

Performance modelling and analysis of systems under attack and misbehaviour



Ohud Mohammed Almutairi

School of Computing
Newcastle University

This dissertation is submitted for the degree of
Doctor of Philosophy

June 2022

I would like to dedicate this thesis to my great parents, my beloved husband, my lovely siblings and my gorgeous children.

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification at Newcastle University, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Ohud Mohammed Almutairi

June 2022

Acknowledgements

I would like to express my sincere gratitude and appreciation to my supervisor, Dr. Nigel Thomas, for his continuous support and guidance throughout my research journey. I am also thankful to the School of Computing for giving me this unique learning opportunity. Additionally, I would like to thank all of the people and staff at the school who assisted me at all stages of the research.

Most importantly, without the support of my family, this research would not have been completed. I am deeply grateful to my parents, Mr. Mohammed and Mrs. Wadha, whose constant love and support keep me motivated and confident throughout the entire PhD study and every day. Further, I would like to express my heartfelt gratitude to my husband, Dr. Abdullah Almuttiri, who has been completely understanding and supportive throughout this long journey. Without his love and patience, I would not be where I am now. My heartfelt thanks also go to Miral, Mohammed, and Raneem, my gorgeous children, for their love and being with me through this journey. Finally, I would like to extend my deepest sense of gratitude towards my lovely siblings and friends for their continuous encouragement and support. From the bottom of my heart, I would like to say a big thank you to all of you.

Abstract

Computing systems are growing increasingly complex, incorporating multiple interactive components. Performance is a critical attribute in evaluating computing systems. Most computing systems are now connected to networks, either private or public, raising concerns about vulnerability and exposure to threats and attacks. A secure system requires effective security protocols and techniques that do not negatively compromise performance. Analysing a system's behaviour under attack and misbehaviour can assist in determining where a problem is located so as to direct additional resources appropriately. The overall aim of this thesis is to model the performance of secure systems where behaviour changes in response to attacks and misbehaviour. Performance Evaluation Process Algebra (PEPA) modelling is employed to convert formal security protocols and methods into formal performance models.

This thesis addresses the impact and cost of cyber-attacks on the performance of web-based sales systems. PEPA models are proposed for two scenarios, with and without the attacks, to understand how the system behaves in different scenarios to provide a sustainable level of performance. It also explores the performance cost of a security protocol, an anonymous and failure resilient fair-exchange e-commerce protocol. The proposed PEPA models were formulated with and without anonymity in order to explore its overhead. Additionally, we modelled a basic protocol with no misbehaviour, not requiring the active involvement of a Trusted Third Party (TTP), and an extended protocol, for which the TTP's participation is essential to resolve disputes. These models provide an insight into the protocol's behaviour and the associated performance cost.

An attack graph is a popular method to support a defender in understanding an attacker's behaviour. It also supports the defender in detecting possible threats, thereby improving a system's security status. Developing a PEPA model version of an attack graph can advance understanding and identification of key risks, and assist the defender with implementing appropriate countermeasures. This thesis developed two methods to automate the generation of the PEPA model based on a pre-existing attack graph specification. The first method is simple, generating a single sequential component to represent both a system and an attacker. The second method has more potential, by generating a PEPA model with two sequential components representing a system and an attacker, as well as the system equation to define

how they interact. The attacker component enables us to explicitly incorporate attacker skills into the model. We use case studies to demonstrate how the PEPA models generated are used to perform path analysis and sensitivity analysis, as well as estimate the time required for each path. The defender can use this to determine the amount of safe time remaining before the system is compromised, and rank the risk from all attack paths. In addition, we developed PEPA models for an attack graph considering two criteria: attacker expertise and the availability of exploit code to estimate time needed to breach the system. We proposed three attacker skill levels: beginner, intermediate, and expert. The adaptability of our proposed PEPA models were improved by incorporating learning behaviours for both attacker and defender, to demonstrate how this affects the time required to compromise the system.

The models in this thesis demonstrate an approach to integrating security and performance concerns to advance understanding of system and attacker behaviour. The performance analysis undertaken indicates where problems may arise and additional resources needed. This analysis could be extended in the future to consider alternative design options and dynamic reconfiguration. Understanding the impact of attackers on system behaviour increases our ability to design systems that can adapt and tolerate attacks. This thesis represents an initial step toward greater understanding of the impact of attacks on system performance.

Table of contents

List of figures	xv
List of tables	xxiii
Nomenclature	xxv
1 Introduction	1
1.1 Motivation	1
1.2 Aim and Objectives	3
1.3 Contributions	3
1.4 Thesis outline	4
1.5 Publications	6
2 Background	7
2.1 The Security and Performance Trade-off	7
2.2 Performance modelling methods	8
2.3 Performance Evaluation Process Algebra (PEPA)	9
2.3.1 PEPA syntax	10
2.3.2 Continuous Time Markov Chain (CTMC)	12
2.3.3 Fluid semantics for PEPA	13
2.3.4 PEPA Eclipse plug-in	14
2.3.5 Performance metrics	15
2.3.6 Alternative approaches for performance evaluation	17
2.4 Related work	18
2.4.1 Classical approaches to security protocol evaluation	18
2.4.2 Performance modelling of secure protocols	20
2.4.3 Performance modelling of systems under an attack	24
2.4.4 Evaluation of attacker behaviour	25
2.4.5 Stochastic models of attacker behaviour	29

2.5	Context of this thesis	32
2.6	Chapter summary	34
3	Performance modelling of the impact of cyber-attacks on a web-based sales system	35
3.1	Introduction	35
3.2	Web-Based Sales System	36
3.2.1	System specification	36
3.2.2	PEPA model of the system without attacks	37
3.2.3	Performance evaluation and results	40
3.3	Web-Based Sales system in the presence of the attacks	43
3.3.1	System specification	43
3.3.2	PEPA model of the system in the presence of the attacks	44
3.3.3	Performance evaluation and results for the extended model	46
3.4	Conclusion	56
3.5	Chapter Summary	56
4	Performance modelling of an anonymous and failure resilient fair-exchange protocol	59
4.1	Introduction	59
4.2	Protocol specification	60
4.2.1	The basic failure resilient fair-exchange protocol specification	60
4.2.2	The basic protocol extension for handling misbehaviours and communication problems	62
4.2.3	The optimistic anonymous protocol	64
4.2.4	The optimistic anonymous extended protocol for handling misbehaviours and communication problems	67
4.3	PEPA models	70
4.3.1	A PEPA model of the basic failure resilient fair-exchange ecommerce protocol	70
4.3.2	PEPA model of the extended failure resilient fair-exchange protocol	73
4.3.3	PEPA model of the optimistic anonymous protocol	76
4.3.4	PEPA models of the extended optimistic anonymous protocol	80
4.4	Numerical results	89
4.4.1	Performance evaluation of the basic failure resilient fair-exchange protocol model	89

4.4.2	Performance evaluation of the extended failure resilient fair exchange protocol model	92
4.4.3	Performance evaluation of the optimistic anonymous protocol	102
4.4.4	Performance evaluation of the extended optimistic anonymous protocol	110
4.5	Conclusion	138
4.6	Chapter Summary	139
5	Performance modelling of attack graphs	141
5.1	Introduction	141
5.2	Methods for creating a PEPA model for a given attack graph	142
5.2.1	Basic PEPA model	142
5.2.2	Multiple component PEPA model	146
5.3	Case study 1: Basic PEPA model for attack graph	154
5.3.1	System specification	154
5.3.2	PEPA model	155
5.3.3	Performance evaluation	157
5.3.4	Alternative PEPA model	163
5.3.5	Performance evaluation of alternative PEPA model	165
5.4	Case study 2: Multiple component PEPA model for attack graph	166
5.4.1	System specification	166
5.4.2	PEPA model	167
5.4.3	Performance evaluation	171
5.5	Conclusion	172
5.6	Chapter Summary	172
6	Advanced models of attacker behaviour	175
6.1	Introduction	175
6.2	PEPA models of attacker skill	176
6.2.1	Beginner attacker component	176
6.2.2	Intermediate attacker component	178
6.2.3	Expert attacker component	180
6.3	Performance evaluation of alternative PEPA models	183
6.4	Attacker learning behaviour	189
6.4.1	Attack graph PEPA model with attacker learning behaviour	190
6.4.2	Performance evaluation of the PEPA model with attacker learning behaviour	196
6.5	Defender learning behaviour	198

6.5.1	Attack graph PEPA model with defender learning behaviour	199
6.5.2	Performance evaluation of the PEPA model with defender learning behaviour	200
6.6	Conclusion	202
6.7	Chapter Summary	203
7	Conclusion	205
7.1	Summary of the research	205
7.2	Summary of contributions	207
7.3	Limitations of the research	208
7.4	Suggestions for future work	209
	References	211
	Appendix A The PEPA Plug-in tool features	219
A.1	Overview	219
A.2	Performance evaluation	220
A.2.1	Population level	220
A.2.2	Throughput	222
A.2.3	Average response time	223
A.2.4	Passage time analysis	225

List of figures

2.1	The operational semantics of PEPA models [33].	12
2.2	Attack tree [64].	25
2.3	Example of network and attack graph [27].	26
3.1	The Throughput Analysis of <i>orderSucessfullyPlaced</i>	41
3.2	The Throughput Analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i>	41
3.3	The population level analysis of <i>Customer₁</i> and <i>Webserver₃</i>	42
3.4	The population level analysis of <i>Warehouse₆</i> and <i>Warehouse₉</i>	43
3.5	The Throughput Analysis of <i>destroyedOrder</i> , <i>orderSucessfullyPlaced</i> and <i>repeatOrder</i> when $q = 0.9$	47
3.6	The Throughput Analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> when $q = 0.9$	48
3.7	The Throughput Analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> during a longer time when $q = 0.9$	48
3.8	The throughput analysis of <i>destroyedOrder</i> , <i>orderSucessfullyPlaced</i> and <i>repeatOrder</i> when $q = 0.5$	49
3.9	The throughput analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> when $q = 0.5$	50
3.10	The throughput analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> during a longer time when $q = 0.5$	50
3.11	The throughput analysis of <i>destroyedOrder</i> , <i>orderSucessfullyPlaced</i> and <i>repeatOrder</i> when $q = 0.1$	51
3.12	The throughput analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> when $q = 0.1$	51
3.13	The throughput analysis of <i>destroyedOrder</i> , <i>orderSucessfullyPlaced</i> and <i>repeatOrder</i> in relation to q different values.	52

3.14	The throughput analysis of <i>forwardOrder</i> , <i>removeExpiredObject</i> and <i>removeSoldObjects</i> in relation to q different values.	53
3.15	The population level analysis when $q=0.9$	54
3.16	The population level analysis when $q=0.5$	54
3.17	The population level analysis when $q=0.1$	54
3.18	The population level analysis of <i>Warehouse9</i> and <i>Warehouse9</i> in relation to q different values.	55
3.19	The population level analysis of <i>Customer2</i> , <i>Webserver2</i> and <i>Webserver5</i> in relation to q different values.	55
4.1	The basic failure resilient fair-exchange protocol shows the interaction between a customer (C), a merchant (M) and a trust third party (TTP).	60
4.2	The optimistic anonymous protocol shows the interaction between a customer (C), a merchant (M), a trust third party (TTP) and a bank (B).	65
4.3	The average response time of M_1 and M_3 using ODE.	89
4.4	The population level analysis using ODE with $K=100$ and $N=100$	90
4.5	The population level analysis using ODE with $K=100$ and $N=400$	90
4.6	The throughput analysis using ODE with $K=100$ and $N=100$	91
4.7	The throughput analysis using ODE with $K=100$ and $N=400$	92
4.8	The average response time of M_1 and M_3 using ODE and $K=20$	93
4.9	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=100$ and $N=100$	94
4.10	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=100$ and $N=400$	94
4.11	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=20$ and $N=100$	95
4.12	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=50$ and $N=100$	95
4.13	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=50$ and $N=400$	96
4.14	The population level analysis for C_2 , C_4 and C_8 using ODE with $K=200$ and $N=400$	96
4.15	The throughput analysis for <i>sendCAabort</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=20$ and $N=100$	97
4.16	The throughput analysis for <i>sendCAabort</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=50$ and $N=100$	98

4.17	The throughput analysis for <i>sendCAbort</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=50$ and $N=400$	98
4.18	The throughput analysis for <i>sendCAbort</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=200$ and $N=400$	99
4.19	The throughput analysis for <i>discoverIncorrectPTK</i> , <i>forwardKtoC</i> and <i>send-CkByTTP</i> using ODE with $K=20$ and $N=100$	100
4.20	The throughput analysis for <i>discoverIncorrectPTK</i> , <i>forwardKtoC</i> and <i>send-CkByTTP</i> using ODE with $K=50$ and $N=100$	100
4.21	The throughput analysis for <i>discoverIncorrectPTK</i> , <i>forwardKtoC</i> and <i>send-CkByTTP</i> using ODE with $K=50$ and $N=400$	101
4.22	The throughput analysis for <i>discoverIncorrectPTK</i> , <i>forwardKtoC</i> and <i>send-CkByTTP</i> using ODE with $K=200$ and $N=400$	101
4.23	The average response time of M_1 and M_5 using ODE ($K=20$ and $S=20$). . .	102
4.24	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=5$	103
4.25	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=20$	104
4.26	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=50$	104
4.27	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=100$ and $S=50$	105
4.28	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=400$ and $S=50$	105
4.29	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=400$ and $S=100$	106
4.30	The population level analysis for C_2 , C_4 and C_6 using ODE with $K=100$, $N=400$ and $S=100$	106
4.31	The throughput analysis for <i>sendCAbort</i> , <i>sendCDigitalCoins</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=20$, $N=100$ and $S=5$	108
4.32	The throughput analysis for <i>sendCAbort</i> , <i>sendCDigitalCoins</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=20$, $N=100$ and $S=20$	108
4.33	The throughput analysis for <i>sendCAbort</i> , <i>sendCDigitalCoins</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=50$, $N=400$ and $S=50$	109
4.34	The throughput analysis for <i>sendCAbort</i> , <i>sendCDigitalCoins</i> , <i>sendCEP</i> and <i>sendCPDk</i> using ODE with $K=50$, $N=600$ and $S=50$	109
4.35	The average response time of M_1 and M_5 using ODE.	110

4.36	The average response time of TTP_{5a} , TTP_6 and TTP_{10} using ODE, $K=20$, $S=20$	111
4.37	The average response time of TTP_{5a} , TTP_6 and TTP_{10} using ODE, $K=60$, $S=20$	112
4.38	The population level analysis using ODE with $K=20$, $N=100$ and $S=20$. . .	112
4.39	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$. . .	114
4.40	The population level analysis using ODE with $K=60$, $N=200$ and $S=20$. . .	114
4.41	The population level analysis using ODE with $K=60$, $N=200$ and $S=40$. . .	115
4.42	The throughput analysis of the actions using ODE with $K=20$, $N=200$ and $S=20$	116
4.43	The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=20$	116
4.44	The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=40$	116
4.45	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$. . .	118
4.46	The population level analysis using ODE with $K=60$, $N=200$ and $S=20$. . .	118
4.47	The population level analysis using ODE with $K=60$, $N=200$ and $S=40$. . .	119
4.48	The throughput analysis of the actions using ODE with $K=20$, $N=200$ and $S=20$	120
4.49	The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=20$	120
4.50	The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=40$	120
4.51	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ in relation to different probabilities of M to be honest.	121
4.52	The throughput analysis of actions using ODE with $K=20$, $N=200$ and $S=20$ in relation to different probabilities of M to be honest.	122
4.53	The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ in relation to different probabilities of M to be honest.	123
4.54	The throughput analysis of actions using ODE with $K=20$, $N=600$ and $S=20$ in relation to different probabilities of M to be honest.	123
4.55	The steady-state detection time in relation to the population number ($N=200$ and $N=600$).	124
4.56	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=0.2$	125

4.57	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=0.5$	125
4.58	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=2$	126
4.59	The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=4$	126
4.60	The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=0.2$	127
4.61	The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=0.5$	127
4.62	The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=2$	128
4.63	The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=4$	128
4.64	The throughput analysis using ODE with $K=20$, $N=200$ and $S=20$ and with different rates for the shared actions between M and B.	129
4.65	The population level analysis using ODE with $K=20$, $N=100$ and $S=20$ (10 misbehaving customers).	131
4.66	The population level analysis using ODE with $K=20$, $N=100$ and $S=20$ (90 misbehaving customers).	131
4.67	The throughput analysis of actions using ODE with $K=20$, $N=100$ and $S=20$ (10 misbehaving customers).	132
4.68	The throughput analysis of actions using ODE with $K=20$, $N=100$ and $S=20$ (90 misbehaving customers).	133
4.69	The population level analysis using ODE with $K=20$, $N=500$ and $S=20$ (50 misbehaving customers).	134
4.70	The population level analysis using ODE with $K=20$, $N=500$ and $S=20$ (450 misbehaving customers).	134
4.71	The throughput analysis of actions using ODE with $K=20$, $N=500$ and $S=20$ (50 misbehaving customers).	135
4.72	The throughput analysis of actions using ODE with $K=20$, $N=500$ and $S=20$ (450 misbehaving customers).	135
4.73	The population level analysis using ODE with $K=100$, $N=500$ and $S=20$ (450 misbehaving customers).	136
4.74	The population level analysis using ODE with $K=100$, $N=500$ and $S=40$ (450 misbehaving customers).	137

4.75	The throughput analysis of actions using ODE with $K=100$, $N=500$ and $S=20$ (450 misbehaving customers).	137
4.76	The throughput analysis of actions using ODE with $K=100$, $N=500$ and $S=40$ (450 misbehaving customers).	138
5.1	An example of an attack graph that comprises four nodes: Start, host 1 (H1), host 2 (H2), and host 3 (H3).	144
5.2	Attack graph consists of six nodes: node A is the starting node, and the H1, H2, H3, H4, and H5 nodes are the hosts in a system [69].	145
5.3	An example of the attacker's abilities/steps to compromise a system.	147
5.4	An example of the attacker's abilities to compromise a system.	151
5.5	Attack graph [69].	154
5.6	Passage-time analysis for each path in the attack graph.	157
5.7	Passage-time analysis for exploiting <i>servU5</i> and <i>telnet</i> to compromised action.	159
5.8	Passage-time analysis for all paths when <i>telnet</i> action rate=0.5.	160
5.9	Passage-time analysis for all paths when <i>telnet</i> action rate=0.1.	160
5.10	Passage-time analysis when <i>telnet</i> action rate =0.9.	162
5.11	Passage-time analysis when <i>telnet</i> action rate =0.5.	162
5.12	Passage-time analysis when <i>telnet</i> action rate =0.1.	163
5.13	Passage-time analysis for each path in the alternative PEPA models.	166
5.14	The attacker's abilities/steps to compromise the system.	167
5.15	Passage-time analysis of each path in the attack graph.	171
6.1	The beginner attacker's capabilities to compromise the system.	176
6.2	The intermediate attacker's capabilities to compromise the system.	178
6.3	The expert attacker's capabilities to compromise the system.	181
6.4	Passage-time analysis of each path in the attack graph for the beginner attacker when the probability of exploit code availability is 0.2.	184
6.5	Passage-time analysis of each path in the attack graph for the intermediate attacker when the probability of exploit code availability is 0.2.	184
6.6	Passage-time analysis of each path in the attack graph for the expert attacker when the probability of exploit code availability is 0.2.	185
6.7	Path 6 in the attack graph for beginner attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).	186
6.8	Path 6 in the attack graph for intermediate attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).	186

6.9	Path 6 in the attack graph for expert attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).	187
6.10	Passage-time analysis for the beginner attacker when $p = 0.2$	188
6.11	Passage-time analysis for the expert attacker when $p = 0.2$	188
6.12	Passage-time analysis for each path when <i>gainKnowledge</i> action rate $a1=0.01$ 196	
6.13	Passage-time analysis for each path when <i>gainKnowledge</i> action rate $a1=0.1$. 197	
6.14	The impact of implementing learning behaviour for the attacker on path 1. . .	198
6.15	The impact of implementing learning behaviour for the attacker on path 6. . .	198
6.16	The impact of implementing learning behaviour for the defender on path 1. . .	201
6.17	The impact of implementing learning behaviour for the defender on path 6. . .	202
A.1	The PEPA Eclipse Plug-in interface.	220
A.2	The population level analysis.	221
A.3	The population level analysis pop-up box.	221
A.4	The graph view of population level.	222
A.5	The throughput analysis.	222
A.6	The throughput analysis pop-up box.	223
A.7	The graph view of throughput.	223
A.8	The average response time.	224
A.9	The average response pop-up box.	224
A.10	The average response's result window.	225
A.11	The Passage Time Analysis.	225
A.12	The Passage Time pop-up box.	226
A.13	The graph view for the passage time result.	226

List of tables

5.1	The probabilities of vulnerabilities to be breached [69].	145
5.2	The probabilities of vulnerabilities to be breached [69].	155
5.3	Attack Paths and probabilities, Sun <i>et al.</i> [69].	158
5.4	The path order from our results and Sun <i>et al.</i> [69].	158
5.5	The attack probability of each path when the probability of <i>telnet</i> to be breach is equal to 0.8, 0.5 and 0.1.	161
6.1	The average time to compromise the system for beginner attacker.	189
6.2	The average time to compromise the system for expert attacker.	189

Nomenclature

List of abbreviations

AST	Abstract Syntax Tree
B	Bank
C	Customer
C'	Customer's pseudo identifier
CPSM	Combined Performance and Security Metric
CSV	Comma Separated Values
CTMC	Continuous Time Markov Chain
CVSS	Common Vulnerability Scoring System
DTMC	Discrete Time Markov Chains
FTP	File Transfer Protocol
GSPN	Generalized Stochastic Petri Net
HMM	Hidden Markov Model
IDS	Intrusion Detection Systems
M	Merchant
MANETs	Mobile Ad hoc Networks
MTTSF	Mean Time to Security Failure
ODE	Ordinary Differential Equation

PEPA	Performance Evaluation Process Algebra
PO	Purchase Order
RSH	Remote Shell
SANs	Stochastic Activity Networks
SPA	Stochastic Process Algebras
SPN	Stochastic Petri Nets
SSA	Stochastic Simulation Algorithms
SSH	Secure Shell
TTP	Trust Third Party
TUDT	Trusted and Untrusted Double Threshold-based rekeying

Chapter 1

Introduction

1.1 Motivation

The advancements to date in computing and technology have involved developing complex systems comprised of multiple interacting parts. The need for secure systems also leads to the development of a wide range of sophisticated security protocols and algorithms. Performance and security are essential factors when developing and evaluating almost all systems. Security mechanisms impose additional costs on a system, possibly adversely affecting its performance. Developing a system that has a high level of security could result in poor performance of the system, and the reduction of some security features and functions can improve the performance of the system but can impact the security status of the system. Modelling and measuring the performance of security protocols can help to understand their behaviour and which parts have the most impact on the performance of a system, in order to assist in developing a system with acceptable levels of both security and performance and to support the development of lightweight protocols. Therefore, we are motivated by whether we can use a performance modelling formalism to formally model security protocols in different scenarios, such as with and without misbehaviours, and then evaluate models and derive performance measures such as response time and throughput to explore a protocol.

The growing demand for online sale systems has led to the development of many web-based sale systems. The online environment suffers from a high-security vulnerability, as it is an open medium, exposing such a system to threats and attacks. Attacks on such a system can affect both a merchant and a customer. They can also delay and disrupt sales. The impact and cost that cyber attacks contribute to the performance of web-based sales system need to be studied to understand better how the system behaves and adapts in different scenarios, such as with and without an attack, to remain secure and provide a sustainable performance level. Although several researchers have studied the performance of a system under an

attack [30, 55, 76, 87], further research is needed regarding modelling and investigating the performance cost introduced by the attacks in a web-based sale system as it is clear from the related work that there are few existing models in this domain. Therefore this can be valuable additional work. This motivates us to whether we can use a performance modelling formalism to formally model a web-based sale system in two scenarios, with and without the presence of attack, and then numerically analyse the models and derive performance measures such as population level and throughput to explore the effects of an attack on a performance of a web-based sales system.

Keeping a system secure is not trivial. A defender must keep a system secure by preventing or early detecting an attacker's intent in order to response and recover in good time. This can be done by identifying and analysing the possible ways that an attacker can use to attack a system in order to assist a defender to implement suitable protection measures. An attack graph is a popular graph-based method that can support a defender to understand the attacker's behaviour and then work to protect a system. By presenting the possible attack paths using the attack graph, the defender can see the possible attacker's journeys to achieve its goal and/or compromise a system. Thus, the defender can evaluate the security status of a system and analyse the attacks from the attacker's perspective. Estimating the actual time to compromise a system is important to indicate how much safe time the system has before it is compromised. It also supports a defender to implement suitable countermeasures in timely manner to keep a system secure. This motivates us to study whether it is possible to translate a pre-existing attack graph to a performance model to represent the interaction between an attacker and a system, the attacker behaviour and the progression in an attack. The performance model can be then analysed via a continuous-time Markov chain with rates to support estimating the time to compromise a system and the time it takes the attacker to get to a particular vulnerability on a system. We can perform path analysis and sensitivity analysis on the model in order to explore the security status of a system. This approach also enables us to estimate the time taken until an attacker compromises a system for each path by using the vulnerabilities that existed in the system and considering different factors such as exploit code availability and the attacker's skill to understand and identify critical threats.

In this thesis, we study whether we can use a performance modelling formalism to model systems under attacks and misbehaviours with time-variable aspects. The method used to model, explore and measure those aspects is Performance Evaluation Process Algebra (PEPA). PEPA is a well-known implementation of Stochastic Process Algebras. It is used to model systems in this thesis as it supports a compositional, formal and abstract approach to construct a model. This approach is important to formally construct models for a complex system in a concise and effective way to be easy to understand, assess, reuse, and modify.

1.2 Aim and Objectives

Our overall aim is to model secure systems where behaviour changes in response to attack or misbehaviour. The following is the list of objectives that we need to fulfil this aim:

- Conduct a review of the existing literature regarding existing performance measurement studies for evaluating the impact of security algorithms and protocols on system performance, current performance studies of systems under attack, and existing studies of attacker behaviour using an attack tree and attack graph.
- Explore the impact and cost of cyber-attacks on the performance of web-based sales system by proposing and evaluating two performance models of the system, one with no attack and the other under a denial of service attack.
- Explore the performance cost of a security protocol (anonymous and failure resilient fair-exchange e-commerce protocol) by proposing and evaluating performance models for different scenarios.
- Propose two methods to automate generation of a PEPA model based on a pre-existing attack graph specification.
- Demonstrate through a case study how we used the developed PEPA models for an attack graph to determine the most and the least system security threatening paths and an attacker's time to compromise a system.
- Consider the attackers' skills and the availability of the exploit code when developing a PEPA model of an attack graph.
- Enhance the adaptability of our proposed PEPA attack graph models by including learning behaviour for both attacker and defender, and then demonstrate the influence of learning behaviour on the attacker's time to compromise the system.

1.3 Contributions

The contributions of this thesis are as follows:

- We explore the development of appropriate models through two PEPA models of a web-based sales system, one without an attack and the other with the presence of a denial of service attack. These models are proposed, analysed and compared based on the performance and security trade-offs in Chapter 3.

- We explore the impact of anonymity and misbehaviour through PEPA models of an anonymous and failure resilient fair-exchange e-commerce protocol. These models proposed, analysed and compared based on the performance and security trade-offs in four main scenarios: with and without an anonymity feature, and with and without misbehaviour of any parties, in Chapter 4.
- We explore the specification of models of attack graphs via two methods to create PEPA models for a pre-existing attack graph in Chapter 5.
- PEPA models of attack graph are explored to enable quantitative analysis via a continuous-time Markov chain with rates to estimate the attacker's time to compromise the system for each attack path and to deduce the most and least security threatening paths in Chapter 5.
- Sensitivity analysis is applied to the proposed PEPA model of the attack graph to support analysing the security status of the system and then improving the system's security status in Chapter 5.
- We consider the effect of attacker expertise by proposing three different skills for the attackers: beginner, intermediate and expert. These are proposed and employed in PEPA models of the attack graph and then the models are numerically analysed and compared in Chapter 6.
- An availability of exploit code factor is introduced and implemented in the PEPA models and then the models are analysed to show the impact of this factor on the attacker's time to compromise the system in Chapter 6.
- Learning behaviour for the attacker and defender is introduced and implemented in PEPA models and then the PEPA models are analysed to show how the learning behaviour for both the attacker and the defender can impact the attacker's time to compromise the system in Chapter 6.

1.4 Thesis outline

This thesis comprises seven chapters, including this introduction chapter. The remainder of this thesis chapters is organised as follows:

Chapter 2 This chapter discusses performance modelling and the PEPA formalism used in this thesis to model the systems. It also discusses a sample of studies related to performance analysis of security-related applications. Additionally, it discusses a sample of studies examining attacker behaviour through the use of attack trees and attack graphs, as well as models of attacker behaviour.

Chapter 3 This chapter examines the impact and cost associated with cyber-attacks on system performance. We proposed PEPA models of a web-based sales system in two different scenarios: with and without denial of service attacks. The models are numerically analysed using a fluid approximation based on ordinary differential equations (ODEs) to avoid the state-space explosion problem. This problem arises as a result of the generation of extremely large state spaces from the interaction of numerous components, which makes computing the numerical solution through linear algebra extremely expensive or very difficult. The two models' results are compared to gain a better understanding of how the system behaves in different scenarios.

Chapter 4 This chapter explores the performance cost introduced by a security protocol known as an anonymous and failure resilient fair-exchange e-commerce protocol. The proposed PEPA models were formulated in two different ways: with and without an anonymity feature. Moreover, both protocol versions were modelled in two ways: as a basic protocol with no misbehaviour of any parties whereby it does not require the active involvement of TTP, and as an extended protocol whereby the TTP's participation is essential to resolving disputes between participants. The models were analysed numerically. These enable understanding the protocol's behaviour and present the performance cost it introduces.

Chapter 5 This chapter provides two methods for generating a PEPA model based on a pre-existing attack graph specification. The first method generates a PEPA model that comprises a single sequential component representing both the system and the attacker. The second method generates a PEPA model that comprises two sequential components representing the system and the attacker. Then, we demonstrated through the case studies how we used the generated PEPA models to perform path and sensitivity analysis and to deduce the most and the least system security-threatening paths and time to compromise the system, which a defender can use as an indicator of how much safe time the system has before it is compromised. In addition, these can rank the risk of all attack paths and help the defender prioritise the countermeasures.

Chapter 6 This chapter considers the effect of attacker expertise by proposing three different skills for the attackers: beginner, intermediate and expert. Based on a attack graph specification and the three proposed attackers' skills, we generated the PEPA models for each attacker's skill. In this chapter, we considered two criteria: attacker skill and the availability of exploit code to estimate the attacker's time to compromise the system. Additionally, we implemented learning behaviours in the model for an attacker and a defender to improve the adaptability of the PEPA models. We then illustrated how learning behaviour for both the attacker and the defender would impact the attacker's time to compromise the system.

Chapter 7 This chapter presents the conclusion of this thesis. we summarise our contributions, limitations of our research and possible future study.

1.5 Publications

Almutairi, O. and Thomas, N. (2019). Performance modelling of an anonymous and failure resilient fair-exchange e-commerce protocol. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, pages 5–12.

Almutairi, O. and Thomas, N. (2020). Performance modelling of the impact of cyber attacks on a web-based sales system. *Electronic Notes in Theoretical Computer Science*, 353:5–20.

Almutairi, O. and Thomas, N., (2021). Modelling a Fair-Exchange Protocol in the Presence of Misbehaviour Using PEPA. In *Performance Engineering and Stochastic Modeling*, pages 96-114. Springer.

Chapter 2

Background

2.1 The Security and Performance Trade-off

Computing systems are becoming increasingly complex and consist of multiple interactive components. Performance has been seen as an important aspect for evaluating computing systems. Performance, as defined by [38], is when a given system, or any of its components, achieves denominated functions specified under certain constraints, i.e. accuracy, speed and/or memory consumption. Common metrics that have been widely applied to measure the performance of a computing system are throughput, which is the amount of work that could be accomplished in a specific time, utilization, which is the value resulting when dividing busy time by the time available to the system, and response time, which is the time between sending a request and receiving a response [38, 84].

Most computing systems are connected to a network, either privately or publicly, and this can pose vulnerability issues, exposing systems to threats and attacks. In addition, the security aspect is now considered a more significant factor when developing computing systems than formerly due to the number of threats and greater awareness. Therefore, many security protocols and methods have been developed and employed in order to develop a secure system. A security protocol is a series of steps and security-related operations designed to support interaction and communication between two or more parties over a network to satisfy some security purposes [15].

However, the steps taken to make systems more secure can affect their overall performance. Security can add an extra overhead to a system, directly influencing its performance; this is a problem for many different domains. For example, the performance of a web server reduces in response to the implementation of the secure sockets layer protocol [8, 9], and the computational cost of security mechanisms adds performance costs to wireless sensor networks during secure communication [88].

Increasing the steps and the functions to make a system more secure can adversely affect its performance. However, improved system performance can be achieved by, for example, decreasing the security steps or lowering the computational cost of security functions, i.e. decreasing the length of the key used in encryption and decryption can reduce the computation cost and subsequently increase the performance of the system [78]. However, this approach can influence the security status of the system. Thus, it is important to develop a system affording an optimal balance between security and performance. Therefore, the extra cost that security functions contribute to the performance of the system has attracted widespread attention. As a result, developers have conducted explorations and taken measurements with the aim of developing a secure system that gives a satisfactory performance. A computing system developer is expected to employ security algorithms and protocols to achieve an acceptable level of system performance and an acceptable level of security. Therefore, studying the security methods and protocols by modelling and investigating their performance could help with the selection of the most suitable option for any computing system.

Performance and security are essential aspects for almost all systems. The methods used in order to explore and measure those aspects are either experimental as in [47] or model-based by employing modelling techniques as in [16, 76, 84]. Moreover, employing a model-based approach can be more tractable and useful to study the performance and security aspects of the system, as it replaces a complex system with a more simplified model that can be easily understood, analysed and modified to investigate the different results that are possible [12].

2.2 Performance modelling methods

A model is a simple version of a system. It requires presenting all relevant aspects of the system that need to be studied and explored. Employing a model-based approach could prove more flexible and beneficial for evaluating performance aspects of the system, as it replaces the pre-existing system with one that can be readily understood, studied and analysed, making it easier to be modified to investigate the different results that are possible [12]. Performance modelling provides a better means of understanding the system's behaviour. Performance modelling of a system is about creating a mathematical model of a system to study and analyse the dynamic behaviour of a system [33]. The model can be evaluated by using an analytical technique to solve equations or by simulating the model [33]. Its evaluation is based on its parameter values to show its behaviour and derive some numerical results that can be used to assess a system's performance for any objective. This is important to gain a

good understanding of which part of the system is most responsible for influencing its overall performance, thereby making it possible to design an efficient system.

Several modelling formalisms have been widely employed and used to model system performance, such as queueing networks [33, 48], Stochastic Petri Nets (SPN) [33] and Stochastic Process Algebras (SPA) [11, 19]. These formalisms have been employed to provide in-depth knowledge and understanding of the system's behaviour and the processes involved in a system, allowing assessment and measuring the system's performance. SPA offers three important features which make it differ from other formalisms. The three key features of SPA are as follow [35]:

- Compositionality, the ability to model a system as a collective of subsystems and the interaction between them produce its behaviour.
- Formality, the ability to define all terms in the language precisely.
- Abstraction, the ability to clearly and explicitly model the components and subcomponents of a system ignoring some internal behaviour when appropriate to construct a complex model.

SPA is a stochastic extension of a process algebra for adding quantification to the model for performance modelling to derive performance measures such as throughput, utilisation and response time [19]. A well-known implementation of SPA is Performance Evaluation Process Algebra (PEPA).

2.3 Performance Evaluation Process Algebra (PEPA)

This subsection provides an overview of Performance Evaluation Process Algebra (PEPA), this approach is used to model the systems under investigation in this thesis. The advancements to date in computing and technology have involved the development of more complex computing systems, comprised of multiple different parts. The complex system needs to decompose into subsystems to clearly and easily be modelled. Unlike other performance modelling formalism, PEPA supports a compositional, formal and abstract approach to construct a model. This approach is important to construct models for a complex system to be easy to understand, assess, reuse and modify.

PEPA is a well-known implementation of Stochastic Process Algebras (SPA). PEPA formalism is a high-level description language used for Markov modelling [32]. It is employed to construct Continuous Time Markov Chain (CTMC) for performance modelling [35]. The two basic elements of PEPA model are components and activities. The component element

can be made up of multiple components. The activity is characterised by a combination of an action type and rate. A system is modelled in PEPA formalism as a set of components that interact and engage individually or with other components in activities in order to evaluate its performance [33]. In PEPA modelling, the components represent the active parts in the system and the behaviour of each part is represented by its activities [35].

PEPA is based on a Markovian Process Algebra in which each action is associated with a random variable with an exponential distribution to represent the rate at which the action occurs. The exponential distribution provides a memoryless property, making modelling easier to specify and analyse because the completion time of the activity depends on the current state and does not depend on the history of the past states. This activity rate can be any positive real number. When an activity is performed, the internal state of the component is changed. PEPA has been used to successfully model the performance of secure systems [83, 85]. The use of PEPA is appropriate because we just need to understand the current state of the system to understand the performance of these systems, and we don't need to understand the history that has been passed. However, the use of PEPA has the limitation that we do not explicitly model the content of messages that are passed and so the security properties are not captured within the security protocols. As a result, we are unable to inspect and prove the security properties associated with security messages using PEPA.

PEPA models have three different interpretations: explicit state of CTMC for an individual component view which leads to generating a complete CTMC, aggregated state of CTMC for studying bigger system and continues approximation of CTMC for fluid flow analysis based on ordinary differential equation (ODE) for studying very large system [31].

2.3.1 PEPA syntax

The two key elements for PEPA formalism are components and activities. The PEPA formalism also has a few combinators that allow constructing the description of a system from its different components and activities. The combinators are used to define the components' behaviours through their activities and interactions with other components [35]. The following are the grammar of the syntax of terms in PEPA model and a brief description of the syntax for PEPA combinators [35]:

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

Prefix, $(\alpha, r).P$, The prefix combinator is a mechanism that constructs the behaviours of the system's components. The component evolves to component P when the activity (α, r) is enabled, and action α performs at rate r . the rate can be any positive real number. It can also

be passive, which is denoted by the top symbol, \top ; in this case, the action occurs depending on the rates of other components.

Choice, $P + Q$, The choice combinator is used to allow the component of a system to evolve either to component P or component Q . All current activities of P and Q are enabled. Moving to either component depends on which component-enabled activities complete first. The resultant component reflects the internal state of the system, and the other component is discarded.

Cooperation, $P \bowtie_L Q$, The cooperation combinator defines the system's behaviour via the cooperation of its components based on the cooperation set. Thus, the synchronisation behaviours of the system's components and the interaction between them are modelled using the cooperation combinator. The cooperation set, denoted by L , contains all actions types that must be shared between components. The shared action only occurs if the activity of that type is enabled by both components P and Q . The duration of the activity is based on the rate of the slowest component. An empty cooperation set means no interaction between the components, and these components work independently from each other. Two components or more work independently can also be denoted by $P || Q$.

Hiding, P/L , The hiding combinator defines a list L of hiding activities for the component P . Any activities of types in the list L are hidden and private for the component P . The hidden activities appear as unknown action type τ and are observed as an internal delay of component P .

Constant, $A \stackrel{def}{=} P$, The constant combinator enables components to be assigned names. For example, constant A , which is a component, is associated with the behaviour of the component P .

Moreover, we can denote the multiple copies of the component P as $P[n]$ where n is the number of copies of P . Figure 2.1 shows the operational semantics of a PEPA model. It defines the semantic rules of how the system evolves based on its components and activities [33]. These rules specify the activities that a component may engage in at any time and the completion of each activity results in a transition within a system. R , in Figure 2.1, means that the rate of a shared action will be at the rate of the slowest participant. A state transition graph, which is also called a derivation graph, can be generated by following these rules to describe the potential behaviour of a component within a model [35]. This graph comprises nodes representing the components and their states and arcs representing the

potential transitions between states and labelled by activity types and rates. Based on these semantics rules, PEPA model can be mapped to a CTMC to compute performance metrics.

