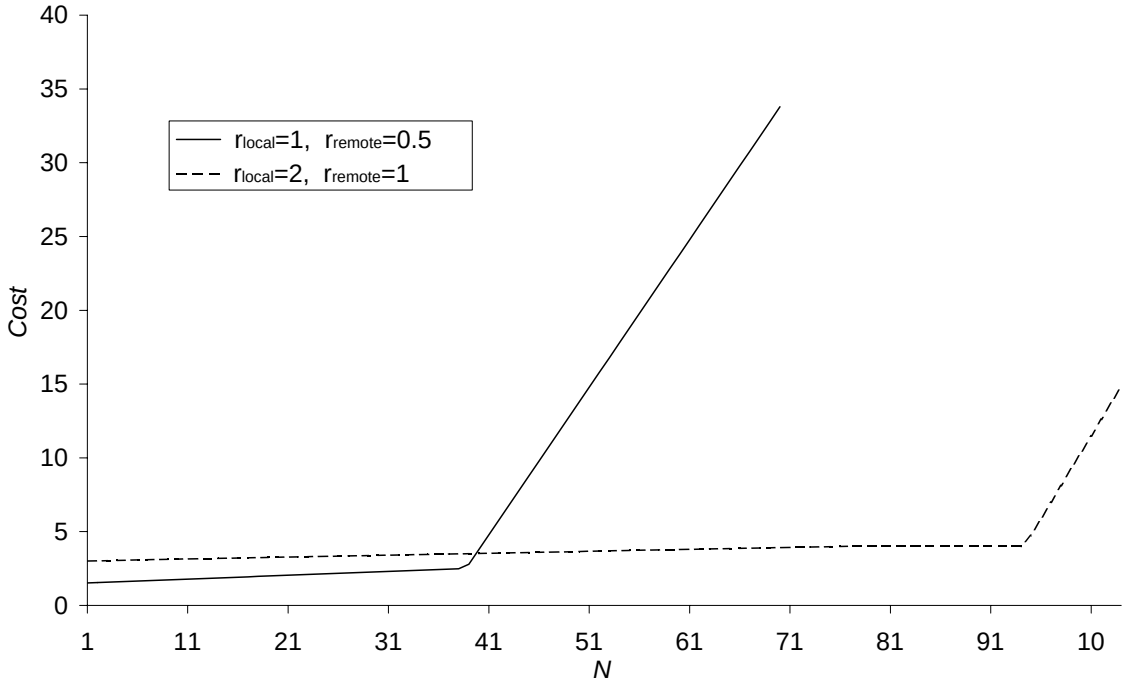

Figure 6.8: Cost varied against the number of clients calculated by ODEs, $p = 0.6$, $M = 3$, $c1 = c2 = 1$, $r_w = 0.01$, $r_{qtgt} = r_{qsgtl} = r_{qtgtr} = r_{qsl} = r_{psl} = 1$, $r_{qsgtrr} = r_{qsrr} = r_{psr} = 0.5$, $r_{local} \in \{r_{ptgt}, r_{psgtl}, r_{ptgtr}\}$, $r_{remote} = r_{psgtr}$

Figure 6.8 shows the cost varied against the number of clients calculated by ODEs. As we expected, the queue length of customer at Kerberos server increases if more clients come to the system. Consequently, the cost goes up under fixed service capacity. When the local rate is 1 and the remote rate is 0.5, the performance significantly degrades at the point where $N = 38$. If we double the rates for all local and remote responses, this point goes to around $N = 93$.

