Efficient solutions of a PEPA model of key distribution centre with a cost function

Y. Zhao, N. Thomas.

TECHNICAL REPORT SERIES

No. CS-TR-1112 July, 2008

TECHNICAL REPORT SERIES

No. CS-TR-1112 July, 2008

Efficient solutions of a PEPA model of key distribution centre with a cost function

Yishi Zhao and Nigel Thomas

Abstract

In this paper we explore the trade-off between security and performance in considering a model of a key distribution centre. The model is specified using the Markovian process algebra PEPA. The basic model suffers from the commonly encountered state space explosion problem, and so we apply some approximate techniques to solve it. First, model reduction techniques and approximation to give a form of the model(in fact, a closed queueing network model) which is more scalable. The approximated model is analysed numerically and results derived from the approximation are compared with a discrete event simulation. We, then, consider the use of a fluid flow approximation based on ordinary differential equations (ODEs) derived from a form of simplified model. The results derived from solving the ODEs are compared with previous closed queueing network approximation. Those results have been found are the same as the asymptotic bounds solution for the queueing network approximation which demonstrates that ODEs gives a alternative solvent of asymptotic bounds, only having been proved, in this circumstance. Base on those techniques above, we, finally, evaluate a cost function of this secure key exchange model. Three questions have been proposed; how many clients can a given KDC configure support? how much service capacity must we provide at a KDC to satisfy a given number of clients? and what is the maximum rate at which keys can be refreshed before the KDC performance begins to degrade in a given demand on a given system? Answers of these three questions are illustrated through numerical examples.

Bibliographical details

ZHAO, Y., THOMAS, N.

Efficient solutions of a PEPA model of key distribution centre with a cost function
[By] Y. Zhao, N. Thomas.

Newcastle upon Tyne: University of Newcastle upon Tyne: Computing Science, 2008.

(University of Newcastle upon Tyne, Computing Science, Technical Report Series, No. CS-TR-1112)

Added entries

UNIVERSITY OF NEWCASTLE UPON TYNE
Computing Science. Technical Report Series. CS-TR-1112

Abstract

In this paper we explore the trade-off between security and performance in considering a model of a key distribution centre. The model is specified using the Markovian process algebra PEPA. The basic model suffers from the commonly encountered state space explosion problem, and so we apply some approximate techniques to solve it. First, model reduction techniques and approximation to give a form of the model(in fact, a closed queueing network model) which is more scalable. The approximated model is analysed numerically and results derived from the approximation are compared with a discrete event simulation. We, then, consider the use of a fluid flow approximation based on ordinary differential equations (ODEs) derived from a form of simplified model. The results derived from solving the ODEs are compared with previous closed queueing network approximation. Those results have been found are the same as the asymptotic bounds solution for the queueing network approximation which demonstrates that ODEs gives a alternative solvent of asymptotic bounds, only having been proved, in this circumstance. Base on those techniques above, we, finally, evaluate a cost function of this secure key exchange model. Three questions have been proposed; how many clients can a given KDC configure support? how much service capacity must we provide at a KDC to satisfy a given number of clients? and what is the maximum rate at which keys can be refreshed before the KDC performance begins to degrade in a given demand on a given system? Answers of these three questions are illustrated through numerical examples.

About the author

Yishi Zhao is a PhD student within the School of Computing Science, Newcastle University.

Suggested keywords

SECURE KEY EXCHANGE,
STOCHASTIC PROCESS ALGEBRA,
SIMULATION,
APPROXIMATION,
ODE,
COST FUNCTION

Efficient solutions of a PEPA model of key distribution centre with a cost function

Yishi Zhao and Nigel Thomas

School of Computing Science, Newcastle University, UK
{Yishi.Zhao,Nigel.Thomas}@ncl.ac.uk

Abstract. In this paper we explore the trade-off between security and performance in considering a model of a key distribution centre. The model is specified using the Markovian process algebra PEPA. The basic model suffers from the commonly encountered state space explosion problem, and so we apply some approximate techniques to solve it. First, model reduction techniques and approximation to give a form of the model(in fact, a closed queueing network model) which is more scalable. The approximated model is analysed numerically and results derived from the approximation are compared with a discrete event simulation. We, then, consider the use of a fluid flow approximation based on ordinary differential equations (ODEs) derived from a form of simplified model. The results derived from solving the ODEs are compared with previous closed queueing network approximation. Those results have been found are the same as the asymptotic bounds solution for the queueing network approximation which demonstrates that ODEs gives a alternative solvent of asymptotic bounds, only having been proved, in this circumstance. Base on those techniques above, we, finally, evaluate a cost function of this secure key exchange model. Three questions have been proposed; how many clients can a given KDC configure support? how much service capacity must we provide at a KDC to satisfy a given number of clients? and what is the maximum rate at which keys can be refreshed before the KDC performance begins to degrade in a given demand on a given system? Answers of these three questions are illustrated through numerical examples.

1 Introduction

One of the more intriguing areas of performance engineering to emerge over recent years has been the study of the overhead introduced by making a system secure. It is clear that in order to add more functionality to a system that more execution time is required. 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 that 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. For this reason cryptographic protocols are one of the few areas of security to have received much attention from the performance community [1–3]. 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.

In this paper we tackle a different, but related, problem in the area of the performance - security trade-off, namely key exchange. Our initial inspiration for this work has been the study of the wide mouth frog protocol by Buchholz *et al* [4]. The authors used the stochastic process algebra PEPA to analyse timing properties of the protocol. Although their motivation was to investigate timing attacks, the models developed in [4] showed how authentication protocols can be modelled effectively in PEPA. In our previous work [5], three bisimilarity models have been modelled and analyzed numerically, but encountered state space explosion problem. The main assignment of this paper is exploring possible approaches to solve the problem and evaluating a cost function, base on those techniques, to better understand the behaviour of this system.