<p>Prefix</p> $\frac{}{(\alpha, r).E \xrightarrow{(\alpha, r)} E}$	<p>Cooperation</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E \bowtie_L F \xrightarrow{(\alpha, r)} E' \bowtie_L F} \quad (\alpha \notin L)$	<p>Hiding</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E/L \xrightarrow{(\alpha, r)} E'/L} \quad (\alpha \notin L)$
<p>Choice</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{E + F \xrightarrow{(\alpha, r)} E'}$	$\frac{F \xrightarrow{(\alpha, r)} F'}{E \bowtie_L F \xrightarrow{(\alpha, r)} E \bowtie_L F'} \quad (\alpha \notin L)$	$\frac{E \xrightarrow{(\alpha, r)} E'}{E/L \xrightarrow{(\tau, r)} E'/L} \quad (\alpha \in L)$
$\frac{F \xrightarrow{(\alpha, r)} F'}{E + F \xrightarrow{(\alpha, r)} F'}$	$\frac{E \xrightarrow{(\alpha, r_1)} E' \quad F \xrightarrow{(\alpha, r_2)} F'}{E \bowtie_L F \xrightarrow{(\alpha, R)} E' \bowtie_L F'} \quad (\alpha \in L)$ <p>where $R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min(r_\alpha(E), r_\alpha(F))$</p>	<p>Constant</p> $\frac{E \xrightarrow{(\alpha, r)} E'}{A \xrightarrow{(\alpha, r)} E'} \quad (A \stackrel{def}{=} E)$

Fig. 2.1 The operational semantics of PEPA models [33].

2.3.2 Continuous Time Markov Chain (CTMC)

A Continuous Time Markov Chain (CTMC) is a type of quantitative method that shows the evolution of a system in the time and space dimensions [31]. It is a continuous time stochastic process with a discrete state space [77]. It is also called Markov process as it satisfies the Markov property, which is also called the memoryless property. The Markov property means the future evaluation of the Markov process only depends on the given present state and does not depend on the history of the past states [77].

CTMCs are widely employed to model stochastic systems and analyse systems performance. A CTMC can be used to model system in real time such as computer networks [66]. A PEPA model can be solved by deriving an equivalent CTMC [32]. The system performance measures based on a CTMC can be derived from the PEPA model of the system by computing the system's steady-state probability distribution through the system's behaviour in the long run [31]. Markovian PEPA models analysed using CTMCs suffer from the problem of state-space explosion when evaluating a system that has a large number of states and large numbers of replicated components [31]. The problem occurs when models of a system become very large, even for systems that are not very complex. This results in the generation of very large state spaces, which makes computing the numerical solution through linear algebra costly or very difficult [34]. As a result, the size of models and consequently the complexity of systems that can be efficiently studied and analysed using CTMCs are limited. Hence, other

analysis techniques must be employed to study larger and more complex models, such as some of those in this thesis.

2.3.3 Fluid semantics for PEPA

Fluid flow analysis based on ordinary differential equation (ODE) is an alternative technique for analysing PEPA and obtaining performance measures. It is considered as one of PEPA interpretation. Hillston [34] is the first who introduces this technique to tackle the problem of state-space explosion. Hillston shows how to analyse the performance of the large-scale systems that are modelled in PEPA by using the ODE-based analysis [34]. ODE-based analysis approximates large discrete states to continuous real-valued variables which are represented by a set of coupled ordinary differential equations to describe the time evaluation of the continuous variables [34]. It can compute the average behaviour of the system with high populations in a fraction of a second, unlike simulation analysis which needs to calculate the average behaviour of the system via multiple simulation runs, which sometimes takes days [34]. The PEPA model of the system can be solved by computing the system's steady-state probability distribution and/or transient probability distribution to derive the system performance measures based on ODE [34].

The following is a simple example of N processes competing to use M resources:

$$P_1 \stackrel{def}{=} (think, a).P_2$$

$$P_2 \stackrel{def}{=} (use, u).P_1$$

$$R_1 \stackrel{def}{=} (use, u).R_2$$

$$R_2 \stackrel{def}{=} (update, b).R_1$$

The system equation is given by

$$System \stackrel{def}{=} P_1[N] \underset{use}{\bowtie} R_1[M]$$

The processes independently *think* at rate a , before each trying to *use* a resource. Each instance of resource component shares the *use* action, before an *update* is performed at rate b and it can be used again.

The CTMC for this model is trivial if $N=1$. There are four states: $R_1|P_1$, $R_1|P_2$, $R_2|P_1$ and $R_2|P_2$. If we add additional processes and resources, then we rapidly increase the state space such that it soon becomes difficult to analyse.

In the CTMC semantics for this model, we ensure that exactly one process and one resource can perform a *use* action at any time. However, in the ODE semantics, we are only

concerned with the rates of change of the number of processes and resources at any given time. Thus, we can compute the rate at which processes move from P_1 to P_2 and R_1 to R_2 (and vice versa). The rate at which processes leave the derivative P_1 , is simply the rate at which each process does a *think* action, multiplied by the number of actions doing a *think* action (in behaviour P_1). However, the rate at which processes enter P_1 is more complicated, because it involves the shared action *use*.

The rate of a shared action is given as the minimum of the rates of each participant. If there are $P_2(t)$ processes wanting to *use* the resource at time t , then the demand would be $uP_2(t)$, however the single resource can only be used at u . If, at time t , $R_1(t)$ resources are available to be used the resultant rate is $\min(uP_2(t), uR_1(t))$.

Thus, we can write an expression for the rate of change of P_1 as follows:

$$\frac{d}{dt}P_1 = u \min(P_2(t), R_1(t)) - a P_1(t)$$

We can write similar expressions for the other derivatives in the model as follows:

$$\frac{d}{dt}P_2 = a P_1(t) - u \min(P_2(t), R_1(t))$$

$$\frac{d}{dt}R_1 = b R_2(t) - u \min(P_2(t), R_1(t))$$

$$\frac{d}{dt}R_2 = u \min(P_2(t), R_1(t)) - b R_2(t)$$

There are various methods we could use to solve such sets of differential equations. The simplest approach would be to simulate the evolution of the model in small time increments, starting from time $t = 0$ where $P_1(0) = N$, $P_2(0) = 0$, $R_1(0) = M$ and $R_2(0) = 0$.

In this model it would be trivial to find the steady state solution given when all the rates of evolution equal zero, i.e.

$$a P_1(t) = u \min(P_2(t), R_1(t)) = b R_2(t)$$

Bear in mind that $P_1(t) = N - P_2(t)$ and $R_1(t) = M - R_2(t)$, so this is a trivial calculation. A general approach to the calculation of steady state results from ODEs can be found in [71].

2.3.4 PEPA Eclipse plug-in

In this thesis, the creation and performance evaluation of a PEPA model is supported by the PEPA Eclipse plug-in [25, 74]. This tool has been developed to support the Markovian steady-

state analysis, Stochastic Simulation Algorithms (SSA) analysis, and Ordinary Differential Equations (ODE) analysis of the PEPA models in the Eclipse Platform [25, 74]. It also creates the transition graph, calculates the number of states, and provides the Markov process matrix of the PEPA models. Therefore, it is used in the thesis to support the creation of the PEPA models of the systems under investigation and the calculation of the performance measures. We provide an overview in Appendix A of how to derive performance measures using the "PEPA Eclipse plug-in" tool.

2.3.5 Performance metrics

In this thesis, we are primarily concerned with three measures of performance; the population, the throughput and the response time.

Population

The population at time t is defined as the number of instances of a component currently behaving as a given derivative. At time $t = 0$, the population of each derivative is given by the system equation, which defines the starting conditions for the PEPA model. The ODEs derived from the PEPA model represent the rate of change of the population of each derivative. Thus, we can simulate the evolution of the model by applying these rates of change incrementally using small time steps Δt , starting from the known condition at $t = 0$. In this way we can find the population of all derivatives at any time t and hence plot the population over any given time interval.

Throughput

Each action in PEPA is defined with a given rate, or the combination of given rates in the case of shared actions. As the population of each derivative varies over time, so does the rate at which the actions in each derivative are observed to occur. We refer to this observed rate of an action as its throughput at time t . By computing the population at time t , we can calculate the throughput of each action that arises from those populations. This is given directly in the ODEs. Thus, in our example, the throughput of the action *think* at time t depends on its given rate a and the population of the derivative P_1 at time t . Thus $T_{think}(t) = aP_1(t)$. For shared actions the situation is only slightly more complicated, as the rate at which a shared action occurs will depend on the minimum of the number of derivatives which enable that action in each component. Thus, the throughput of the action *use* at time t is given by $umin(P_2(t), R_1(t))$.

Response time

The notion of response time originates from queueing theory, where it is typically defined as the time taken from a job entering a queue until it completes service and moves on. The relationship between the number of jobs, arrival rate and response time is given by the famous Little's Law. In PEPA the notion is slightly different, in that we have only components, derivatives and actions. We can define the response time for a single action to be the time it takes for that action to occur from being enabled in an instance of a component. For independent actions, such as *think* in our example above, this notion of response time is trivial to compute as it is simply the exponential distribution with the given mean. Thus, in the simple example, the average response time for *think* is simply $1/a$. However, for shared actions it is significantly more complicated as we need to take account of the number of instances of each component which enable the shared action. An instance of the process component which does a *think* action to become P_2 , requires an instance of the resource to be R_1 in order for a *use* action to occur. Furthermore, if there are n instances of P_2 enabled, then there needs to be n occurrences of *use* and the rate of each occurrence will depend on the number of P_1 and R_2 derivatives in the current evolution of the model.

In [13], the response time is defined as the time it takes for a CTMC to evolve from one state, the source, to another state, the target. This is then mapped to the PEPA abstraction to give source and target in terms of the derivatives of a model. In general, a response time can therefore not only entail more than one action, but potentially also more than one path, if there are multiple means to evolve the model from source to target. To compute this evolution, Bradley *et al.* [13] translate the cyclic PEPA model into an absorbing model by employing a STOP derivative, as in [72]. The system equation is modified to define the specified source. The ODEs resulting from the absorbing model can then be simulated, as described previously and the population over time computed until the required target condition is met.

The method implemented in the PEPA Eclipse Plug-in is based on this approach, but differs by correcting an error that occurs when instances of a component which have reached their target are blocked from competing for resources in future, resulting in an overly optimistic estimation of the response time for multiple occurrences of the same action. Instead of modifying the component to become absorbing, an additional absorbing one-run probe component is specified which is tightly coupled to the original component and is used to detect the target condition without affecting the future behaviour of model. The resulting model can be simulated from the ODEs as before, or the time to absorption can be calculated directly from the ODEs to compute the average response time.

Steady state solution

If the model ergodic, then the population will reach a steady state as $t \rightarrow \text{infinity}$ and $\frac{d}{dt}C \rightarrow 0$ for all derivatives C , as is shown for the example in Section 2.3.3. This arises when the rate of flow into each derivative matches the rate of flow out of each derivative. In order to find the steady state population in general, we can therefore set each ODE to be zero and solve the resulting set of simultaneous equations, together with the limiting equations for the total populations within each component. The complication relative to standard elimination methods for linear solvers is that the set of ODEs will contain one or more minimum function arising from shared actions. Thus, for each minimum function we get two possible conditions, arising from either term being the minimum value. The steady state solution to the ODEs is then found when all these conditions converge as $t \rightarrow \text{infinity}$ (which requires another set of simultaneous equations). However, a proof of convergence is not given in [71]. Furthermore, it is shown in [71] that the presence of multiple minimum functions can affect the agreement between the steady state solution found by solving the ODEs compared to that found by solving the CTMC directly. The PEPA Eclipse plug in does not find the steady state solution in this theoretical way. Instead, the set of ODEs is simulated as described previously until all solutions are found to be unchanging (subject to a degree of tolerance) or the specified duration of simulation time has been reached (in which case no solution is found).

2.3.6 Alternative approaches for performance evaluation

A variety of approaches have been used to evaluate system performance. The following summaries exemplify some of them.

PRISM [45] is a tool for checking probabilistic models. It was developed at the University of Oxford. The approach enables modelling and analysis of the performance of systems with random or probabilistic behaviour from a wide range of application domains, including computer networks, communication systems and distributed protocols. The tool offers a graphical user interface that incorporates a model editor, simulator and graph-plotting tools for modelling and analysing a system. It supports the construction and analysis of a variety of different types of probabilistic models, including CTMC, discrete time Markov chains (DTMC) and the Markov decision processes. It can be used to perform transient and steady-state analyses.

Möbius [20, 21] is an integrated multi-formalism and multi-solution approach. It is primarily designed to support the evaluation of performance and dependability. The tool can support a wide range of modelling formalisms, including Petri nets, Markov chains, and stochastic process algebras. It applies numerical and analytical techniques to study Markovian

models and discrete-event simulations, utilising a well-defined abstract functional interface that enables the models and their solvers to interact. Möbius is capable of constructing a CTMC based on a high-level stochastic model, and analysing it, and also derives the performance metrics for the model. The tool includes a number of analytical solvers that can perform steady-state and transient analyses, utilising diverse of linear algebra techniques.

SMART tool [17, 18] is a stochastic modelling and analysis software tool designed to solve the large models, which are necessary to tackle practical systems. It supports multiple formalisms to define the systems under investigation, such as Petri nets representing a high-level and DTMCs and CTMCs as a low-level. The tool enables the construction of parametric models from which a number of performance metrics can be derived. Although it lacks a graphical user interface for either input models or output visualisation, its rich textual language supports the construction of complicated parametric models. It also provides numerical solutions and simulations for stochastic timing analyses.

Mercury [67] is another performance modelling tool, and was developed by the MODCS research group. It is typically used to assess the dependability and performance of a variety of systems. It offers a graphical user interface with a model editor and evaluator. Mercury supports the creation, modification, and evaluation of performance models, including the CTMC and SPN models. The tool enables users to graphically construct models and conduct both stationary and transient analyses.

STORM [22] is a probabilistic model checker tool developed at RWTH Aachen University. The tool supports a variety of Markov model input languages, such as dynamic fault trees and stochastic Petri nets. It can also analyse CTMC and DTMC models.

However, in this thesis, we used PEPA to formally model and analyse the systems. PEPA has a well-established toolset that permits flexibility when analysing the model, by supporting a variety of approaches, as presented in Subsection 2.3.4. It has well-defined semantics that allow us to explicitly define the models for systems of interest. In addition, prior studies relevant to our field have made use of PEPA, such as [83, 85].

2.4 Related work

2.4.1 Classical approaches to security protocol evaluation

The methods used to explore and measure the performance of security protocols and methods are either experimental via measurements-based experimental or a mathematical-based model. This section presents some works to evaluate security protocols and methods following

classical approaches via experimental approaches such as using a testbed and a computer simulation.

Potlapally and Raghunathan [61] propose a measurement-based experimental testbed to evaluate the performance of cryptographic algorithms and security protocol for small hand-held devices. The energy consumption of many cryptographic algorithms is tested based on their three main classes, which are asymmetric, symmetric, and hash. Their evaluation result shows that asymmetric algorithms have the highest energy costs, whereas hash algorithms have the least energy costs. It also presents that the energy consumption of asymmetric algorithms depends on the key size, whereas the energy consumption of symmetric algorithms does not impact by the key size. Also, they study the energy consumption of the secure sockets layer protocol over wireless networks. The energy costs of cryptographic functionalities of the protocol were explored. Their evaluation results illustrate how the additional cryptographic overheads of implementing security protocols cause performance degradation. In addition, they discuss the possible ways to optimise the energy consumption of cryptographic algorithms and security protocols.

Dick and Thomas [24] analyse the performance overhead of the pretty good privacy protocol via measurements-based experiments. The performance measures were computed based on running the protocol on different machines under Fedora Linux and MS Windows XP. The performance costs of the protocol are analysed based on different encryption/decryption algorithms and key lengths.

Furthermore, the performance of encryption/decryption algorithms was studied by Lamprecht *et al.* in [47]. To investigate the performance of the algorithms, they created a test bed using existing Java implementations from two libraries, which are VeriSign's Trust Services Integration Kit (TSIK) and Java Cryptography Extensions (JCE). They conducted a performance comparison of the most commonly used encryption algorithms for online transactions. They focused on the performance cost when employing different encryption algorithms. They measured operation time as the performance metrics when testing each common algorithm, i.e. key generation, encryption and decryption. Their study's findings can aid in selecting a suitable encryption algorithm for an online transaction system and associated applications. However, as their study is not model-based, they indicate how different implementations for the same algorithm could result in different runtimes. Therefore, this might not be accurate enough to generalise from. A more recent study by Alrowaithy [4–6] conducts a more comprehensive set of experiments using different languages and libraries on different scales of devices, focussing on systems with limited capacity.

Oluwaranti and Adejumo [58] use a network simulator tool to study the performance of security protocols across two operating system platforms, which are Linux and Windows.

Their study investigates the behavioural patterns of IPsec and SSL security protocols based on the encryption algorithm, cryptographic methods to provide the optimal platform that has the least performance overhead. They provide scenarios for each protocol to be examined using OPNET, a tool that provides a realistic networks simulation and a performance data collection. They evaluate the performance of the protocols by computing the response time of some services such as Email, HTTP and Remote Login to compare the two protocols.

Moreover, Hirschler and Sauter [36] study the performance overhead of IPsec for resource-limited devices based on low-performance platforms, which are Intel and ARM-based platforms. IPsec is a secure network protocol developed to provide secure communication between two nodes in a network. Their study is measurement-based experiments to compute the transmission and receiving delays based on different operation modes of the protocol. It shows the processing delay costs introduced by the security features of the protocol on devices with limited processing power.

Although the experimental approach can provide useful information, employing a model-based approach can be more tractable and understandable to study the performance costs of security methods and protocols on a system, as it replaces a complex system with a more simplified model that can be easily understood, analysed and modified to investigate the different results that are possible.

2.4.2 Performance modelling of secure protocols

There are number of researchers have studied and measured the performance of security related algorithms and protocols via employing a mathematical-based model. For example, in [78], Wolter and Reinecke studied the interrelation of security with performance, examining how increasing one affects the other in a model-based evaluation using Generalized Stochastic Petri Net (GSPN) formalism and the TimeNET tool. The issue their model explored was the impact of different encryption key lengths on system performance. They hypothesized that, the longer the key is, the more secure the system. This would clearly also mean, the longer the key, the greater the computation cost introduced, resulting in implications for system performance. Therefore, it is important to choose a suitable encryption key length that ensures the system has an acceptable level of security and performance.

Their model was used to quantify some metrics introduced to study system performance, security and the interrelation between them. Throughput metrics was used to quantify performance and the probability that the system is secure was used as the security metric. As their study focused on the interrelatedness of security and performance, they proposed metrics to be used to measure a combination of performance and security. They combined measures of throughput and the probability of the system being secure into a single metric, called

combined performance and security (CPSM), as a way to measure the trade-off between performance and security. They showed that increasing the length of the key results in linearly increasing the probability of the system being secure, and in turn reduces throughput and maximizes the value of the CPSM metric. They were the first to build a simple model to explore the trade-off between performance and security. However, their model was simple and so does not accurately represent a complex system. Moreover, the formalism they used to build their model does not support the modelling of a complex system comprised of interacting subsystems [19].

As in the previous study, Cho *et al.* [16] has proposed a system model based on a stochastic Petri net (SPN). They outline a mathematical model for a secure group communication system in mobile ad hoc networks (MANETs) to analyse performance-security trade-off. The MANETs environment suffers high security vulnerability, as it is an open medium [16]. Additionally, in MANETs, the members of a group communication system can dynamically join and leave the group. Therefore, this type of system has to be protected from inside and outside attackers. In a secure system, any compromised member of a group has to be detected and evicted. Furthermore, the group has to share a group key for encrypting the communication messages that go between them. Therefore, the system under study employs two security techniques, which are intrusion detection systems (IDS) and rekeying techniques. IDS and rekeying techniques maintain secure communication between group members, protecting them from inside and outside attackers, respectively. They studied the impact of three different rekeying protocols, which are Individual rekeying, Trusted and Untrusted Double Threshold-based rekeying (TUDT) and Join and Leave Double Threshold-based rekeying (JLDT).

In Cho *et al.*'s study, the performance of the system is measured by quantifying the average response time for the message transmitted between the group members. Also, the system's security aspect was measured by a Mean Time to Security Failure (MTTSF) metric, which is when a time till the security failure state is reached. In their model, performance is affected by members that joining and leave the group and also by rekeying mechanisms, whereas security is influenced by the rate of compromised and detected members and also how efficient IDS is as a detection tool. The results of their study clearly show how best to provide an optimal setting of security techniques in their system, in order to achieve a satisfactory performance level and guarantee maximum MTTSF value as a security indicator. Moreover, they show that the Individual rekeying protocol achieved the worst value for system response time and MTTSF metrics, whereas the TUDT protocol achieved the best value.

Moreover, Montecchi *et al.* [56] studied the trade-off between scalability, performance and the security aspects of a multi-service web-based platform. Stochastic Activity Networks (SANs) formalism, which is an extension of stochastic Petri nets, was adopted to construct a model of the target system. They measured utilization and response time metrics, as the system performance indicators. The scalability aspect of a system is studied by changing the parameter value, which represents the number of users with access to the system, and then evaluated the performance metrics. They study the impact of two security mechanisms, which are the Intrusion Prevention System (IPS) and input validation mechanism which takes two approaches: blacklist and whitelist.

However, Montecchi *et al.*'s study mainly focuses on studying and exploring the scalability of a system with respect to performance, and the security aspect was considered as having an indirect effect on scalability, through its influence on performance. Therefore, the security aspect was complementary when studied to support quantification of different systems' performance; as security and performance are two aspects of the system that are strongly interrelated. The positive aspect of their developed system model is that it comprises multiple small template models, which together represent the overall system model. The evaluation of that model is based on rearranging small models in different way, to provide different system configurations. The results of their study show that employing the chosen security mechanisms to the studied system would have a significant influence on system performance, which in turn would be expected to influence system scalability.

Furthermore, the security impact on a system's performance can also be studied using the Performance Evaluation Process Algebra (PEPA) formalism. This formalism is employed by Zhao and Thomas [83] to model two security protocols. Zhao and Thomas model Zhou and Gollman's non-repudiation protocols which are a type of security protocol, designed to ensure that when two or more parties interact with one another, no party can have an advantage over the other. In their study, the performance of each protocol is quantified by average response time and average queuing length metrics. They clearly illustrate how those metrics could be influenced by a different population size, which is a number of clients requesting a service from a Trust Third Party (TTP) in the protocols. Further, they offer a performance comparison between the two protocols. This could assist developers to carefully choose between the two protocols.

Elsewhere, Zhao and Thomas studied the performance of a key distribution centre [84]. A key distribution centre is a trusted party that generates a key session for users upon request, in order that they can communicate securely. They proposed a PEPA model to model the interaction between a key distribution centre and the users of the centre so as to evaluate the performance cost of key exchange functions. They trialled their proposed model for

analysing and quantifying two performance measures. The measures were average utilization at the key distribution centre, and the key distribution centre's average response time to users' requests. The security aspect of the system was based on the reality that the more frequently a fresh session key is regenerated, the greater the rate to achieve secure communication [68]. Therefore, the key lifetime can be considered an indicator denoting secure communication, and the longer the key lifetime the less secure the communication. This would increase the demand imposed on the key distribution centre. Therefore, the security aspect is explored by increasing and decreasing the rate of the key usage and rekey requests.

Zhao and Thomas clearly illustrate that the performance of the system is influenced by the security mechanism involving key generating and requesting. The utilization of the system increases when the number of requests to generate a key rises. In addition, average response time increases when requests to generate key increase.

Moreover, Zhao and Thomas [85] conducted a performance study of another type of non-repudiation protocol, called an optimistic fair exchange protocol. This type of security protocol employs a trusted third party, when one or more parties misbehave during their interaction in an e-commerce environment. They proposed PEPA models to investigate the performance costs introduced by the protocol. They modelled the protocol in two ways. First, they modelled the protocol when there was no misbehaviour or problem event during the exchanges between the participants. Then they modelled the protocol with a misbehaviour event from one or more parties during the exchange processes, which required the involvement of a trusted third party to resolve the issue and ensure a fair exchange between the parties. The results of their model evaluation clearly showed that a misbehaviour event leading to TTP involvement increases the performance cost of a system compared to cases in which there is no misbehaviour between the participants. However, they concentrated exclusively on steady-state performance and did not employ the technique of transient analysis.

The influence of different encryption algorithms on a Networked Control System performance is studied by Zeng and Chow in [80]. They proposed a mathematical model to analyse the trade-off between security and performance; evaluating the real-time dynamic performance of Networked Control Systems with respect to adopting Data Encryption Standard (DES) and Advanced Encryption Standard (AES) algorithms with different encryption key lengths. The key lengths represent the security level of the encryption algorithms in their study. Their model was evaluated by developing a Simulink based test-bed in MATLAB. The results of their evaluation show the effectiveness of their proposed model as a means to illustrate the interrelation between providing acceptable performance and establishing

sufficient system security. However, they did not show an optimal balance between security and performance in the system model they considered.

Finally, although many studies have been conducted, more research is needed regarding modelling and investigating the performance cost introduced by security protocols in order to support the development of security protocols that offer acceptable levels of both security and performance. In this thesis, we demonstrate the performance cost introduced by a particular type of security protocol by developing PEPA models to represent the protocol's behaviour and then deriving various performance measures such as throughput and response time using different analytical techniques such as ODE and transient analysis. Consideration of various performance metrics enables in-depth understanding of the security protocol's behaviour in various scenarios and any associated performance costs.

2.4.3 Performance modelling of systems under an attack

Several researchers have studied and measured the performance of a system under an attack. For example, Meng *et al.* studied the performance cost of the security mechanisms of mobile offloading systems under timing attacks in [55]. They proposed a hybrid Continuous-time Markov chain (CTMC) and queueing model to explore the performance cost introduced by the quantified security attributes. They formulate measures that would optimize the trade-off between security and performance in the studied system. The result of their study presents the best rekeying rate that provides the optimal balance between security and performance for their system.

Furthermore, the performance of an email system under three types of attacks was explored by Wang *et al.* in [76]. Their study is a model-based system, which utilizes a queueing network approach. Wang *et al.* proposed a system model comprised of four queueing models to evaluate the performance of an email security system by subjecting it to attacks. One queueing model represents the email information unit and each of the remaining models represent one attack. They present the effects on the system under the following attacks: mail bombs such as denial of service attacks, password cracked and a malicious mail attack that contains, for example, a Trojan Horse. The study evaluates the trade-off between performance and security system according to three proposed metrics: system availability and information leakage probability as security metrics and average queue length as a performance metric. The numerical findings from the study show how effective and efficient their approach is when analysing the security aspect of email systems under attack.

Gelenbe and Wang studied the performance of a warehouse in the presence of a denial of service attack on its webserver [30]. They focused on the economic performance of the

warehouse that sells perishable products. They predicted the income loss which results from such an attack. A queuing theory-based technique was applied in their study and their approach is suggested to be used as an optimization problem to such a system. Zhu and Martinez studied a resilient control problem for linear systems, which is subject to replay attacks [87]. They studied the influence of replay attacks on system stability and performance. Model predictive control method was employed in their study.

Finally, more research is needed regarding modelling and investigating the performance cost introduced by the attacks in a system. It can be used to understand better how the system behaves and adapts in different scenarios (with and without a threat) to remain secure and provide a sustainable performance level.

2.4.4 Evaluation of attacker behaviour

Attack trees and attack graphs are the most popular graph-based representation methods of an attack [46]. They can aid a defender to understand the different ways that an attacker can use to breach the system. Attack tree is introduced by Bruce Schneier [64] to model the different ways to attack a computing system. It has a root node representing the attacker goal and leaf nodes representing the attacks [64]. Figure 2.2 illustrates an example of an attack tree defined by Bruce Schneier. The root node is Open Safe which is the attacker's main goal. The leaf nodes such as Pick Lock and Learn Combo represent the different attacks that the attacker can do to achieve the main goal.

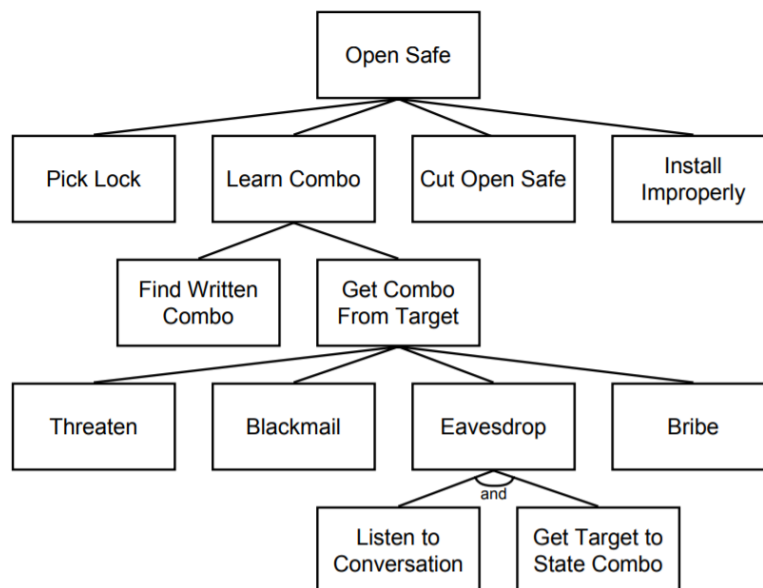


Fig. 2.2 Attack tree [64].

The other popular graph-based representation method of an attack is an attack graph. Attack graph is proposed by Swiler *et al.* [70]. It has nodes representing the possible states of a system and/or a vulnerability and edges representing the transitions between different states due to the attacker's actions moving from node to node [70].

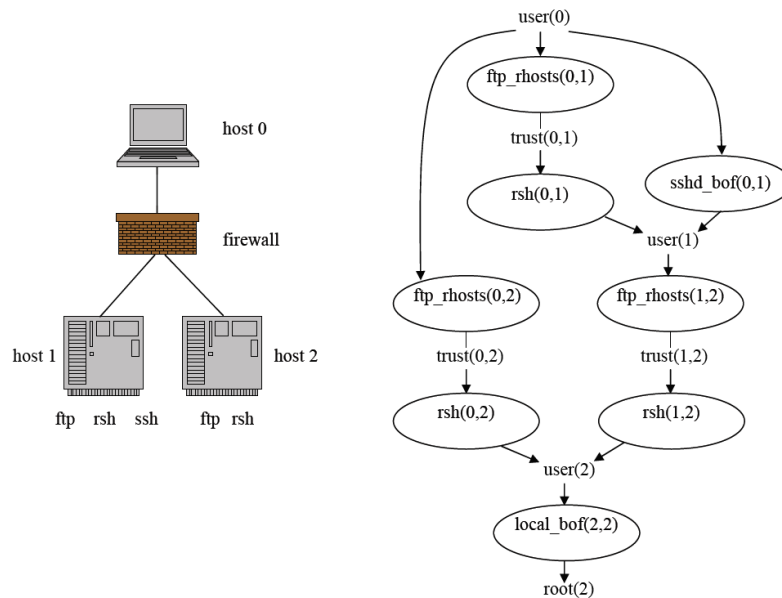


Fig. 2.3 Example of network and attack graph [27].

Figure 2.3 shows an example of a network configuration (left side) and an attack graph (right side) when a malicious workstation user (host 0) tries to get a root access to a database server (host 2), adopted from [27]. The attacker starts the attack from host 0 aiming to get a root access on host 2. In Figure 2.3, the network comprise of 3 hosts and firewall. Host 0 is a user workstation. Host 1 is a file server offering file transfer protocol (ftp), secure shell (ssh), and remote shell (rsh) services. Host 2 is an internal database server offering ftp and rsh services. Firewall in the network permits the traffic of ftp, ssh and rsh from host 0 to other hosts. The initial state of the attacker and the starting point is as a normal user in host 0 ($user(0)$). The target and the final state of the attacker is as a root user having full privileges in host 2 ($root(2)$).

The methods can present one, two or more alternative attack paths to achieve the attacker goal and/or compromise a system [46]. The attack paths are the different sequence of steps that attacker can do to exploit the vulnerabilities existed in the system and to reach the attacker's main goal. In Figure 2.2, the attack tree has eight possible paths to achieve the attacker's main goal, such as the attacker can pick the lock to open a safe. In Figure 2.3, the attack graph has three possible paths for the attacker to follow in order to achieve root

access on the database server (host 2). For example, one possible attack path is *sshd_bof(0,1)* -> *ftp_rhosts(1,2)* -> *rsh(1,2)* -> *local_bof(2,2)*. The attacker is a malicious user in host 0 and can run an ssh buffer overflow attack (*sshd_bof(0,1)*) from host 0 to host 1. Then, the attacker can exploit the ftp vulnerability (*ftp_rhosts(1,2)*) to be able to edit the trusted hosts list on host 2 from host 1. After that, he is able to use the new trusted hosts list to remotely run shell commands (*rsh(1,2)*) on host 2 from host 1. Then, he can run the buffer overflow attack (*local_bof(2,2)*) on host 2 to get a root access to host 2 (the database server).

By presenting the possible attack paths, the defender can see the possible attacker's journeys to achieve its goal and/or compromise a system. So that the defender can evaluate the security status of a system and analyse the attacks from the attacker's perspective. The defender can also analyse and predict the attacker's action and behaviour.

Moreover, each node in the graphs can be assigned a weight representing cost, risk, or the likelihood of the vulnerability to be breached [73]. The weights can be used to evaluate the security level of a system. They can be used by the defender to calculate, for example, the attack probability of each path [69] which it is important to prioritise the countermeasures and make attacks more harder for the attacker to reach its final goal.

Researchers have done many works related to study and predict the attacker behaviour and action using attack tree and attack graph to aid a defender in making a security-related decision. Yousefi *et al.* [79] propose a novel algorithm to create a transition graph from an attack graph to clearly present the possible movements of an attacker between the vulnerabilities that existed in the system. Then they use a Markov model to predict the behaviour of an attacker. They calculate the transition probabilities based on the Common Vulnerability Scoring System (CVSS) score. Then they use the Absorbing Markov Chain to predict the behaviour of the attacker. Their results present three key information about an attacker behaviour. First, they clearly present the number of steps the attacker needs to achieve its final goals. Second, their results show the number of the attacker's visit for each node when the attacker starts from a specific node. Third, they present the probability that the attacker succeeds and achieves its main goals when starting from a specific node in the graph. Their evaluation results can provide important information that can support the defender on the security-related decision such as implementing security measures.

Buldas *et al.* [14] propose a model for the attacker's decision-making process and then use an attack tree to analyse the process. They employ the game-theoretic paradigm to build the model. In their study, they consider an attacker as a player in the attack game. They assume that the attacker has a rational behaviour which means that the attacker will attack a system if the cost is less than the gained benefit and chooses the most profitable attacks. They also consider multiple parameters in their analysis of the attacker's action and behaviour,

such as the attacker's gains, the success probability of the attack, the attack's cost, penalties and the caught probability of the attacker. Their study and analysis of the attacker's action are based on parametrising an attack tree with these parameters to deduce the success probability and the cost of attacks.

Lenin *et al.* [49] introduce a new security analysis tool called ApproxTree+ based on attack tree and attacker profile. They augmented the power of the attack tree with the attacker profiling. The attacker profiling is to identify the attacker's skill and the sufficient tools and resources that support the attacker to exploit a vulnerability [70]. They made an attacker profile explicit by including the attacker's skill, budget, and available time to the threat modelling to make it possible to identify the attacker's possible steps and the attack's cost. Their study made a clear difference between the vulnerability and threat landscape by employing attacker profiles.

Beek *et al.* [10] propose a framework of a graphical security model based on combining an attack tree with a behavioural model for quantitative security modelling. In their study, the attack tree represents the vulnerabilities that existed in a system and the behavioural model represents the attacker's steps to attack the system successfully. They present the probability of each attack succeeding from the first try and the average number of steps. Also, They show the probabilities that each attack succeeded over time.

Zheng *et al.* [86] propose a quantitative method based on an attack graph and the international standard CVSS to measure network security risk level and quantify the maximum reachable possibility of nodes and the importance of nodes. The evaluation of their model can support security professionals to provide a suitable countermeasure to the network. Moreover, Sun *et al.* [69] propose Network Security Risk Assessment Model (NSRAM) based on attack graph and Markov chain to provide the optimal attack path. Their proposed model generates the attack graph, and then each vulnerability is assigned attack probability from the international standard CVSS. Then they use a discrete Markov chain method to calculate the transition probability from node to node. The enterprise network security risk level and the attack probability of each path are provided using their proposed model. Their analysis is based on counting the steps till the system compromised. However, estimating the actual time to compromise the system is critical to indicate how much a safe time the system has before it is compromised.

Kotenko and Stepashkin [43] propose a new network security analysis approach. Their approach is to generate an attack graph based on simulating the attacker's actions and behaviour and then computing different security metrics based on CVSS score. The generated attack graph shows the possible attack scenarios considering the attacker's skill and location,

the configuration of the network, and the used security policy. The result of calculating the metrics support the defender to improve the security level of the system.

Wang and Liou [75] propose an approach for predicting the attack strategy of the attacker based on an attack graph to support a defender to understand the attacker behaviour and be a step ahead of the attacker to be better prepared for the attack and defend the system more efficiently. Their approach appends the attack graph with probabilities based on the vulnerability database (CVE) and the vulnerability scoring system (CVSS). Then, it matches the alerts from the intrusion detection system with a specific related node in the attack graph. Then, the probabilities of the nodes are updated and propagated from the matched node to the goal node based on that to dynamically update the prediction of the attack strategy to the change on the current security status of the system.

Durkota *et al.* [26] present a framework based on a game theory and attack graph to model the attacker's behaviour and the interaction between a defender and an attacker in order to compute the optimal defence strategies. The attacker's possible steps and actions are provided based on the attack graph. They compute the optimal attacker's policies by using Stackelberg equilibrium, mixed-integer linear programming and Markov decision process. Their results can support a defender in the decision-making process to defend the system against complex attacks.

In this thesis, we provide approaches that are based on the PEPA modelling formalism and attack graphs to model the attacker's behaviour and the interaction between a system and an attacker. The PEPA model allows us to perform dynamic analysis to identify the critical threats and estimate the attacker's time to compromise a system. The results can help a defender prioritise countermeasures.

The examples of attack graphs that we considered in Chapters 5 and 6 are all cycle free. However, the approach should be able to handle cycles but we have not investigated that in this thesis and so it remains as future work. Work has been done by Matthews *et al.* [53] on handling cycles in the attack graph without considering time.

2.4.5 Stochastic models of attacker behaviour

Modelling the interaction between an attacker and a system and/or defender is important to deeply understand the attacker behaviour to effectively defend a system, evaluate the security of a system and prioritise countermeasures.

Zhang *et al.* [81] propose two stochastic models for attacker behaviour based on attack graph using Markov Decision Process. The two models were used to assess network security at two stages: design and defense.

Katipally, Yang and Liu [41] introduce an approach to predict and analyse the attacker behaviour using a stochastic model called Hidden Markov Model (HMM). They represent an abstract model of multistep attack. They collect previously learned alerts and intrusions information to be used as an input to train the HMM model to predict the attacker's behaviour effectively. Furthermore, they analyse the optimal possible attack sequence and determine the type of attack by using the clustering algorithm and HMM. Their study explains how their proposed approach can profile attackers into different groups based on their goals, intention, and level of expertise to estimate the attack risk level.

Krautsevich *et al.* [44] present their initial ideas to model an attacker's behaviour based on Markov Decision Processes theory to allow understanding the attacker's behaviour. The models present the possible steps and actions that an attacker could take to attack a system depending on the attacker's resources and knowledge of the system. This can support the defender to prevent the most likely attacks. Their analysis is based on the attacker's view of a system.

Almasizadeh and Azgomi [3] propose a state-based stochastic model to model the attacker actions and defender's reactions. Their focus is to model an attack process over time. They provide an abstract state-transition diagram and then assign time distribution to the transitions to translate the diagram into a state-based stochastic model. In their model, the activities of the attacker and defender are implicitly represented in the model's transitions. Then, they computed security quantitative metrics: mean-time to security failure and steady-state solution to represent the system security status.

Ibidunmoye, Alese and Ogundele [37] propose a stochastic model based on a game-theoretic approach to model the interaction between the attacker and defender. They developed a zero-sum stochastic game for the interaction. They explained how the results could be used to predict the attacker's actions, identify the most network's vulnerable assets, and suggest the best defence strategies. However, their study could not provide analysis on the attacker's behaviour and how the attacker exploits the vulnerabilities in the network.

Sedaghatbaf and Abdollahi Azgomi [65] introduce a probabilistic attack modelling method based on a hierarchical and coloured extension of stochastic activity networks (HCSAN). In their approach, an attacker behaviour is modelled as a strategic decision-making process considering the attack's goal, the costs of the attack, and the system's defence strategies. In their paper, they show how their proposed method can be used to compute the attack success probability and Mean-Time to Security Failure (MTTSF) measures.