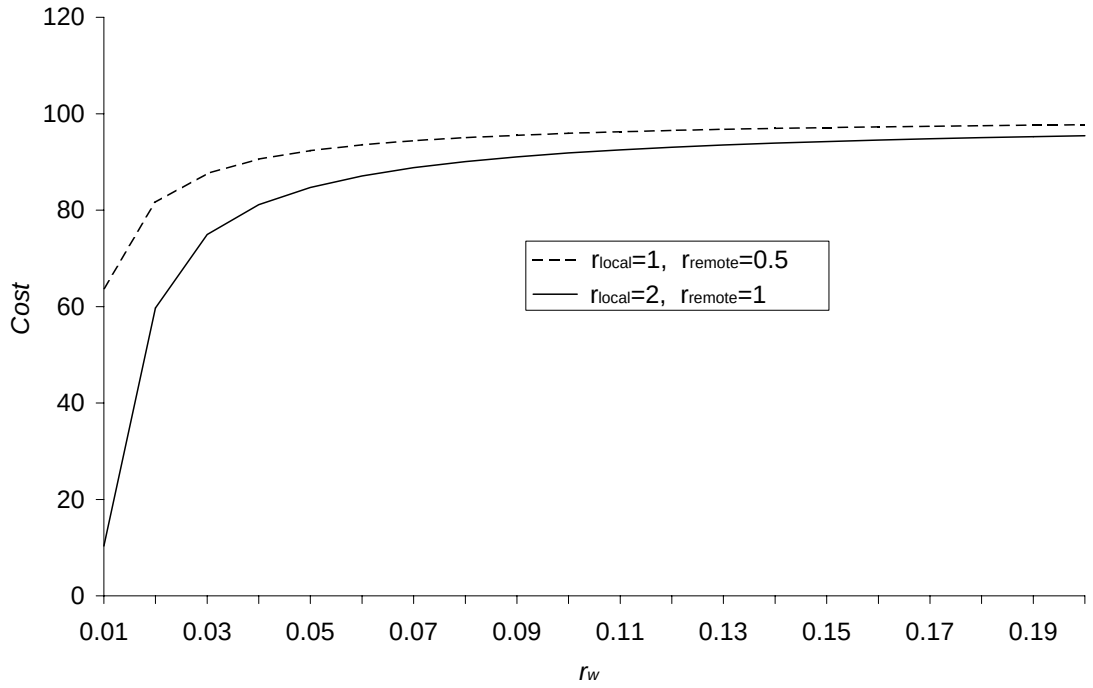

Figure 6.9: Cost varied with rate of *work* calculated by ODEs, $p = 0.6$, $M = 3$, $N = 100$, $c1 = c2 = 1$, $r_w = 0.01$, $r_{qtgt} = r_{qsgtl} = r_{qtgtr} = r_{qsl} = r_{psl} = 1$, $r_{qsgtrr} = r_{qsrr} = r_{psr} = 0.5$, $r_{local} \in \{r_{ptgt}, r_{psgtl}, r_{ptgtr}\}$, $r_{remote} = r_{psgtr}$

Figure 6.9 presents the cost varied with the rate of refresh key, r_w. Generally, frequently refreshing the session key leads more workload has been added to the system. Hence, cost increases with the number of waiting customers goes up. When we double the response rates, the cost is smaller. The reason is that the cost of customer waiting is reduced more than the increment of cost of service capacity. However, the two curves becomes very close when r_w is large, and this is because the difference of costs of customer waiting for two curves become smaller with a very large workload involved. Different from previous case studies, there is no obvious point here to make a balance between security and performance. Nevertheless, a security manager still could choose a heuristic balance point in particular applications.

6.8 Summary

In this chapter we modelled a version of the Kerberos protocol with cross realm requests. Even under an homogeneous assumption, the original model is cumbersome to formulate and suffers from commonly known state space explosion problem. We then simplified the original model by aggregating some relevant states, and it has been analysed by ODE approximation then verified by stochastic simulation. Through comparison of the aggregated model and the original model, this simplification is shown to obtain exact results for the overall performance of the system. Additionally, adding a realm to the system does not affect the performance of an individual server in this case. In this work, we assume there are K but only one type of S in each realms. It is possible to add more types of application server by naming it differently, however, this would obviously increase the complexity, with more extra behaviours added. Finally, we apply the techniques to a utility function to make a efficient capacity analysis to answer our proposed performance questions.