The paper is organised as follows. In the next section we introduce the system to be modelled, the key distribution centre (KDC). This is followed by a brief overview of the Markovian process algebra PEPA. Section 4 introduces the basic model of the KDC, followed by a simplified (equivalent) version and an approximation in Section 5. Some numerical results of approximation are presented in Section 6, including comparison of the approximation results with simulation. After that, OED analysis has been introduced in section 7, and section 8 shows its numerical results which compared with approximation and stochastic simulation. The cost model and its numerical results analysis are illustrated in section 9 and 10 respectively. Finally we draw some conclusions and discuss some avenues for future research.

2 Key Distribution Centre

We now describe the specific problem we seek to model. This is the secure exchange of secret keys (also known as symmetric keys) using a trusted third party known as a key distribution centre (KDC). The protocol is illustrated below, following the description in [6].

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

1. Alice sends request to KDC with nonce N_1
2. E$\{K_A\}\,[K_S|request|N_1|E\,\{K_B\}\,[K_S|ID_A]]$
   - K_S is a session key for Alice and Bob to use.
   - Alice can't decrypt the part encoded with Bob's key, she can only send it on.
3. E$\{K_B\}\,[K_S|ID_A]$
4. E$\{K_S\}\,[N_2]$
5. E$\{K_S\}\,[f(N_2)]$

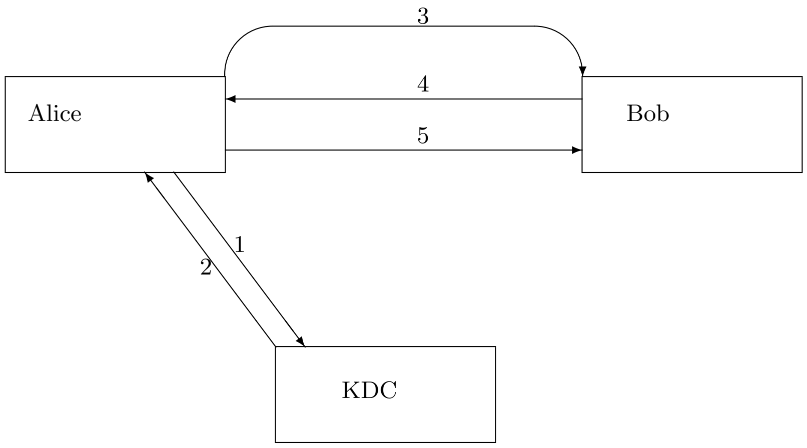

Fig. 1. Key Distribution Scenario.

where,

- N_1 and N_2 are nonces (random items of data),
- ID_A is a unique identifier for Alice,
- E$\{K_A\}$[X] denotes that the data X is encrypted using the key K_A, and
- f(N_2) denotes a predefined function applied to the nonce N_2, signifying that Alice has read the encrypted message sent by Bob.

The key features of this protocol are that only Alice can read the message sent by the KDC (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 (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).

3 PEPA

In this paper we model the performance of the key distribution centre using the Markovian process algebra PEPA. This approach has a number of advantages over a direct approach of 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 progressive 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.

A formal presentation of PEPA is given in [7], 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 $\top$.

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. $A \overset{def}{=} P$ gives the constant A the behaviour of the component P.

In this paper 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.

4 The Models

This scheme can be easily modelled for a single pair of clients in PEPA [7] as follows (*Model 1*).

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

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

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

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

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

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

$$System \stackrel{\text{def}}{=} KDC \underset{\mathcal{L}}{\bowtie} Alice \underset{\mathcal{K}}{\bowtie} Bob$$

Where, $\mathcal{L} = \{request, response\}$, $\mathcal{K} = \{sendBob, sendAlice, confirm, usekey\}$.

In Model 1, above, Alice's behaviour is separated into getting a session key (Alice), authentication with Bob ($Alice'$) and using the session key ($Alice''$). Similarly Bob's behaviour is separated into the key exchange and authentication with Alice (Bob) and the use of the session key (Bob'). In this model Alice only requests (and uses) one session key at a time. Thus the model is limited such that if Alice wishes to start a new session, she must first finish the previous session. This observation will be important when considering models with multiple clients.

According to Stallings [6]:

> *"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."*

In brief, this means there is a trade-off to be achieved between performance and security in the handling of session keys. In our model (above), this would be represented by varying the values of r_u and r_q. If these values are high then keys are being refreshed more regularly, putting more demand on the KDC and the network.

In our work we are primarily interested in studying the performance of the KDC, rather than the network. In [5] we developed three approaches to modelling multiple clients requesting session keys from the KDC. These approaches all formally represent the same protocol definition and are notionally equivalent at the syntactic level (they have a form of *bisimilarity*). However, they are not isomorphic and hence can give different values for important performance metrics. In the most intuitive version, presented in this paper, multiple clients are manually added using different names with parallel requests and responses allowed; meaning that the KDC can receive (and queue) several distinct requests before responding to them. A model with N pairs of clients is illustrated in Figure 2.

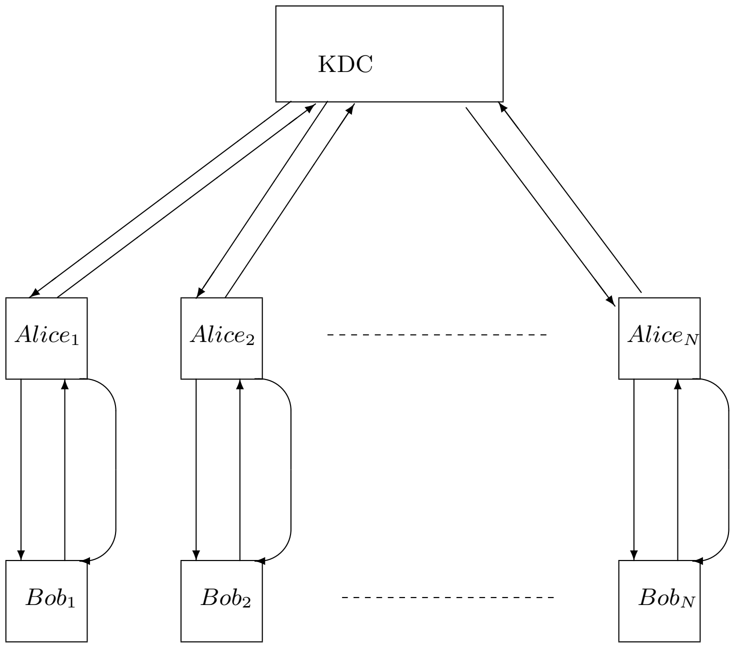

Fig. 2. Diagrammatic representation of a key distribution centre.