Abraham and Nair [1] propose a non-homogenous Markov model using an attack graph to assess the security state of the network by considering the temporal factor associated with the vulnerabilities. They focus on their security analysis to estimate Expected Path

length, Probabilistic Path and Expected Impact metrics. In addition, Abraham [2] proposes a stochastic model based on a non-homogenous continuous-time Markov model to estimate the overall mean time to compromise by considering the casual relationship between the vulnerabilities in the attack graph and the attacker skill. In their model, the coefficients of different attacker skills were estimated by analysing 15 years of vulnerability data from the National Vulnerability Database (NVD). His results show that when the attacker has a higher skill to compromise the system, that causes the time to compromise the system to reduce.

Kaluarachchi, Tsokos and Rajasooriya [40] propose a stochastic model using a Markovian approach to predict the Expected Path Length, which is the number of attacker's step to reach the final goal. Also, they provide the minimum attacker's steps to reach the final goal. Their finding is used to support security administrators to understand their system security's status clearly.

Jhawar, Lounis and Mauw [39] introduce a new method for evaluating the security of attack-defense trees based on Continuous Time Markov Chains (CTMC). Attack-defense trees are a visual representation of security scenarios. They systematically represent the various actions that an attacker can take to achieve a security goal, as well as the various actions that a defender can take to prevent the attacker from achieving its goal. To conduct a quantitative security analysis, attack-defense trees uses a bottom-up approach which assumes that all (attacks and defenses) actions are independent [39]. This is unrealistic because usually the actions are dependent on one another [39]. Their proposed method solves this limitation and considers the actions (attacks and defenses) dependencies. They convert attack-defense trees to Continuous Time Markov Chains (CTMC) representing attack and defense actions to allow performing the security assessments and a continuous-time analysis. Also, they study one type of countermeasure that delay an attack's success. Then, Lounis [51] proposes a new stochastic operational semantics based on stochastic Petri-nets for attack-defense trees for security evaluation. Their proposed stochastic model allows them to conduct qualitative and quantitative security assessments. Recently, Lounis and Ouchani [52] propose a new CTMC model with the objective of representing countermeasures to address attacker attempts in attack-defense trees.

Furthermore, Pokhrel and Tsokos in [60] propose a stochastic model using a host access attack graph to evaluate the overall security risk of a network. Their model is based on Markov chains and host access attack graph with CVSS score of each vulnerability. They introduce a Bias factor to model the attacker's skill. Their result can support network administrators in their decision on prioritising the patching of vulnerable nodes existing in the network.

Deshmukh, Rade and Kazi [23] introduce a framework to model the attacker behavioural aspects to attack a system. They modelled the attacker behaviour based on their proposed Fusion Hidden Markov Model (FHMM). Their approach can help to understand attacker behaviour to support predicting the attacker's future action and then prevent the attacker from compromising a system.

Sadu *et al.* [63] propose a stochastic approach for create Petri Nets model for an attack tree, allowing for the investigation of attacker's attacks. Their work investigates cyber-physical attacks on critical infrastructures. They also address the dependency between transitions in the attack tree in their proposed approach because the attack tree follows a bottom-up approach that implies the transitions are independent in assessing the security status of the system. The stochastic model allows them to estimate the probability that the attacker will reach the attack tree's root node with and without countermeasures, as well as the average time it will take the attacker to reach the root with and without countermeasures. Their approach can aid system designers in assessing the attack's risk which helps to implement appropriate countermeasures.

In contrast to previous studies, we propose methods for converting an attack graph to PEPA models. PEPA supports a compositional, formal and abstract approach to construct a model for an attack graph. The model represented in PEPA involves the attack graph's evolution over time. Other approaches using CTMC have previously been explored [39]. The novelty of our work is that we developed these methods to translate the attack graph to PEPA models, and then used the PEPA tool to explore the time-dependent aspect of these models by performing dynamic analysis. Additionally, the model represents an attacker behaviour and the interaction between an attacker and a system. Evaluating the PEPA models of attack graph can assist the defender in assessing the system's security status and determining where the critical threat exists and where additional countermeasures may be required.

2.5 Context of this thesis

This thesis explores the performance of a web-based sale system in the presence of cyber-attacks in Chapter 3 and studies the performance of a type of e-commerce security protocol in Chapter 4. The proposed PEPA models demonstrate an approach to integrating security and performance concerns in order to gain a better understanding of system behaviour. The performance analysis undertaken indicates where problems may arise and where additional resources may be required.

Moreover, this thesis provides methods to generate PEPA models of the interaction between attacker and system based on a pre-existing attack graph in Chapter 5. It proposes

PEPA models for the attack graph, taking into two criteria: attacker skill and the availability of exploit code to estimate the attacker's time to compromise the system, in Chapter 6. A better understanding of the impact of attackers on a system improves our ability to design systems capable of adapting to and tolerating attacks. This also can help the defender to identify critical threats and prioritise the countermeasures. In Chapter 6, we implement learning behaviours for the attacker and the defender.

Chapter 3 studies the impact and cost that cyber-attacks contribute to system performance. We proposed PEPA models of web-based sale system in two scenarios, with and without the presence of denial of service attacks, to better understand how the system behaves in different scenarios to provide a sustainable level of performance.

Chapter 4 explores the performance cost introduced by a security protocol known as an anonymous and failure resilient fair-exchange e-commerce protocol. The proposed PEPA models were formulated in two different ways: with and without an anonymity feature. Moreover, both protocol versions were modelled in two ways: as a basic protocol with no misbehaviour of any parties whereby it does not require the active involvement of TTP, and as an extended protocol whereby the TTP's participation is essential to resolving disputes between participants. These enable understanding the protocol's behaviour and evaluate the performance cost it introduces.

Chapter 5 provides two methods to automate the generation of the PEPA model based on a pre-existing attack graph specification. The first method generates a PEPA model that comprises one sequential component and system equation. This component represents the system and attacker coupled together. The second method generates a PEPA model that comprises two sequential components and the system equation. One component represents a system or network, and the other component represents an attacker. We used Java to implement the algorithms; then, we generated PEPA models for a case study. Furthermore, we demonstrated through the case study how we used the generated PEPA models to do a sensitivity analysis and deduce the most and the least system security threatening paths and time to compromise, which the defender can use as an indicator of how much a safe time the system has before it is compromised. In addition, the time taken to compromise a system for each attack path resultant from evaluating the model can rank the risk of all attack paths and help the defender prioritise the countermeasures.

Chapter 6 proposes PEPA models for the attack graph, taking into account three different skills for the attackers: beginner, intermediate and expert. We considered two criteria: attacker skill and the availability of exploit code to estimate the attacker's time to compromise the system. Also, we implemented learning behaviours in the model for the attacker and the

defender. We illustrated how learning behaviour for both the attacker and the defender would impact the time to compromise the system.

Chapter 7 presents the conclusion of this thesis. we summarise our contributions, limitations of our research and our future study.

2.6 Chapter summary

This chapter presents performance modelling and PEPA formalism that is used to model the systems in this thesis. It also discusses a sample of existing performance measurement studies designed to test the impact of security algorithms and protocols on systems performance. It also presents some works done on the performance of systems under attack and evaluation of attacker behaviour using attack tree and attack graph.

Chapter 3

Performance modelling of the impact of cyber-attacks on a web-based sales system

3.1 Introduction

The growing demand for online sale systems has led to the development of many web-based sale systems. The online environment suffers from a high-security vulnerability, as it is an open medium, which can expose such a system to threats and attacks such as a denial of service attack. A denial of service attack is a cyber-attack in which an attacker floods a targeted victim with messages to temporarily or permanently prevent legitimate users from accessing resources [7].

Performance has been seen as an important aspect of evaluating such systems. The impact and cost that cyber-attacks contribute to the system's performance need to be studied. By modelling the performance of such a system, we can better understand the system's behaviour in different scenarios, with and without attacks, to enhance both the security and performance levels of the system. Although several researchers have studied the performance of a system under an attack [30, 55, 76, 87], further research is required in terms of modelling and investigating the performance cost imposed by attacks. It can be used to understand better how the system behaves and adapts in different scenarios, such as with and without the presence of a threat, to remain secure and provide a sustainable performance level.

In this chapter, the description of the system we study is based on Gelenbe and Wang [30]. Gelenbe and Wang studied the performance of a warehouse in the presence of a denial of service attack on its webserver by applying an analytical modelling approach using a queuing theory-based technique [30]. In their study, they focused on the economic performance of a warehouse that sells perishable products. Their study focused on exploring and predicting

the income loss which results from such an attack. Their approach is suggested to be used as an optimization problem to such a system.

In this study, we use a different approach, using PEPA, to study the system. This approach offers a formality feature to model a system, as we mentioned in Section 2.2 in Chapter 2. This feature is not available in the approach that Gelenbe and Wang [30] used. We propose models of a web-based sale system in two scenarios, with and without the presence of denial of service attack messages, in order to be able to assess the impact of the attack on the system performance. Then, we perform an analytical technique on the proposed models and derive some performance metrics such as throughput and population in order to demonstrate how an attack on a warehouse's webserver, which customers use to place orders, affects the warehouse's performance, how the attacks prevent some or all customer orders from being fulfilled, and how the delay in selling products results in products being discarded. In addition, we illustrate how such an attack on the webserver disrupts warehouse sales.

3.2 Web-Based Sales System

3.2.1 System specification

In this section, we provide the description of the web-based sales system based on Gelenbe and Wang [30]. The description of the system with no attack is as follows (the words in bold are the actions name that we used in our proposed PEPA model):

- A warehouse has an online webserver to sell perishable products.
- Perishable products have a limited shelf life.
- Products arrive from supplier to warehouse at a rate s_1 (**objectArrive**).
- The warehouse's webserver receives the orders from customers at rate r_1 order per time unite (**order**).
- Webserver forwards the successful order messages to the warehouse at rate r_2 (**forwardOrder**).
- The expiry date of perishable products will be checked periodically in warehouse (**checkObject**).
- The two ways to remove the products from the warehouse are:
 - Products are removed due to their perishability (**removeExpiredObject**).
 - Products are removed after they are successfully sold (**removeSoldObjects**).

- If the warehouse does not have the product that the customer ordered, then the warehouse will order it from a supplier and pay for it (**requestObject**, **payForObject**). Then the product will be delivered to the warehouse (**objectArrive**).
- We assume that the warehouse already has some products.

3.2.2 PEPA model of the system without attacks

This section presents the proposed PEPA model for the web-based sales system without the presence of an attack. The model comprises five parts, one for each component: Customer, Webserver, Warehouse and Timer, which move sequentially between their different behaviours based on the activities specified in the model. The model is formulated as follows:

Customer component

$$\begin{aligned} Customer_0 &\stackrel{def}{=} (order, c_1).Customer_1 \\ Customer_1 &\stackrel{def}{=} (orderSucessfullyPlaced, r_1).Customer_2 \\ Customer_2 &\stackrel{def}{=} (complete, c).Customer_0 \end{aligned}$$

This part of the model specifies the customer's different behaviours, moving from $Customer_0$ to $Customer_2$. The first state is $Customer_0$. It is the state when a customer visits a webserver and performs the action $order$ at rate c_1 leading to $Customer_1$. Then, in state $Customer_1$, the only action that happens is $orderSucessfullyPlaced$ at rate r_1 leading to $Customer_2$. $Customer_2$ is the state when the customer performs the action $complete$ at rate c leading back to $Customer_0$, which means that the interaction between the customer and the webserver has finished. After $Customer_2$, the behaviour returns to $Customer_0$ so that the model becomes cyclic and steady-state measures can be obtained.

Webserver component

$$\begin{aligned} Webserver_0 &\stackrel{def}{=} (order, c_1).Webserver_1 \\ Webserver_1 &\stackrel{def}{=} (processingOrder, r_3).Webserver_2 \\ Webserver_2 &\stackrel{def}{=} (orderSucessfullyPlaced, r_1).Webserver_3 \\ Webserver_3 &\stackrel{def}{=} (forwardOrder, r_2).Webserver_0 \end{aligned}$$

The above component represents the webserver's different behaviours, moving from $Webserver_0$ to $Webserver_3$. The first state is $Webserver_0$. When webserver in the state $Webserver_0$, the webserver performs the action $order$ to receive an order from a customer at rate c_1 leading to $Webserver_1$. Then, in the state $Webserver_1$, the only action that happens is $processingOrder$ at rate r_3 leading to $Webserver_2$. In the state $Webserver_2$, the only

orderSuccessfullyPlace can occur at rate r_1 leading to *Webserver₃* which is the state when the webserver forwards the successful order to the warehouse at rate r_2 .

Warehouse component

$$\begin{aligned}
\text{Warehouse}_0 &\stackrel{\text{def}}{=} (\text{forwardOrder}, r_2). \text{Warehouse}_1 + (\text{checkObject}, w_8). \text{Warehouse}_8 \\
&\quad + (\text{objectArrive}, s_1). \text{Warehouse}_0 \\
\text{Warehouse}_1 &\stackrel{\text{def}}{=} (\text{objectExist}, w_1). \text{Warehouse}_5 + (\text{objectNotExist}, w_2). \text{Warehouse}_2 \\
\text{Warehouse}_2 &\stackrel{\text{def}}{=} (\text{requestObject}, w_3). \text{Warehouse}_3 \\
\text{Warehouse}_3 &\stackrel{\text{def}}{=} (\text{payForObject}, w_4). \text{Warehouse}_4 \\
\text{Warehouse}_4 &\stackrel{\text{def}}{=} (\text{objectArrive}, s_1). \text{Warehouse}_5 \\
\text{Warehouse}_5 &\stackrel{\text{def}}{=} (\text{addObjectToOrderList}, w_5). \text{Warehouse}_6 \\
&\quad + (\text{addObjectToOrderList}, w_6). \text{Warehouse}_1 \\
\text{Warehouse}_6 &\stackrel{\text{def}}{=} (\text{removeSoldObjects}, w_7). \text{Warehouse}_7 \\
\text{Warehouse}_7 &\stackrel{\text{def}}{=} (\text{sendOrderToCustomer}, w_{12}). \text{Warehouse}_0 \\
\text{Warehouse}_8 &\stackrel{\text{def}}{=} (\text{expired}, w_9). \text{Warehouse}_9 + (\text{notExpired}, w_{10}). \text{Warehouse}_0 \\
\text{Warehouse}_9 &\stackrel{\text{def}}{=} (\text{removeExpiredObject}, w_{11}). \text{Warehouse}_0
\end{aligned}$$

The above part of the model is for the warehouse component. In the state *Warehouse₀*, one of three actions could happen either *forwardOrder* at rate r_2 leading to *Warehouse₁* if warehouse receives an order from the webserver, *checkObject* at rate w_8 leading to *Warehouse₈* when warehouse checks periodically the expiry date of the products, or *objectArrive* at rate s_1 when some products arrive in the warehouse from a supplier leading back to *Warehouse₀*. In the state *Warehouse₁*, either actions could be performed *objectExist* at rate w_1 leading to *Warehouse₅* or *objectNotExist* at rate w_2 leading to *Warehouse₂* in order to request the product from a supplier. In the state *Warehouse₂*, the only action that happens is *requestObject* at rate w_3 leading to *Warehouse₃*. Then, in the state *Warehouse₃*, there is only one action that could happen which is *payForObject* at rate w_4 leading to *Warehouse₄* which is the state when the warehouse receives the object from the supplier after performing the action *objectArrive* at rate s_1 leading to *Warehouse₅*. In the state *Warehouse₅*, one of two actions could be performed either *addObjectToOrderList* at w_5 leading to *Warehouse₆* in order to remove these products from the warehouse or *addObjectToOrderList* at rate w_6 leading to *Warehouse₁* when there is more than one product in the order. In *Warehouse₆*, the only action that happens is *removeSoldObjects* at rate w_7 leading to *Warehouse₇* which is the state when the warehouse sends the order to the customer by performing action *sendOrderToCustomer* at rate w_{12} leading back to *Warehouse₀*. In *Warehouse₈*, one of two actions can be performed either *expired* at rate w_9 leading to *Warehouse₉* in order to remove the expired product from the warehouse or *notExpired* at rate w_{10} leading back to *Warehouse₀*. Finally, in the state

Warehouse₉, the only action that happens is *removeExpiredObject* at rate w_{11} leading back to *Warehouse₀*.

Timer component

$$\begin{aligned}
 Timer_0 &\stackrel{def}{=} (tick, t).Timer_1 + (notExpired, w_{10}).Timer_0 \\
 &\quad + (sendOrderToCustmer, w_{12}).Timer_0 \\
 Timer_1 &\stackrel{def}{=} (tick, t).Timer_2 + (notExpired, w_{10}).Timer_1 \\
 &\quad + (sendOrderToCustmer, w_{12}).Timer_0 \\
 Timer_2 &\stackrel{def}{=} (expired, w_9).Timer_0 + (sendOrderToCustmer, w_{12}).Timer_0
 \end{aligned}$$

This part of the model is for the timer component, which is the part that counts down the time life of the product in the warehouse. It has three behaviours starting from *Timer₀*. In *Timer₀*, one of three actions can happen either *tick* at rate t to count down the time life of the product in the warehouse leading to *Timer₁*, action *notExpired* at rate w_{10} leading back to *Timer₀* or action *sendOrderToCustomer* at rate w_{12} when the product is sent to the customer before reaching expiry date leading back to *Timer₀*. *Timer₁* state has the same actions as in *Timer₀*. In the state *Timer₂*, one of two actions can be performed either *expired* when the product *expired* at rate w_9 leading back to *Timer₀* or action *sendOrderToCustomer* at rate w_{12} when the product is sent to the customer before reaching expiry date leading back to *Timer₀*.

Supplier component

$$Supplier \stackrel{def}{=} (requestObject, w_3).(payForObject, w_4).(objectArrive, s_1).Supplier$$

The last part of the model is for the supplier component. There is just one state which is *Supplier*. The supplier's main action is *objectArrive*. The rate of this action is controlled by the supplier.

System equation

The system equation and complete specification are given by

$$System \stackrel{def}{=} Customer_0[N] \bowtie_B Webserver_0[N] \bowtie_L Warehouse_0 \bowtie_S (Supplier || Timer_0)$$

Where $B = \{order, orderSuccessfullyPlaced\}$, $L = \{forwardOrder\}$, $S = \{requestObject, payForObject, objectArrive, notExpired, expired, sendOrderToCustmer\}$, any action in list B , L and S is a shared action between the components specified in system equation. N is the number of Customer and Webserver instances in the system. The five components are initially in the states *Customer₀*, *Webserver₀*, *Warehouse₀*, *Supplier* and *Timer₀*.

3.2.3 Performance evaluation and results

The current investigation seeks to calculate the throughput of some main actions of the Webserver and Warehouse which are *orderSucessfullyPlaced* and *forwardOrder* actions of the Webserver, and *removeSoldObjects* and *removeExpiredObject* actions of the Warehouse using the ordinary differential equations (ODEs) method. The throughput is the amount of work that can be accomplished in a specific time [38]. Moreover, we want to investigate the population level of some states of the components using ODE. The population is the average number of the component's copies in a state waiting for action in the model. We apply transient analysis to derive the population level and throughput measures. The accuracy of the model changes with N (the number of customers) and converges as $N \rightarrow \textit{infinity}$ [71, 82, 84]. In this study, we keep the total number of customers to 100.

All the action rates for the customer is 1 except *complete* action is 0.01 to let a customer waits to use the product before starting again. All the action rates of the webserver are 100 except *forwardOrder* action which we assume to be faster and have a rate of 200. They are divided by the number of the customers using the webserver. All the action rates of the warehouse are 100 except *sendOrderToCustmer* action, which we assume to be slower and has a rate value of 24, and the rates of *checkObject* and *objectArrive* actions, which we assume they have rate values equal to *forwardOrder* action's rate to avoid action race as they are in the same state $Warehouse_0$ in the model. The action rate for the timer component that counts down the time life of the products in the warehouse is 24.

Figures 3.1 and 3.2 show the throughput values of the actions *orderSucessfullyPlaced*, *forwardOrder*, *removeSoldObjects* and *removeExpiredObject*. In Figure 3.1, the throughput values of the action *orderSucessfullyPlaced* reach the peak of about 27 and then reach the steady-state at 0.7 at a short period of time. Figure 3.2 clearly illustrate that the throughput values of *forwardOrder* and *removeSoldObjects* are larger than the throughput values of *removeExpiredObject*. This indicates that larger products are sold and less products are discarded when there is no attack in the system.

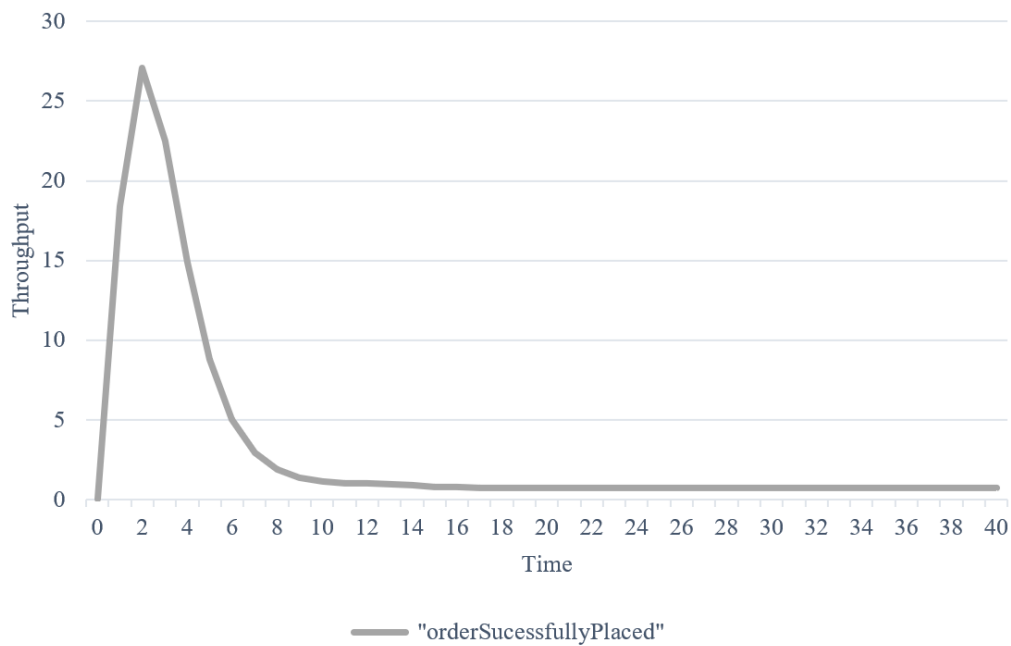


Fig. 3.1 The Throughput Analysis of *orderSuccessfullyPlaced*.

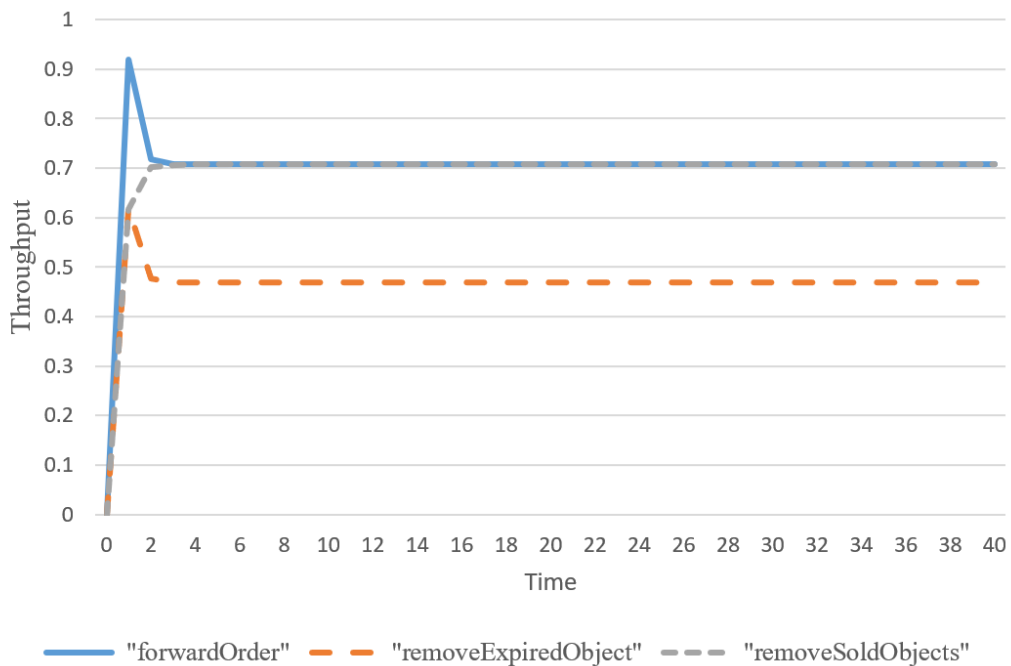


Fig. 3.2 The Throughput Analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects*.

The following two graphs show the population level analysis for the following states: $Customer_1$ (the customer's state when waiting to receive a successful order confirmation

message from the webserver), *Webserver₃* (the webserver’s state when waiting to forward the successful order to the warehouse), *Warehouse₆* (the warehouse’s state when removes sold products from the warehouse) and *Warehouse₉* (the warehouse’s state when removes expired products from warehouse due to their perishability). As you can see from Figure 3.3 and 3.4, all the customers have their orders forwarded to the warehouse. Figure 3.4 illustrates that, in the warehouse, the number of sold products is larger than the number of discarded products when there is no attack.

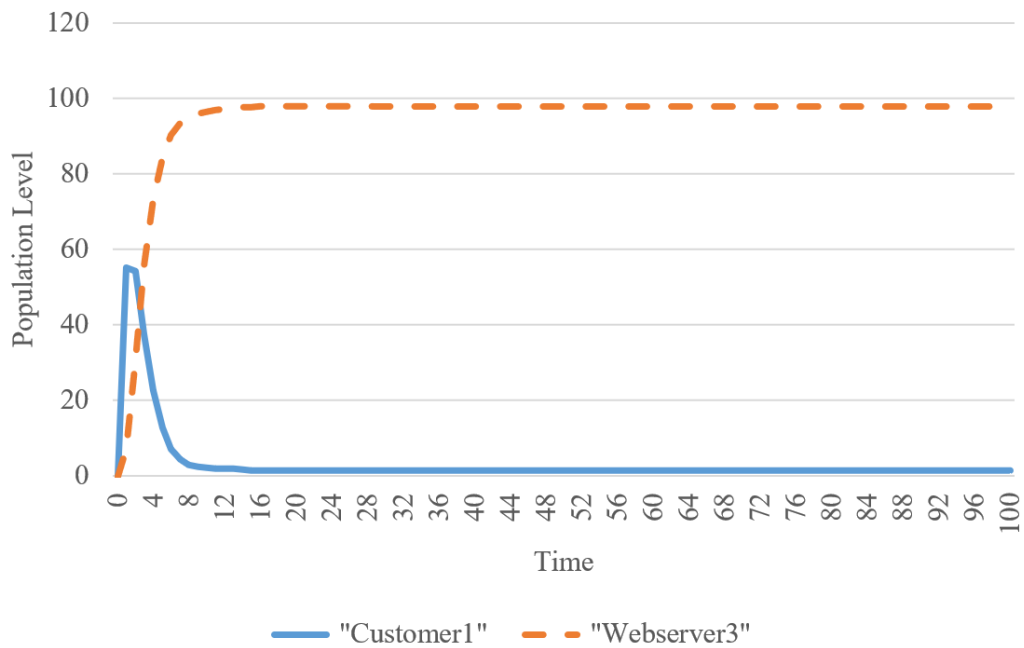


Fig. 3.3 The population level analysis of *Customer₁* and *Webserver₃*.

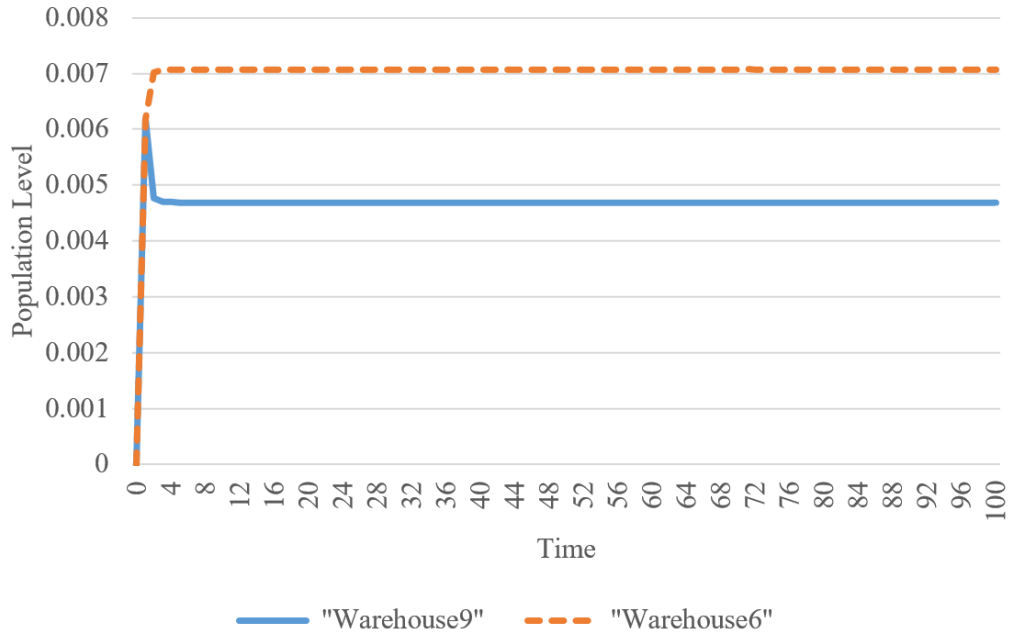


Fig. 3.4 The population level analysis of $Warehouse_6$ and $Warehouse_9$.

3.3 Web-Based Sales system in the presence of the attacks

3.3.1 System specification

In this section, we are going to introduce an attack on the system. The kind of attack that we want to consider is a denial of service attack, whereby an attacker floods the system with bogus requests. The attacker behaves similarly to a negative customer in G-networks [29]. The effect of this increase in traffic is to delay the processing of legitimate orders. If this delay is sufficiently long, orders will fail due to timeouts, and hence revenue will be lost and goods may perish. A real-world denial of service attack may cause systems to overflow, resulting in message loss, although we do not model this aspect of the attack. Our aim in modelling this form of attack is to investigate the relationship between the rate of attack and the impact on order throughput and products perishing in the warehouse.

The system has the same steps as in Section 3.2.1 and the following are steps of introducing the attack to the system as described in [30] (the words in bold are the actions name used in our proposed PEPA model):

- The webserver receives an attack message at some rate a_1 per time units based on their vulnerability to attack message with a probability q (**attackM**).
- Each attack message destroy one processing order (**destroyedOrder**).

- The customer that has a destroyed order could repeat the same order with a probability p (**repeatOrder**).

We assume that the number of attack message is equal to the number of website copy created for each customer. If the website is vulnerable to attack message, the current processing order will be destroyed.

3.3.2 PEPA model of the system in the presence of the attacks

This section presents the proposed PEPA model for the web-based sales system in the presence of an attack. In our PEPA model, there are six types of components: Customer, Attacker messages, Webserver, Warehouse, Timer and Supplier. The model comprises six parts, one for each component. Customer, Attacker messages, Webserver, Warehouse and Timer move sequentially from their different behaviours based on the activities specified in the model. The model is formulated as follows:

Customer component

$$\begin{aligned}
Customer_0 &\stackrel{def}{=} (order, c_1).Customer_1 \\
Customer_1 &\stackrel{def}{=} (orderSuccessfullyPlaced, r_1).Customer_3 \\
&\quad + (destroyedOrder, a_2 * p).Customer_2 \\
&\quad + (destroyedOrder, a_2 * (1 - p)).Customer_3 \\
Customer_2 &\stackrel{def}{=} (repeatOrder, c_2).Customer_1 \\
Customer_3 &\stackrel{def}{=} (complete, c).Customer_0
\end{aligned}$$

This part of the model specifies customer's different behaviours. The first state is the same as the first customer state in Section 3.2.2. Then, in state $Customer_1$, one of three actions can happen either *orderSuccessfullyPlaced* at rate r_1 leading to $Customer_3$, *destroyOrder* at rate $a_2 * p$ leading to $Customer_2$ or *destroyOrder* at rate $a_2 * (1 - p)$ leading to $Customer_3$ which is the state when a customer performs action *complete* at rate c leading back to $Customer_0$ which it means that the interaction between the customer and the webserver has finished. In $Customer_2$, the only action that happens is *repeatOrder* at rate c_2 leading back to $Customer_1$.

AttackerM component

$$\begin{aligned}
AttackerM_0 &\stackrel{def}{=} (attackM, a_1).AttackerM_1 \\
AttackerM_1 &\stackrel{def}{=} (destroyedOrder, a_2).AttackerM_0
\end{aligned}$$

The above component represents the two attacker message's behaviours. The first state is $AttackerM_0$. It is when the attacker sends an attack message by performing *attackM* action

at rate a_1 leading to $AttackerM_1$. The second state is $AttackerM_1$. It is when the attacker message destroys the current processing order based on the webserver's vulnerability to attack message by performing $destroyedOrder$ at rate a_2 leading back to $AttackerM_0$.

Webserver component

$$\begin{aligned}
Webserver_0 &\stackrel{def}{=} (order, c_1).Webserver_1 \\
&\quad + (repeatOrder, c_2).Webserver_1 \\
Webserver_1 &\stackrel{def}{=} (processingOrder, r_3 * q).Webserver_4 \\
&\quad + (processingOrder, r_3 * (1 - q)).Webserver_2 \\
Webserver_2 &\stackrel{def}{=} (orderSucessfullyPlaced, r_1).Webserver_3 \\
Webserver_3 &\stackrel{def}{=} (forwardOrder, r_2).Webserver_0 \\
Webserver_4 &\stackrel{def}{=} (attackM, a_1).Webserver_5 \\
Webserver_5 &\stackrel{def}{=} (destroyedOrder, a_2).Webserver_0
\end{aligned}$$

The above component of the model is for the webserver's different behaviours. The local states increase to five states compared to the webserver component when there is no attack (Section 3.2.2). The first state is $Webserver_0$. When the webserver in state $Webserver_0$, the webserver performs one of two actions either $order$ to receive an order from customer at rate c_1 leading to $Webserver_1$ or $repeatOrder$ at rate c_2 leading also to $Webserver_1$. Then, in state $Webserver_1$, one of two actions can happen either $processingOrder$ at rate $r_3 * q$ leading to $Webserver_4$ or $processingOrder$ at rate $r_3 * (1 - q)$ leading to $Webserver_2$. The states $Webserver_2$ and $Webserver_3$ are the same as the webserver's states in Section 3.2.2. In the state $Webserver_4$, the only $attackM$ can occur at rate a_1 leading to $Webserver_5$, which is when a current processing order is destroyed in the webserver by performing $destroyedOrder$ at rate a_2 .

Other components

Warehouse, Timer and Supplier components have the same behaviours as in Section 3.2.2.

System equation

The system equation and complete specification are given by

$$\begin{aligned}
System &\stackrel{def}{=} ((Customer_0[N] \bowtie_H AttackerM_0[N]) \bowtie_M Webserver_0[N] \bowtie_L Warehouse_0 \\
&\quad \bowtie_B (Supplier || Timer_0))
\end{aligned}$$

Where $H = \{destroyedOrder\}$, $M = \{order, orderSucessfullyPlaced, destroyedOrder, repeatOrder, attackM\}$, $L = \{forwardOrder\}$, $B = \{requestObject, payForObject, objectArrive, notExpired, expired, sendOrderToCustmer\}$ any action in list H , M , L and B is a shared action between the components specified in the system equation. N is the number of Customer, AttackerM and Webserver instances in the system. The six components are initially in the states $Customer_0$, $AttackerM_0$, $Webserver_0$, $Warehouse_0$, $Supplier$ and $Timer_0$.

3.3.3 Performance evaluation and results for the extended model

In this section, we seek to illustrate how the performance of the warehouse's sale would be affected by a denial of service attack by preventing some or all customers' orders from being fulfilled. Then, the delay in selling perishable products would result in products being discarded. So the warehouse would lose customers and waste products which would also affect the warehouse economically. All the same actions as in Section 3.2.2 have the same rate values as specified in Section 3.2.3. However, we assume all AttackerM's actions rate is 4, which is faster than customer's rates. We also assume that a customer with a destroyed order can repeat the same order with a probability of 0.5 (50%).

Our investigation seeks to calculate the throughput of some main actions of *Customer*, *AttackerM*, *Webserver* and *Warehouse* components using ODEs. These actions are *repeatOrder* action of *Customer*, *destroyedOrder* action of *AttackerM*, *orderSucessfullyPlaced* and *forwardOrder* actions of *Webserver*, and *removeSoldObjects* and *removeExpiredObject* actions of *Warehouse*. Moreover, we want to investigate the population level of some states of the components using ODE. We assume all perishable products are fresh at the beginning of the interaction then the Timer component counts down its lifetime.

The throughput analysis

The following graphs, Figures 3.5 to 3.12, show the throughput values of *repeatOrder*, *destroyedOrder*, *orderSucessfullyPlaced*, *forwardOrder*, *removeSoldObjects* and *removeExpiredObject* actions. The throughput values of the actions were calculated based on changing the probability of the webserver to be vulnerable to the attacker's message to 90%, 50%, and then 10%.

Figures 3.5 to 3.12 show how increasing the probability of the webserver being vulnerable to the attacker's message resulted in more orders were destroyed, sale delay and then more products were discarded due to their perishability. The throughput values of *destroyedOrder* and *removeExpiredObject* are larger than *orderSucessfullyPlaced* and *removeSoldObjects*, when the probabilities of the webserver to be vulnerable to the attacker's message are 90%

and 50%. They also show that the larger the probability of the webserver being vulnerable to the attacker's message, the faster the system loses its control to deal with the orders and attacker's messages which then causing a delay in the sale due to preventing some or all customers' orders from being fulfilled and then increasing the number of products that have been discarded.

In Figures 3.5, 3.6 and 3.7, the probability of the webserver to be vulnerable to the attacker's message is 90% ($q = 0.9$). Figure 3.5 shows that the throughput values of *destroyedOrder* start larger than the throughput values of *orderSucessfullyPlaced* for a period of time. This means many orders are prevented from being fulfilled due to the attacks. In Figures 3.6 and 3.7, the throughput values of *removeExpiredObject* start less than the throughput values of *forwardOrder* and *removeSoldObjects* and then start to increase during a short period of time. This indicates that the delay in the sale of the products cause a larger amount of products to be discarded due to their limited shelf life.

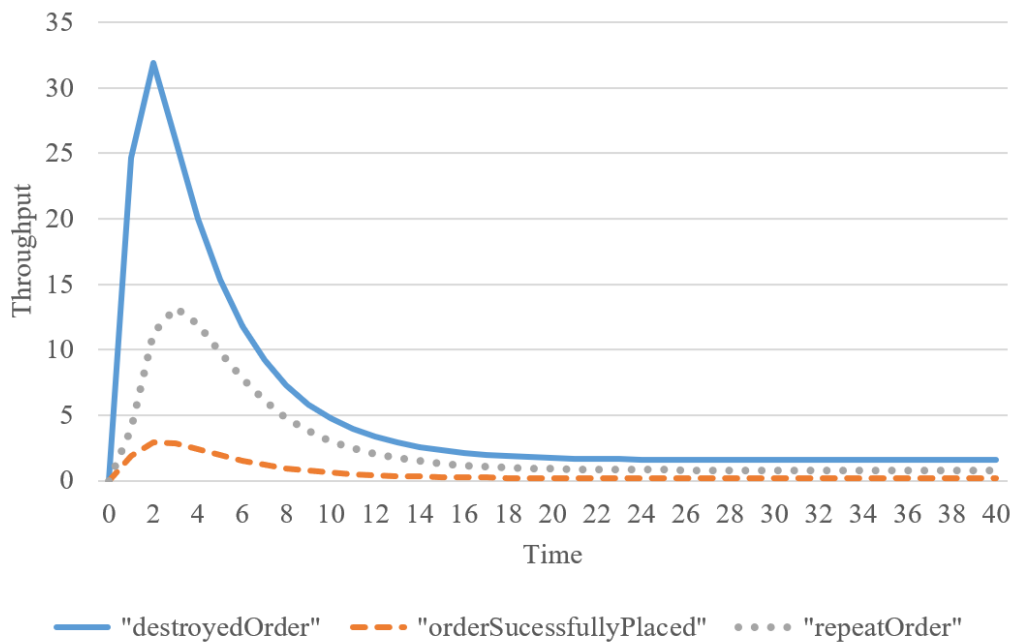


Fig. 3.5 The Throughput Analysis of *destroyedOrder*, *orderSucessfullyPlaced* and *repeatOrder* when $q = 0.9$.