Through this case study, an aggregation technique can be added to the proposed work flow in simplification stage to decrease the difficulty of analysis when partial evaluation is not applicable.

Chapter 7

Conclusion and Further Work

7.1 Conclusions

In this thesis five security protocols have been modelled and analysed, and these protocols can be classified into different categories in order of complexity in terms of system behaviour. We have applied the analysis techniques that have been learned from former ones to latters. Firstly, in Chapter 3, we explored the performance of a model of a key distribution centre, as it is a simple and basic protocol for an initial case study. In the preliminary stage, we have proposed three modelling choices to discuss and selected one which is simple, but which captures the necessary behaviour. However, the chosen modelling method still suffers from the commonly known state space explosion problem. We therefore firstly implemented a simulation of the model, then an approximation is proposed to simplify system behaviour, and finally ODE analysis has been applied as a deterministic fluid flow analysis. The approximation shows good accuracy of prediction compared with simulation, scales exceptionally well and is fast to compute. ODE results have been compared with those derived from the approximation, unfortunately it is not always as accurate as our simplification and we have not been able to obtain all our desired metrics. On the other hand, the ODE approach does not suffer the same numerical problems as the approximation, it

is extremely efficient to solve and it is shown to be extremely accurate when the number of clients is huge. Finally, these techniques have been applied to a utility function to explore the capacity planning for a system associated with this protocol.

Then, two non-repudiation protocols, to be referred as $ZG1$ and $ZG3$, have been studied in Chapter 4. $ZG1$ has similar system behaviour to KDC in the previous chapter, therefore we can directly apply partial evaluation and the ODE approximation to solve it. However, the $ZG3$ model introduces an unintended race between different service actions for the non-repudiation server. This problem has been solved by specifying functional rates to the servers for each type of jobs. The $ZG3$ is able to be modelled and analysed through partial evaluation and ODE approximation. The ODE analysis for PEPA with functional rates can be either solved by coding in Matlab, or by translating to a $CMDL$ model and solved with the PEPA Eclipse plugin. Mean value analysis (MVA) has been firstly applied for PEPA models here, and compared with ODE analysis through numerical results. Both approaches enable systems with extremely large numbers of components to be solved efficiently. When the population size, N, is very large, there is almost no difference between the values obtained by either method, although the fluid approximation is slightly quicker. However, when N is small the fluid approximation diverges in the region around a known point, hence in such cases, mean value analysis is preferable.

In Chapter 5, we modelled an optimistic fair exchange protocol, which is another type of non-repudiation protocol. In contrast to $ZG1$ and $ZG3$ in the previous chapter, this optimistic non-repudiation protocol employs an off-line Third Trust Party (TTP) that is only being activated if a dispute occurs between involved clients. Therefore, two models are formulated: one for the case where there is no dispute and one for the case where the TTP is involved with resolving a dispute. Furthermore, we consider the Merchant server as consisting of several threads and each thread is partially evaluated with a client. Hence, the response action rates of the Merchant server becomes a functional rate, which depends on the active number of relevant clients. In the model of the extended protocol (TTP

involved), the version of the functional rates used in $ZG3$ is employed to avoid the race between different actions of the TTP server. Following the previous study, the ODE approximation is adopted as the most efficient analysis technique here, and the results are verified by stochastic simulation.

Finally, a cross-realm Kerberos protocol is investigated in Chapter 6. The original PEPA model not only suffers from the state space explosion problem with large populations and the large number of realms, but is also cumbersome to formulate it. Adding additional realms to the whole system leads more actions being specified. We therefore conduct a simplification process by aggregating relevant states from different realms to a new state in the whole network. By this simplification, one can easily modify parameters of population and number of realms rather than change the system behaviour in the original model. Again, according to our previous experience, ODE based fluid flow analysis has been applied, and the results have been verified by stochastic simulation. This simplified model is compared with original model for a small number of realms to show the accuracy of this aggregation. However, this simplification only works with steady state distribution and would not be suitable if we wished to analyse transient behaviour.