This approach can be modelled in PEPA as follows (*Model 2*).

$$\begin{aligned} KDC &\overset{def}{=} (request_1, \top).KDC_1 + (request_2, \top).KDC_2 \\ &\quad + \cdots + (request_N, \top).KDC_N \end{aligned}$$

$$\begin{aligned} KDC_1 &\overset{def}{=} (response_1, r_p).KDC + (request_2, \top).KDC_{N+1} \\ &\quad + \cdots + (request_N, \top).KDC_{2N-1} \end{aligned}$$

$$\cdots$$

$$\begin{aligned} KDC_{2^N} &\overset{def}{=} (response_1, r_p/N).KDC_{2^N-N} + (response_2, r_p/N).KDC_{2^N-N+1} \\ &\quad + \cdots + (response_N, r_p/N).KDC_{2^N-1} \end{aligned}$$

$$Alice_i \overset{def}{=} (request_i, r_q).(response_i, \top).(sendB_i, r_B).(sendA_i, \top).$$

$$(confirm_i, r_c).(usekey_i, r_u).Alice_i \ , \ 1 \leq i \leq N$$

$$Bob_i \overset{def}{=} (sendB_i, \top).(sendA_i, r_A).(confirm_i, \top).(usekey_i, \top).Bob_i \ , \ 1 \leq i \leq N$$

$$System \overset{def}{=} KDC \underset{\mathcal{K}}{\bowtie} ((Alice_1 \underset{\mathcal{L}_1}{\bowtie} Bob_1) || \cdots || (Alice_N \underset{\mathcal{L}_N}{\bowtie} Bob_N)$$

Where,

$$\mathcal{K} = \{request_1, response_1, \cdots, request_N, response_N\}$$

and

$$\mathcal{L}_i = \{sendB_i, sendA_i, confirm_i, usekey_i\}$$

In Model 2 we introduce 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.

Specific notation in Model 2 is introduced as follows. The subscript j in KDC_j corresponds to a binary representation of the request status of node i, such that the i^{th} bit is 1 if $Alice_i$ is awaiting a response from the KDC and 0 otherwise. The rate of each $response_i$ action in KDC_j is r_p divided by the number of $response_i$'s enabled.

It is worth observing here that Model 2 is cumbersome to specify; if we want to consider an extra client that no only needs to be specified as new $Alice_j$ and Bob_j components, but also the KDC component needs to be modified to incorporate the additional behaviours, $request_j$ and $response_j$.

5 Model Simplification and Approximation

Model 2 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 matrices with over 60 million elements (although admittedly mostly zeros). Even the best distributed Markov chain solvers generally only tackle state spaces of a few million states at most. To counter this, and to make the model easier to specify and understand, we have applied some simplification techniques to derive a form of the model which gives the same results for 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 3*).

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

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

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

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

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

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

$$System \overset{def}{=} KDC \underset{\mathcal{L}}{\bowtie} \left(Alice \underset{\mathcal{K}}{\bowtie} Bob || \ldots || Alice \underset{\mathcal{K}}{\bowtie} Bob\right)$$

Where, $\mathcal{L} = \{request, response\}$, $\mathcal{K} = \{sendBob, sendAlice, confirm, usekey\}$.

Clearly the component *Bob* is almost redundant, and the sharing for the action *request* and its enabling in *KDC* has no effect on the behaviour of the model. Hence an even simpler equivalent specification would be (*Model 4*):

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

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

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

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

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

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 [7] introduced the *weak isomorphism* equivalence for exactly this purpose). Thus we obtain the following simple form of the model (*Model 5*).

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

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

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

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 5 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 [8] pages 195-197).

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

and

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

6 Numerical Results of Approximation

The 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 [9] 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.

In Figure 3 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 4 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 5 and 6, 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.

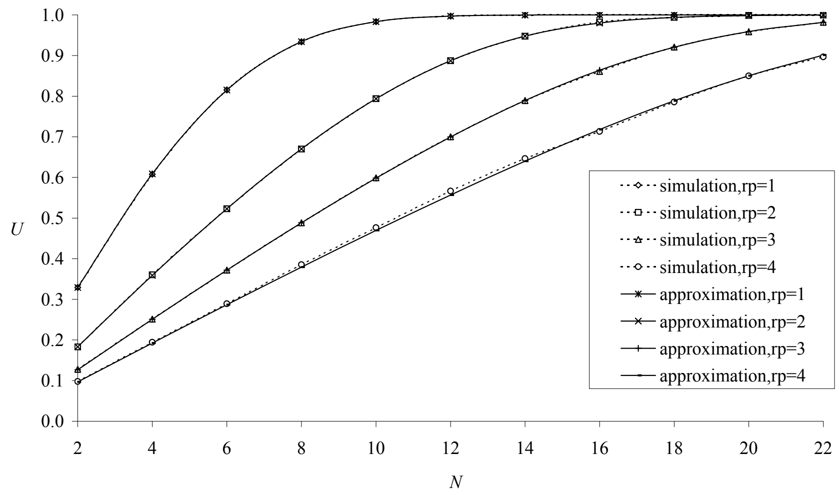

Fig. 3. Average utilisation varied against the number of client pairs. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