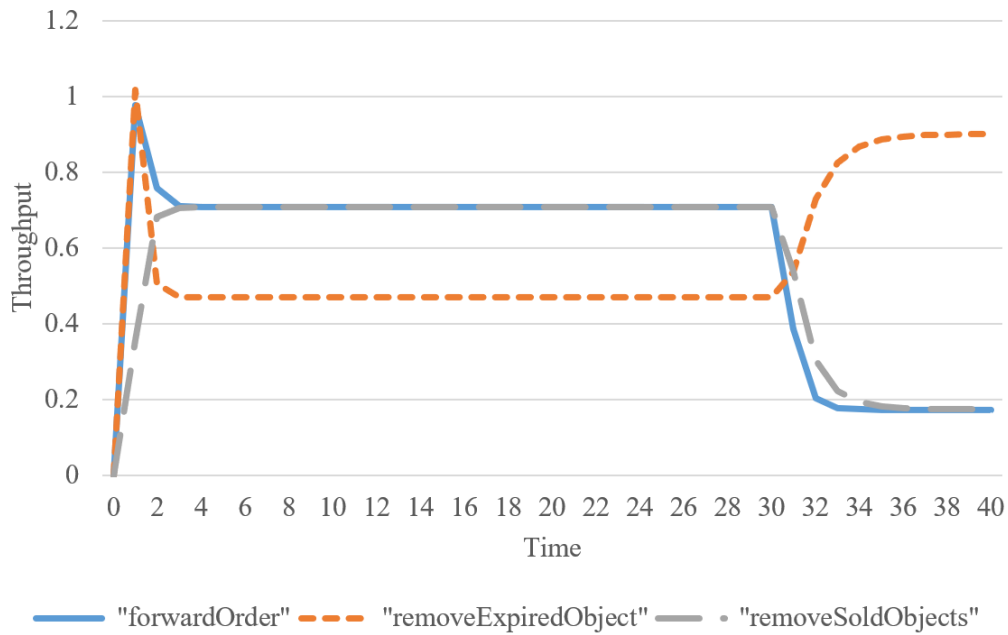


Fig. 3.6 The Throughput Analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* when $q = 0.9$.

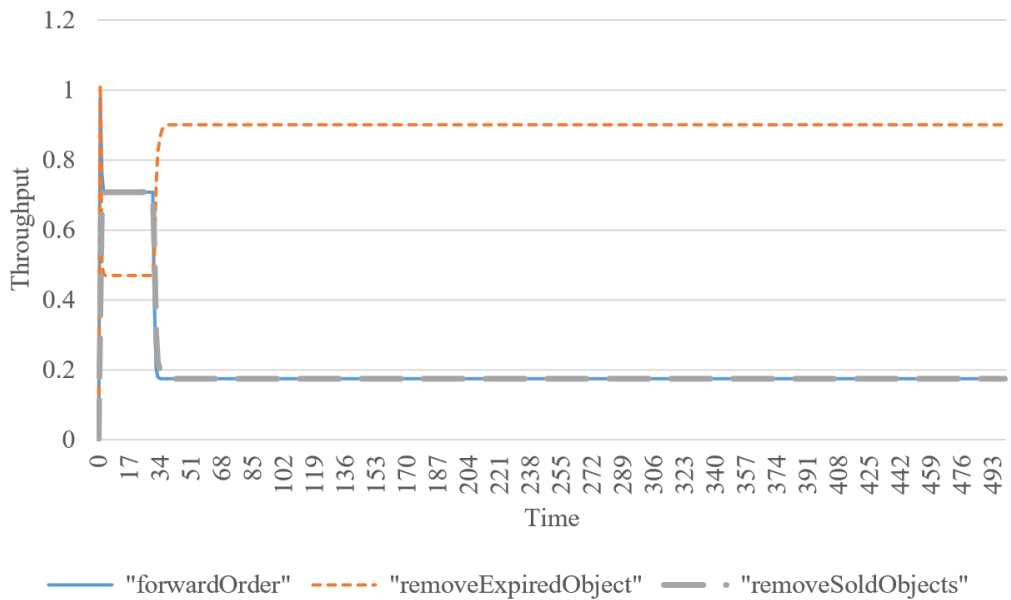


Fig. 3.7 The Throughput Analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* during a longer time when $q = 0.9$.

In Figures 3.8, 3.9 and 3.10, the probability of the webserver to be vulnerable to the attacker’s message is 50% ($q = 0.5$). Decreasing the probability of the webserver to be

vulnerable to the attacker's message causes the throughput values of *destroyedOrder* to decrease and the throughput values of *orderSucessfullyPlaced* to increase as shown in Figure 3.8 compared to Figure 3.5 when the probability is larger. Moreover, when the probability of the webserver to be vulnerable is larger, as in shown Figures 3.6 and 3.7, the system loses control of dealing with customer orders and attacks faster than when the probability is decreased, as shown in Figure 3.10.

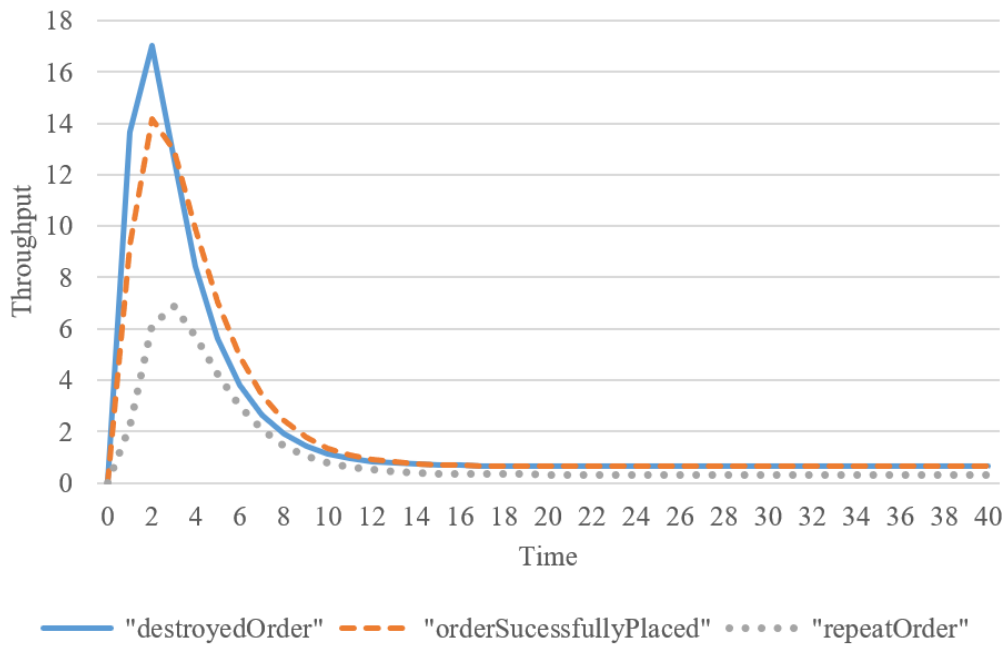


Fig. 3.8 The throughput analysis of *destroyedOrder*, *orderSucessfullyPlaced* and *repeatOrder* when $q = 0.5$.

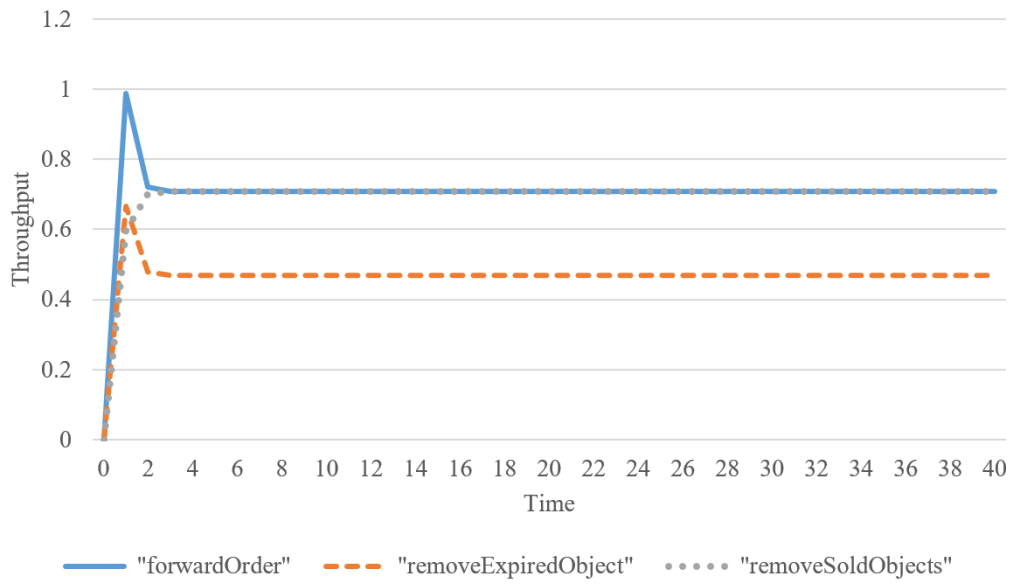


Fig. 3.9 The throughput analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* when $q = 0.5$.

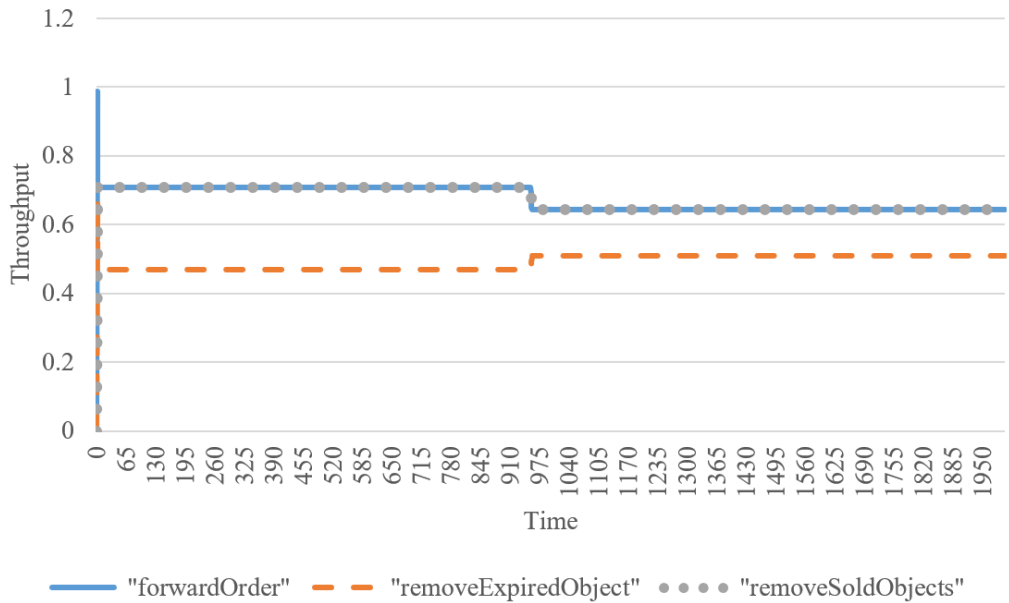


Fig. 3.10 The throughput analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* during a longer time when $q = 0.5$.

In Figures 3.11 and 3.12, the probability of the webserver to be vulnerable to the attacker’s message is 10% ($q = 0.1$). When the probability of the webserver being vulnerable is low, the larger number of customers orders is fulfilled. The throughput values of the

actions *orderSucessfullyPlaced*, *forwardOrder* and *removeSoldObjects* are larger than the throughput value of *destroyedOrder* and *removeExpiredObject*.

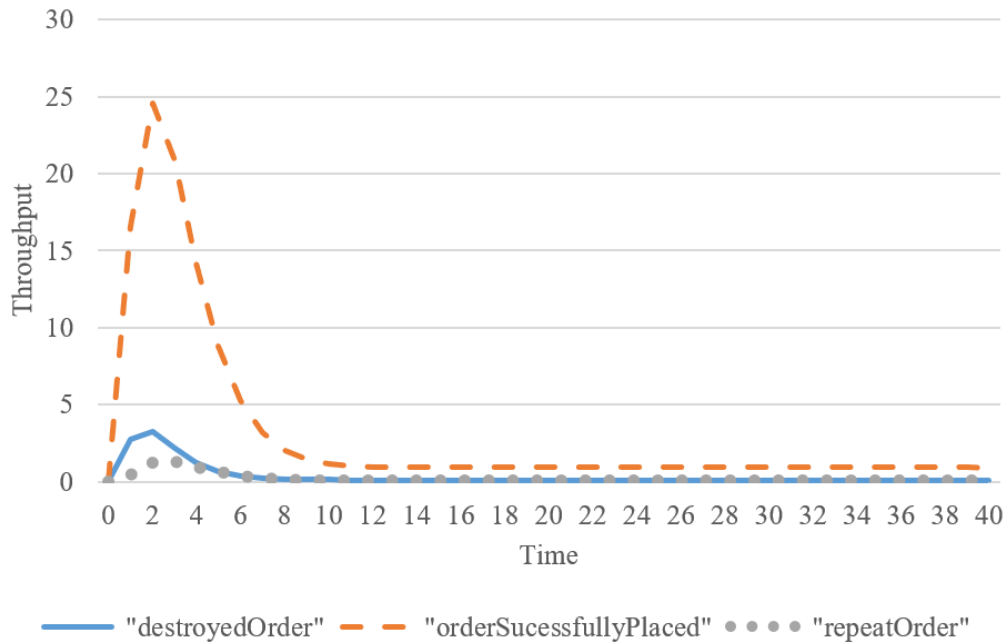


Fig. 3.11 The throughput analysis of *destroyedOrder*, *orderSucessfullyPlaced* and *repeatOrder* when $q = 0.1$.

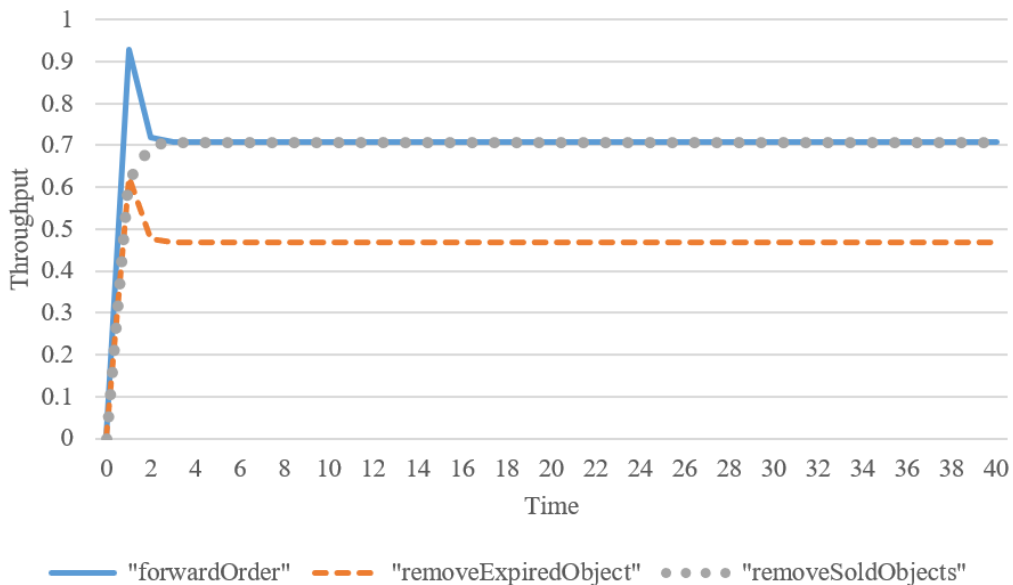


Fig. 3.12 The throughput analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* when $q = 0.1$.

Moreover, Figures 3.13 and 3.14 shows the different throughput values of *destroyedOrder*, *orderSucessfullyPlaced*, *repeatOrder*, *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* actions in relation to different probability values (q) of the webserver to be vulnerable to the attacker’s message. They illustrate how increasing q would have a significant impact on the throughput values of the actions.

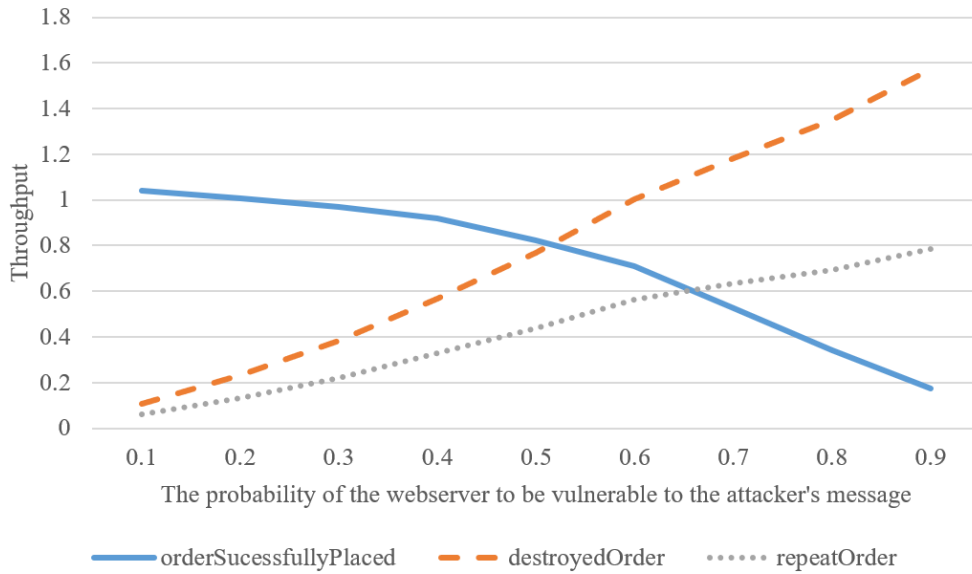


Fig. 3.13 The throughput analysis of *destroyedOrder*, *orderSucessfullyPlaced* and *repeatOrder* in relation to q different values.

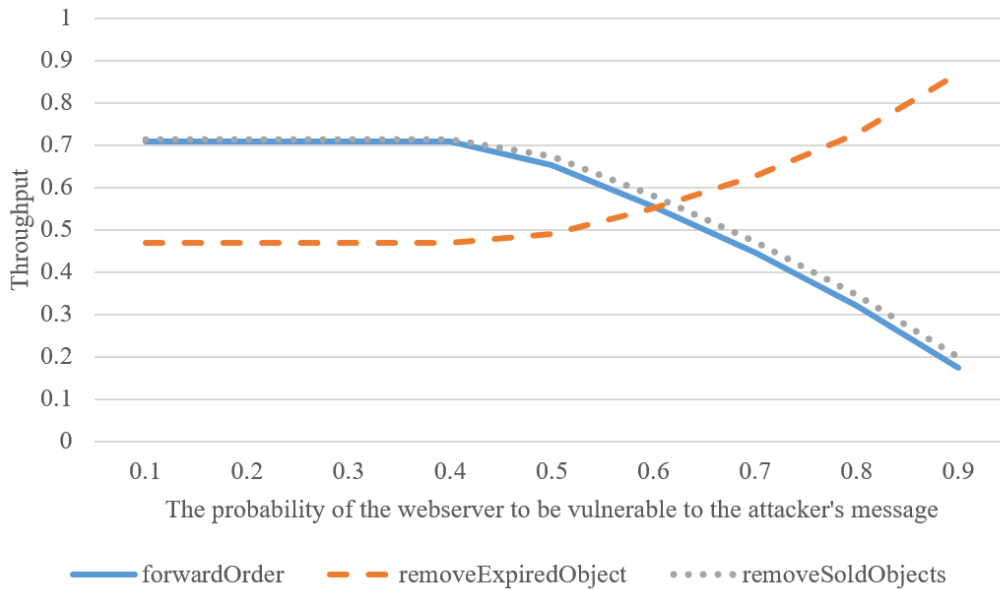


Fig. 3.14 The throughput analysis of *forwardOrder*, *removeExpiredObject* and *removeSoldObjects* in relation to q different values.

The population level analysis

The following graphs, Figures 3.15 to 3.17, show the population level analysis for the following states: *Customer₂* (the customer's state when waiting to repeat their destroyed order), *Webserver₂* (the webserver's state when waiting to forward the successful order to the warehouse), *Webserver₅* (the webserver's state when the order is destroyed), *Warehouse₆* (the warehouse's state when removing sold products from the warehouse) and *warehouse₉* (the warehouse's state when removing expired products from the warehouse due to their perishability). Figures 3.15 to 3.17 illustrate how increasing the probability of the webserver to be vulnerable to the attacker's message causes the average number of *Warehouse₆* and *Webserver₂* copies to decrease and *Warehouse₉*, *Customer₂* and *Webserver₅* to increase. In Figure 3.15, the probability of the webserver being vulnerable to the attacker's message is 90% ($q = 0.9$). In Figure 3.16, the probability is 50% ($q = 0.5$). In Figure 3.17, the probability is 10% ($q = 0.1$).

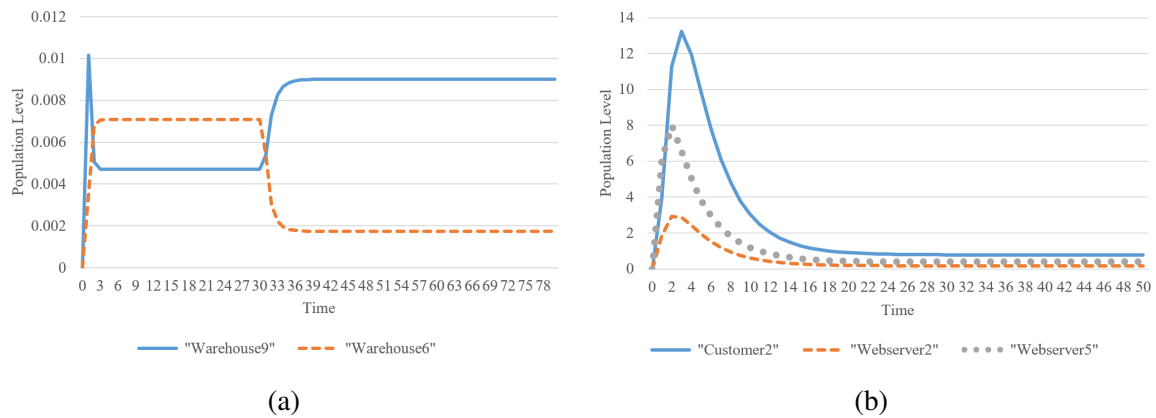


Fig. 3.15 The population level analysis when $q=0.9$.

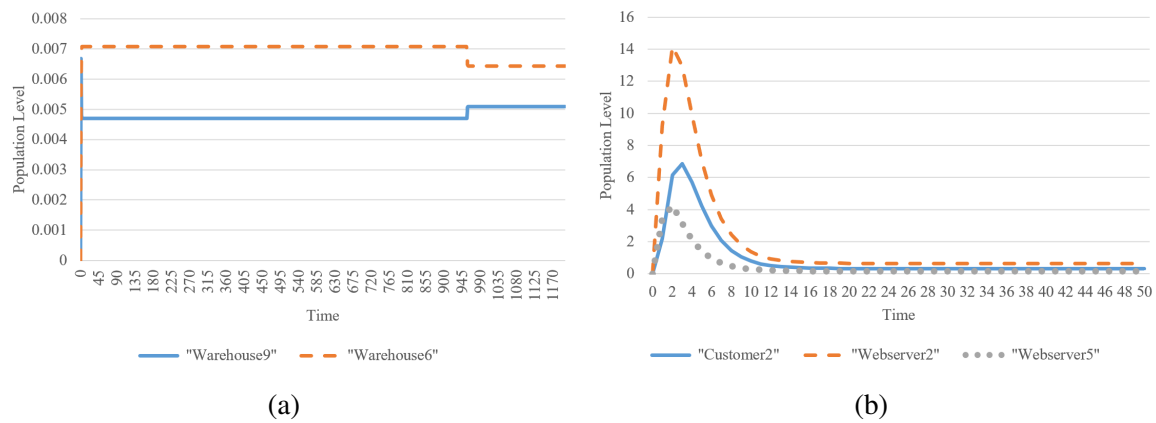


Fig. 3.16 The population level analysis when $q=0.5$.

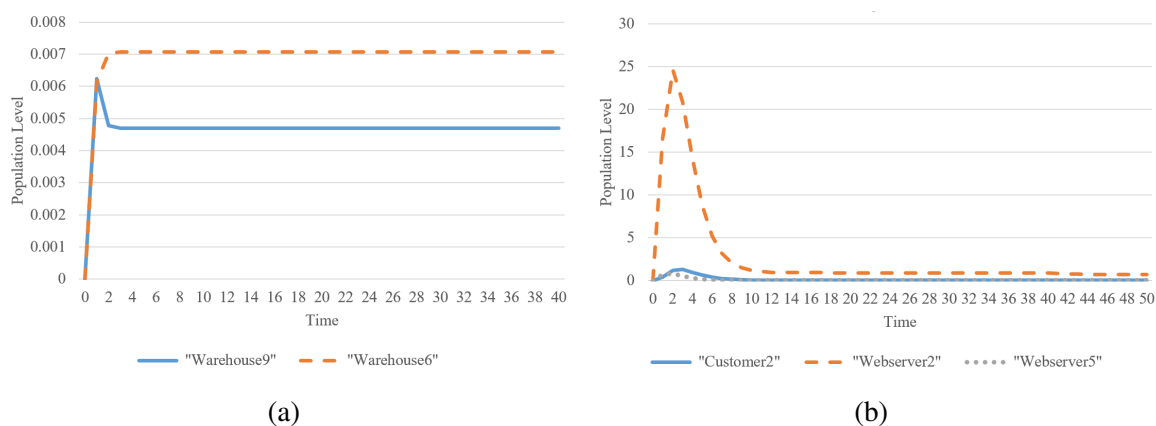


Fig. 3.17 The population level analysis when $q=0.1$.

Moreover, Figures 3.18 and 3.19 show the different population levels of *Warehouse9*, *Warehouse6*, *Customer2*, *Webservers2* and *Webservers5* states in relation to different proba-

bility values (q) of the webserver to be vulnerable to the attacker’s message. They illustrate how increasing q would have a significant impact on the population level of the states.

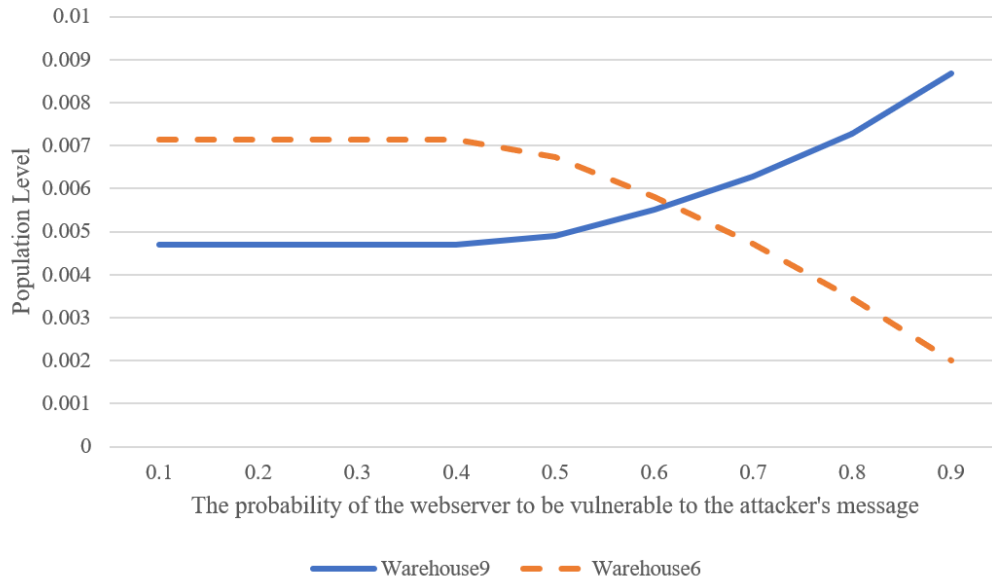


Fig. 3.18 The population level analysis of *Warehouse9* and *Warehouse9* in relation to q different values.

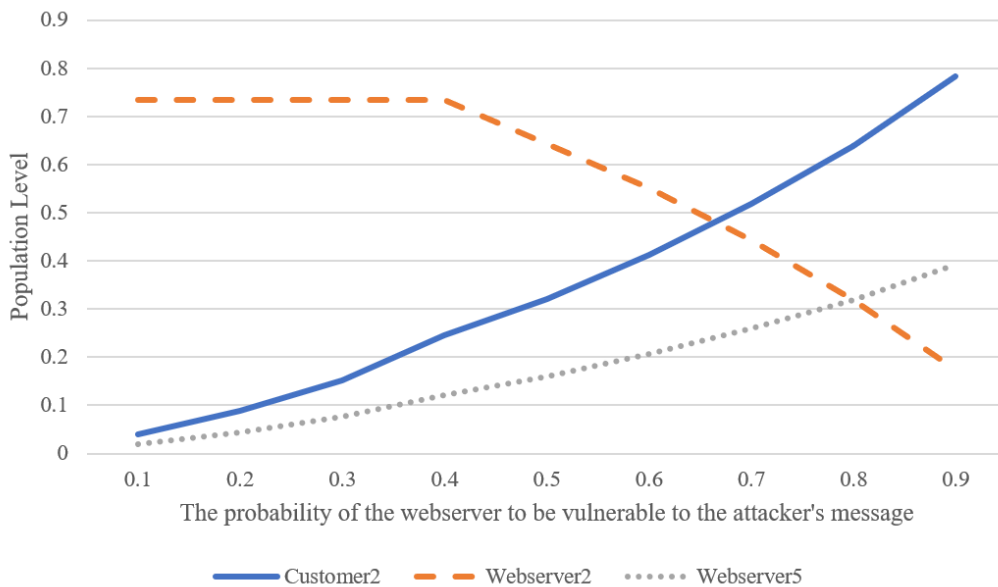


Fig. 3.19 The population level analysis of *Customer2*, *Webserver2* and *Webserver5* in relation to q different values.

Moreover, in order to make the system less vulnerable to the attacks, some work is needed to be done to protect the system and that might slow down the system. Therefore, varying q rate could have some other implications in the system, for example, slowing down the processing of the orders.

3.4 Conclusion

In this chapter, we present the performance models of web-based sale system in two scenarios, with and without the presence of denial of service attacks. It is clear from the related work in Chapter 2 that there are few existing models in this domain and therefore this work is a potentially valuable addition. The proposed PEPA models illustrated the high-level interaction between the components. The parameters used are somewhat arbitrary, nevertheless, the evaluation of the throughput and population level of the proposed models shows how the attacks would prevent some or all positive customers' orders from being fulfilled and how the delay in selling products would result in products being discarded.

The action rates for the proposed model were chosen arbitrarily. Clearly, the rates of actions used in the model will impact the effect of any attack. Therefore, it is desirable to obtain more realistic parameter values from a real or similar system and thereby validate the model. However, even without this, the model clearly demonstrates the impact of a denial of service attack on this system. This means that the model could be extended to explore the cost and benefit of potential defensive mechanisms and further understand the performance security trade-off in this context.

3.5 Chapter Summary

This chapter presents two performance models of a web-based sales system, one without an attack and the other with a denial of service attack. The approach used to model a system under investigation is PEPA. The PEPA Eclipse plug-in supports the creation of the PEPA models and the calculation of the performance measures. We used ordinary differential equations (ODEs) to evaluate and investigate some aspects of system performance. The evaluation of the models illustrates how the performance of the warehouse's sale is negatively affected by the denial of service attack by preventing some or all customers' orders from being fulfilled. The resultant delay in selling perishable products would result in products being discarded.

A more complex system will be discussed in the following chapter. It studies the performance overheads introduced by an e-commerce security protocol that ensures a fair

exchange between two parties by modelling and then evaluating the protocol's performance. We investigate the performance costs associated with the protocol's security features and behaviour. Additionally, we investigate the performance cost imposed by a security protocol in the event of misbehaviour during online commercial transactions.

Chapter 4

Performance modelling of an anonymous and failure resilient fair-exchange protocol

4.1 Introduction

In the previous chapter, we explored the performance of a web-based sales system, without and with a denial of service attack. In this chapter, we extended our study by considering a more complex system. We study the performance of an anonymous and failure resilient fair-exchange e-commerce protocol that was proposed by Ray *et al.* [62]. Ray *et al.* provide in [62] a detailed description of their proposed protocol and its security properties. In this study, we formally convert their proposed protocol to PEPA models in order to explore the performance overheads introduced by the security features and protocol's behaviour. The performance overheads introduced by a misbehaviour of any parties in e-commerce security protocol are also examined. This e-commerce protocol guarantees a fair-exchange between two parties. It satisfies the following features: first, fairness – no party can have any advantages over the other party during the exchange course; second, the anonymity of the parties – the parties, a customer, and/or a merchant can interact without disclosing any personal information; third, no manual dispute resolution; fourth, not relying on the service of a single trusted third party (TTP) – instead, multiple TTPs are available to provide services; fifth, offline TTP – the involvement of such a party must be at a minimum level, only when any problem occurs; and finally, any types of digital merchandise can be exchanged. Moreover, the protocol is based on an approach called 'cross-validation', which allows the customer to validate the encrypted electronic product without decrypting it.

An anonymous and failure resilient fair-exchange e-commerce protocol relies on TTPs but does not need them to be active at any time except if a problem occurs. Therefore, the protocol has two main descriptions depending on the type of TTP involvement: offline TTP (basic) and online TTP (extension) [62]. With offline TTP involvement type, there is no TTP active involvement as no parties misbehave or prematurely terminate the protocol. However, with online TTP, when parties misbehave or prematurely terminate the protocol, the TTP must be involved in resolving the problem and ensuring fair-exchange. Following the description provided by Ray *et al.* in [62], we first present PEPA models of a failure resilient fair-exchange protocol without a customer anonymity feature. Then, PEPA models of an optimistic anonymous protocol with a customer anonymity feature are presented and evaluated. Both versions of the protocol are modelled in two ways: with and without dispute between parties. In addition, the discussion focuses on the behaviour aspects of the protocols in order to analyse their performance.

4.2 Protocol specification

4.2.1 The basic failure resilient fair-exchange protocol specification

This protocol preserves all the features identified in the introduction (Section 4.1) except for the customer's anonymity. In this version of the protocol, the true identity of a party can be disclosed by obtaining a payment token. A formal description of this protocol with the security-related details is given in [62]. The basic protocol with no misbehaviour of any parties is as follows:

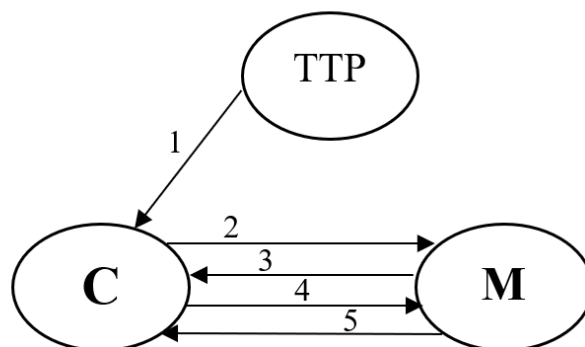


Fig. 4.1 The basic failure resilient fair-exchange protocol shows the interaction between a customer (C), a merchant (M) and a trust third party (TTP).

Before the protocol starts, the environment needs to be set up with the following two steps:

- I. A customer (C) needs to create an account with a financial institution (Bank) (B). B creates a pair of keys. One is sent to C, and the other is kept by B.
- II. A merchant (M) needs to register with a Trust Third Party (TTP). TTP creates a pair of keys. One is sent to M, and the other is kept by TTP. For every product M wants to sell, M sends a product and its description along with the product identifier (PID) to TTP. TTP then encrypts the electronic product with the same key that is sent to M before uploading the product to its website to be advertised.

Then, the protocol has the following five interaction steps, Figure 4.1. The words in bold are the actions name that we used in our proposed PEPA model:

1. **download** (TTP→C): C visits the TTP website and downloads the encrypted electronic product and the product identifier (PID) from the TTP server. C cannot obtain the product without a decryption key. This encrypted electronic product can be used to validate the product received from M. So if C is interested in the product, s/he contacts M and begins an interaction with them by sending the next message.
2. **sendMPO** (C→M): C sends a message containing the purchase order (PO) to M. This message contains a digitally signed PO cryptographic checksum, the payment token (PT), and the identity of its financial institution (B).
3. **sendCEP** or **sendCAbort** (M→C): M sends the encrypted product to C or sends a transaction abort statement. After M receives the message from C containing the PO (step 2), it checks its content to see if it meets the satisfaction criteria. If M is satisfied, it sends the encrypted electronic product to C along with a signed cryptographic checksum of the PO, a signed cryptographic checksum of the encrypted electronic product, an encrypted random number, and a signed cryptographic checksum of the encrypted random number.
4. **sendMPTDk** or **sendMAbort** (C→M): C sends the decryption key for the PT or a transaction abort statement to M. After C receives the message from M (step 3), C checks it. If it contains an abort statement, then C aborts the transaction. If it contains the encrypted electronic product, then C validates it with the encrypted electronic product received from the TTP (step 1). If the product is valid, C sends the decryption key for the PT. The key is encrypted with M's public key. Also C sends a signed cryptographic checksum of the encrypted electronic product received to M. Then C

waits for the product decryption key by setting a timer (if C does not receive the key within this time, C will require TTP involvement to resolve the dispute). If the product is not valid, C requests the product once again from M (step 2) or sends an abort statement along with a signed cryptographic checksum of the encrypted electronic product received to M.

5. **sendCPDk** (M→C): M sends the electronic product decryption key to C or ends the transaction. On receiving the message from C, M checks its content. If the message contains an abort statement, then M terminates the transaction. If the message includes the decryption key for the PT, M obtains the PT and then sends the decryption key for the electronic product to C. The decryption key is encrypted with C's public key.

4.2.2 The basic protocol extension for handling misbehaviours and communication problems

Subsection 4.2.1 presents the basic version of the failure resilient fair-exchange protocol, which is executed when there is no dispute between the exchange parties. However, when misbehaviours and/or communication problems occur, the extended protocol is initiated and TTP status is changed to online during protocol execution. The execution of the extended protocol is started when the customer's timer expires (during Step 4 in the basic protocol) and the basic protocol does not reach completion status, or when any misbehaviour occurs during the execution of the basic protocol. However, the extended protocol is always initiated by the customer [62]. This is because the merchant always receives the payment and can check its validity before sending the product decryption key to the customer.

Therefore, when the merchant (M) misbehaves, the customer (C) starts the extended protocol by sending the signed cryptographic checksums of the encrypted product and the random number received from M together with the signed cryptographic checksum of the PO and the payment token that have been sent to M (as evidence of M's misbehaviour) to TTP; this is called an initiation message. The misbehaviour scenarios considered by Ray et al [62] and solved by the extended protocol are illustrated as follows (all texts in bold indicate the names of the actions used in the PEPA model presented in this chapter):

Merchant behaves improperly

Scenario 1: M sends an invalid product decryption key after receiving the payment token decryption key (Step 5 in the basic protocol). The interaction actions for resolving this dispute are as follows:

1. *sendTTPinfo*: C initiates the execution of the extended protocol by sending an initiation message after receiving an invalid product decryption key.
2. *askMForValidK*: TTP orders M to send a valid product decryption key and then sets a timer for M's response.
3. *sendTTPvalidK* or *timeoutTTP*: M responds within the timeout period by sending the valid product decryption key to TTP, or M does not respond and the timeout expires.
4. *forwardKtoC* or *sendCkByTTP* and *takeActionAgainstM*: If TTP receives the valid product decryption key from M, it forwards it to C, or if TTP does not receive the valid product decryption key within a specified timeout period, TTP sends C the preserved product decryption key and takes action against M.

Scenario 2: M disappears without sending a valid product decryption key after receiving a payment token decryption key (Step 5 in the basic protocol). The interaction actions for resolving this dispute are as follows:

1. *cTimeoutExpired* then *sendTTPinfo*: C initiates the extended protocol by sending an initiation message after the timeout period for receiving the decryption key has expired.
2. *askMForValidK*: TTP asks M to send a valid product decryption key and then sets a timer for M's response.
3. *timeoutTTP* or *sendTTPvalidK*: M does not respond and the timeout expires, or M responds within the timeout period by sending the valid product decryption key to TTP.
4. *forwardKtoC* or *sendCkByTTP* and *takeActionAgainstM*: If TTP receives the valid product decryption key from M, it forwards it to C, or if TTP does not receive the valid product decryption key within the specified timeout period, TTP sends C the preserved product decryption key and takes action against M.

Scenario 3: M claims that a valid product decryption key has not been sent because payment from C has not been received. The interaction actions for resolving this dispute are as follows:

1. *sendTTPreason*: M responds to TTP by identifying the reason for not sending the valid product decryption key to C.
2. *sendTTPvalidK*: M must still send TTP the valid product decryption key.
3. *sendMpKbyTTP* and *sendCkbyTTP*: When TTP receives the valid product decryption key from M, it sends M the valid payment decryption key and C the valid product decryption key.