Through these case studies, we can enhance our proposed work flow as follow in Figure 7.1.

Now, we have several modelling choices in the modelling stage. In security protocol, one can model the clients as the same replicated component (first and third modelling choice in Chapter 3) or assign different name to it (second modelling choice in Chapter 3). For the server, it can be modelled as a single component (Chapter 3, 4, 6) or several threads (Chapter 5). However, the modelling form only depends on the application and scenarios. If the performance model is simple and small scaled, one can directly analyse it, otherwise, the model should be simplified. Partial evaluation is able to be employed to reduced the components of clients (Chapter 3, 4, 5), if the clients in security protocols are tightly coupled. Aggregation technique is also helpful to decrease the difficulty of analysis when

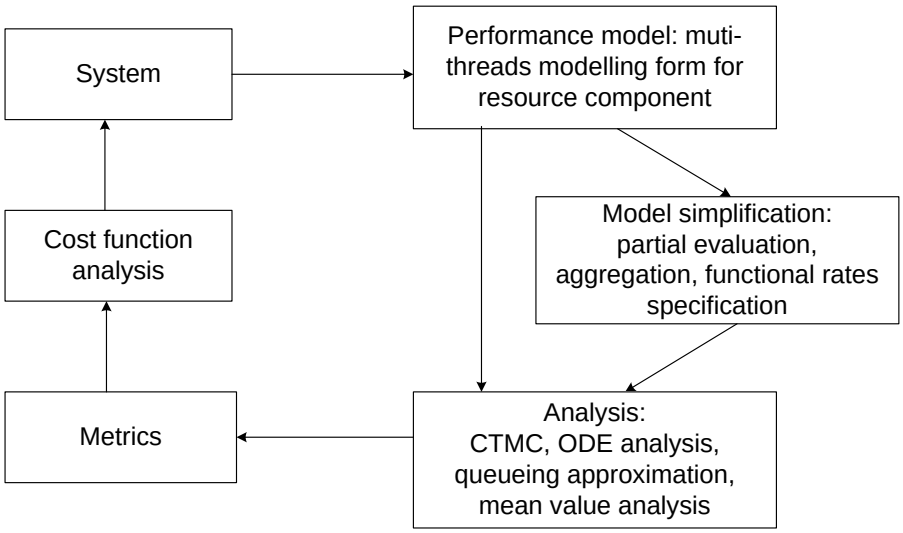

Figure 7.1: Enhanced work flow of performance analysis in context

partial evaluation is not applicable (Chapter 6). Furthermore, functional rates specification is alternative choice to avoid to write full detailed behaviour of the server when there is an unintended competition between different service actions (Chapter 4, 5, 6). If the simplified model is small enough, CTMC method can be applied (few clients in Chapter 3). In some cases (Chapter 3), the PEPA model can be transformed to a simple closed queueing network model, one only need to calculated the balance equation to solve the model. Another efficient analysis is mean value analysis, however, it only can be employed with our defined class of PEPA model (Chapter 4), though, this class will be extended in future. Finally, one can resort to ODE analysis to very large scale models (Chapter 3, 4, 5, 6). This is an approximation which only gives accurate results when system is very large. Therefore, it is a good complement to other exact solutions. After analysis, one can use the obtained metrics to the cost functions to analyse the performance of the security protocols or systems. This is able to be considered as a initial framework that can be followed for the further case studies.

7.2 Further work

There is a large scope for further work. This work can be classified into three categories: addressing the limitations of current work, applying other relevant analysis techniques to the five protocols in context, and exploring more types of security protocols. Though the works, we aim to further enhance our framework and generalise the performance evaluation techniques for security protocols.