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

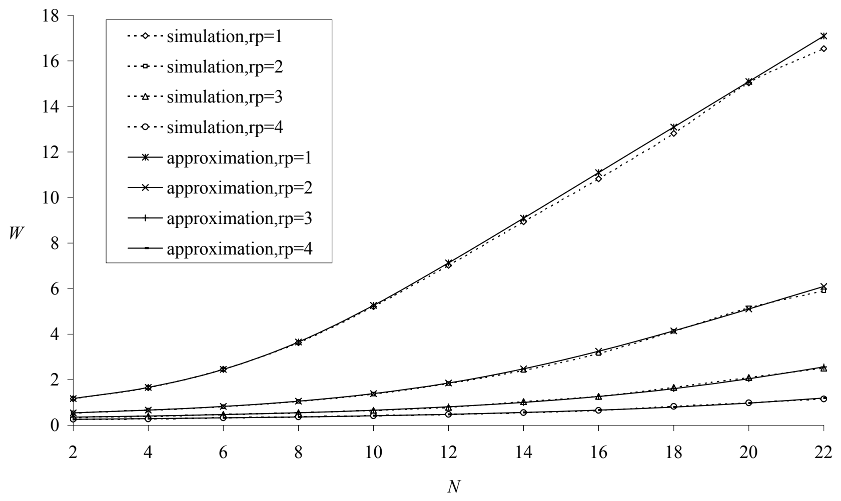

Fig. 4. Average response time varied against the number of client pairs. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

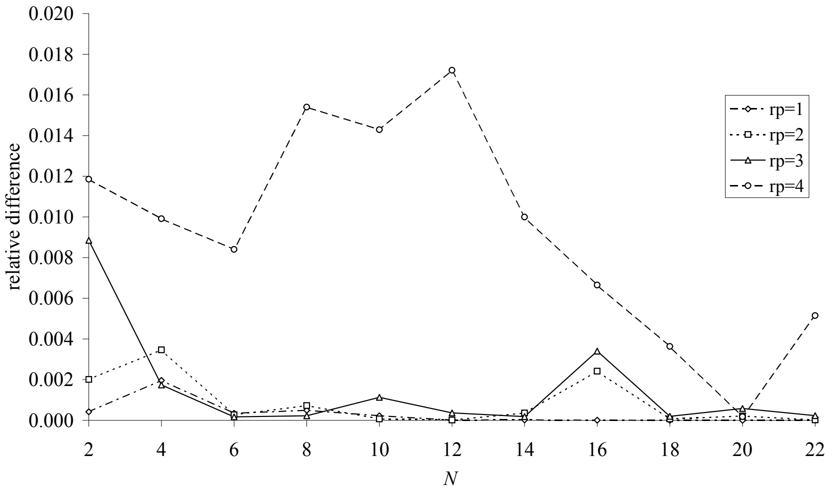

Fig. 5. Relative error in utilisation of approximation compared to simulation. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

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

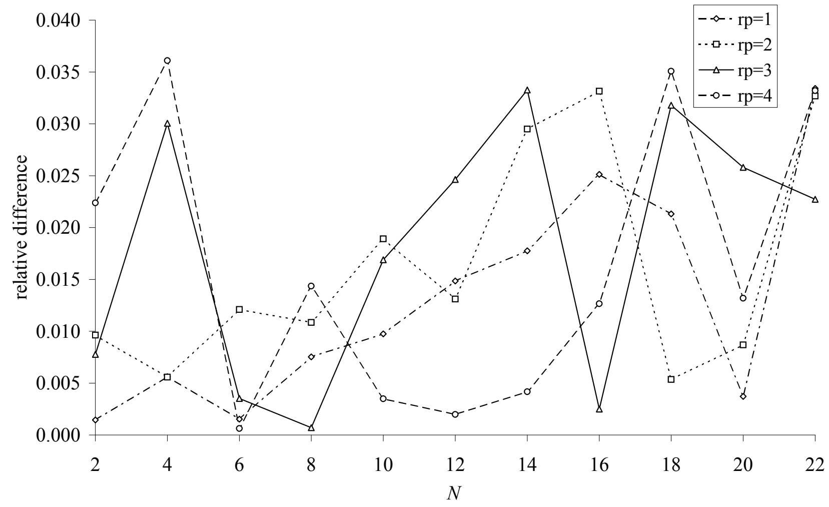

Fig. 6. Relative error in average response time of approximation compared to simulation. $r_u = 1.1$, $r_A = r_B = r_c = r_q = 1$.

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 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 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. Note that although in theory the approximation scales very well, in practise there can be numerical problems relating to the representation and manipulation of large factorials, hence in this instance we have restricted the experimentation to $N = 150$ ($150! \approx 5.7 * 10^{262}$)

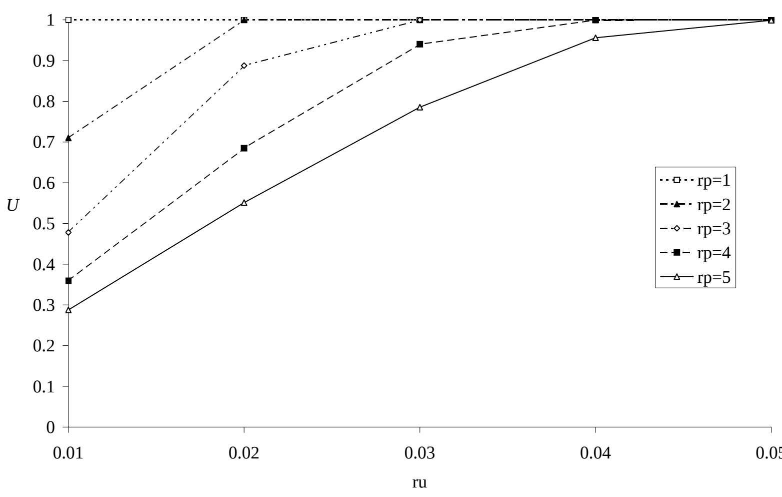

Fig. 7. Average utilisation varied against the rate of session key use, r_u. $r_q = r_A = r_B = r_c = 1$, $N = 150$

Figures 7 and 8 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 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.