Customer behaves improperly

In this case, after C sends TTP an initiation message for the extended protocol, TTP checks all information and discovers that C has sent an invalid payment decryption key. As a result, TTP will not forward the valid product decryption key to C (*discoverIncorrectK*).

4.2.3 The optimistic anonymous protocol

This version of the protocol ensures that customer privacy is protected from any other parties. The customer does not need to share any personal information with a merchant to buy. Thus, the customer's true identity is hidden from the merchant. In the protocol described previously (see Subsection 4.2.1), the payment token that the customer sends to the merchant contains some personal information, such as the customer's identity and bank account information. Therefore, the merchant will have detailed personal information of the customer once it receives the payment token. This will deter some customers from buying from some merchants as they are not willing to share these personal details. Thus, as well as delivering all the features provided by the failure resilient fair-exchange protocol (see Subsection 4.2.1), the optimistic anonymous protocol also preserves the customer's anonymity.

Ray *et al.* [62] modified the basic failure resilient fair-exchange protocol to prevent the customer's personal information from being known by the merchant by following the electronic cash system described in [57]. The customer uses digital base money to buy from merchants. By using this method, merchants cannot obtain any personal information from the customer or create a customer profile without permission.

The formal description of the protocol and security-related details are provided in [62]. The following is an informal description of the protocol. As with the basic failure resilient fair-exchange protocol, before the protocol is initiated, the environment needs to be set up with the same steps detailed in Subsection 4.2.1. Unlike the basic failure resilient fair-exchange protocol, the customer (C) uses a pseudo identifier (C') when starting a new transaction with the merchant (M) to preserve the anonymity of C. Thus, no parties in the protocol except the customers themselves have sufficient information to link the C' used in the transaction with C, which is the real customer identity. The optimistic anonymous protocol with no dispute between M and C has the following nine interaction steps [62] (all texts in bold indicate the names of the actions used in the PEPA model):

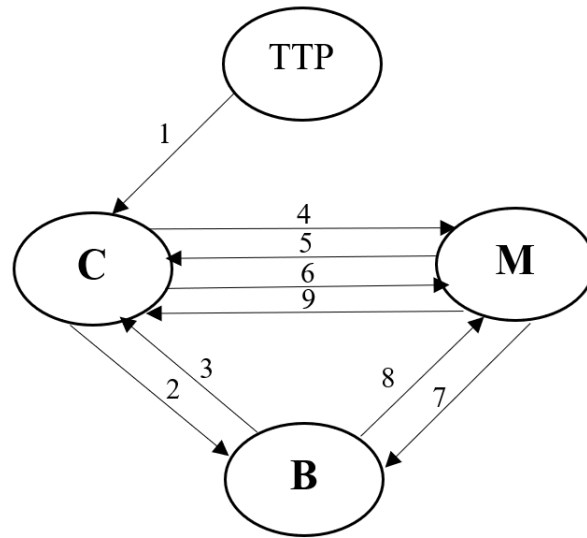


Fig. 4.2 The optimistic anonymous protocol shows the interaction between a customer (C), a merchant (M), a trust third party (TTP) and a bank (B).

1. **download** (TTP→C): C visits the TTP website and downloads the encrypted electronic product and the product identifier (PID) from the TTP server. The customer cannot get the product without a decryption key. This encrypted electronic product can be used to validate the product received from M, so if C is interested in the product, they must contact their bank (B) to request digital coins to buy the product with, as per the following message.
2. **requestBDigitalCoins** (C→B): C sends a request to B for digital coins. C's request message contains an unsigned blinded coin, a signed cryptographic checksum of the blind coin, the true identity of the customer, the account number, and a nonce. The signed cryptographic checksum ensures that the unsigned blinded coin is correct and not altered during the transfer. The nonce is used in the request message to prevent replying the same message again.
3. **sendCDigitalCoins** (B→C): Once B has received the request message, C's bank account is debited for the same amount of money as the value of the unsigned blinded coin. Then B generates the digital coin by signing the blinded coin, and sends the digital coin to C.
4. **sendMPO** (C'→M): C' sends a message containing the purchase order (PO) to M. The PO contains the product identifier, the customer's pseudo identifier (C'), the product price, and a nonce which prevents replying the PO. This message also contains a

digitally signed PO cryptographic checksum and the digital coin encrypted with C 's secret key. Note that the customer uses a one-time private/public key pair.

5. *sendCEP* or *sendCAbort* ($M \rightarrow C'$): M sends the encrypted product to C' or sends an abort statement to end the transaction. After receiving the message from C' containing the PO (step 4), M checks the message's contents to ensure they are satisfactory. If M is satisfied, it sends the encrypted electronic product to C' along with a signed cryptographic checksum of the PO, a signed cryptographic checksum of the encrypted electronic product, an encrypted random number, and a signed cryptographic checksum of the encrypted random number. If M is not satisfied, it sends an abort message to C' to terminate the exchange process.
6. *sendMCoinDk* or *sendMAbort* ($C' \rightarrow M$): C' sends the decryption key of the digital coin to M or sends an abort message to end the transaction. After receiving the message from M (step 5), C' checks it. If it contains an abort statement, then C' aborts the transaction. If it contains the encrypted electronic product, then C' validates it with the encrypted electronic product received from TTP (step 1). If the product is valid, C' sends the decryption key for the digital coin, which has been encrypted with M 's public key, along with a signed cryptographic checksum of the encrypted electronic product received to M and then waits for the product decryption key by setting a timer. If C' does not receive the key within the time set, they will require TTP involvement (as in Subsection 4.2.2). If the product is not valid, C' requests the product once again from M (step 4) or sends an abort statement with a signed cryptographic checksum of the encrypted electronic product received to M .
7. *sendBCoinByM* or *sendCAbort* ($M \rightarrow B$ or $M \rightarrow C'$): M sends B their identity and the digital coin that they have signed, or M sends C' an abort message to terminate the transaction. On receiving the decryption key for the digital coin from C' , M checks B 's signature on the coin and whether or not the amount received is equal to the price of the product. If M is satisfied, M sends B their identity and the signed digital coin and then sets a timer. If the timeout period expires, M sends B the message again. If M is unsatisfied for any reason, M sends C' an abort message to terminate the transaction.
8. *sendMyes* or *sendMno* ($B \rightarrow M$): B sends M either 'yes' or 'no'. Once B receives the coin from M , B checks whether or not the coin has been spent. If the coin has been spent, B sends M 'no'. If the coin has not been spent, B credits M 's account with the same amount of money as the digital coin and then sends M 'yes'.

9. *sendCPDk* or *sendCAbort* ($M \rightarrow C'$): M sends the electronic product decryption key to C' after receiving 'yes' from B, or ends the transaction by sending an abort message to C' after receiving 'no' from B. On receiving the 'yes' message from B, M sends the electronic product decryption key to C' . The decryption key is encrypted with C' public key.

4.2.4 The optimistic anonymous extended protocol for handling misbehaviours and communication problems

This is an extended version of the optimistic anonymous protocol (Subsection 4.2.3). The protocol in Subsection 4.2.3 is executed when there is no dispute between the exchange parties. In this subsection, we present the description of the protocol's extension to solve any dispute between the merchant and the customer. Therefore, when misbehaviours and/or communication problems occur, the extended protocol is initiated and TTP status is changed to online during the protocol execution. The execution of the extended protocol is started when the customer's timer expires (after Step 6 in the optimistic anonymous protocol) and the protocol does not reach completion status, or when customer receive an abort message or an invalid product decryption key as a reply to Step 9.

A dispute resolution is initiated when a customer sends TTP an initiation message similar to one in the extended basic protocol (Subsection 4.2.2) that it contains an evidence of misbehaving. The misbehaviour scenarios solved by the extended protocol are illustrated as follows (all texts in bold indicate the actions names employed in the PEPA models):

Merchant behaves improperly

Scenario 1: M sends an invalid product decryption key (Step 9 in the optimistic anonymous protocol). The interaction actions for resolving this dispute are as follows:

1. *seekingHelpFromTTP* and then *sendTTPinfo*: C initiates the execution of the extended protocol by seeking help from TTP. Then C sends an initiation message to TTP to seek a resolution of the problem of receiving an invalid product decryption key.
2. *validateCoinToB*: TTP receives the customer's dispute resolution request. Then it starts to contact B to validate the coin.
3. *sendTTPyes*: B confirms that the coin is valid, which means that the customer play fairly. Then TTP starts the following message to solve the dispute.

4. *askMForValidK*: TTP orders M to send a valid product decryption key and then sets a timer for M's response.
5. *sendTTPvalidK* or *timeoutTTP*: M responds within the timeout period by sending the valid product decryption key to TTP, or M does not respond and the timeout expires.
6. *forwardKtoC* or *sendCkByTTP* and *takeActionAgainstM*: If TTP receives the valid product decryption key from M, it forwards it to C. However, if TTP does not receive the valid product decryption key within a specified timeout period, TTP sends C the preserved product decryption key and then takes action against M.

Scenario 2: M sends an invalid product decryption key (Step 9 in the optimistic anonymous protocol). The coin in this scenario is invalid. The interaction actions for resolving this dispute are as follows:

1. *seekingHelpFromTTP* and then *sendTTPinfo*: C initiates the execution of the extended protocol by seeking help from TTP. Then C sends an initiation message to TTP to seek a resolution of the problem of receiving an invalid product decryption key.
2. *validateCoinToB*: TTP receives the customer's dispute resolution request. Then it starts to contact B to validate the coin.
3. *sendTTPno*: B confirms that the coin is invalid, which means that TTP needs to investigate who spent the coin. Then, TTP starts the following message to solve the dispute.
4. *investigationInvalidCoinToB*: TTP contacts B to investigate who spent the coin.
5. *mspentTheCoinToTTP*: B confirms that the coin is spent by M. Then, TTP starts the following message to solve the dispute.
6. *askMForValidK*: TTP orders M to send a valid product decryption key and then sets a timer for M's response.
7. *sendTTPvalidK* or *timeoutTTP*: M responds within the timeout period by sending the valid product decryption key to TTP, or M does not respond and the timeout expires.
8. *forwardKtoC* or *sendCkByTTP* and *takeActionAgainstM*: If TTP receives the valid product decryption key from M, it forwards it to C. However, if TTP does not receive the valid product decryption key within a specified timeout period, TTP sends C the preserved product decryption key and then takes action against M.

Scenario 3: M disappears without sending a valid product decryption key (Step 9 in the optimistic anonymous protocol). The interaction actions for resolving this dispute are as follows:

1. *cTimeoutExpired* and then *sendTTPinfo*: C initiates the extended protocol by sending an initiation message after the timeout period for receiving the decryption key has expired.
2. *validateCoinToB*: TTP receives the dispute resolution request from customer. Then it starts to contact B to validate the coin.
3. *sendTTPyes*: B confirms that the coin is valid, which means that the customer plays fairly. Then TTP starts the following message to solve the dispute.
4. *askMForValidK*: TTP orders M to send a valid product decryption key and then sets a timer for M's response.
5. *sendTTPvalidK* or *timeoutTTP*: M responds within the timeout period by sending the valid product decryption key to TTP, or M does not respond and the timeout expires.
6. *forwardKtoC* or *sendCkByTTP* and *takeActionAgainstM*: If TTP receives the valid product decryption key from M, it forwards it to C. However, if TTP does not receive the valid product decryption key within a specified timeout period, TTP sends C the preserved product decryption key and then takes action against M.

Scenario 4: M claims that a valid product decryption key has not been sent because an invalid coin's decryption key has been received from C. The interaction actions for resolving this dispute are as follows:

1. *sendTTPreason*: M responds to TTP by identifying the reason for not sending the valid product decryption key to C after TTP contacts it to send a valid decryption key.
2. *sendTTPvalidK*: M must still send TTP the valid product decryption key.
3. *sendMpKbyTTP* and *forwardKtoC*: When TTP receives the valid product decryption key from M, it sends M the valid coin decryption key and forwards the valid product decryption key to C.

Customer behaves improperly

In this case, after C sends TTP an initiation message for the extended protocol, TTP starts contacting B to validate the coin. If the coin is invalid (*sendTTPno*), TTP then starts contacting B to investigate who spent the coin (*investigationInvalidCoinToB*). If B confirms that the customer is who spent the coin (*cspentTheCoinToTTP*), TTP will not forward the valid product decryption key to C (*discoverMisbehavingC*).

4.3 PEPA models

This section presents our proposed PEPA models for the anonymous and failure resilient fair-exchange e-commerce protocol. We proposed PEPA models for each version of the protocol, as specified in the protocol specification Section 4.2.

4.3.1 A PEPA model of the basic failure resilient fair-exchange e-commerce protocol

This subsection presents PEPA model for the basic failure resilient fair-exchange e-commerce protocol. This protocol is the basic protocol with no misbehaviour of any parties, as described in Subsection 4.2.1. In our proposed PEPA model, there are three types of components: customer (C), merchant (M) and trust third party (TTP). C and M are sequential components whereas TTP is static component. The model comprises of three parts, one for each component. C and M move sequentially from their different behaviours based on the activities specified in the model. The model is formulated as follows:

Merchant component

$$\begin{aligned}
 M_0 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).M_1 \\
 M_1 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).M_2 + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_5 \\
 M_2 &\stackrel{\text{def}}{=} (\text{sendMPTdk}, r_{\text{sendMPTdk}}).M_3 + (\text{sendMAbort}, r_{\text{sendMAbort}}).M_4 \\
 M_3 &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).M_4 \\
 M_4 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_0 \\
 M_5 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_4
 \end{aligned}$$

The above model component specifies M's different behaviours, moving from M_0 to M_5 . When M is in state M_0 (step 2 in the protocol's description), M performs action *sendMPO* at rate r_{sendMPO} leading to M_1 . Then, in state M_1 (step 3 in the protocol's description), either can happen action *sendCEP* at rate r_{sendCEP} leading to M_2 or action *sendCAbort* at rate

$r_{sendCAbort}$ leading to M_5 . In state M_2 (step 4), M can perform either action $sendMPTdk$ at rate $r_{sendMPTdk}$ leading to M_3 or action $sendMAbort$ at rate $r_{sendMAbort}$ leading to M_4 . Then, in state M_3 (step 5), the only action that happens is $sendCPDk$ at rate $r_{sendCPDk}$ leading to M_4 , which is the state when M acts *complete* at rate $r_{complete}$ leading back to M_0 , which it means that the exchange between C and M has finished. The state M_5 is when M performs action $sendMAbort$ due to performing action $sendCAbort$ in state M_1 leading to M_4 . This is to keep smooth communication between M and C. Therefore, when M sends abort to C (during state M_1), C also sends abort to M (during state M_5) and then moves to the last state M_4 , to terminate the interaction. After M_4 , the behaviour returns to M_0 so that the model becomes cyclic and steady-state measures can be obtained.

Customer component

$$\begin{aligned}
C_0 &\stackrel{def}{=} (download, r_d).C_1 \\
C_1 &\stackrel{def}{=} (sendMPO, r_{sendMPO}).C_2 \\
C_2 &\stackrel{def}{=} (sendCEP, r_{sendCEP}).C_3 + (sendCAbort, r_{sendCAbort}).C_6 \\
C_3 &\stackrel{def}{=} (sendMPTdk, r_{sendMPTdk}).C_4 + (sendMAbort, r_{sendMAbort}).C_5 \\
C_4 &\stackrel{def}{=} (sendCPDk, r_{sendCPDk}).C_5 \\
C_5 &\stackrel{def}{=} (complete, r_{complete}).C_0 \\
C_6 &\stackrel{def}{=} (sendMAbort, r_{sendMAbort}).C_5
\end{aligned}$$

The above component presents C's different behaviours, moving from C_0 to C_6 . The first state is C_0 . It represents step 1 in the protocol's description, see Subsection 4.2.1. It is the state when C visits the TTP website and performs action *download* for a specific product at rate r_d leading to C_1 . Then, in state C_1 (step 2 in the protocol's description), the only action happens $sendMPO$ at rate $r_{sendMPO}$ leading to C_2 . In state C_2 (step 3), one of two actions can happen, either $sendCEP$ at rate $r_{sendCEP}$ leading to C_3 or $sendCAbort$ at rate $r_{sendCAbort}$ leading to C_6 . In state C_3 (reflects step 4), there is one of two actions that can happen, either $sendMPTdk$ at rate $r_{sendMPTdk}$ leading to C_4 or $sendMAbort$ at rate $r_{sendMAbort}$ leading to C_5 . In C_4 (represents step 5 in the protocol's description), $sendCPDk$ can occur at rate $r_{sendCPDk}$ leading to C_5 . The state C_5 is a termination step when C performs action *complete* at rate $r_{complete}$ leading back to C_0 to finish the interaction and maybe use the product before starting again. In state C_6 , C performs action $sendMAbort$ at rate $r_{sendMAbort}$ to receive abort from M as a result of performing action $sendCAbort$ in state C_2 . Performing C_6 's action leads to C_5 . This step allows C to respond to M and starts the interaction again, providing correct information.

TTP component

$$TTP \stackrel{def}{=} (download, r_d).TTP$$

In the model, TTP has only one state. In state TTP (represents step 1 in protocol's description), the only action can happen is $download$ at rate r_d leading to the same state TTP .

The system equation

The system equation and complete specification are given by

$$System \stackrel{def}{=} TTP[K] \bowtie_J C_0[N] \bowtie_L M_0[N]$$

Where $J=\{download\}$, $L=\{sendMPO, sendCEP, sendCAabort, sendMPTDk, sendMAabort, sendCPDk, complete\}$, any action in the list J and L is a shared action between the components specified in the system equation. N is the number instances of C and M copies in the system. K is the number of TTPs. The three components are initially in the states TTP , C_0 and M_0 .

Moreover, the number of M 's copies depends on the number of C . M can has multiple copies, and each copy is associated with one C to serve it. This indicates that the rates of M 's main activities are divided by the number of C s that interact with it. M 's main activities are $sendCEP$, $sendCAabort$ and $sendCPDk$. The rates are calculated as follows:

$$r_{sendCEP} = \frac{r_{sendCEP1}}{N}$$

$$r_{sendCAabort} = \frac{r_{sendCAabort1}}{N}$$

$$r_{sendCPDk} = \frac{r_{sendCPDk1}}{N}$$

N is the number of instances of the merchant component M , which is related to the number of customers using M 's website. $r_{sendCEP1}$, $r_{sendCAabort1}$ and $r_{sendCPDk1}$ are the rates of M 's main activities.

4.3.2 PEPA model of the extended failure resilient fair-exchange protocol

This subsection presents the proposed PEPA model for the basic protocol extension for handling misbehaviours and communication problems. This extended protocol is executed when the disputes occur, as described in Subsection 4.2.2. There are three types of components in our proposed PEPA model: customer (C), merchant (M) and trust third party (TTP). C and M are sequential components whereas TTP is static component. The PEPA model comprises of three main parts, one for each component: M, C and TTP. The model is formulated as follows:

Merchant component

$$\begin{aligned}
M_0 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).M_1 \\
M_1 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).M_2 + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_5 \\
M_2 &\stackrel{\text{def}}{=} (\text{sendMPTdk}, r_{\text{sendMPTdk}}).M_3 + (\text{sendMAbort}, r_{\text{sendMAbort}}).M_4 \\
M_3 &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).M_4 + (\text{cTimeoutExpired}, r_{\text{cTimeout}}).M_4 \\
M_4 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_0 + (\text{askMForValidK}, r_{\text{askMForValidK}}).M_6 \\
M_5 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_4 \\
M_6 &\stackrel{\text{def}}{=} (\text{sendTTPvalidK}, r_{\text{sendTTPvalidK}}).M_4 + (\text{timeoutTTP}, r_{\text{timeout1}}).M_7 \\
&\quad + (\text{sendTTPreason}, r_{\text{sendTTPreason}}).M_8 \\
M_7 &\stackrel{\text{def}}{=} (\text{takeActionAgainstM}, r_{\text{takeActionAgainstM}}).M_4 \\
M_8 &\stackrel{\text{def}}{=} (\text{sendTTPvalidK}, r_{\text{sendTTPvalidK}}).M_9 \\
M_9 &\stackrel{\text{def}}{=} (\text{sendMpKbyTTP}, r_{\text{sendMpKbyTTP}}).M_4
\end{aligned}$$

The local states of the M component increase to 10 states compared to the M component of the basic protocol presented in Subsection 4.3.1. This has similar states as in the basic protocol described in Subsection 4.3.1 except from state M_3 . In M_3 , either sendCPDk occurs at rate r_{sendCPDk} leading to M_4 or action cTimeoutExpired at rate r_{cTimeout} leading also to M_4 , which it reflects step 1 in scenario 2 in the protocol description, see Subsection 4.2.2. When M is in state M_4 , M can perform either action complete if both parties are happy to finish the interaction or action askMForValidK at rate $r_{\text{askMForValidK}}$ leading to M_6 when misbehaviour occurs (as described in step 2 of scenario 1 and 2). Then, in state M_6 any of three actions can happen; sendTTPvalidK at rate $r_{\text{sendTTPvalidK}}$ leading to M_4 , timeoutTTP at rate r_{timeout1} leading to M_7 or sendTTPreason at rate $r_{\text{sendTTPreason}}$ leading to M_8 as described in step 3 in scenario 1 and 2 for the first two actions and in step 1 in scenario 3 for the last action. In state M_7 , the only action happens is $\text{takeActionAgainstM}$ at rate $r_{\text{takeActionAgainstM}}$ leading to M_4 , reflecting step 4 in both scenario 1 and 2, see Subsection 4.2.2. When M is in state

M_8 , it can do action $sendTTPvalidK$ at rate $r_{sendTTPvalidK}$ leading to M_9 . This reflects step 2 in scenario 3 in the protocol description. Then in state M_9 , the only action that happens is $sendMpKbyTTP$ at rate $r_{sendMpKbyTTP}$ leading to M_4 , as described in step 3 in scenario 3.

Customer component

$$\begin{aligned}
C_0 &\stackrel{def}{=} (download, r_d).C_1 \\
C_1 &\stackrel{def}{=} (sendMPO, r_{sendMPO}).C_2 \\
C_2 &\stackrel{def}{=} (sendCEP, r_{sendCEP}).C_3 + (sendCAbort, r_{sendCAbort}).C_6 \\
C_3 &\stackrel{def}{=} (sendMPTdk, r_{sendMPTdk}).C_4 + (sendMAbort, r_{sendMAbort}).C_5 \\
C_4 &\stackrel{def}{=} (sendCPDk, r_{sendCPDk}).C_5 + (cTimeoutExpired, r_{cTimeoutExpired}).C_7 \\
C_5 &\stackrel{def}{=} (complete, r_{complete}).C_0 + (seekingHelpFromTTP, r_{seekingHelp}).C_7 \\
C_6 &\stackrel{def}{=} (sendMAbort, r_{sendMAbort}).C_5 \\
C_7 &\stackrel{def}{=} (sendTTPinfo, r_{sendTTPinfo}).C_8 \\
C_8 &\stackrel{def}{=} (forwardKtoC, r_{forwardKtoC}).C_5 + (sendCkByTTP, r_{sendCkByTTP}).C_5 \\
&\quad + (discoverIncorrectK, r_{discoverIncorrectK}).C_5
\end{aligned}$$

The C component's part of the model has similar states to the basic protocol PEPA model except from state C_4 . Moreover, the local states of this component increases to 9 compared to the basic protocol PEPA model presented in Subsection 4.3.1. In state C_4 , one of two actions can happen either action $sendCPDk$ in order to receive the product decryption key from M leading to C_5 or action $cTimeoutExpired$ at rate $r_{cTimeout}$ leading to C_7 , as described in step 1 in scenario 2 (Subsection 4.2.2). In state C_5 , If C has not received a valid product decryption key, action $seekingHelpFromTTP$ happens at rate $r_{seekingHelp}$ leading to C_7 (as described in Subsection 4.2.2 in step 1 in scenario 1) otherwise action $complete$ happens to finalise the interaction. When C is in state C_7 as a result of performing action $cTimeoutExpired$, the only action that happens is $sendTTPinfo$ at rate $r_{sendTTPinfo}$ leading to C_8 . This reflects step 1 in scenario 2. Then, in state C_8 , either $forwardKtoC$ occurs at rate $r_{forwardKtoC}$, $sendCkByTTP$ at rate $r_{sendCkByTTP}$ or $discoverIncorrectK$ at rate $r_{discoverIncorrectK}$ all of which lead to C_5 to finish the interaction. The first two actions reflect step 4 in both scenario 1 and 2 whereas action $discoverIncorrectPTK$ is a result of customer misbehaviour of sending invalid payment decryption key so TTP will not forward the valid product decryption key to C as described in Subsection 4.2.2.

TTP component

$$\begin{aligned}
TTP \stackrel{def}{=} & (download, r_d).TTP + (sendTTPinfo, r_{sendTTPinfo}).TTP \\
& + (askMForValidK, r_{askMForValidK}).TTP + (timeoutTTP, r_{timeoutTTP}).TTP \\
& + (discoverIncorrectK, r_{discoverIncorrectK}).TTP \\
& + (forwardKtoC, r_{forwardKtoC}).TTP + (sendCkByTTP, r_{sendCkByTTP}).TTP \\
& + (takeActionAgainstM, r_{takeActionAgainstM}).TTP \\
& + (sendMpKbyTTP, r_{sendMpKbyTTP}).TTP + (sendTTPreason, r_{sendTTPreason}).TTP
\end{aligned}$$

The last part of the model is the TTP component. The TTP has more actions to control the interaction between the other two components and to resolve the dispute between them compared to the PEPA model of the basic protocol presented in Subsection 4.3.1. There is just one state. The TTP's main actions to resolve the misbehaviour are *askMForValidK*, *timeoutTTP*, *discoverIncorrectK*, *forwardKtoC*, *takeActionAgainstM* and *sendMpKbyTTP*, as described in the protocol description, see Subsection 4.2.2. The rates of those actions are controlled by TTP.

The system equation

The system equation and complete specification are given by

$$System \stackrel{def}{=} TTP[K] \bowtie_R (C_0[N] \bowtie_L M_0[N])$$

Where $R = \{download, sendTTPinfo, askMForValidK, timeoutTTP, sendTTPvalidK, discoverIncorrectK, forwardKtoC, sendCkByTTP, takeActionAgainstM, sendMpKbyTTP, sendTTPreason\}$, $L = \{sendMPO, sendCEP, cTimeoutExpired, sendCAbort, sendMPTDk, sendMAbort, sendCPDk, complete\}$, any action in the list R and L is a shared action between the components specified in the system equation. N is the number of C and M copies in the system. K is the number of TTPs. The three components are initially in the states TTP , C_0 and M_0 .

In this PEPA model of the extended protocol, the number of actions is larger than in the basic protocol. The service rates of all the main actions carried out by M depend on the number of C s interacting with M . M 's main actions are *sendCEP*, *sendCAbort*, *sendCPDk*, *sendTTPreason*, and *sendTTPvalidK*. So, each service rate is divided by the number of C s interacting with M , as in the basic protocol:

$$\begin{aligned}
r_{sendCEP} &= \frac{r_{sendCEP1}}{N} \\
r_{sendCAbort} &= \frac{r_{sendCAbort1}}{N}
\end{aligned}$$

$$r_{sendCPDk} = \frac{r_{sendCPDk1}}{N}$$

$$r_{sendTTPreason} = \frac{r_{sendTTPreason1}}{N}$$

$$r_{sendTTPvalidK} = \frac{r_{sendTTPvalidK1}}{N}$$

Where N is M copies instances which is related to the number of customers using M 's website. $r_{sendCEP1}$, $r_{sendCAbort1}$, $r_{sendCPDk1}$, $r_{sendTTPreason1}$ and $r_{sendTTPvalidK1}$ are the rates of M 's main activities.

Furthermore, the service rates of all TTP actions depend on the number of both Cs and TTPs interacting with each other. TTP's main actions are *forwardKtoC*, *sendCkByTTP*, *download*, *askMForValidK*, *takeActionAgainstM*, *discoverIncorrectPTK*, and *sendMpKbyTTP*. One, two or more TTPs can be involved in the protocol [62]. Thus, each service rate can be calculated as follows:

$$r_d = \left(\frac{r_{download}}{N} \right) * K$$

$$r_{forwardKtoC} = \left(\frac{r_{forwardKtoC1}}{N} \right) * K$$

$$r_{sendCkByTTP} = \left(\frac{r_{sendCkByTTP1}}{N} \right) * K$$

$$r_{askMForValidK} = \left(\frac{r_{askMForValidK1}}{N} \right) * K$$

$$r_{takeActionAgainstM} = \left(\frac{r_{takeActionAgainstM1}}{N} \right) * K$$

$$r_{sendMpKbyTTP} = \left(\frac{r_{sendMpKbyTTP1}}{N} \right) * K$$

$$r_{discoverIncorrectK} = \left(\frac{r_{discoverIncorrectK1}}{N} \right) * K$$

Where N is the number of customer and K is the number of TTP instances. $r_{download}$, $r_{forwardKtoC1}$, $r_{sendCkByTTP1}$, $r_{askMForValidK1}$, $r_{takeActionAgainstM1}$, $r_{sendMpKbyTTP1}$ and $r_{discoverIncorrectK1}$ are the rates of the TTP's main actions.

4.3.3 PEPA model of the optimistic anonymous protocol

This subsection presents the proposed PEPA model for the optimistic anonymous protocol. This version of the protocol ensures that customer privacy is protected from any other parties, as presented in Subsection 4.2.3. The following PEPA model is for the optimistic anonymous protocol when there is no dispute between M and C . The proposed PEPA model contains four

components; Merchant (M), Customer (C), Trust Third Party (TTP) and Bank (B). In our PEPA model, M and C are sequential components whereas TTP and B are static components. The model comprises four main parts, one for each component: M, C, TTP and B. The model is formulated as follows:

Merchant component

$$\begin{aligned}
M_0 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).M_1 \\
M_1 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).M_2 + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_8 \\
M_2 &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCoinDk}}).M_3 + (\text{sendMAbort}, r_{\text{sendMAbort}}).M_6 \\
M_3 &\stackrel{\text{def}}{=} (\text{startContactB}, r_{\text{startContactB}}).M_{3a} + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_8 \\
M_{3a} &\stackrel{\text{def}}{=} (\text{sendBCoinByM}, r_{\text{sendBCoinByM}}).M_4 \\
M_4 &\stackrel{\text{def}}{=} (\text{sendMyes}, r_{\text{sendMyes}}).M_5 + (\text{sendMno}, r_{\text{sendMno}}).M_7 \\
M_5 &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).M_6 \\
M_6 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_0 \\
M_7 &\stackrel{\text{def}}{=} (\text{sendCAbort}, r_{\text{sendCAbort}}).M_8 \\
M_8 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_6
\end{aligned}$$

The above part of the model is for the M component. The first two states are the same as states M_0 and M_1 described in the previous protocol models. The states M_0 and M_1 reflect step 4 and 5 of the optimistic anonymous protocol description presented in Subsection 4.2.3, respectively. In the state M_2 , one of two actions can happen either sendMCoinDk at rate r_{sendMCDk} leading to M_3 if a customer receives an encrypted product or sendMAbort at rate $r_{\text{sendMAbort}}$ leading to M_6 if the customer is not satisfied. This state reflects step 6 of the optimistic anonymous protocol description. When M reaches state M_3 , one of two actions happen either startContactB at rate $r_{\text{startContactB}}$ leading to M_{3a} if M is satisfied with the amount of the digital coins or sendCAbort at rate $r_{\text{sendCAbort}}$ leading to M_8 if M is not satisfied, as described in step 7. In state M_{3a} , the only action can happen is sendBCoinByM at rate $r_{\text{sendBCByM}}$. In state M_4 , either action can be performed sendMyes at rate r_{sendMyes} leading to M_5 to have a confirmation from the bank that the coins are valid or sendMno at rate r_{sendMno} leading to M_7 in order to send an abort to the customer when M has a confirmation from the bank that the coins are invalid, as described in step 8. Then, in state M_5 , the only action that happens is sendCPDk at rate r_{sendCPDk} leading to M_6 , which is the state when M acts complete at rate r_{complete} leading back to M_0 , which it means that the exchange between C and M has finished. M_5 reflects step 9 in the protocol description.

Customer component

$$\begin{aligned}
C_0 &\stackrel{\text{def}}{=} (\text{download}, r_d).C_1 \\
C_1 &\stackrel{\text{def}}{=} (\text{requestBDigitalCoins}, r_{\text{requestBDC}}).C_2 \\
C_2 &\stackrel{\text{def}}{=} (\text{sendCDigitalCoins}, r_{\text{sendCDC}}).C_3 \\
C_3 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).C_4 \\
C_4 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).C_5 + (\text{sendCAbort}, r_{\text{sendCAbort}}).C_8 \\
C_5 &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCDk}}).C_6 + (\text{sendMAbort}, r_{\text{sendMAbort}}).C_7 \\
C_6 &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).C_7 + (\text{sendCAbort}, r_{\text{sendCAbort}}).C_8 \\
C_7 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).C_0 \\
C_8 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).C_7
\end{aligned}$$

The different states of the C component are formulated above. After C performs action *download* in C_0 , it moves to C_1 . In state C_1 , the only action that happens is *requestBDigitalCoins* at rate $r_{\text{requestBDC}}$ in order to request a digital coin from the bank leading C_1 to C_2 , as described in step 2. Then in state C_2 , there is only one action that can happen: *sendCDigitalCoins* at rate r_{sendCDC} to get the digital coin from the bank leading to C_3 , as described in step 3 in Subsection 4.2.3. The states C_3 and C_4 are the same as states C_1 and C_2 described in previous protocol models. They reflect step 4 and 5 in the description of this protocol, respectively. Then when C reaches C_5 , one of two possible actions happens either *sendMCoinDk* at rate r_{sendMCDk} leading to C_6 when C gets valid encryption product or *sendMAbort* at rate $r_{\text{sendMAbort}}$ leading to C_7 when C gets invalid encryption product M during state C_4 , as described in step 6. The states C_6 , C_7 and C_8 are similar to states C_4 , C_5 and C_6 in the basic protocol model presented in Subsection 4.3.1, respectively. State C_6 reflects step 9 of the optimistic anonymous protocol description.

TTP component

$$TTP \stackrel{\text{def}}{=} (\text{download}, r_d).TTP$$

In this model, TTP has only one state. In state TTP , the only action could happen is *download* at rate r_d leading to the same state TTP . This reflects step 1 in the protocol's description.

Bank component

$$\begin{aligned}
B &\stackrel{\text{def}}{=} (\text{requestBDigitalCoins}, r_{\text{requestBDC}}).B + (\text{sendCDigitalCoins}, r_{\text{sendCDC}}).B \\
&+ (\text{sendBCoinByM}, r_{\text{sendBCByM}}).B + (\text{sendMyes}, r_{\text{sendMyes}}).B \\
&+ (\text{sendMno}, r_{\text{sendMno}}).B
\end{aligned}$$

The last part of the model is for the B component. There is just one state which is B . The B's main actions to support the purchase processes between C and M are $sendCDigitalCoins$, $sendMyes$ and $sendMno$ as described in step 3 and 8 of the optimistic anonymous protocol description. The rates of those actions are controlled by B.

The system equation

The system equation and complete specification are given by

$$System \stackrel{def}{=} TTP[K] \bowtie_R (C_0[N] \bowtie_L M_0[N]) \bowtie_M B[S]$$

Where the cooperation sets $R=\{download\}$, $L=\{sendMPO, sendCEP, sendCAbort, sendM-CoinDk, sendMAbort, sendCPDk, complete\}$, and $M=\{requestBDigitalCoins, sendCDigitalCoins, sendMno, sendBCoinByM, sendMyes\}$, any action in list R , L and M is shared actions between the components specified in the system equation. N is the number of customers and merchant copies on the system, K is the number of TTPs, S is the number of Bs. The four components are initially in the states TTP , C_0 , M_0 and B .

Moreover, the rates of all the main actions carried out by M depend on the number of Cs interacting with M. The M's main actions are $sendCEP$, $sendCAbort$, $startContactB$, $sendBCoinByM$ and $sendCPDk$. So, each rate is divided by the number of Cs interacting with M, as in the basic and extended protocols, as described in Subsections 4.3.1 and 4.3.2.

Furthermore, the service rate of all actions for B – $sendCDigitalCoins$, $sendMno$, and $sendMyes$ – is dependent on the number of Cs, Ms and Bs involved in the interaction. One, two or more Bs can be involved in the protocol to serve C and M. So, each rate is calculated as follows:

$$\begin{aligned} r_{sendCDC} &= \left(\frac{r_{sendCDigitalCoins1}}{N} \right) * S \\ r_{sendMno} &= \left(\frac{r_{sendMno1}}{N} \right) * S \\ r_{sendMyes} &= \left(\frac{r_{sendMyes1}}{N} \right) * S \end{aligned}$$

Where S is the number of financial institutions and N is the number of customers and merchant copies. $r_{sendCDigitalCoins1}$, $r_{sendMno1}$ and $r_{sendMyes1}$ are the rates of the B's main actions.

4.3.4 PEPA models of the extended optimistic anonymous protocol

This subsection presents three proposed PEPA models for the optimistic anonymous protocol extension for handling misbehaviours and communication problems. This extended protocol is executed when the disputes occur between the parties, as described in Subsection 4.2.4. We provide three different PEPA models based on different scenarios. The first PEPA model presented is a basic extended optimistic anonymous protocol to solve the misbehaving event between M and C parties. The second PEPA model is similar to the first one but with probabilities of M misbehaving. The third model is a PEPA model comprises two types of C interacting with M. One type of Cs is honest C and one is misbehaving C.

The PEPA models comprise of four main components. The four components are Merchant (M), Customer (C), Trust Third Party (TTP) and Bank (B). In the PEPA models, M, C and TTP are sequential components whereas B is a static component. Unlike the previous PEPA models, TTP component becomes a sequential component to effectively interact and solve the dispute between M and C in this extended protocol.

PEPA model for the extended optimistic anonymous protocol

The extended PEPA model for the extended optimistic anonymous protocol is formulated as follows:

Merchant component

$$\begin{aligned}
M_0 &\stackrel{\text{def}}{=} (sendMPO, r_{sendMPO}).M_1 \\
M_1 &\stackrel{\text{def}}{=} (sendCEP, r_{sendCEP}).M_2 + (sendCAabort, r_{sendCAabort}).M_8 \\
M_2 &\stackrel{\text{def}}{=} (sendMCoinDk, r_{sendMCoinDk}).M_3 + (sendMAabort, r_{sendMAabort}).M_6 \\
M_3 &\stackrel{\text{def}}{=} (startContactB, r_{startContactB}).M_{3a} + (sendCAabort, r_{sendCAabort}).M_8 \\
M_{3a} &\stackrel{\text{def}}{=} (sendBCoinByM, r_{sendBCoinByM}).M_4 \\
M_4 &\stackrel{\text{def}}{=} (sendMyes, r_{sendMyes}).M_5 + (sendMno, r_{sendMno}).M_7 \\
M_5 &\stackrel{\text{def}}{=} (sendCPDk, r_{sendCPDk}).M_6 + (cTimeoutExpired, r_{cTimeoutExpired}).M_6 \\
M_6 &\stackrel{\text{def}}{=} (complete, r_{complete}).M_0 + (askMforValidK, r_{askMforValidK}).M_9 \\
M_7 &\stackrel{\text{def}}{=} (sendCAabort, r_{sendCAabort}).M_8 \\
M_8 &\stackrel{\text{def}}{=} (sendMAabort, r_{sendMAabort}).M_6 \\
M_9 &\stackrel{\text{def}}{=} (sendTTPvalidK, r_{sendTTPvalidK}).M_6 + (timeoutTTP, r_{timeoutTTP}).M_{10} \\
&\quad + (sendTTPreason, r_{sendTTPreason}).M_{11} \\
M_{10} &\stackrel{\text{def}}{=} (takeActionAgainstM, r_{takeActionAgainstM}).M_6 \\
M_{11} &\stackrel{\text{def}}{=} (sendTTPvalidK, r_{sendTTPvalidK}).M_{12} \\
M_{12} &\stackrel{\text{def}}{=} (sendMpkbyTTP, r_{sendMpkbyTTP}).M_6
\end{aligned}$$

The local states of the M component increased to fourteen states compared to the protocol without a dispute between M and C (Subsection 4.3.3). This model has similar states as in the optimistic anonymous protocol described in Subsection 4.3.3 except from state M_6 which is when the dispute occurs and M has been asked for a valid product decryption key. Then appropriate actions based on the specified rate are followed to solve the dispute between M and C.