Some limitations exist in this work. Firstly, we assume two clients are tightly coupled in the process of partial evaluation in Chapter 3 and Chapter 4. However, two clients might not have equal populations (e.g. 2 Alice with 3 Bob) in some cases, therefore partial evaluation can not be applied in the case without approximation. Then, we are not able to calculate average response time in the situation of multiple server and multiple jobs, as the order of jobs are not known. Finally, the whole system of Kerberos network is under the assumption of homogeneous realms in Chapter 7. Obviously, a heterogeneous Kerberos cross realm environment does not fit for our simplification process. All these limitations remain issues of further exploration.

There are many other analysis techniques that can be attempted for these four protocols. In this thesis we have focused exclusively on steady state performance, whereas in practice many important metrics are transient in nature. Therefore applying transient analysis techniques would extend the practical applicability of this work. As ODE presents a deterministic approximation, stochastic differential equations based fluid flow analysis [29] can be applied by introducing noise to ODE to approach the fact. Kronecker representation [36] provides a exact solution to efficiently solve Markov chain with less memory needed. Product form and semi-product form solution with Harrison's RACT [24, 27] also can be employed to investigate those protocols in this thesis. Bradley and Hayden's high order moment analysis [32] can be utilised to analysis more performance metrics. Additionally, performance measurement of prototype implementations is another analysis approach, by which the results are not only able to be verified, but also provide a path to validate the performance models.

As our long term objective is a general performance analysis approach to security protocols based on trade-off analysis between security and performance, we therefore need to explore a greater range of security protocols with respect to system behaviour. Two such protocols are proposed here. The first one is Netbill Security and Transaction protocol [15], which provides a layered request and response architecture. In this protocol, one of the components acts both as client and server. Furthermore, the Bull/Otway authentication protocol [8] aims at establishing fresh session keys between a number of participants and a server (one key for each pair of agents adjacent in the chain). To formulate its performance model, partial evaluation can not be applied and additional participants changes all system behaviour. Therefore, further scalable modelling and analysis approaches need to be explored.

References

[1] P. Argyroudis, R. Verma, H. Tewari and D. O'Mahony, Performance analysis of cryptographic protocols on handheld devices, in: *Proceedings of the Third IEEE International Symposium on Network Computing and Applications*, Massachusetts, 2004.

[2] J. Banks, J. Carson, B. Nelson and D. Nicol, *Discrete-event system simulation - fourth edition*, Pearson, 2005.

[3] C. Bodei, M. Buchholtz, M. Curti, P. Degano, F. Nielson, H. Nielson and C. Priami, Performance evaluation of security protocols specified in LySa, in: *Journal of Electronic Notes in Theoretical Computer Science (ENTCS)*, **112**, 2005.

[4] J. Bradley and W. Knottenbelt, The ipc/HYDRA Tool Chain for the Analysis of PEPA Models, in: *Proceedings of the 1st IEEE International Conference on the Quantitative Evaluation of Systems*, 2004.

[5] J. Bradley, S. Gilmore and J. Hillston, Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models, in: *Journal of Computer and System Sciences*, **74**(6), 2008.

[6] J. Bradley, S. Gilmore and N. Thomas, Performance analysis of Stochastic Process Algebra models using Stochastic Simulation, in: *Proceedings of 20th IEEE International Parallel and Distributed Processing Symposium*, IEEE Computer Society, 2006.

[7] M. Buchholtz, S. Gilmore, J. Hillston and F. Nielson, Securing statically-verified communications protocols against timing attacks, in: *Electronic Notes in Theoretical Computer Science*, **128**(4), Elsevier, 2005.

[8] J. Bull and D. Otway, The authentication protocol, APM Technical Report, Defence Research Agency, 1997.

[9] J. Cho, I. Chen and P. Feng, Performance analysis of dynamic group communication systems with intrusion detection integrated with batch rekeying in Mobile Ad Hoc Networks, in: *Proceddings of the 22nd International Conference on Advanced Information Networking and Applications*, Okinawa, Japan, IEEE, 2008.