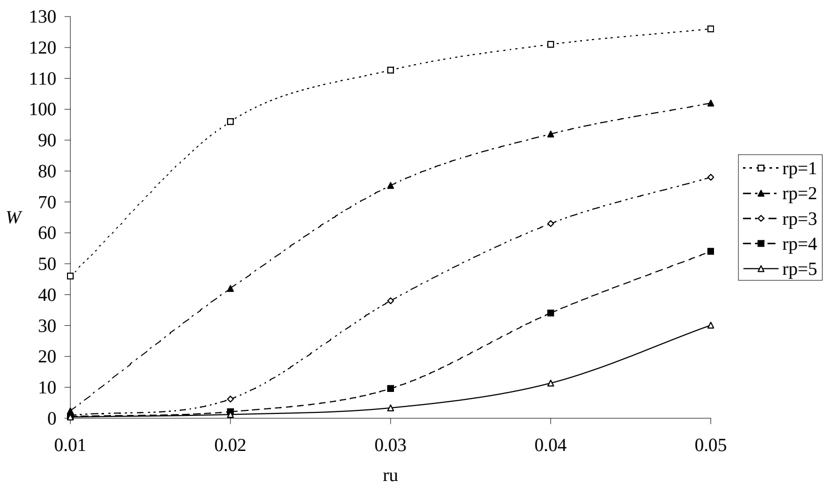

Fig. 8. Average Response time varied against the rate of session key use, r_u. $r_q = r_A = r_B = r_c = 1$, $N = 150$.

7 ODE analysis

Thus far we have considered a traditional approach to modelling and analysis. The approximation shows good accurarcy of prediction compared with a discrete event simulation, scales well and is fast to compute. Nevertheless, it is frustrated by large factorial computation. In this section we consider an alternative approach proposed by Hillston [10], based on the solution of ordinary differential equations (ODEs). In this style of model 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.

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

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

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

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

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.

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

There are a number of approaches to solving this set of ODEs. For simplicity we have simulated 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, $\delta t \leq 1/(r_{max}N)$, where $r_{max} = max(r_q, r_p, r_B, r_A, r_c, r_u)$.

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 [8] 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).

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

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.

8 Numerical results of ODEs

Figure 9 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 clearly due to the fact that the flow out of $Alice_1$ is much greater when $r_p = 4$.

Figure 10 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 6.

We expect the ODE method to be accurate when N is large. Figure 10 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 11, where the evolution of the ODEs is compared with the stochastic simulation of the PEPA model [11] derived directly using the PEPA Eclipse Plug-in. When N is sufficiently far from the gradient change there is good agreement between the ODE solution and the stochastic simulation. However, at $N = 6$ the divergence is significant; 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 [12] pages 245-246 for example).

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

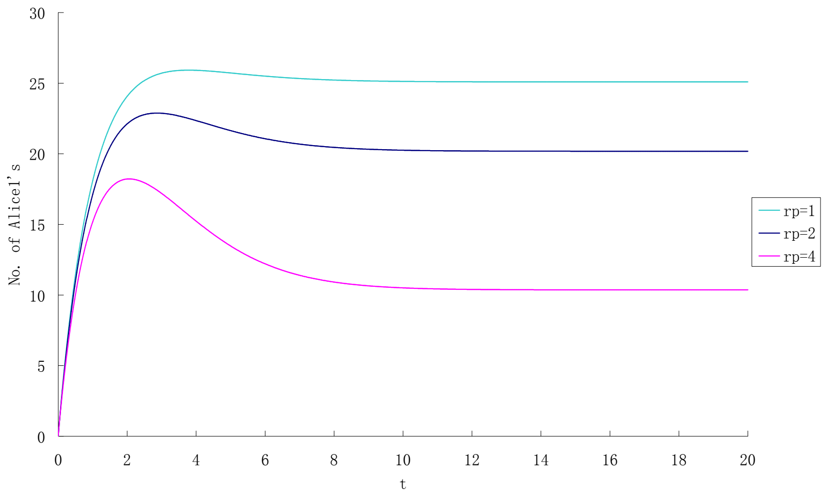

Fig. 9. Number of waiting clients over time, $N = 30$, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

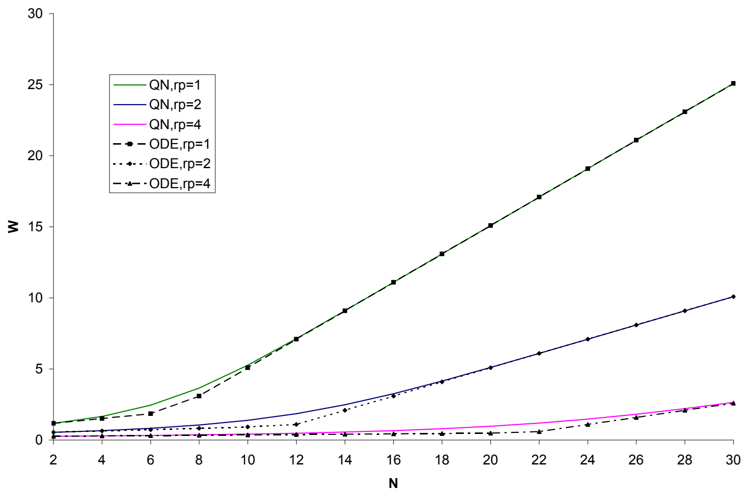

Fig. 10. Average response time calculated by the ODE method and QN approximation, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