Customer component

$$\begin{aligned}
C_0 &\stackrel{def}{=} (download, r_d).C_1 \\
C_1 &\stackrel{def}{=} (requestBDigitalCoins, r_{requestBDC}).C_2 \\
C_2 &\stackrel{def}{=} (sendCDigitalCoins, r_{sendCDC}).C_3 \\
C_3 &\stackrel{def}{=} (sendMPO, r_{sendMPO}).C_4 \\
C_4 &\stackrel{def}{=} (sendCEP, r_{sendCEP}).C_5 + (sendCAbort, r_{sendCAbort}).C_8 \\
C_5 &\stackrel{def}{=} (sendMCoinDk, r_{sendMCDk}).C_6 + (sendMAbort, r_{sendMAbort}).C_7 \\
C_6 &\stackrel{def}{=} (sendCPDk, r_{sendCPDk}).C_7 + (sendCAbort, r_{sendCAbort}).C_8 \\
&\quad + (cTimeoutExpired, r_{cTimeoutExpired}).C_9 \\
C_7 &\stackrel{def}{=} (complete, r_{complete}).C_0 + (seekingHelpFromTTP, r_{seekingHelp}).C_9 \\
C_8 &\stackrel{def}{=} (sendMAbort, r_{sendMAbort}).C_7 \\
C_9 &\stackrel{def}{=} (sendTTPinfo, r_{sendTTPinfo}).C_{10} \\
C_{10} &\stackrel{def}{=} (forwardKtoC, r_{forwardKtoC}).C_7 + (sendCkByTTP, r_{sendCkByTTP}).C_7 \\
&\quad + (discoverMisbehavingC, r_{discoverMisbehavingC}).C_7
\end{aligned}$$

The above component presents C's different behaviours, moving from C_0 to C_{10} . In this model, the local states of C increased to eleven compared to the model without misbehaviour between C and M. This also has similar states as in the optimistic anonymous protocol (Subsection 4.3.3) except from state C_6 which when the dispute occurs and/or C's timeout expired. Then C seeks help form TTP and sends some evidence to TTP to solve the problem. Then, the appropriate actions based on the specified rate are followed to solve the dispute between M and C, as presented in the model.

TTP component

$$\begin{aligned}
TTP_0 &\stackrel{def}{=} (download, r_d).TTP_0 + (sendTTPinfo, r_{sendTTPinfo}).TTP_1 \\
TTP_1 &\stackrel{def}{=} (validateCoinToB, r_{vr}).TTP_2 \\
TTP_2 &\stackrel{def}{=} (sendTTPyes, r_{yes}).TTP_3 + (sendTTPno, r_{no}).TTP_7 \\
TTP_3 &\stackrel{def}{=} (askMforValidK, r_{askMforValidK}).TTP_4 \\
TTP_4 &\stackrel{def}{=} (sendTTPvalidK, r_{sendTTPvalidK}).TTP_{5a} + (timeoutTTP, r_{timeoutTTP}).TTP_6 \\
&\quad + (sendTTPreason, r_{sendTTPreason}).TTP_{4a} \\
TTP_{4a} &\stackrel{def}{=} (sendTTPvalidK, r_{sendTTPvalidK}).TTP_5 \\
TTP_5 &\stackrel{def}{=} (sendMpkbyTTP, r_{sendMpkbyTTP}).TTP_{5a} \\
TTP_{5a} &\stackrel{def}{=} (forwardKtoC, r_{forwardKtoC}).TTP_0 \\
TTP_6 &\stackrel{def}{=} (sendCkByTTP, r_{sendCkByTTP}).TTP_8 \\
TTP_7 &\stackrel{def}{=} (investigationInvalidCoinToB, r_{investigationInvalidCoinToB}).TTP_9 \\
TTP_8 &\stackrel{def}{=} (takeActionAgainstM, r_{takeActionAgainstM}).TTP_0 \\
TTP_9 &\stackrel{def}{=} (cspentTheCoinToTTP, r_{cspentTheCoin}).TTP_{10} \\
&\quad + (mspentTheCoinToTTP, r_{mspentTheCoin}).TTP_3 \\
TTP_{10} &\stackrel{def}{=} (discoverMisbehavingC, r_{discoverMisbehavingC}).TTP_0
\end{aligned}$$

In this model, TTP becomes a sequential component and has thirteen states compared to TTP component in previous PEPA models. TTP moves from states TTP_0 to TTP_{10} to solve the dispute between C and M. The number of actions increased and are performed based on the specified rates in order for TTP to involve in the interaction and provide a fair resolution for the disputed parties. TTP's main actions are *download*, *validateCoinToB*, *askMforValidK*, *timeoutTTP*, *sendMpkbyTTP*, *forwardKtoC*, *sendCkByTTP*, *investigationInvalidCoinToB*, *takeActionAgainstM* and *discoverMisbehavingC*. TTP controls the rates of those actions.

Bank component

$$\begin{aligned}
B &\stackrel{def}{=} (requestBDigitalCoins, r_{requestBDC}).B + (sendCDigitalCoins, r_{sendCDC}).B \\
&\quad + (sendBCoinByM, r_{sendBCByM}).B + (sendMyes, r_{sendMyes}).B \\
&\quad + (sendMno, r_{sendMno}).B + (cspentTheCoinToTTP, r_{cspentTheCoin}).B \\
&\quad + (mspentTheCoinToTTP, r_{mspentTheCoin}).B + (validateCoinToB, r_{vc}).B \\
&\quad + (sendTTPyes, r_{yes}).B + (sendTTPno, r_{no}).B \\
&\quad + (investigationInvalidCoinToB, r_{investigationInvalidCoinToB}).B
\end{aligned}$$

The last part of the model is for B component. As in the optimistic anonymous protocol model, B just has one state. In this model, B's actions increased compared to the B

component in the previous model. The B's main actions to support the purchase processes between the components C and M are *sendCDigitalCoins*, *sendMyes* and *sendMno* and to support the dispute resolution are *sendTTPyes*, *sendTTPno*, *cspentTheCoinToTTP* and *mspentTheCoinToTTP* as described in the scenarios of the extended optimistic anonymous protocol specification (Subsection 4.2.4). The rates of these actions are controlled by B.

The system equation, The system equation and complete specification are given by

$$\text{System} \stackrel{\text{def}}{=} \text{TTP}[K] \bowtie_R (B[S] \bowtie_M (C_0[N] \bowtie_L M_0[N]))$$

Where the cooperation sets $R = \{\text{download}, \text{sendTTPinfo}, \text{validateCoinToB}, \text{sendTTPyes}, \text{sendTTPno}, \text{askMforValidK}, \text{sendTTPvalidK}, \text{timeoutTTP}, \text{sendTTPreason}, \text{sendMpkbyTTP}, \text{forwardKtoC}, \text{sendCkByTTP}, \text{investigationInvalidCoinToB}, \text{takeActionAgainstM}, \text{cspentTheCoinToTTP}, \text{mspentTheCoinToTTP}, \text{discoverMisbehavingC}\}$, $M = \{\text{requestBDigitalCoins}, \text{sendCDigitalCoins}, \text{sendMno}, \text{sendBCoinByM}, \text{sendMyes}\}$ and $L = \{\text{sendMPO}, \text{sendCEP}, \text{sendCAbort}, \text{sendMCoinDk}, \text{sendMAbort}, \text{sendCPDk}, \text{complete}, \text{cTimeoutExpired}\}$, any action in the lists R , L and M is shared action between the components specified in the system equation. N is the number of customers and merchant copies on the system, K is the number of TTPs, S is the number of Bs. The four components are initially in the states TTP_0 , C_0 , M_0 and B .

Furthermore, like all the previous PEPA models, the service rates of all the main actions carried out by M depend on the number of Cs interacting with M. The M's main actions are *sendCEP*, *sendCAbort*, *startContactB*, *sendBCoinByM*, *sendCPDk*, *sendTTPvalidK* and *sendTTPreason*. Also, the service rates of all the main actions of B are calculated based on the number of C and M's copies as well as the number of Bs involved in the interaction, as described in Subsection 4.3.3. The B's main actions are *sendCDigitalCoins*, *sendMno*, *sendMyes*, *mspentTheCoinToTTP*, *cspentTheCoinToTTP*, *sendTTPyes* and *sendTTPno*. Further, the service rates of all TTP's main actions depend on the number of both Cs and TTPs interacting with each other, as mentioned in Subsection 4.3.2. TTP's main actions are *forwardKtoC*, *download*, *sendCkByTTP*, *askMForValidK*, *takeActionAgainstM*, *discoverIncorrectPTK*, *sendMpKbyTTP*, *validateCoinToB*, *sendMpkbyTTP* and *investigationInvalidCoinToB*.

Alternative PEPA model for the extended optimistic anonymous protocol with probabilities distribution of misbehaviour

The extended PEPA model with the probabilities of M misbehaviour is formulated as follows:

Merchant component

$$\begin{aligned}
M_0 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).M_1 \\
M_1 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).M_2 + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_8 \\
M_2 &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCoinDk}}).M_3 + (\text{sendMAbort}, r_{\text{sendMAbort}}).M_6 \\
M_3 &\stackrel{\text{def}}{=} (\text{startContactB}, r_{\text{startContactB}}).M_{3a} + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_6 \\
M_{3a} &\stackrel{\text{def}}{=} (\text{sendBCoinByM}, r_{\text{sendBCoinByM}}).M_4 \\
M_4 &\stackrel{\text{def}}{=} (\text{sendMyes}, r_{\text{sendMyes}}).M_5 + (\text{sendMno}, r_{\text{sendMno}}).M_7 \\
M_5 &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).M_6 + (\text{cTimeoutExpired}, r_{\text{cTimeoutExpired}}).M_6 \\
M_6 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_0 + (\text{askMforValidK}, r_{\text{askMforValidK}}).M_9 \\
M_7 &\stackrel{\text{def}}{=} (\text{sendCAbort}, r_{\text{sendCAbort}}).M_6 \\
M_8 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_6 \\
M_9 &\stackrel{\text{def}}{=} (\text{sendTTPvalidK}, r_{\text{sendTTPvalidK}}).M_6 + (\text{timeoutTTP}, r_{\text{timeoutTTP}}).M_{10} \\
&\quad + (\text{sendTTPreason}, r_{\text{sendTTPreason}}).M_{11} \\
M_{10} &\stackrel{\text{def}}{=} (\text{takeActionAgainstM}, r_{\text{takeActionAgainstM}}).M_6 \\
M_{11} &\stackrel{\text{def}}{=} (\text{sendTTPvalidK}, r_{\text{sendTTPvalidK}}).M_{12} \\
M_{12} &\stackrel{\text{def}}{=} (\text{sendMpkbyTTP}, r_{\text{sendMpkbyTTP}}).M_6
\end{aligned}$$

The states and actions of M in this model are similar to those in the previous PEPA model except in state M_3 and M_7 . In these states, the merchant can send customer abort and then move to M_6 instead of moving to M_8 . So merchant just sent an abort message without the necessity to receive an abort from the customer, if the merchant is not satisfied.

Customer component

$$\begin{aligned}
C_0 &\stackrel{\text{def}}{=} (\text{download}, r_d).C_1 \\
C_1 &\stackrel{\text{def}}{=} (\text{requestBDigitalCoins}, r_{\text{requestBDC}}).C_2 \\
C_2 &\stackrel{\text{def}}{=} (\text{sendCDigitalCoins}, r_{\text{sendCDC}}).C_3 \\
C_3 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).C_4 \\
C_4 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).C_5 + (\text{sendCAbort}, r_{\text{sendCAbort}}).C_8 \\
C_5 &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCDk}}).C_6 + (\text{sendMAbort}, r_{\text{sendMAbort}}).C_7 \\
C_6 &\stackrel{\text{def}}{=} (\text{sendCPDk}, p * r_{\text{sendCPDk}}).C_7 + (\text{sendCPDk}, (1 - p) * r_{\text{sendCPDk}}).C_9 \\
&\quad + (\text{sendCAbort}, p * r_{\text{sendCAbort}}).C_7 + (\text{sendCAbort}, (1 - p) * r_{\text{sendCAbort}}).C_9 \\
&\quad + (\text{cTimeoutExpired}, p * r_{\text{cTimeoutExpired}}).C_7 \\
&\quad + (\text{cTimeoutExpired}, (1 - p) * r_{\text{cTimeoutExpired}}).C_9 \\
C_7 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).C_0 \\
C_8 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).C_7
\end{aligned}$$

$$\begin{aligned}
C_9 &\stackrel{\text{def}}{=} (sendTTPinfo, r_{sendTTPinfo}).C_{10} \\
C_{10} &\stackrel{\text{def}}{=} (forwardKtoC, r_{forwardKtoC}).C_7 + (sendCkByTTP, r_{sendCkByTTP}).C_7 \\
&\quad + (discoverMisbehavingC, r_{discoverMisbehavingC}).C_7
\end{aligned}$$

The above component represents C's different behaviours which are similar to the C component in the previous PEPA model except for state C_6 . In C_6 , we introduced a probability of M misbehaving in the actions rates as the customer is the one who always initiates contact with TTP seeking dispute resolution. p indicates the probability that C will receive an honest or satisfactory response from M (assuming M is honest), and $(1 - p)$ indicates the probability that C will receive an invalid or no response from M (assuming M is misbehaving). When C receives an invalid or no response from M, C will initiate contact with TTP to resolve the dispute. Therefore, in C_6 , there are 6 actions could happen either $sendCPDk$ at rate $r_{sendCPDk} * p$ moving to C_7 , $sendCPDk$ at rate $r_{sendCPDk} * (1 - p)$ moving to C_9 to seek a dispute resolution form TTP, $sendCAbort$ at rate $r_{sendCAbort} * p$ moving to C_7 , $sendCAbort$ at rate $r_{sendCAbort} * (1 - p)$ moving to C_9 , $cTimeoutExpired$ at rate $r_{cTimeoutExpired} * p$ moving to C_7 or $cTimeoutExpired$ at rate $r_{cTimeoutExpired} * (1 - p)$ moving to C_9 .

TTP and B components, TTP and B components have the same states and actions as TTP and B components in the previous PEPA model.

The system equation, The system equation is the same as the equation in the previous model.

Alternative PEPA model for the extended optimistic anonymous protocol when an honest M interacts with two types of C

This subsection proposes a PEPA model presenting honest M's copies interacting with two types of C; honest C and misbehaving C. In this model, we introduced two honest M. Each M has the appropriate states and behaviours to interact with one type of customer. The model comprises six main parts: two Ms, two Cs, one TTP and one B. The PEPA model is formulated as follows:

Merchant component

$$\begin{aligned}
M_0 &\stackrel{\text{def}}{=} (sendMPO, r_{sendMPO}).M_1 \\
M_1 &\stackrel{\text{def}}{=} (sendCEP, r_{sendCEP}).M_2 + (sendCAbort, r_{sendCAbort}).M_8 \\
M_2 &\stackrel{\text{def}}{=} (sendMCoinDk, r_{sendMCoinDk}).M_3 + (sendMAbort, r_{sendMAbort}).M_6
\end{aligned}$$

$$\begin{aligned}
M_3 &\stackrel{\text{def}}{=} (\text{startContactB}, r_{\text{startContactB}}).M_{3a} + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_8 \\
M_{3a} &\stackrel{\text{def}}{=} (\text{sendBCoinByM}, r_{\text{sendBCoinByM}}).M_4 \\
M_4 &\stackrel{\text{def}}{=} (\text{sendMno}, r_{\text{sendMno}}).M_5 + (\text{sendMno}, r_{\text{sendMno}}).M_7 \\
M_5 &\stackrel{\text{def}}{=} (\text{cTimeoutExpired}, r_{\text{cTimeoutExpired}}).M_6 \\
M_6 &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_0 \\
M_7 &\stackrel{\text{def}}{=} (\text{sendCAbort}, r_{\text{sendCAbort}}).M_6 \\
M_8 &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_6
\end{aligned}$$

The above part of the model presents the different states and actions of the honest merchant when interacting with the misbehaving customer. In state M_5 , M does not send C the product decryption key when M is informed by B that the coin is invalid during the state M_4 . This means that C is playing unfairly.

The following part of the model from M_{00} to M_{08} shows the different states and actions of the honest merchant when interacting with an honest customer. The two parts have the same actions and states except for state M_5 and M_{05} which have different actions to deal with different customer type.

$$\begin{aligned}
M_{00} &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).M_{01} \\
M_{01} &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).M_{02} + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_{08} \\
M_{02} &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCoinDk}}).M_{03} + (\text{sendMAbort}, r_{\text{sendMAbort}}).M_{06} \\
M_{03} &\stackrel{\text{def}}{=} (\text{startContactB}, r_{\text{startContactB}}).M_{03a} + (\text{sendCAbort}, r_{\text{sendCAbort}}).M_{08} \\
M_{03a} &\stackrel{\text{def}}{=} (\text{sendBCoinByM}, r_{\text{sendBCoinByM}}).M_{04} \\
M_{04} &\stackrel{\text{def}}{=} (\text{sendMyes}, r_{\text{sendMyes}}).M_{05} + (\text{sendMno}, r_{\text{sendMno}}).M_{07} \\
M_{05} &\stackrel{\text{def}}{=} (\text{sendCPDk}, r_{\text{sendCPDk}}).M_{06} \\
M_{06} &\stackrel{\text{def}}{=} (\text{complete}, r_{\text{complete}}).M_{00} \\
M_{07} &\stackrel{\text{def}}{=} (\text{sendCAbort}, r_{\text{sendCAbort}}).M_{06} \\
M_{08} &\stackrel{\text{def}}{=} (\text{sendMAbort}, r_{\text{sendMAbort}}).M_{06}
\end{aligned}$$

Customer component

$$\begin{aligned}
C_0 &\stackrel{\text{def}}{=} (\text{download}, r_d).C_1 \\
C_1 &\stackrel{\text{def}}{=} (\text{requestBDigitalCoins}, r_{\text{requestBDC}}).C_2 \\
C_2 &\stackrel{\text{def}}{=} (\text{sendCDigitalCoins}, r_{\text{sendCDC}}).C_3 \\
C_3 &\stackrel{\text{def}}{=} (\text{sendMPO}, r_{\text{sendMPO}}).C_4 \\
C_4 &\stackrel{\text{def}}{=} (\text{sendCEP}, r_{\text{sendCEP}}).C_5 + (\text{sendCAbort}, r_{\text{sendCAbort}}).C_8 \\
C_5 &\stackrel{\text{def}}{=} (\text{sendMCoinDk}, r_{\text{sendMCDk}}).C_6 + (\text{sendMAbort}, r_{\text{sendMAbort}}).C_7 \\
C_6 &\stackrel{\text{def}}{=} (\text{sendCAbort}, r_{\text{sendCAbort}}).C_9 \\
&+ (\text{cTimeoutExpired}, r_{\text{cTimeoutExpired}}).C_9
\end{aligned}$$

$$\begin{aligned}
C_7 &\stackrel{\text{def}}{=} (complete, r_{complete}).C_0 \\
C_8 &\stackrel{\text{def}}{=} (sendMAbort, r_{sendMAbort}).C_7 \\
C_9 &\stackrel{\text{def}}{=} (sendTTPinfo, r_{sendTTPinfo}).C_{10} \\
C_{10} &\stackrel{\text{def}}{=} (discoverMisbehavingC, r_{discoverMisbehavingC}).C_7
\end{aligned}$$

The above part of the model from C_0 to C_{10} are the different states and actions for a misbehaving customer. In state C_6 , the only two actions that could happen are $sendCAbort$ or $cTimeoutExpired$, which lead to state C_9 seeking help from TTP. So the misbehaving customer does not get the product decryption key from M as a result of their misbehaviour.

The following part of the model is for the honest customer different behaviours, moving from C_{00} to C_{08} . This part of the model is the same as the C component in the PEPA model of the optimistic primitive protocol when there is no dispute between the two parties.

$$\begin{aligned}
C_{00} &\stackrel{\text{def}}{=} (download, r_d).C_{01} \\
C_{01} &\stackrel{\text{def}}{=} (requestBDigitalCoins, r_{requestBDC}).C_{02} \\
C_{02} &\stackrel{\text{def}}{=} (sendCDigitalCoins, r_{sendCDC}).C_{03} \\
C_{03} &\stackrel{\text{def}}{=} (sendMPO, r_{sendMPO}).C_{04} \\
C_{04} &\stackrel{\text{def}}{=} (sendCEP, r_{sendCEP}).C_{05} + (sendCAbort, r_{sendCAbort}).C_{08} \\
C_{05} &\stackrel{\text{def}}{=} (sendMCoinDk, r_{sendMCDk}).C_{06} + (sendMAbort, r_{sendMAbort}).C_{07} \\
C_{06} &\stackrel{\text{def}}{=} (sendCAbort, r_{sendCAbort}).C_{08} + (sendCPDk, r_{sendCPDk}).C_{07} \\
C_{07} &\stackrel{\text{def}}{=} (complete, r_{complete}).C_{00} \\
C_{08} &\stackrel{\text{def}}{=} (sendMAbort, r_{sendMAbort}).C_{07}
\end{aligned}$$

TTP component

$$\begin{aligned}
TTP_0 &\stackrel{\text{def}}{=} (download, r_d).TTP_0 + (sendTTPinfo, r_{sendTTPinfo}).TTP_1 \\
TTP_1 &\stackrel{\text{def}}{=} (validateCoinToB, r_{vr}).TTP_2 \\
TTP_2 &\stackrel{\text{def}}{=} (sendTTPno, r_{no}).TTP_3 \\
TTP_3 &\stackrel{\text{def}}{=} (investigationInvalidCoinToB, r_{investigationInvalidCoinToB}).TTP_4 \\
TTP_4 &\stackrel{\text{def}}{=} (cspentTheCoinToTTP, r_{cspentTheCoin}).TTP_5 \\
TTP_5 &\stackrel{\text{def}}{=} (discoverMisbehavingC, r_{discoverMisbehavingC}).TTP_0
\end{aligned}$$

The TTP component contains just 6 states to solve the dispute between an honest M and a misbehaving C. The TTP checks the validation of the coin with B during TTP_1 . The B informs TTP that the coin is invalid during TTP_2 . In state TTP_3 , TTP performs $investigationInvalidCoinToB$ at rate $r_{investigationInvalidCoinToB}$ to investigate on who spent the digital coin. In state TTP_4 and TTP_5 , TTP performs $cspentTheCoinToTTP$ and $discoverMisbehavingC$, respectively. TTP discovers that C plays unfairly.

Bank component

$$\begin{aligned}
B \stackrel{def}{=} & (requestBDigitalCoins, r_{requestBDC}).B + (sendCDigitalCoins, r_{sendCDC}).B \\
& + (sendBCoinByM, r_{sendBCByM}).B + (sendMyes, r_{sendMyes}).B \\
& + (sendMno, r_{sendMno}).B + (cspentTheCoinToTTP, r_{cspentTheCoin}).B \\
& + (validateCoinToB, r_{vc}).B + (sendTTPno, r_{no}).B \\
& + (investigationInvalidCoinToB, r_{investigationInvalidCoinToB}).B
\end{aligned}$$

The last part of the model is for the B component which has only one state. In this model, B performs nine actions. B's main actions to support the purchase processes between the components C and M are *sendCDigitalCoins*, *sendMyes* and *sendMno* and to support the dispute resolution are *sendTTPno* and *cspentTheCoinToTTP*. The rates of these actions are controlled by B.

The system equation, The system equation and complete specification are given by

$$System \stackrel{def}{=} TTP[K] \bowtie_R (B[S] \bowtie_M (C_0[Y] \bowtie_L M_0[Y] || C_{00}[N-Y] \bowtie_I M_{00}[N-Y]))$$

Where the cooperation sets $R=\{download, sendTTPinfo, validateCoinToB, sendTTPno, investigationInvalidCoinToB, cspentTheCoinToTTP, discoverMisbehavingC\}$, $M=\{requestBDigitalCoins, sendCDigitalCoins, sendMno, sendBCoinByM, sendMyes\}$ and $L=\{sendMPO, sendCEP, sendCAbort, sendMCoinDk, sendMAbort, complete, cTimeoutExpired\}$, $I=\{sendMPO, sendCEP, sendCAbort, sendMCoinDk, sendMAbort, sendCPDk, complete\}$, any action in list R , L , I and M is shared action between the components specified in the system equation. N is the number of clients and merchant copies on the system, Y is a number of misbehaving customers and an honest merchant copies interacting in the system. K is the number of TTPs, S is the number of Bs. The six components are initially in state TTP_0 , C_0 , C_{00} , M_0 , M_{00} and B .

4.4 Numerical results

This section presents the evaluation results of the proposed PEPA models.

4.4.1 Performance evaluation of the basic failure resilient fair-exchange protocol model

In this subsection, the evaluation of PEPA models proposed in Subsection 4.3.1 is presented. We evaluate and study some aspects of the system performance using the ordinary differential equations (ODEs) method. The investigation seeks to calculate the average response time of the merchant that customers will observe, population-level during some states and throughput analyses of some main actions. We apply steady state analysis to calculate the average response time and transient analysis to derive the population level and throughput. We assigned 1 to the rate of all actions except the rate $r_{complete}$ is assigned 0.01 to allow the customer waits before starting again. The M's actions rates are calculated based on the number of customers M interacts with (as mentioned in Subsection 4.3.1).

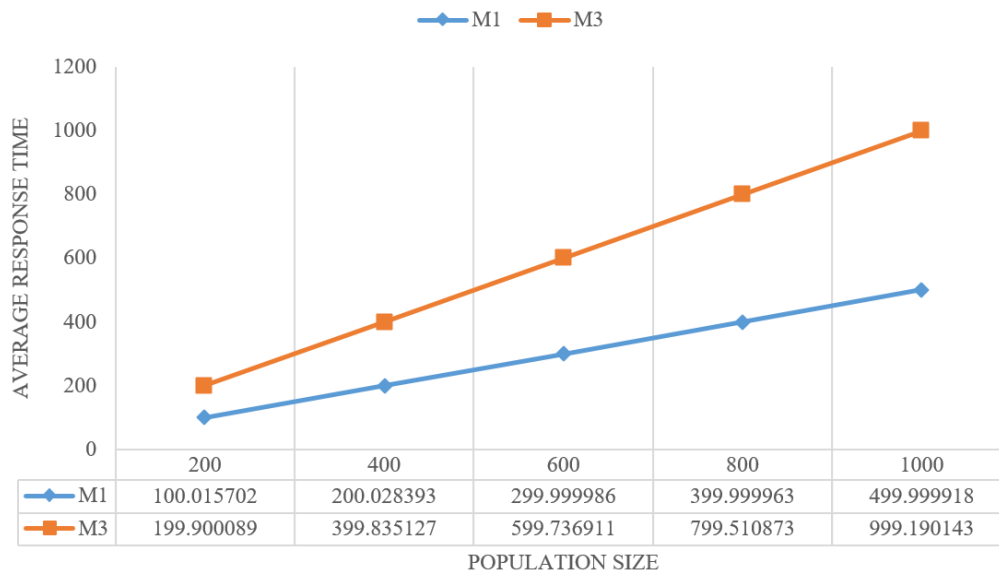


Fig. 4.3 The average response time of M_1 and M_3 using ODE.

Figure 4.3 shows clearly how increasing the number of customers affected the merchant's average response time for the actions $sendCEP$, $sendCAbort$, and $sendCPDk$. M_1 and M_3 are the states of a merchant when performing these actions to respond to customers during the interaction course.

Figures 4.4 and 4.5 shows the population level analysis that shows the average number of C's copies in state C_2 and C_4 waiting to receive a response from M. In the state C_2 , the customers wait to receive a response from M to receive the encrypted product or abort. In the state C_4 , the customers wait to receive the product decryption key.

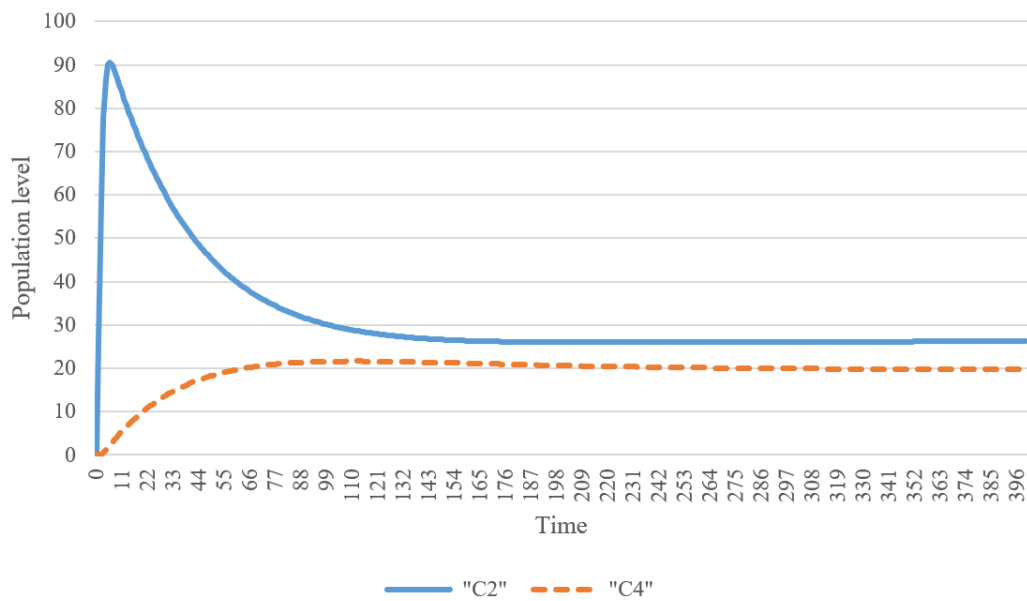


Fig. 4.4 The population level analysis using ODE with $K=100$ and $N=100$.

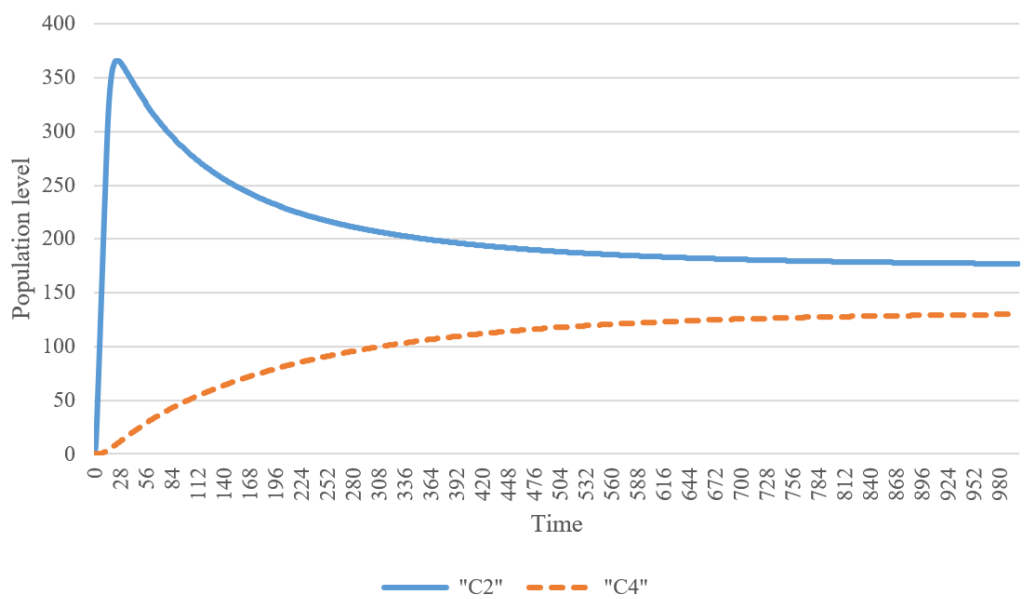


Fig. 4.5 The population level analysis using ODE with $K=100$ and $N=400$.

The average number of copies of C_2 and C_4 increased when N (the number of C and M copies) was increased from 100 to 400. Figure 4.5 shows that having larger customer population size causes the system to take longer time to reach the steady state compared to Figure 4.4. Moreover, more customers waiting to be served in C_2 than in C_4 . This means that some customers abort the process and would not reach the state C_4 .

Figures 4.6 and 4.7 illustrate the throughput values for *sendCAbort*, *sendCEP* and *sendCPDk* actions which are the M 's main actions to response to and serve C . The actions *sendCAbort* and *sendCEP* have the same throughput values whereas action *sendCPDk* has lower values which mean some customers abort the interaction before the stage where the product decryption key is sent. Further, the larger the customer population size, the longer time it takes the system to stabilise and reach a steady state.

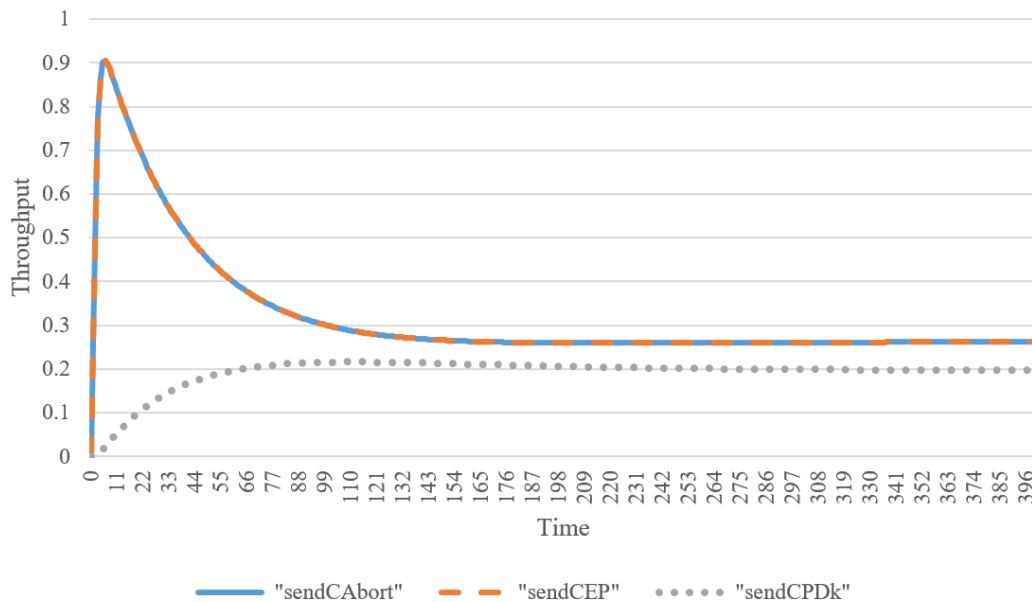


Fig. 4.6 The throughput analysis using ODE with $K=100$ and $N=100$.

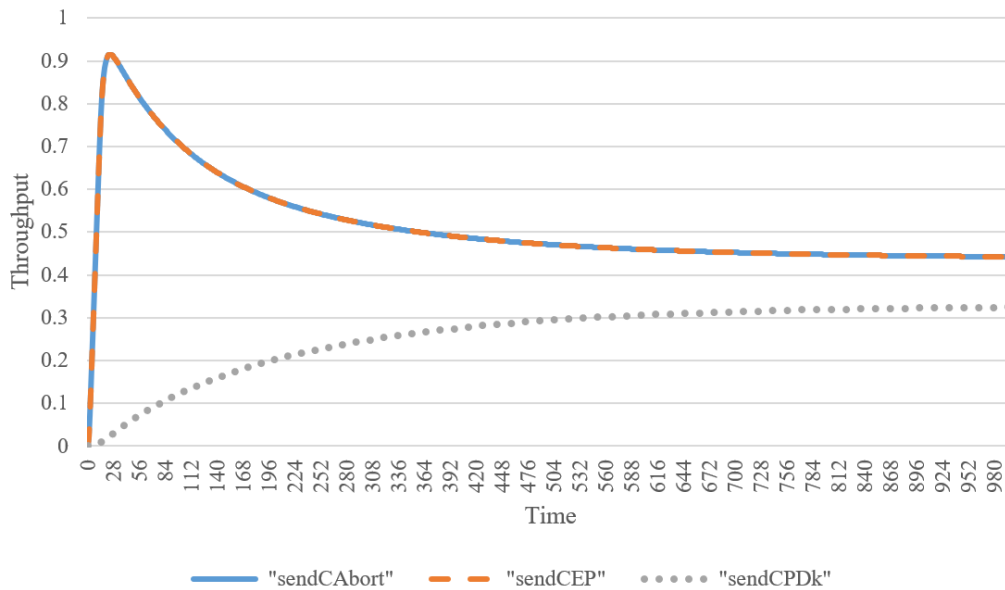


Fig. 4.7 The throughput analysis using ODE with $K=100$ and $N=400$.

4.4.2 Performance evaluation of the extended failure resilient fair exchange protocol model

In this subsection, we seek to investigate the average response time of merchant, population-level and throughput analyses when TTP involve in the interaction to solve the dispute between the parties. The rate of all actions is 1 except the rate $r_{complete}$ is assigned 0.01. M's actions rates are calculated based on the number of customers that M interacts with, as mentioned in Subsection 4.3.2.

The average merchant response times are calculated through M_1 and M_3 behaviours using ODE. M_1 and M_3 are the main states when M is responding to serve the customers in the system. Figure 4.8 shows that the average response time of M_1 is slightly larger when misbehaviour occurred compared to the average response time of M_1 when no misbehaviour occurred, as shown in Figure 4.3. In Figure 4.8, the average response time of M_1 is larger and significantly increasing when the number of customers is increased. Whereas the average response time of M_3 is low and slightly increasing when the number of customer increases. Also, the average response times of M_3 are lower compared to M_3 when no misbehaviour occurred, as shown in Figure 4.3. We believe this is because a large number of customers are seeking for resolving the dispute from TTP.

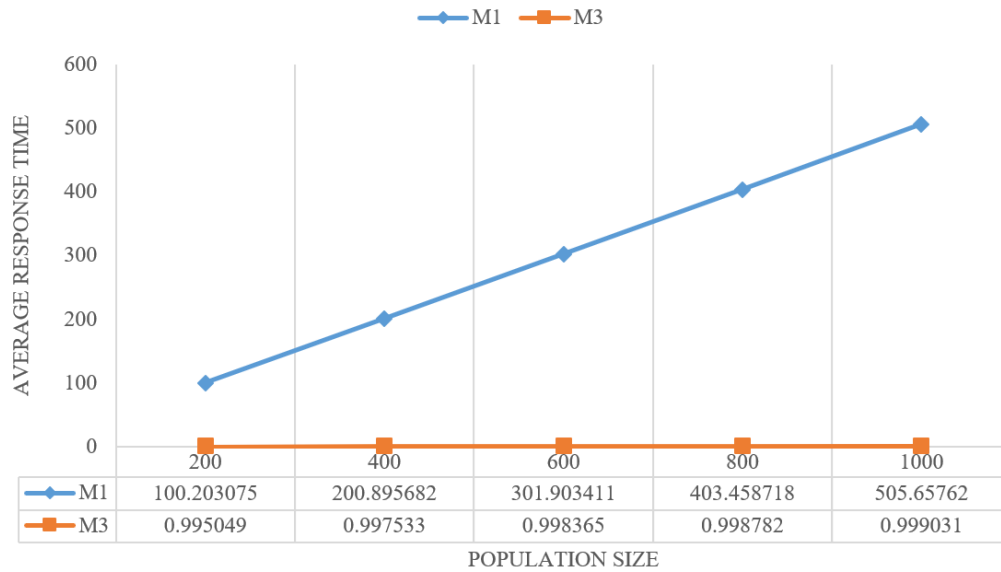


Fig. 4.8 The average response time of M_1 and M_3 using ODE and $K=20$.

Figures from 4.9 to 4.14 show the population level analysis that shows the average number of customers in C_2 and C_4 waiting to have a response from M and in C_8 waiting to get a dispute resolution from TTP in relation to different population size of customers (N) and TTP (K).

Figures 4.9 and 4.10 show that the average number of C 's copies in the states C_2 and C_4 waiting to receive a response from M is decreased when the disputes happen and the TTPs involve solving the disputes compared to the average number of C 's copies in the states C_2 and C_4 when there are no disputes between C and M , as shown in Figures 4.4 and 4.5.

Moreover, Figures from 4.9 to 4.14 clearly illustrate when the number of K (the number of TTP) increases, the number of customers in C_8 decreased. This suggests that the more TTPs participated in the protocol, the better the performance of the extended protocol would be. Figure 4.11 shows the increasing number of C_8 copies when the number of TTP is 20 whereas the number of C_8 copies is decreased when number of TTP increased to 50, as shown in Figure 4.12. However, in Figure 4.13, when the number of customers in the system becomes larger (400) and the number of TTPs involved in solving the disputes remain 50, this creates loads for TTPs and increases the number of customers waiting to receive the dispute resolution in C_8 .