[10] M. Chromiak and Z. Lojewshi, Stream security particularities in Java, *Journal of Annales UMCS, Informatica*, **8**(5-13), 2009.

[11] A. Clark, A. Duguid, S. Gilmore and M. Tribastone, Partial evaluation of PEPA models for fluid-flow analysis, in: *Computer Performance Engineering: Proceedings of the 5th European Workshop on Performance Engineering (EPEW)*, LNCS 5261, Springer-Verlag, 2008.

[12] G. Clark and W. Sanders, Implementing a stochastic process algebra within the Möbius Modeling framework, in: *Proceedings of the 9th International Workshop on Process Algebra and Performance Models*, Aachen, Germany, 2001.

[13] G. Clark and S. Gilmore and J. Hillston and N. Thomas, Experiences with the PEPA Performance Modelling Tools, in: *IEE Proceedings - Software*, pp: 11-19, **146**(1), 1999.

[14] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. Doyle, W. Sanders, and P. Webster. The Möbius Modeling Tool, in: *Proceedings of the 9th International Workshop on Petri Nets and Performance Models*, pp: 241-250, Aachen, Germany, 2001.

[15] B. Cox, J. Tygar, and M. Sirbu, NetBill security and transaction protocol, in: *Proceedings of the 1st conference on USENIX Workshop on Electronic Commerce*, Berkeley, CA, USA, 1995.

[16] S. Dick and N. Thomas, Performance analysis of PGP, in: *Proceedings of 22nd UK Performance Engineering Workshop (UKPEW)*, Bournemouth University, 2006.

[17] M. El-Hadidi, N. Hegazi and H. Aslan, Performance analysis of the Kerberos protocol in a distributed environment, in: *Proceedings of the 2nd IEEE Symposium on Computers and Communications (ISCC '97)*, Alexandria, Egypt, IEEE, 1997.

[18] M. El-Hadidi, N. Hegazi and H. Aslan, Performance evaluation of a new hybrid encryption protocol for authentication and key distribution, in: *International Symposium on Computers and Communications*, Egypt, IEEE, 1999.

[19] W. Freeman and E. Miller, An experimental analysis of cryptographic overhead in performance-critical systems, in: *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, IEEE Computer Society, 1999.

[20] D. Gillespie, Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry*, **81**(25), pp: 2340-2361, 1977.

[21] S. Gilmore, J. Hillston and M. Ribaudo, An Efficient Algorithm for Aggregating PEPA Models, *IEEE Transactions on Software Engineering*, **27**(5), 2001.

[22] P. Gutmann, Performance characteristic of application-level security protocols, draft paper at *http://www.cs.auckland.ac.nz/ pgut001/pubs/app_sec.pdf.*

[23] A. Harbitter and D. Menasce, A methodology for analyzing the performance of authentication protocols, *Journal of ACM Transactions on Information and System Security*, **5**(4), 2002.

[24] P.G. Harrison, Turning back time in Markovian process algebra, *Theoretical Computer Science*, **290**(3), 2003.

[25] P.G. Harrison, G-networks with Propagating Resets via RCAT, *ACM SIGMETRICS Performance Evaluation Review*, **31**(2), pp: 2-3, 2003.

[26] P.G. Harrison, Compositional reversed Markov processes, with applications to G-networks, *Performance Evaluation*, **57**(3), 2004.

[27] P. Harrison and N. Thomas, State-Dependent Rates and Semi-Product-Form via the Reversed Process. in: *Proceeding of the 7th European Performance Engineering Workshop*, Springer-Verlag, 2010.

[28] B. Haverkort, *Performance of Computer Communication Systems: A model Based Approach*, Wiley, 1998.

[29] R. Hayden, *Addressing the state space explosion problem for PEPA models through fluid-flow approximation*, Undergraduate thesis, Imperial College London, 2007.

[30] R. Hayden and J. Bradley, Fluid-flow solutions in PEPA to the state space explosion problem, in: *6th Workshop on Process Algebra and Stochastically Timed Activities*, Imperial College, London, 2007.