Below N^* the asymptotic bound is given as

$$L(N) = \frac{N r_x}{r_x + r_p} \tag{2}$$

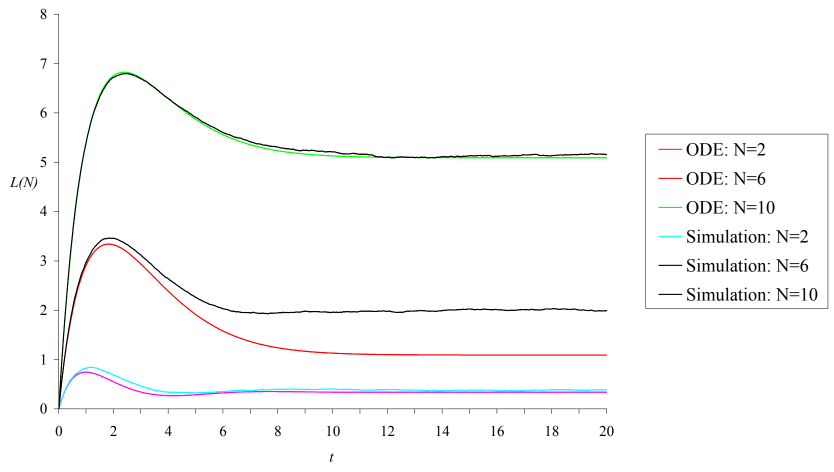

Fig. 11. Number of waiting clients over time, $r_p = r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

Above N^* the asymptotic bound is given as

$$L(N) = \frac{Nr_x - Kr_p}{r_x} \tag{3}$$

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. 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.

8.1 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 a multi-clients issue in PEPA model should be specified 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 a passive action would be subject to the *apparent rate* in PEPA. Hence, K KDCs and 1 Alice would give rise to *response* occuring 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,

$$(N-i)r_x\Pi_i = (i+1)r_p\Pi_{i+1} \;, 0 \le i < K$$
$$(N-i)r_x\Pi_i = Kr_p\Pi_{i+1} \;, K \le i < N$$

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 12 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 $N \ge 25$) 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.

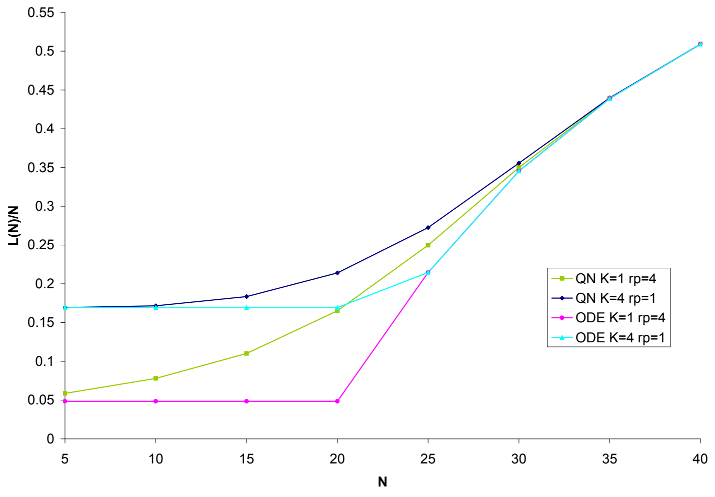

Fig. 12. Proportion of $Alice_1$ components, calculated by the ODE method and QN approximation, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

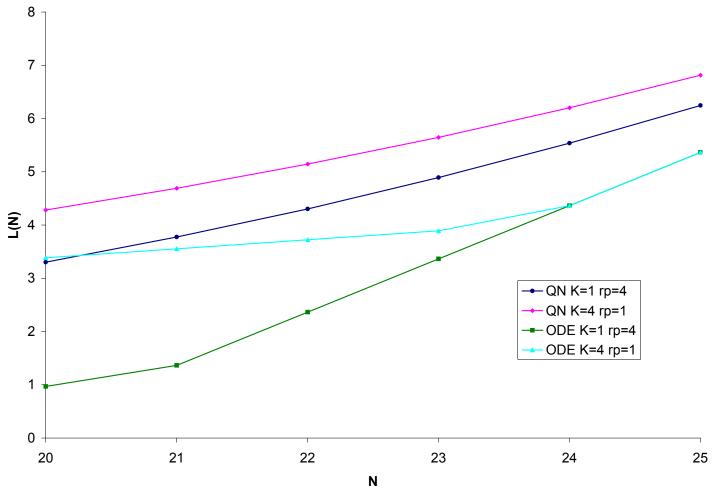

Fig. 13. Average queue length calculated by the ODE method and QN approximation, $r_q = r_B = r_A = r_c = 1$ and $r_u = 1.1$

There is a clear divergence between the ODE and QN results around the change in gradient as we have already observed in Figure 10. Figure 13 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 QN solutions at this point, near to N^*.

9 The cost model

We now wish to explore the question of capacity planning for a system involving a key distribution centre based on those techniques which introduced above. In doing this there are two closely related questions we wish to address. 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 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. We have introduced a number of performance measures already, 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 cost function to be optimised. This function is based on the assumption that there is a cost in keeping customers waiting (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 cost function.

$$C = c_1 L + c_2 K r_p \ \ , \ c_1, c_2 \geq 0 \tag{4}$$

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.

Taking the asymptotic results (2) and (3) for L gives a simple exposition of this cost function. If $N < N^*$ then

$$C = \frac{c_1 N r_x}{r_x + r_p} + c_2 K r_p \tag{5}$$

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

$$C = c_1 \frac{N r_x - K r_p}{r_x} + c_2 K r_p \tag{6}$$

If we seek to find the largest value of N before the performance begins to significantly degrade, then (5) and (6) 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 6 is increasing with K if $c_1 < r_x c_2$.

Now, recall that (1) 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, (5) and (6) 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 $c_1 > r_x c_2$.

10 Numerical results of Cost Model

We now illustrate the scenarios described above through numerical examples. Figures 14 and 15 show the cost varied against the number of clients, calculated by queueing network model equations and asymptotic bounds respectively. Clearly, under these parameter values with $r_p = 1$, the cost raises rapidly at around 80 clients, which is the approximate maximum capability that the KDC can handle before performance starts to degrade. In a small system ($N < 60$), the cost is greater for a faster KDC. The reason for this is that some servers will be idle when the queue size is small enough, 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 costly, and so we adopt the use of the asymptotic bound. Figure 15 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. Thus, doubling the service rate from $r_p = 2$ to $r_p = 4$ effectively doubles the capacity of the system in this case.