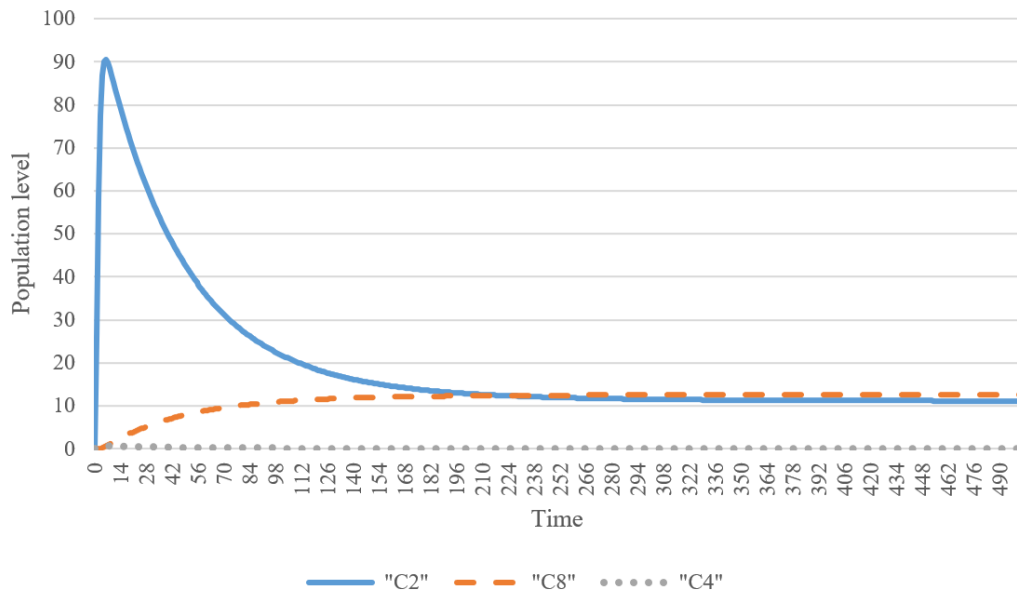


Fig. 4.9 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=100$ and $N=100$.

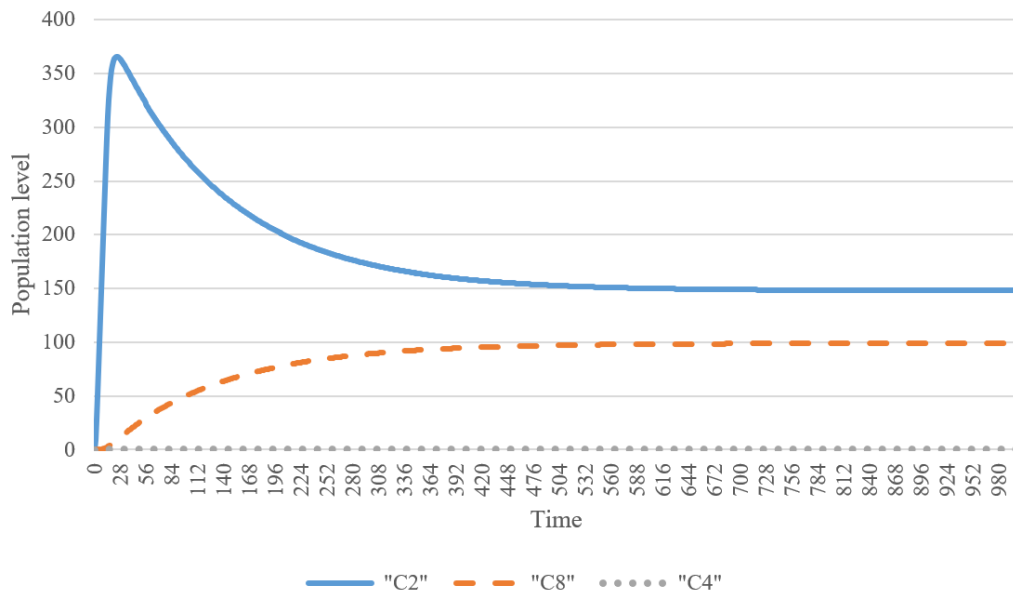


Fig. 4.10 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=100$ and $N=400$.

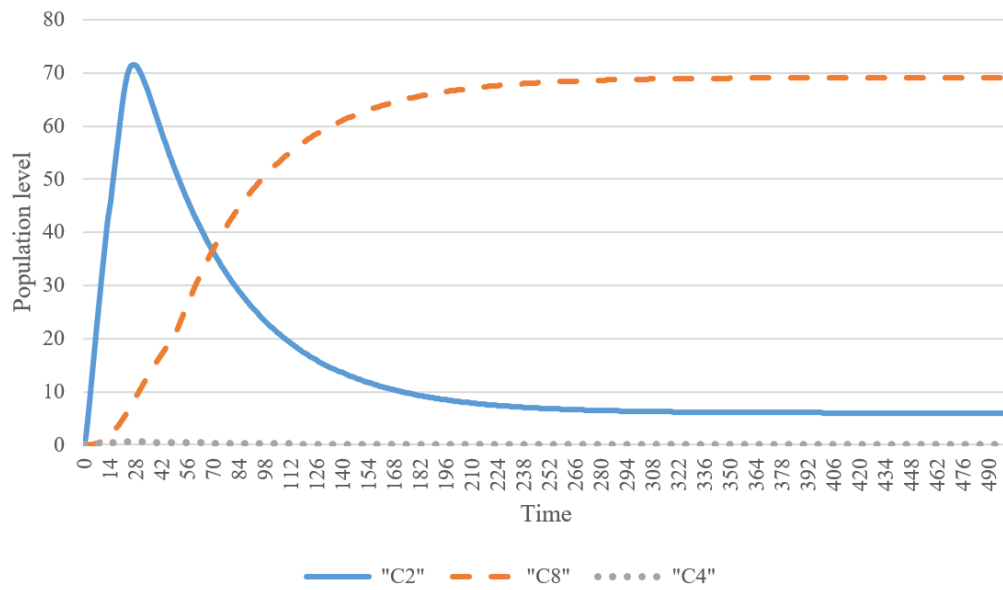


Fig. 4.11 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=20$ and $N=100$.

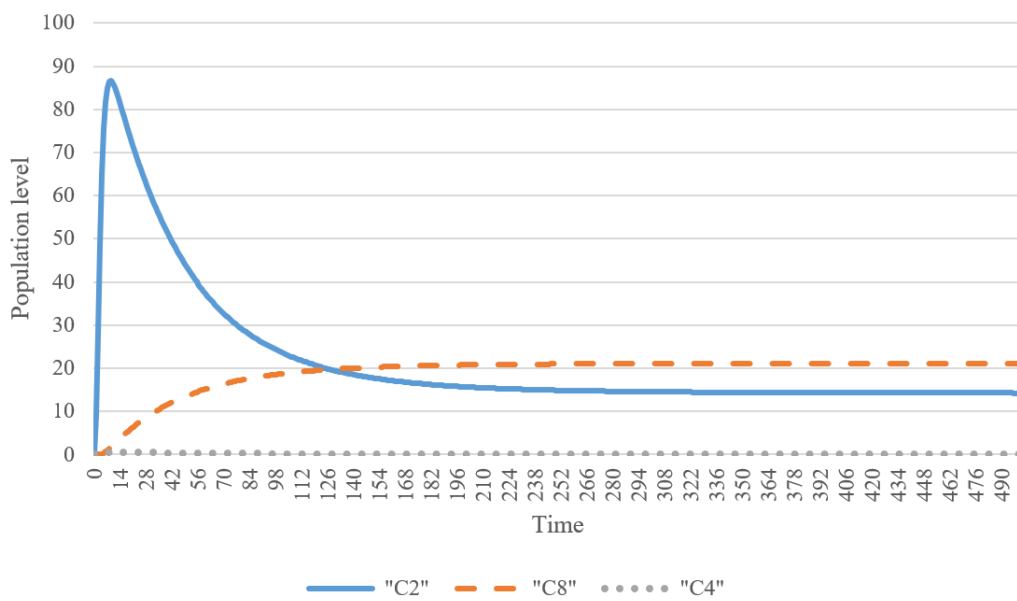


Fig. 4.12 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=50$ and $N=100$.

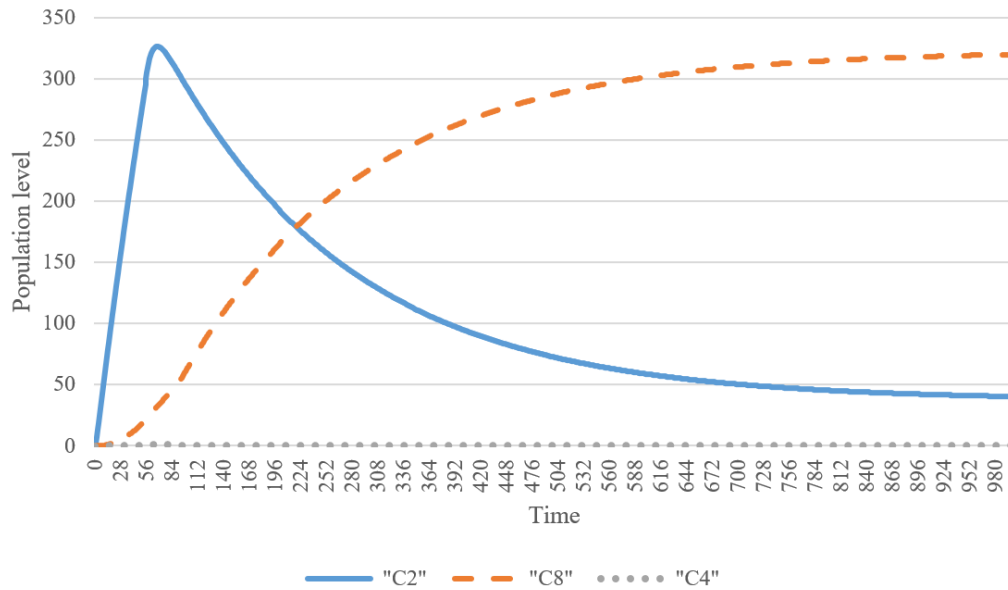


Fig. 4.13 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=50$ and $N=400$.

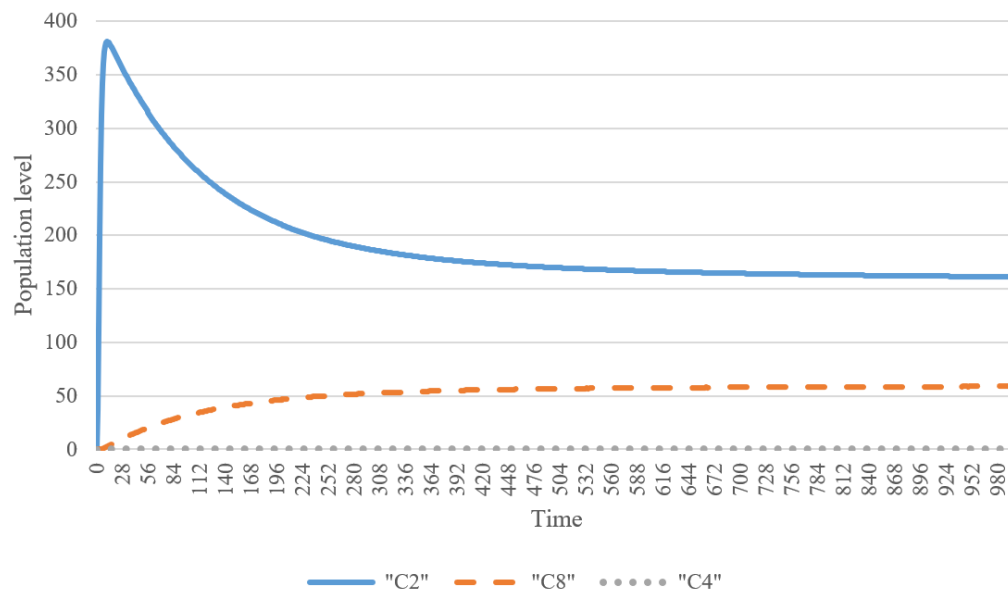


Fig. 4.14 The population level analysis for C_2 , C_4 and C_8 using ODE with $K=200$ and $N=400$.

Figures from 4.15 to 4.22 show the throughput values of the six actions, which are *sendCAbort*, *sendCEP*, *sendCPDk*, *discoverIncorrectPTK*, *forwardKtoC* and *sendCkByTTP*, and how having different population sizes of K and N could affect the throughput values of these actions.

Figures 4.15, 4.16, 4.17 and 4.18 show the throughput values for *sendCAbort*, *sendCEP* and *sendCPDk* actions, which are the M's main actions to respond to and serve C. You can notice from the graphs that the first two actions have the same throughput values. In contrast, action *sendCPDk* has extremely low values, indicating the misbehaviour that occurred from M. Moreover, the figures clearly show the throughput improvement for *sendCAbort*, *sendCEP* actions when the number of TTPs is larger. This is because in our model the customers first download the encrypted product from TTP to validate the encrypted product received from M before sending the payment token decryption key.

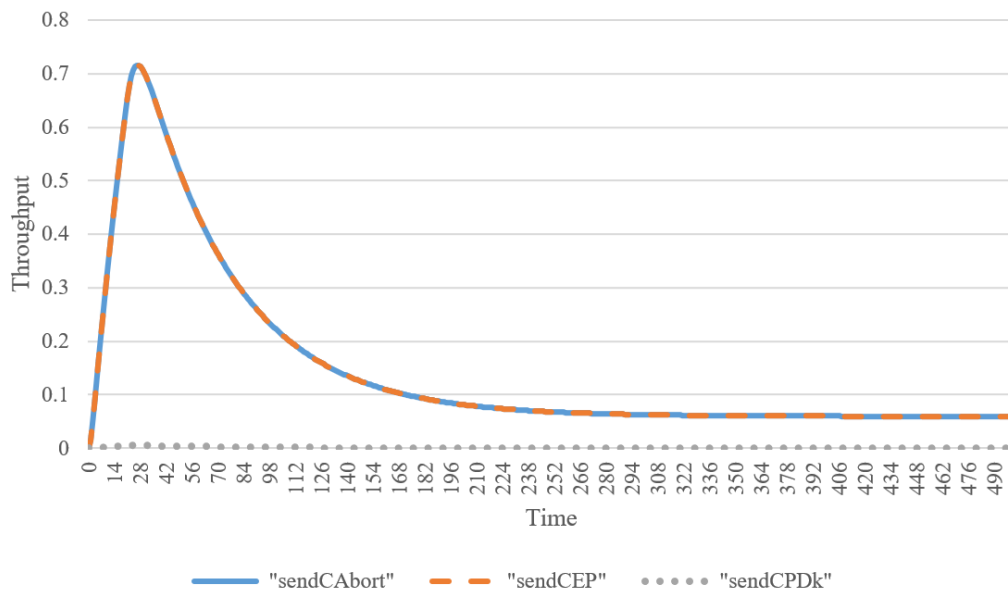


Fig. 4.15 The throughput analysis for *sendCAbort*, *sendCEP* and *sendCPDk* using ODE with $K=20$ and $N=100$.

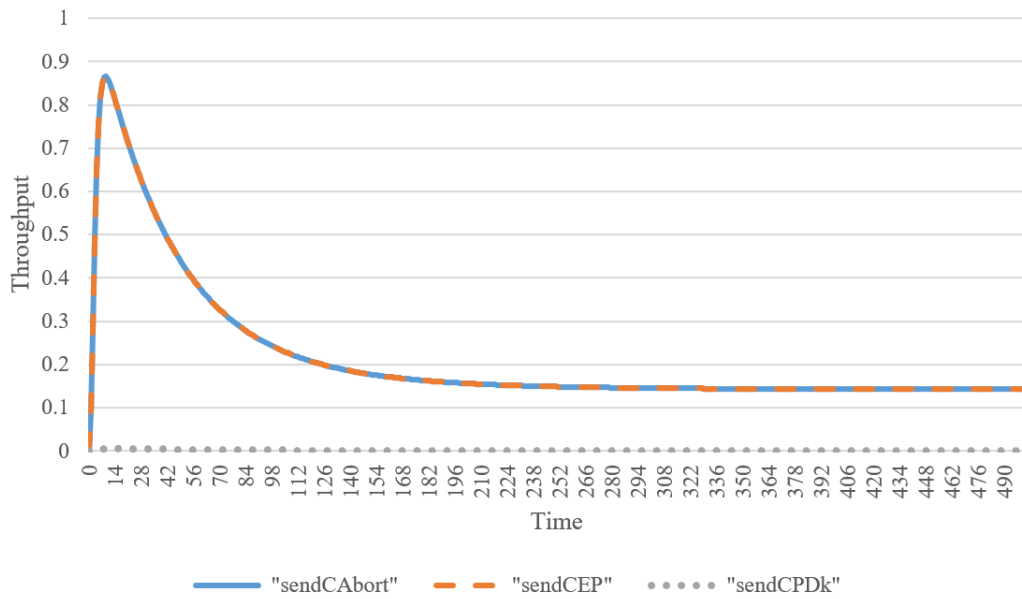


Fig. 4.16 The throughput analysis for *sendCAbort*, *sendCEP* and *sendCPDk* using ODE with $K=50$ and $N=100$.

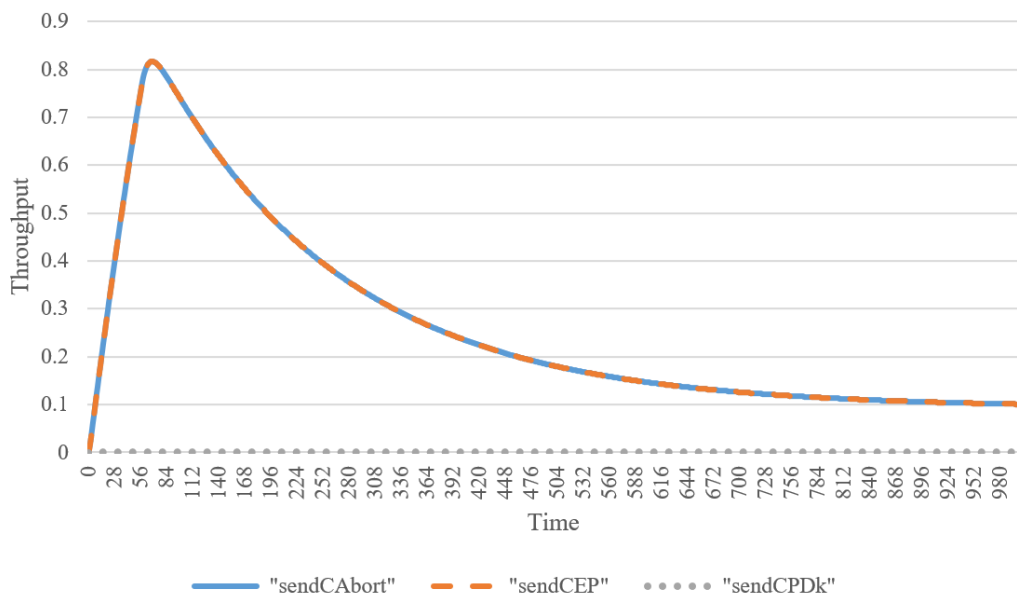


Fig. 4.17 The throughput analysis for *sendCAbort*, *sendCEP* and *sendCPDk* using ODE with $K=50$ and $N=400$.

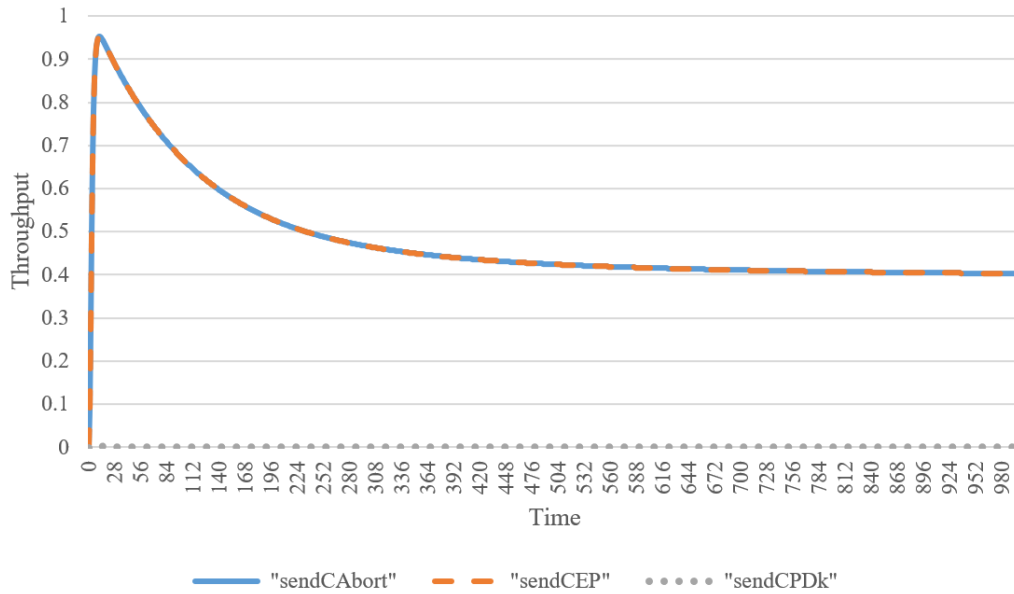


Fig. 4.18 The throughput analysis for *sendCAbort*, *sendCEP* and *sendCPDk* using ODE with $K=200$ and $N=400$.

Figures 4.19, 4.20, 4.21 and 4.22 illustrate the throughput values for *discoverIncorrectPT-K*, *forwardKtoC* and *sendCkByTTP*, which are the main actions of TTP to resolve the dispute between C and M. As you can see from the figures, the throughput values of all three actions have the same values. Graphs reveal that more customers asked for a resolution from TTPs for misbehaviour during the interaction between the customer and the merchant. The throughput of the actions increased when the number of TTP increased. Therefore, we believe that having more TTPs to serve the customers would mitigate the performance cost.

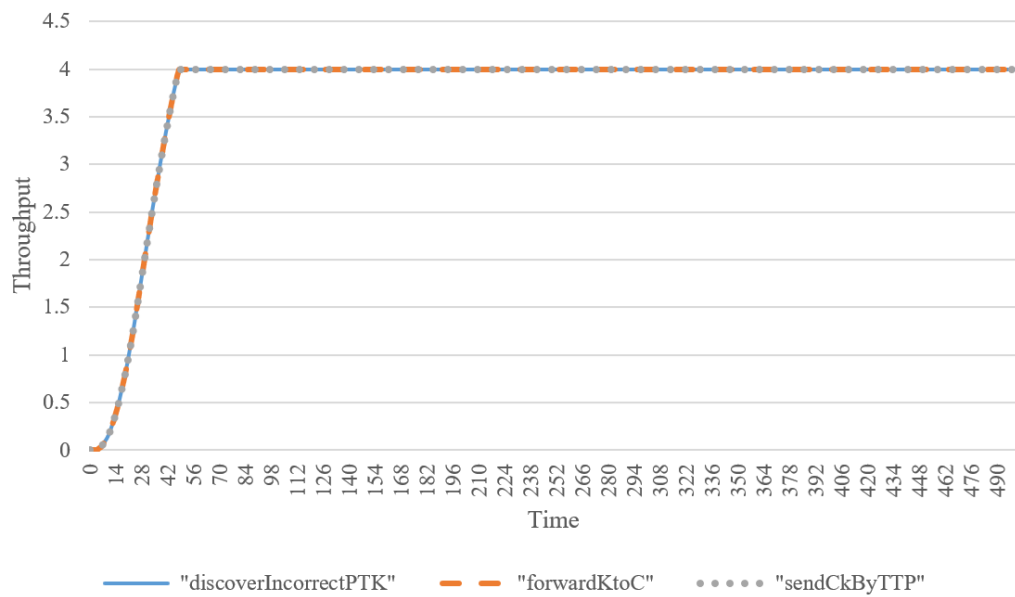


Fig. 4.19 The throughput analysis for *discoverIncorrectPTK*, *forwardKtoC* and *sendCkByTTP* using ODE with $K=20$ and $N=100$.

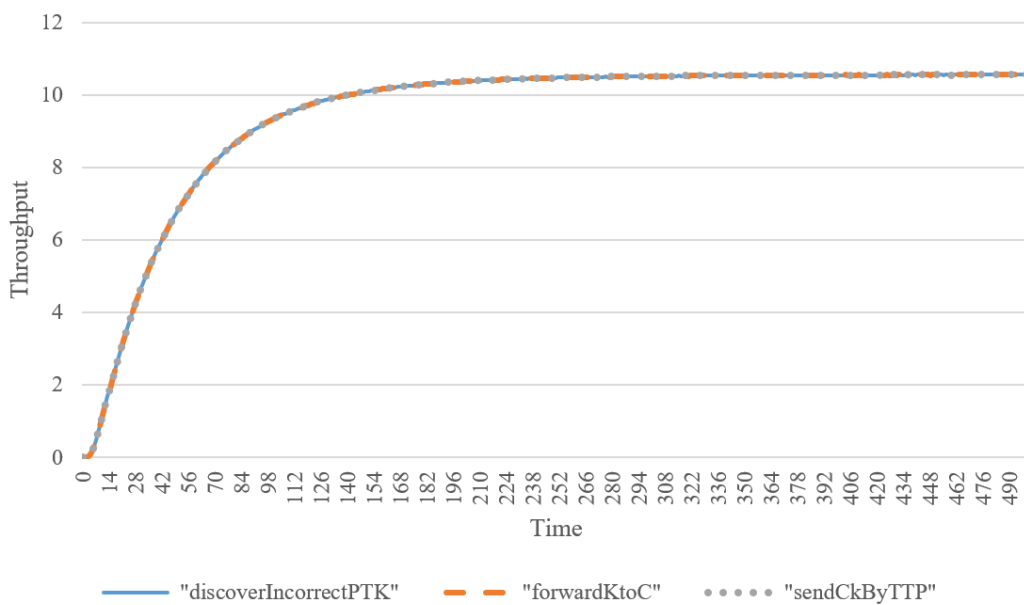


Fig. 4.20 The throughput analysis for *discoverIncorrectPTK*, *forwardKtoC* and *sendCkByTTP* using ODE with $K=50$ and $N=100$.

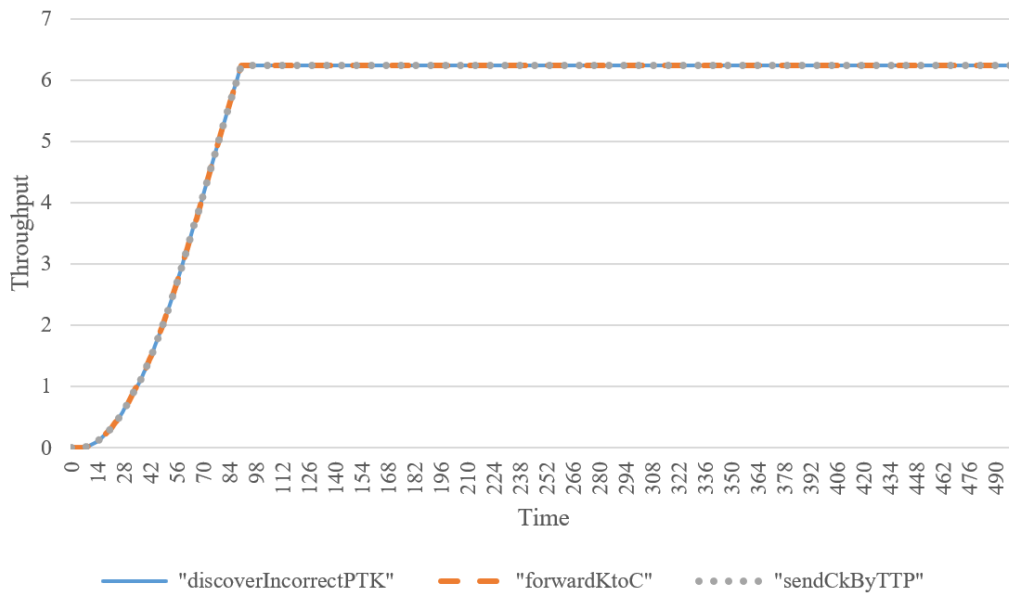


Fig. 4.21 The throughput analysis for *discoverIncorrectPTK*, *forwardKtoC* and *sendCkByTTP* using ODE with $K=50$ and $N=400$.

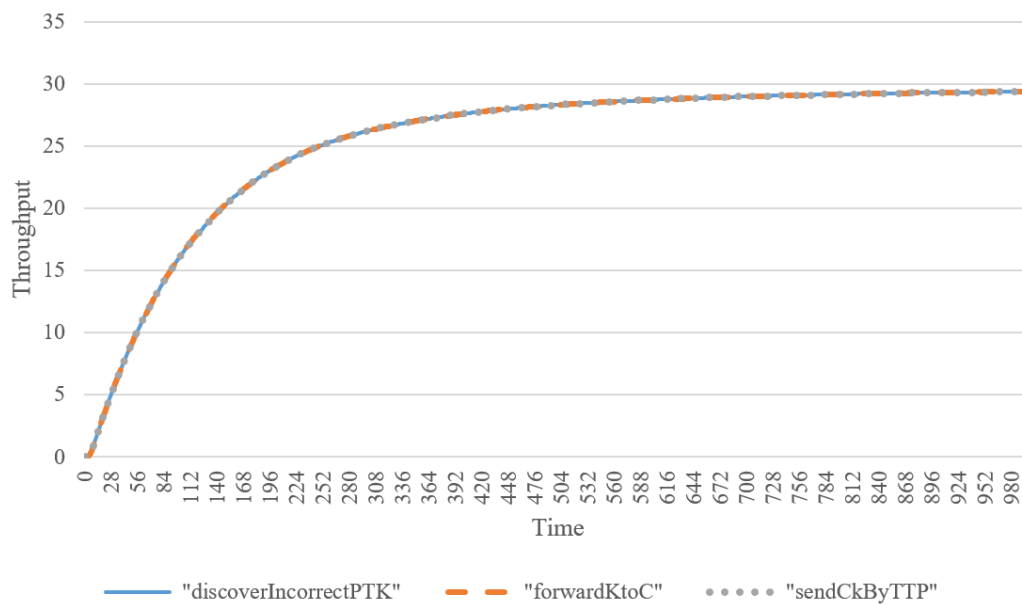


Fig. 4.22 The throughput analysis for *discoverIncorrectPTK*, *forwardKtoC* and *sendCkByTTP* using ODE with $K=200$ and $N=400$.

4.4.3 Performance evaluation of the optimistic anonymous protocol

This investigation also seeks to calculate the average response times of M . We assigned 1 as a value for all rates. However, the main actions of M are calculated based on the number of customers in the system. The average response times of M_1 and M_5 in relation to the number of customers is presented in Figure 4.23. The figure clearly shows that the average response time of M_1 and M_5 increased when the number of customers increased. Moreover, the average response times of M_1 and M_5 are larger when introducing the customer's anonymous feature on this protocol compared to the average response times of M_1 and M_3 of the basic protocol without this feature in Figure 4.3 in Subsection 4.4.1. For instance, in Figure 4.23, when the protocol preserves anonymity and the customer number is 1000, the response time of M_1 is 960.97 time units, which is longer than the response time of M_1 when the protocol does not preserve anonymity; the response time of M_1 is 499.99 time units, as shown in Figure 4.3. M_1 and M_5 in this protocol are same as M_1 and M_3 in the basic protocol, respectively. Additionally, Figures 4.3 and 4.23 indicate that M_5 's response time is slightly longer than M_3 's response time. Therefore, adding such a feature to the protocol would add more performance overhead to the system.

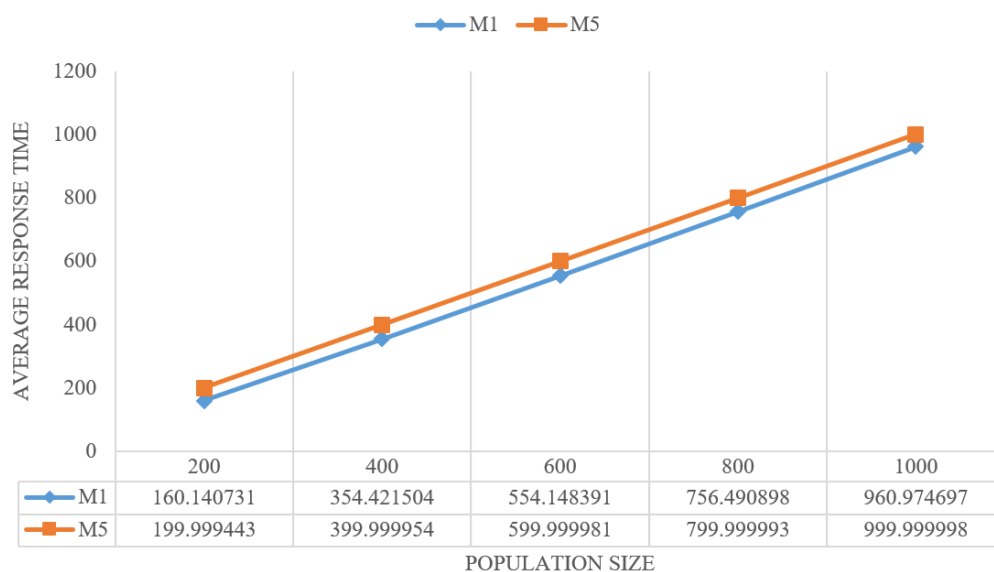


Fig. 4.23 The average response time of M_1 and M_5 using ODE ($K=20$ and $S=20$).

The following graphs are the population level analysis for the average number of customers in C_4 and C_6 for having a service from M and in C_2 for having a service from B to get a digital coin. C_4 is the state when the customer waiting to receive the encrypted product or

abort from M. C_6 is the state when the customer is waiting for the product decryption key or abort from M.

In Figures 4.24, 4.25 and 4.26, we changed the number of the banks involved in the C and M interaction from 5 in Figure 4.24, to 20 in Figure 4.25, and then to 50 in Figure 4.26 whereas the number of customer and TTP are 100 and 20, respectively. The Figures show a decrease in the average number of C_2 copies when the number of B is increased.

Then, in Figure 4.27, we increased the number of TTP to 50, which did not show significant change except the peak of C_2 and C_4 for just a short period of time. Moreover, the figures show a significant increase in the number of customers who are waiting to receive the product decryption key or abort (C_6 copies) from M when the number of bank is relatively small, as in Figure 4.24, compared to C_6 copies when the number of B is larger, as in Figures 4.25 and 4.26.

Then, in Figures 4.28 and 4.29, we increased the number of customer to 400 and kept TTP's to 50 whereas the number of banks are 50 in Figure 4.28 and 100 in Figure 4.29. The Figures also show a decrease in the average number of C_2 copies. Then, in Figure 4.30, we increased the number of TTP to 100, which did not also show significant change except the peak of C_2 and C_4 for just a short period of time. This is because there is no dispute between the parties.

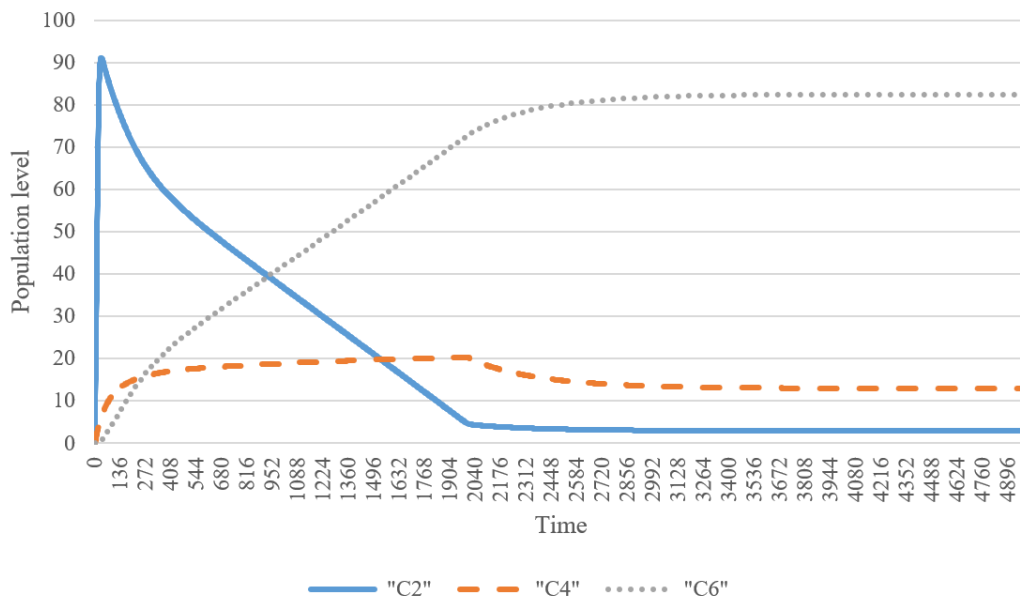


Fig. 4.24 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=5$.

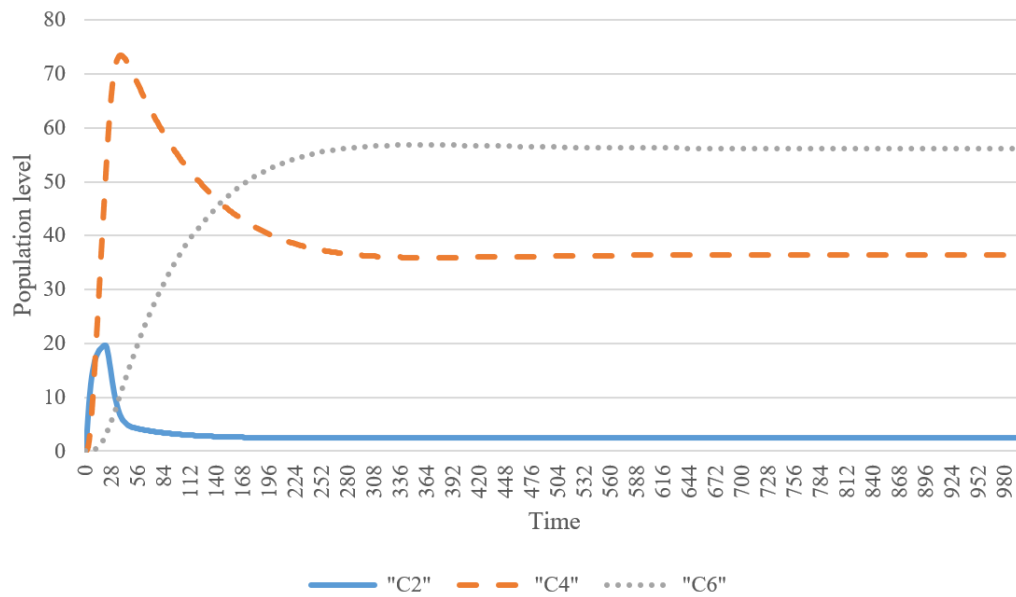


Fig. 4.25 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=20$.

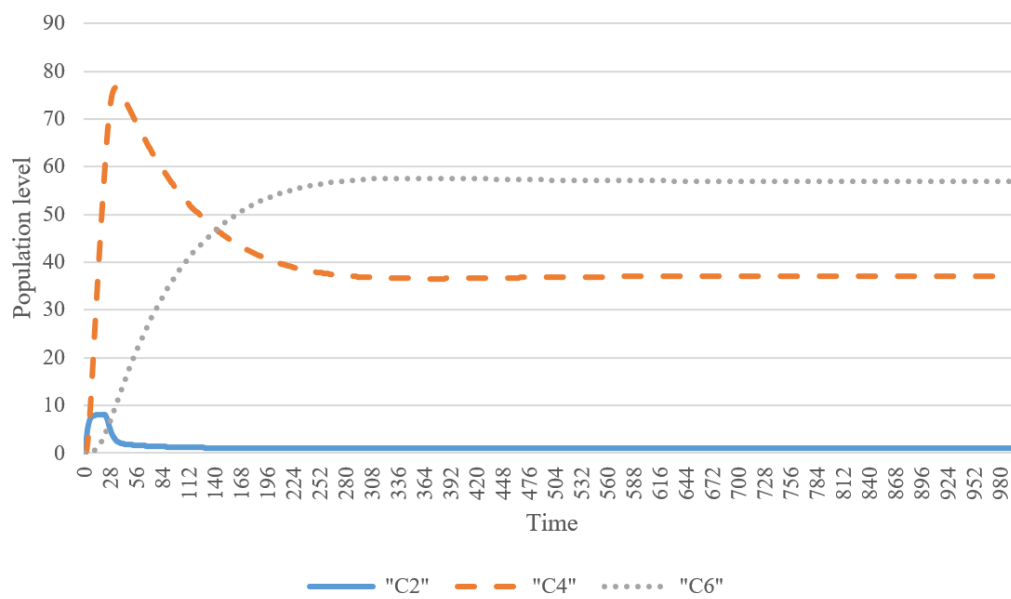


Fig. 4.26 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=20$, $N=100$ and $S=50$.