[31] R. Hayden and J. Bradley, ODE-based general moment approximations for PEPA, in: *7th Workshop on Process Algebra and Stochastically Timed Activities*, Edinburgh, 2008.

[32] R. Hayden and J. Bradley, A fluid analysis framework for a Markovian process algebra, in: *Theoretical Computer Science*, **411**(22-24), pp: 2260-2297, April, 2010.

[33] J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.

[34] J. Hillston, Fluid flow approximation of PEPA models, in: *Proceedings of 2nd International Conference on the Quantitative Evaluation of Systems (QEST)*, pp:33-43, IEEE Computer Society, 2005.

[35] J. Hillston and N. Thomas, Product form solution for a class of PEPA models, *Performance Evaluation*, **35**(3-4), 1999.

[36] J. Hillston and L.J. Kloul, An efficient Kronecker representation for PEPA models, in: *Proceeding of the first Joint PAPM-PROBMIV Workshop*, LNCS 2165, pp: 120-135, Aachen, Springer, 2001.

[37] J. Hillston and L.J. Kloul, From SAN to PEPA: A Technology Transfer, in: *Proceedings of the 1st Workshop on Process Algebras and Timed Activities (PASTA'02)*, Edinburgh, 2002.

[38] A. Hinton, M. Kwiatkowska, G. Norman and D. Parker, PRISM: A tool for automatic verification of probabilistic systems. in: *Proceeding of the 12th International Conference of Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 3920, pp: 441-444. Springer, 2006.

[39] W. Knottenbelt, *Parallel Performance Analysis of Large Markov Models*, PhD thesis, Department of Computing, Imperial College of Science, 1999.

[40] W. Knottenbelt and J. Bradley, Tackling Large State Spaces in Performance Modelling, in: *7th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*, Springer, 2007.

[41] M. Kwiatkowska, G. Norman and D. Parker, Stochastic Model Checking, in: *7th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*, Springer, 2007.

[42] C. Lamprecht, A. van Moorsel, P. Tomlinson and N. Thomas, Investigating the efficiency of cryptographic algorithms in online transactions, *International Journal of Simulation: Systems, Science & Technology*, **7**(2), pp: 63-75, 2006.

[43] S. Lavenberg and M. Reiser, Stationary state space probabilities at arrival instants for closed queueing networks with multiple types of customers, in: *Journal of Applied Probability*, **17**)(4), pp: 1048-1061, 1980.

[44] W. Liu, L. Yang, Q. Li, H. Dai and B. Hou, Performance analytic model for authentication mechanism, in: *International Conference on Networking, Sensing and Control*, pp: 1097 - 1102, Sanya, China, 2008.

[45] I. Mitrani, *Simulation techniques for discrete event systems*, Cambridge University Press, 1982.

[46] I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.

[47] A. Moralis, V. Pouli, M. Grammatikou, S. Papavassilious, and V. Maglaris, Performance comparison of web services security: Kerberos token profile against X.509 token profile, in: *Third International Conference on Networking and Services (ICNS)*, Athens, IEEE, 2007.

[48] R. Needham and M. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, **21**(12), 1978.

[49] N. O'Shea, Concerning Performance Driven Cryptographic Protocol Development, in: *8th Workshop on Process Algebgra and Stochastically Timed Activities*, University of Edinburgh, 2009.

[50] N. Potlapally, S. Ravi, A. Ragunathan, and N. Jha, Analyzing the energy consumption of security protocols, in: *International Symposium on Low Power Electronics and Design*, Seoul, Korea, ACM, 2003.

[51] I. Ray and I. Ray, An Optimistic Fair Exchange E-commerce Protocol with Automated Dispute Resolution, in: *Proceedings of the First International Conference on Electronic Commerce and Web Technologies*, LNCS **1875**, pp: 84-93, 2000.