We now start to seek the optimal value of K to minimize the cost of system. According to the analysis in last 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 asymptotic bound as the more efficient approach in optimisation, because 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 16 shows the results of cost varied with number of KDCs. Generally, more clients need more KDCs to support them, in order to reduce the optimal

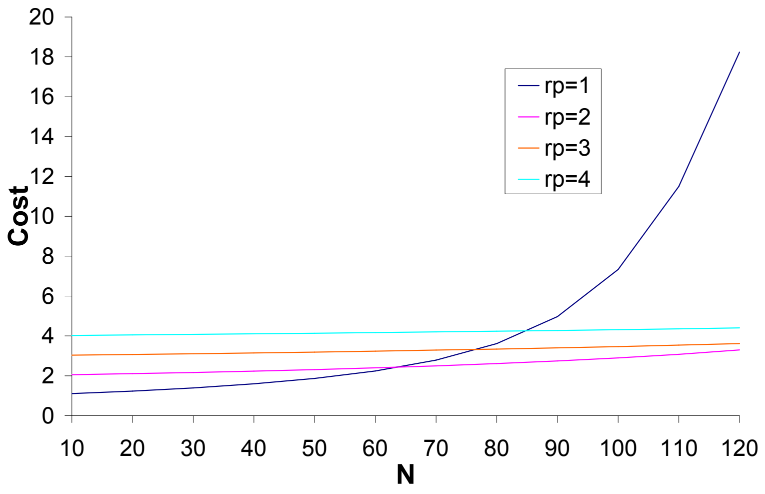

Fig. 14. Cost varied against the number of clients calculated by queueing network model, $K = 1, c1 = c2 = 1, r_q = r_A = r_B = r_c = 1, r_u = 0.01$

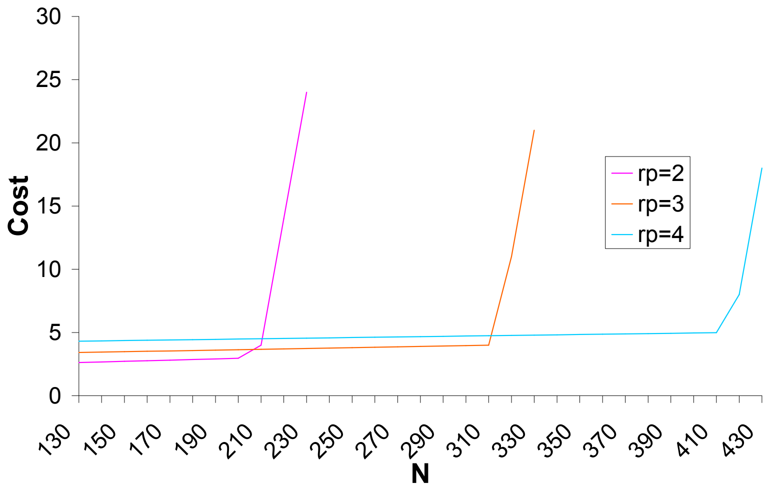

Fig. 15. Cost varied against the number of clients calculated by asymptotic bounds, $K = 1, c1 = c2 = 1, r_q = r_A = r_B = r_c = 1, r_u = 0.01$

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 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 exact queueing network model to show how the true value of K_{opt} can vary from K^*. Figure 17 compares the cost around $K*$ in case of 1000 clients with $c_2 = 0.1$ and 10 calculated by asymptotic bounds and queueing network model. In the case of $c_2 = 10$, K_{opt} is 10 which is the same as asymptotic 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 then case of $K = 14$ and 16 with value 11.184 and 11.178, respectively. The cost at K^* is calculated by asymptotic bounds 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 be use to calculate a near optimal cost extremely efficiently.

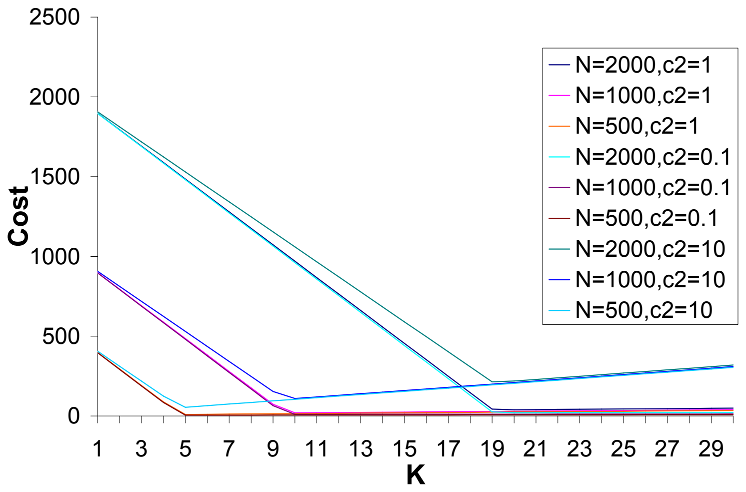

Fig. 16. Cost varied with number of KDCs calculated by asymptotic bounds, $c1 = 1, r_q = r_p = r_A = r_B = r_c = 1.0, r_u = 0.01$

Figure 18 shows the cost varied with the rate of key being used calculated by asymptotic bounds which answered the proposed third question. In this experiment, number of servers and clients have been fixed. Generally, large r_u results more cost which caused by more frequently refresh the key, more workload has been added in KDC servers, consequently, more waiting jobs. According to Stallings [6]: although cost increased, the system becomes more secure as the eavesdropper has less cipher text to crack any given session key. As a security manager must try to make a balance between security and performance. Apparently, the balance point here is the value of r_u for cost starting to increase rapidly. It is not difficult to find $r_u = 0.01$ is the anticipated answer for all three cases($c_2 = 0.1, 1$ and 10). This is not a optimum selection, therefore, the