[52] R. Ramsey, D. Orrell, and H. Bolouri, Dizzy: stochastic simulation of large-scale genetic regulatory networks. in: *Journal of Bioinf. Comp. Biol.*, **3**(2), pp: 415-436, 2005.

[53] M. Reiser and S. Lavenberg, Mean value analysis of closed multichain queueing networks, *Journal of the ACM*, **22**(4), pp: 313-322, 1980.

[54] O. Rodeh, K. Birman, M. Hayden, Z. Xiao and D. Dolev, The architecture and performance of security protocols in the ensemble group communication system, *Journal of ACM Transactions on Information and System Security*, **4**(3), 2001.

[55] K. Sevcik and I. Mitrani, The distribution of queueing network states at input and output instants, *Journal of the ACM*, **28**(2), pp: 358-371, 1981.

[56] X. Sha, X. Tan, N. Zhang and W. Liu, Performance analysis of multiple-access protocol ALOHA/PCD, in: *International Conference on Communication Technology Proceedings (ICCT)*, pp: 938-941, Beijing , China, 1996.

[57] J. Slegers, A Langevin interpretation of PEPA models, *Electronic Notes in Theoretical Computer Science*, **261**, 2010.

[58] M. Smith, Abstraction and Model Checking in the PEPA Plug-In for Eclipse, in: *7th International Conference on the Quantitative Evaluation of Systems*, pp: 155-156, Williamsburg, VA, USA, 2010.

[59] W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 1999.

[60] A. Stefanek, R. Hayden and J. Bradley, GPA - Tool for rapid analysis of very large scale PEPA models, in: *26th UK Performance Engineering Workshop*, University of Warwick, 2010.

[61] W. Stewart, *Introduction to the numerical solution of Markov chains*, Princeton University Press, 1994.

[62] W. Stewart, Performance Modelling and Markov Chains, in: *7th International School on Formal Methods for the Design of Computer, Communication and Software Systems (SFM)*, Springer, 2007.

[63] N. Thomas, Exploiting behavioural independence and control in Markovian process algebra, in: *1st Workshop on Process algebra with stochastic timed activities*, University of Edinburgh, 2002.

[64] N. Thomas, Performability of a secure electronic voting algorithm, *Journal of Electronic Notes in Theoretical Computer Science*, **128**(4), pp: 45-58, 2005.

[65] N. Thomas, Using ODEs from PEPA models to derive asymptotic solutions for a class of closed queueing networks, in: *8th Workshop on Process Algebra and Stochastically Timed Activities*, University of Edinburgh, 2009.

[66] N. Thomas, J. Bradley and D. Thornley, Approximate solution of PEPA models using component substitution, *IEE proceedings on Computers and Digital Techniques*, **150**(2), 2003.

[67] N. Thomas and Y. Zhao, Mean value analysis for a class of PEPA models. in:The Computer Jouneral, 2010; doi: 10.1093/comjnl/bxq064.

[68] M. Tribastone, The PEPA plug-in project, in: *Proceedings of 4th International Conference on the Quantitative Evaluation of Systems (QEST)*, pp: 53-54, IEEE Computer Society, 2007.

[69] Y. Wang, C. Lin and Q. Li, Performance analysis of email systems under three types of attacks, in: *Performance Evaluation*, **67**(6), 2010.

[70] M. Zhao, *Performance evaluation of distributed security protocols using discrete event simulation*, PhD thesis, Dartmouth College, Hanover, New Hampshire, 2005.

[71] J. Zhou and D. Gollmann, A Fair Non-repudiation Protocol, in: *Proceedings of IEEE Symposium on Security and Privacy (SP'96)*, IEEE Computer Society, 1996.

[72] J. Zhou and D. Gollmann, Observation on Non-repudiation, in: *Advances in Cryptology-ASIACRYPT'96*, LNCS 1163, pp: 133-144, Springer-Verlag, 1996.

[73] http://www.dcs.ed.ac.uk/pepa/tools/ipc/.

[74] http://eclipse.org.