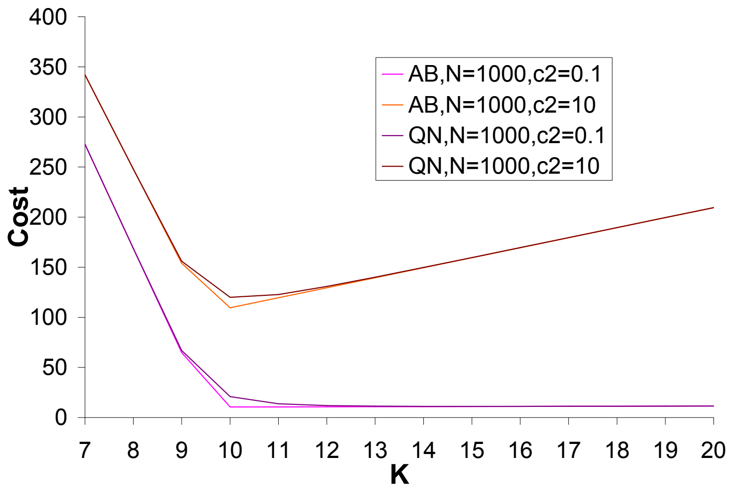

Fig. 17. Cost varied with number of KDCs calculated by asymptotic bounds and QN approximation, $c1 = 1, r_q = r_p = r_A = r_B = r_c = 1.0, r_u = 0.01$

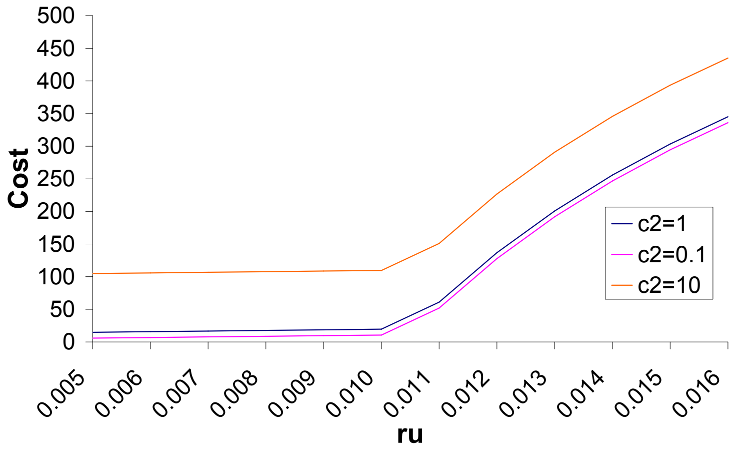

Fig. 18. Cost varied with rate of usekey calculated by asymptotic bounds, $c1 = 1, r_q = r_p = r_A = r_B = r_c = 1.0, N = 1000, K = 10$

accurate results are not quite necessary by a complex calculation. In practical circumstance, the security manager could choose a value for r_u around 0.01 depending on different case.

11 Conclusion and Further Work

In this paper 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 compared with simulation, scales exceptionally well and is fast to compute. ODE results have been compared with those derived from the approximation. This approach as 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 extremely efficient to solve and is shown to be extremely accurate when the number of clients is large. By using the asymptotic results, it is possible to compute the metrics of interest extremely efficiently. Those techniques has been applied to a cost model to explore the capacity planning for a system associated with this protocol. Three proposed questions have been answered through numerical results which has been acquired by a very efficient approach of techniques combination.

This work is part of an on going investigation into performance modelling of secure systems. The next step is to apply these modelling techniques to a class of non-repudiation protocols.

Acknowledgements

The authors are indebted to A. Clark, A. Duguid, S. Gilmore and M. Tribastone of the University of Edinburgh for invaluable comments of an earlier draft of this paper, in particular for clarifying aspects of the PEPA Eclipse Plug-in and the apparent rate.

The work described in this technical report was previously reported in [13–15].

References

1. S. Dick and N. Thomas, *Performance analysis of PGP*, in: F. Ball (ed.) Proceedings of 22nd UK Performance Engineering Workshop (UKPEW), Bournemouth University, 2006.
2. W. Freeman and E. Miller, *"An Experimental Analysis of Cryptographic Overhead in Performance-critical Systems"*, Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS),IEEE Computer Society, 1999.

3. 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.
4. M. Buchholtz, S. Gilmore, J. Hillston and F. Nielson, Securing statically-verified communications protocols against timing attacks, *Electronic Notes in Theoretical Computer Science*, 128(4), Elsevier, 2005.
5. Y. Zhao and N. Thomas, Modelling secure secret key exchange using stochastic process algebra, in: *Proceedings of 23rd UK Performance Engineering Workshop*, Edge Hill University, 2007.
6. W. Stallings, *Cryptography and Network Security: Principles and Practice*, Prentice Hall, 1999.
7. J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.
8. I. Mitrani, *Probabilistic Modelling*, Cambridge University Press, 1998.
9. G. Clark and S. Gilmore and J. Hillston and N. Thomas, *Experiences with the PEPA Performance Modelling Tools*, IEE Proceedings - Software, pp. 11-19, 146(1), 1999.
10. J. Hillston, Fluid flow approximation of PEPA models, in: *Proceedings of QEST'05*, pp. 33-43, IEEE Computer Society, 2005.
11. 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.
12. B. Haverkort, *Performance of Computer Communication Systems: A model Based Approach*, Wiley, 1998.
13. Y. Zhao and N. Thomas, Approximate solution of a PEPA model of a key distribution centre, in: *Performance Evaluation - Metrics, Models and Benchmarks: SPEC International Performance Evaluation Workshop*, LNCS 5119, Springer Verlag, 2008.
14. N. Thomas and Y. Zhao, Fluid flow analysis of a model of a secure key distribution centre, in: *Proceedings 24th Annual UK Performance Engineering Workshop*, Imperial College London, 2008.
15. Y. Zhao and N. Thomas, A cost model analysis of a secure key distribution centre, in: *Proceedings of International Symposium on Trusted Computing*, IEEE Computer Society, 2008.