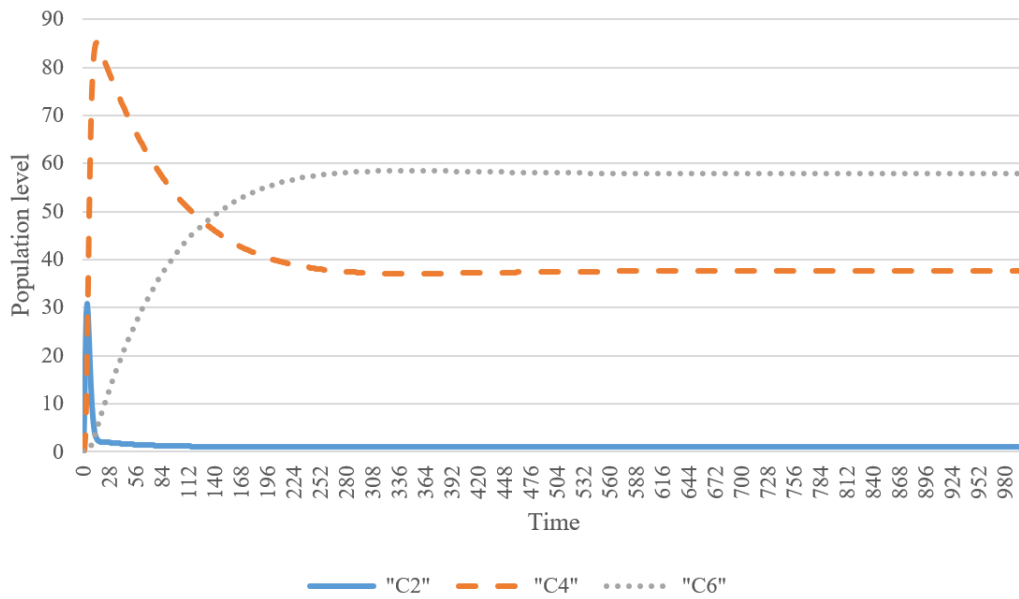


Fig. 4.27 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=100$ and $S=50$.

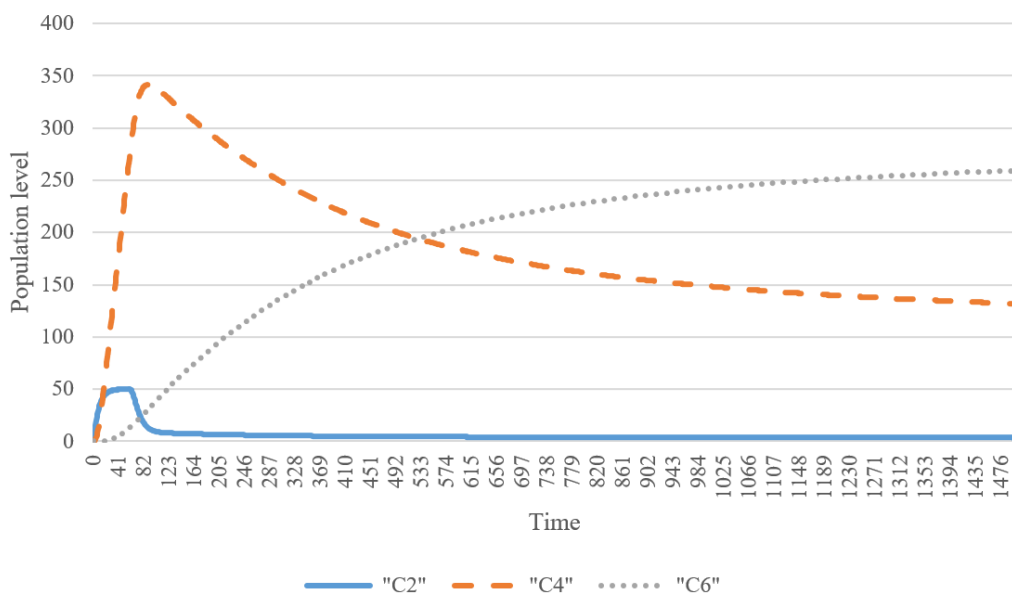


Fig. 4.28 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=400$ and $S=50$.

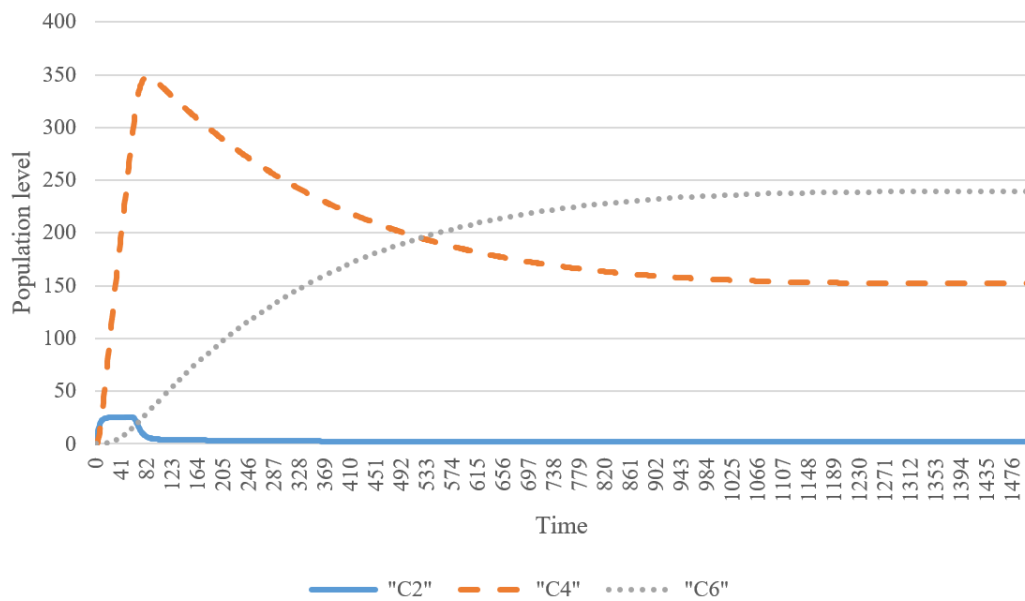


Fig. 4.29 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=50$, $N=400$ and $S=100$.

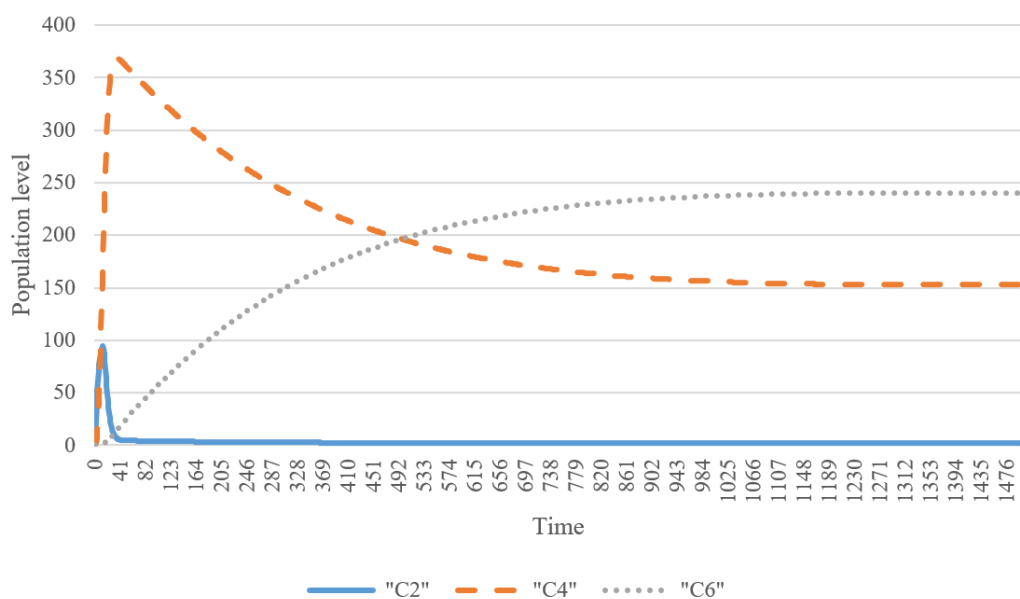


Fig. 4.30 The population level analysis for C_2 , C_4 and C_6 using ODE with $K=100$, $N=400$ and $S=100$.

Figures 4.24 to 4.30 illustrate how increasing the number of B involved in the protocol caused the average number of C_2 copies to decrease. This suggests that the more Bs are involved in the protocol, the better the performance of the protocol would be. Moreover, more

customers are in C_6 than C_4 for a service from M; this is because M needs to check the digital coins with B before sending the product decryption key. Increasing the number of Bs and having a reasonable number of TTP involved in the system would mitigate the performance cost if the system has a large number of customers to serve. Further, this would cause the system to settle faster and reach a steady-state during a shorter time period, as shown in Figures 4.24, 4.25, 4.28 and 4.29. In Figure 4.24, the system reached the steady-state at 4125.383 time units when the number of bank is 5 whereas in Figure 4.25 the system reached the steady-state at 1233.198 time units when the number of banks increased to 20. Moreover, when the number of customers in the system is larger, the system reached the steady-state at 7262.322 time units when the number of banks is 50 (Figure 4.29) whereas the system reached the steady-state at 4073.783 time units when the number of banks increased to 100 (Figure 4.30).

Furthermore, in Figure 4.30, the number of customers who are waiting to receive the product decryption key or abort (C_6 copies) form M is larger compared to the number of customers who are waiting to receive the product decryption key or abort (C_4 copies) when the customer's anonymity feature does not exist in the protocol, as shown in Figure 4.5. For example, at 900 time-units, when the protocol keeps the anonymity feature, the population level of C_6 is 234.37 (as illustrated in Figure 4.30), which is greater than the population level of C_4 ($C_4 = 128.98$, as illustrated in Figure 4.5) when the protocol does not have this feature.

The following graphs, Figures 4.31 to 4.34, illustrate the throughput of *sendCAbort*, *sendCDigitalCoins*, *sendCEP*, and *sendCPDk* actions. They show throughput values of the main actions to serve the customers in this protocol in relation to the different population size of K , N and S . They illustrate how increasing the number of customers (N) would significantly impact the actions throughput. The throughput values of action *sendCPDk* are the lowest values in all graphs compared to the other actions. Moreover, action *sendCDigitalCoins* has the largest throughput values in all graphs. We believe this is because components are moving sequentially from state to state and customer needs first to download product from TTP website and then action *sendCDigitalCoins* have to be performed for all customers to be able to perform other actions. Thus increasing the number of Bs involved in the protocol could improve the protocol performance. Also, having a reasonable number of TTP in the system would influence the throughput of all actions. Actions *sendCAbort* and *sendCEP* have started with relatively similar values and then *sendCEP* action's throughput value becomes larger.

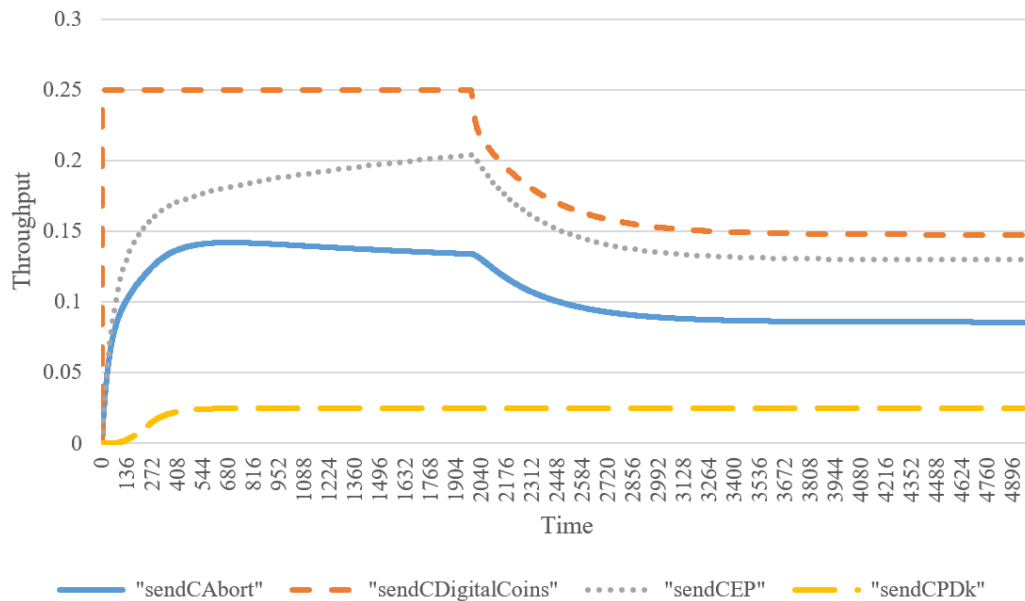


Fig. 4.31 The throughput analysis for *sendCAbort*, *sendCDigitalCoins*, *sendCEP* and *sendCPDk* using ODE with $K=20$, $N=100$ and $S=5$.

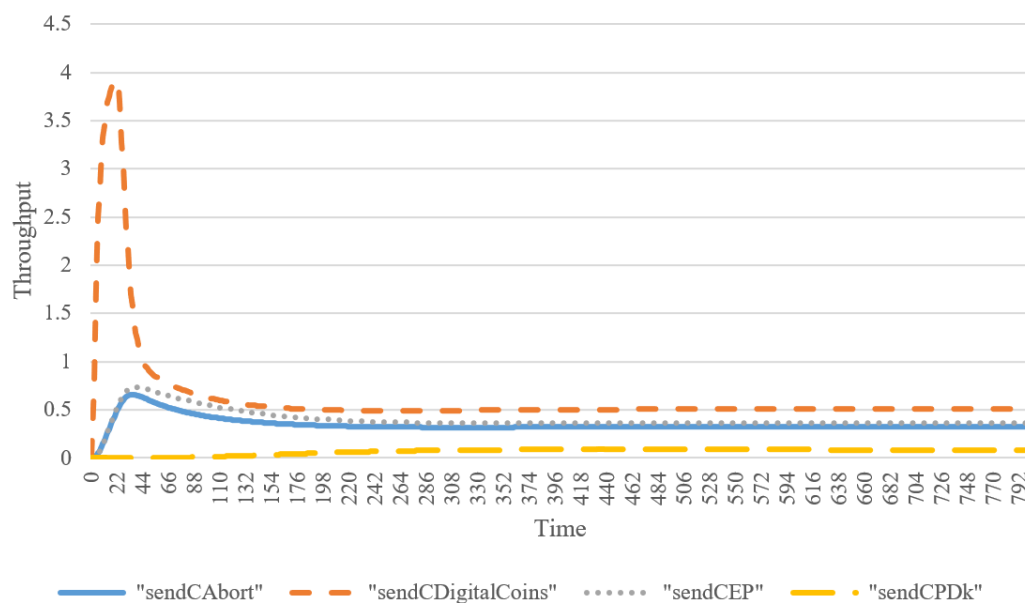


Fig. 4.32 The throughput analysis for *sendCAbort*, *sendCDigitalCoins*, *sendCEP* and *sendCPDk* using ODE with $K=20$, $N=100$ and $S=20$.

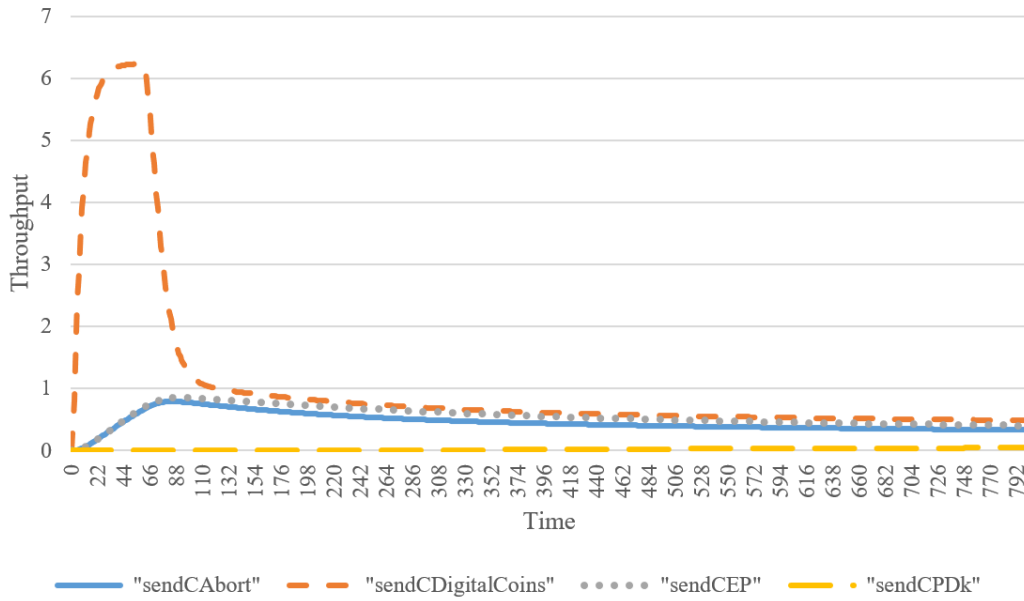


Fig. 4.33 The throughput analysis for *sendCAbort*, *sendCDigitalCoins*, *sendCEP* and *sendCPDk* using ODE with $K=50$, $N=400$ and $S=50$.

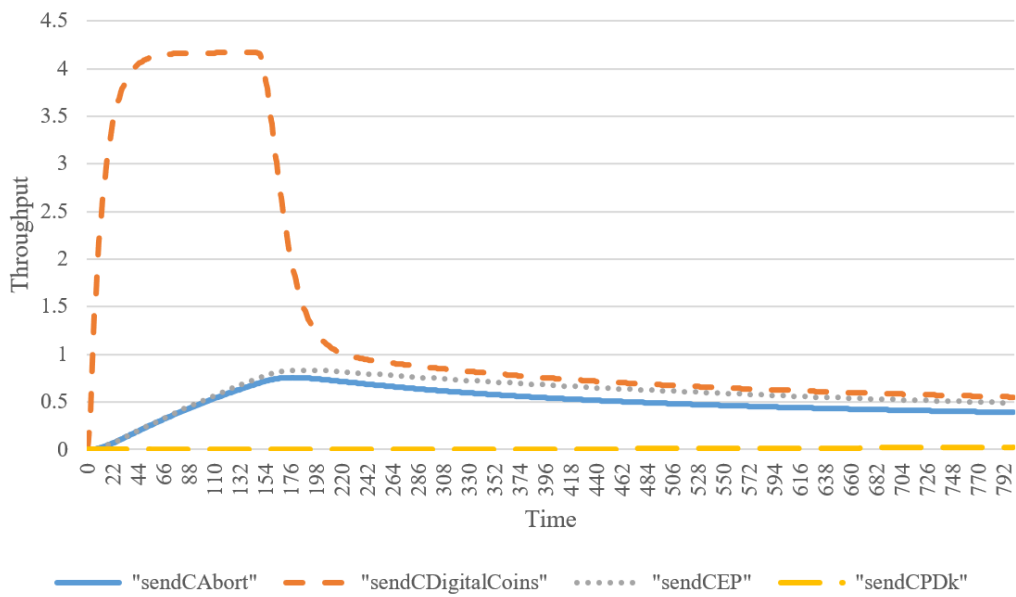


Fig. 4.34 The throughput analysis for *sendCAbort*, *sendCDigitalCoins*, *sendCEP* and *sendCPDk* using ODE with $K=50$, $N=600$ and $S=50$.

4.4.4 Performance evaluation of the extended optimistic anonymous protocol

The basic extended optimistic anonymous protocol

The average response times of M and TTP, we seek to calculate the average response times of M and TTP to serve C. We assigned 1 as a value for all rates. The main actions of M are calculated based on the number of customers in the system and the main actions of TTP are calculated based on the number of customers and TTP in the system, as mentioned in Subsection 4.3.4.

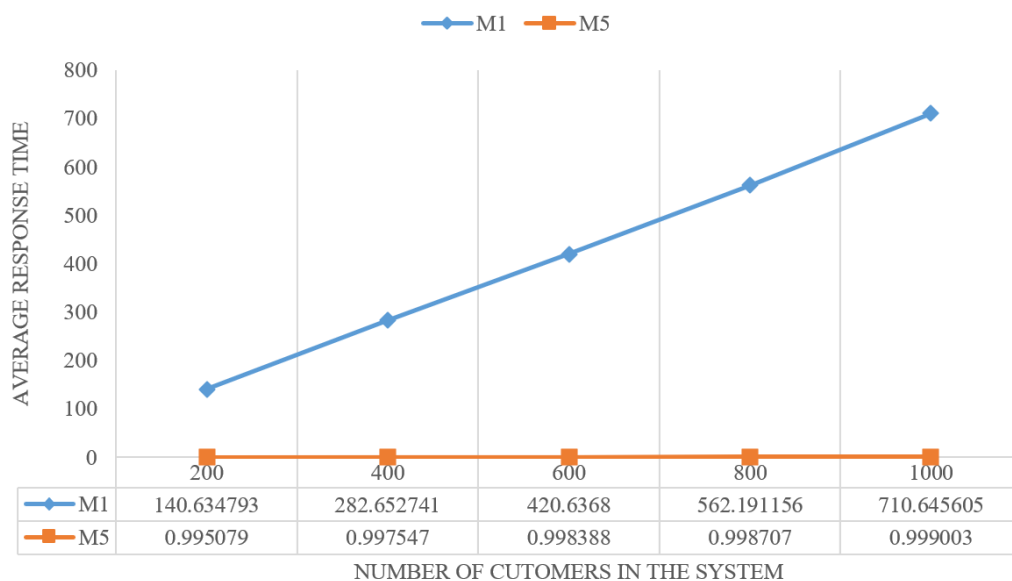


Fig. 4.35 The average response time of M_1 and M_5 using ODE.

Figure 4.35 shows the merchant’s average response time in case of the TTP involvement to solve the dispute between the customer and the merchant. The M_1 and M_5 are the behaviour of sending to a customer the encrypted product and the decryption key of the product, respectively. The average response time of M_1 is large and significantly increased when the number of customers is increased, whereas M_5 has low average response times numbers. We believe this result is because many customers are seeking help from TTP for resolving the dispute. Further, unlike the extended failure resilient fair-exchange protocol, this extended optimistic anonymous protocol preserves customer anonymity, introducing more performance overhead in a protocol when the dispute occurs. Starting from C requesting and receiving a digital coin from B, then M needs to contact B to check the validity of the digital coins before responding to the customer to provide a product decryption key and finally when a

dispute occurs, TTP also needs to contact B to check the validity of the digital coin and/or investigate on who spent the digital coin before starting to solve the dispute between C and M. Figures 4.36 and 4.37 show the average response time of some of TTP's state to serve the customer and solve the dispute by sending/forwarding the product decryption key or discover a misbehaving customer.

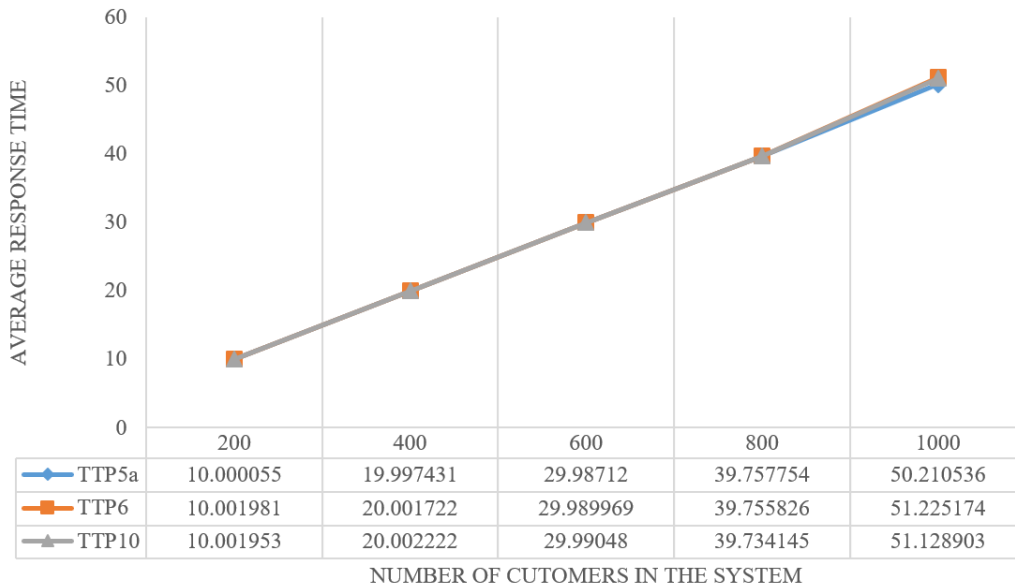


Fig. 4.36 The average response time of TTP_{5a} , TTP_6 and TTP_{10} using ODE, $K=20$, $S=20$.

In Figure 4.36, the number of TTP involved in the system is 20 and we change the number of customers from 200 to 1000 to show how increasing the number of the customer seeking help from TTP would impact the performance of the protocol. The average response time of TTP for all main states (TTP_{5a} , TTP_6 and TTP_{10}) to solve the dispute is the same. Having a large number of customers significantly increases the average response time of TTPs which creates more performance overhead. However, in Figure 4.37, the number of TTP increased to 60 which decreases the response time in relation to the number of customers in the system compared to Figure 4.36. Therefore, having a larger number of TTPs involved in the protocol when the dispute occurs between C and M mitigates the security protocol's performance overhead.

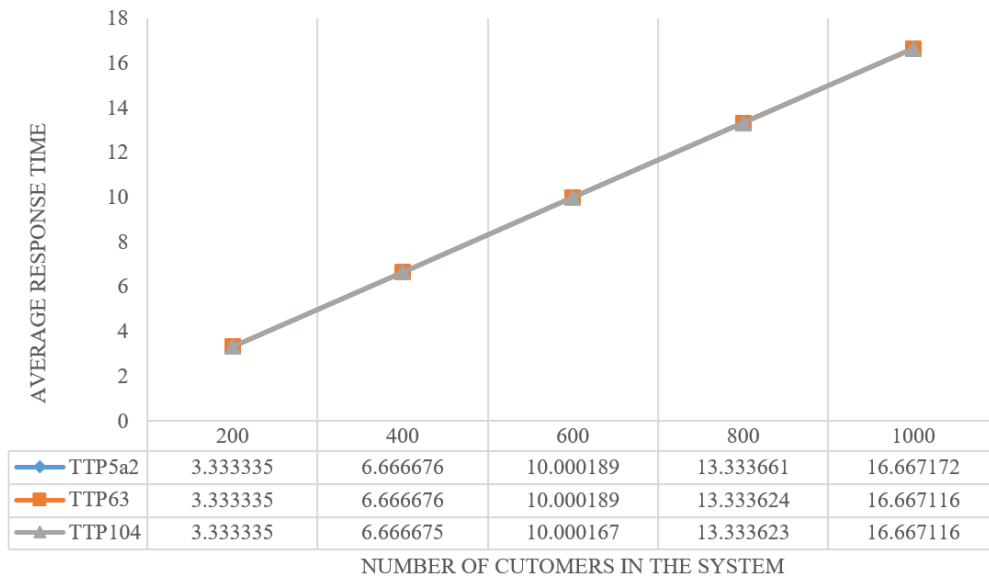


Fig. 4.37 The average response time of TTP_{5a} , TTP_6 and TTP_{10} using ODE, $K=60$, $S=20$.

The population level and throughput analysis, we seek to investigate the population level analysis for the average number of customers in C_4 and C_6 for having a service from M, C_2 for having a digital coin from B, and C_9 and C_{10} for interacting and having a service from TTP to get a dispute resolution.

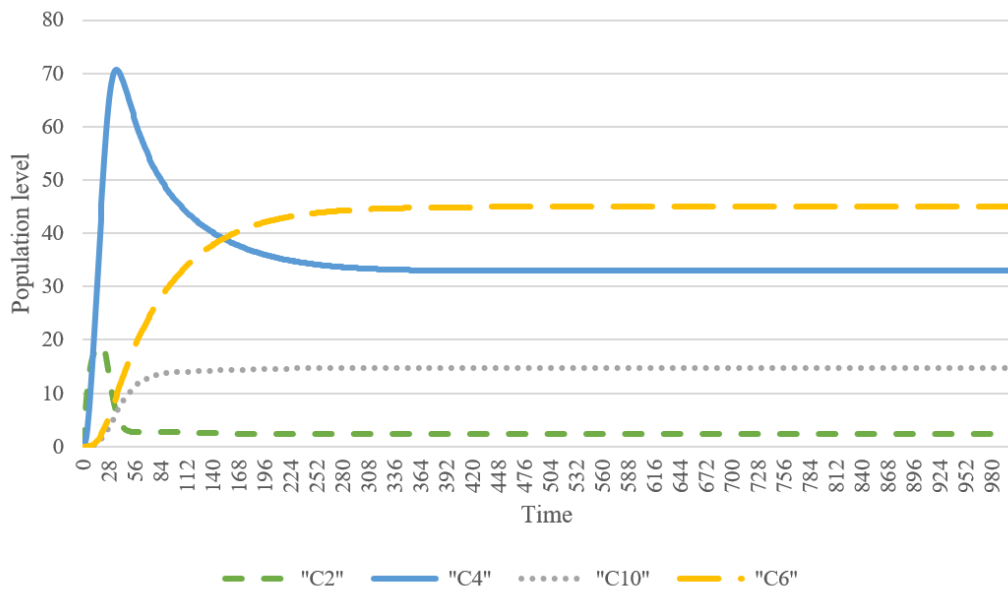


Fig. 4.38 The population level analysis using ODE with $K=20$, $N=100$ and $S=20$.

Figure 4.38 shows the population level of C_2 , C_4 , C_6 and C_{10} when the number of TTP (K), C and M copies (N) and bank (S) are 20, 100 and 20, respectively. The average number of customer in C_6 is significantly decreased compared to C_6 copies in Figure 4.25 when there is no dispute. This is because the customer's timeout expired before receiving the product encryption key. The rate of *cTimeoutExpired* is larger than the rate of the other actions in the same state. Moreover, some customer is waiting to receive a dispute resolution from TTP in C_{10} . However, there is a slight decrease in the average number of customer in C_4 and no change in the average number of customer in C_2 compared to C_4 and C_2 in Figure 4.25 when there is no dispute.

In the following experiments, we changed the rate values of the actions *sendCAbort*, *sendMAbort* and *cTimeoutExpired* to show how this would affect the system performance. The number of customers (C) interacting with the merchant (M) is 200.

First experiment in this experiment, the rate of aborting the customer request (for action *sendCAbort*) is decreased to be slower than the actions of the merchant to send the customer an encrypted product and to send the customer the product encryption key. The rate of *sendCAbort* action is calculated based on the number of customers interacting with the merchant. It is calculated as follows:

$$\begin{aligned} r_{sendCAbort1} &= 0.5, r_{sendCAbort} = 0.5/N, \\ r_{sendCAbort} &= 0.5/200 = 0.0025 \end{aligned}$$

Where N is the number of customer and merchant copies.

Moreover, we assumed that it is more likely that the merchant sends the customer the valid encrypted product, so the customer will not send him an abort message (*sendMAbort* action). So the rate of a customer to abort is decreased to 0.5.

Furthermore, we assumed that it is more likely that the customer will receive a response from the merchant before its timeout expired, so the rate of *cTimeoutExpired* action should be lower than the other optional actions in state C_6 . Therefore, we assigned 0.001 to the rate of *cTimeoutExpired* action.

The following figures show the population level analysis for C_4 and C_6 for having a service from M , and C_9 and C_{10} for interacting and having a service from TTP for solving the dispute.

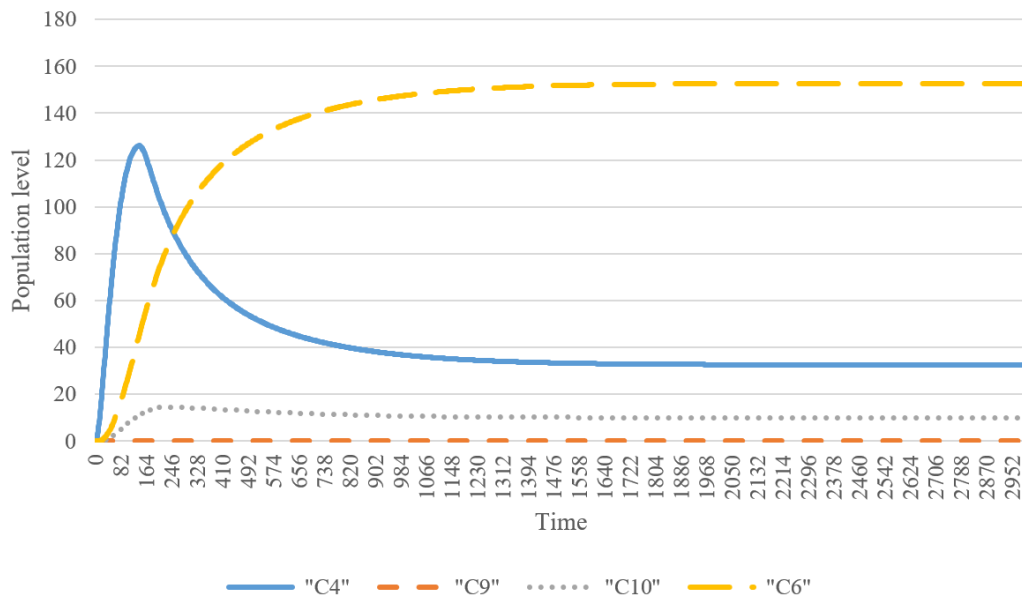


Fig. 4.39 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$.

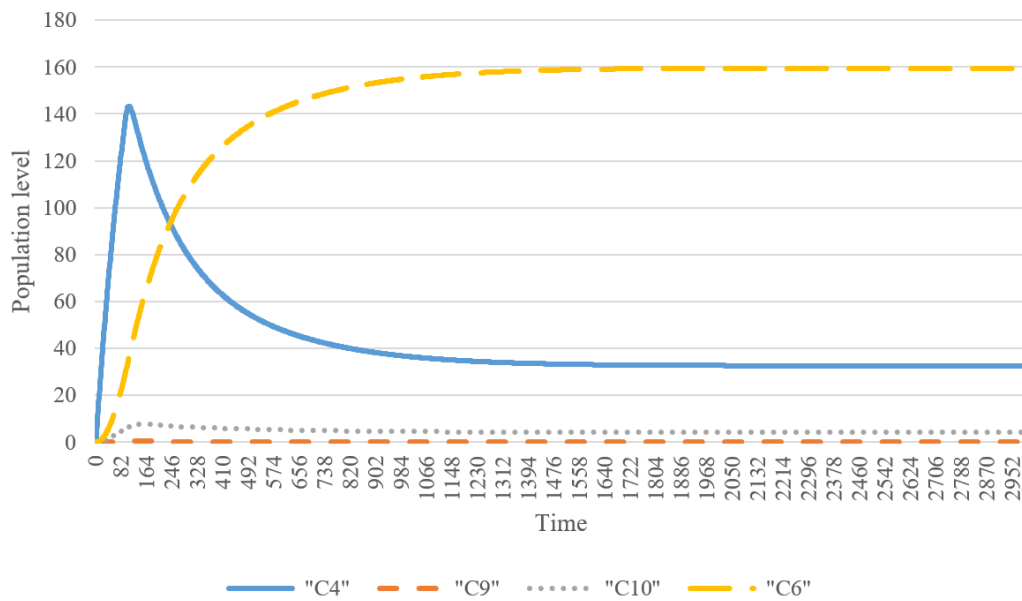


Fig. 4.40 The population level analysis using ODE with $K=60$, $N=200$ and $S=20$.

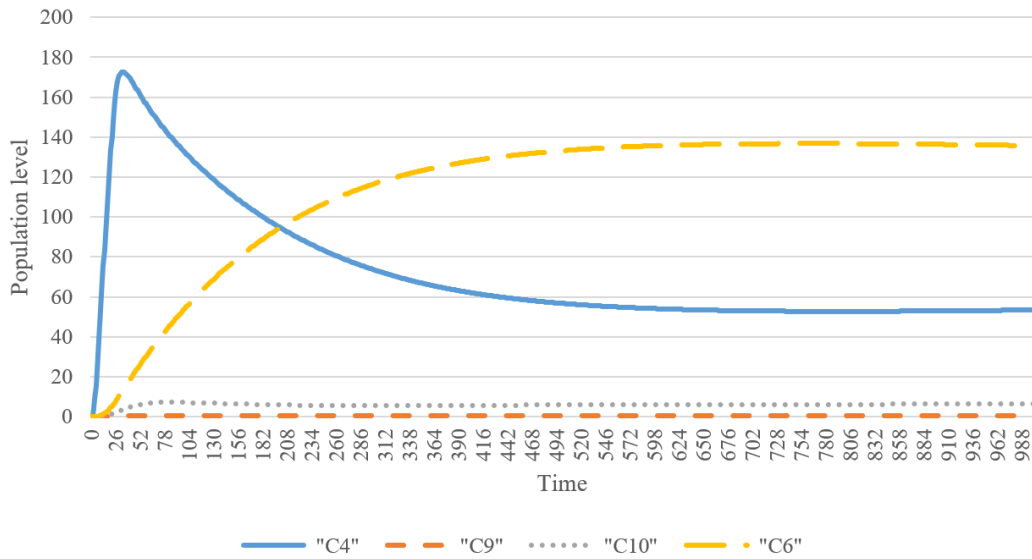


Fig. 4.41 The population level analysis using ODE with $K=60$, $N=200$ and $S=40$.

Figures 4.39 to 4.41 show how increasing the number of TTP would have a positive influence on decreasing the copies of C_{10} whereas increasing the number B would have a significant effect on decreasing the copies of C_6 as M needs to check the validity of coin before sending C the decryption key. However, no clear change happened to the copies of C_9 , which is when the customer starts sending the evidence of M misbehaving to TTP. Moreover, increasing the number of TTP and B involved in the system would make the system reach the stable state faster than having a lower number of these resources. The steady-states of the system are 1930.478, 1874.386 and 1708.432 time units in Figures 4.39, 4.40 and 4.41, respectively.

The following figures are the throughput analysis of some main actions related to the customer (C) when gets a service and interacts with M , B and TTP. *cTimeoutExpired*, *sendCAbort*, *sendCEP*, *sendCPDk* are the actions when the customer interacts with M . *sendCDigitalCoins* is the action when the customer gets digital coins from B . *discoverMisbehavingC*, *forwardKtoC*, *seekingHelpFromTTP*, *sendCkByTTP*, *sendTTPinfo* are actions when the customer gets services from TTP to solve the disputes between C and M .

Figures 4.42 to 4.44 show a clear improvement in the actions throughput of a service that customers get from TTP, M and B and the faster settlement of the system when the number of TTP and B increased. Moreover, the throughput of *sendCEP* action is larger than *sendCAbort* in Figures 4.42(a), 4.43(a) and 4.44(a). This reflects our purpose in this experiment that M is more likely to send the customers the encrypted product than an abort.

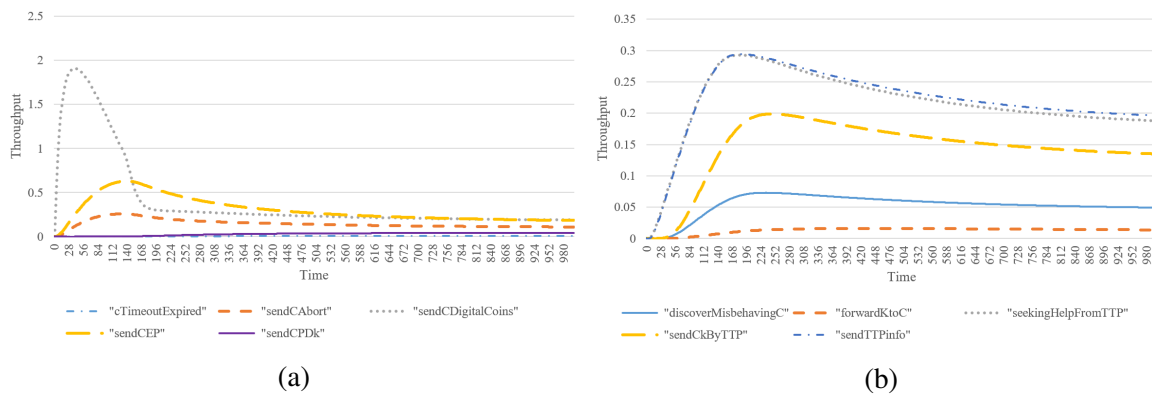


Fig. 4.42 The throughput analysis of the actions using ODE with $K=20$, $N=200$ and $S=20$.

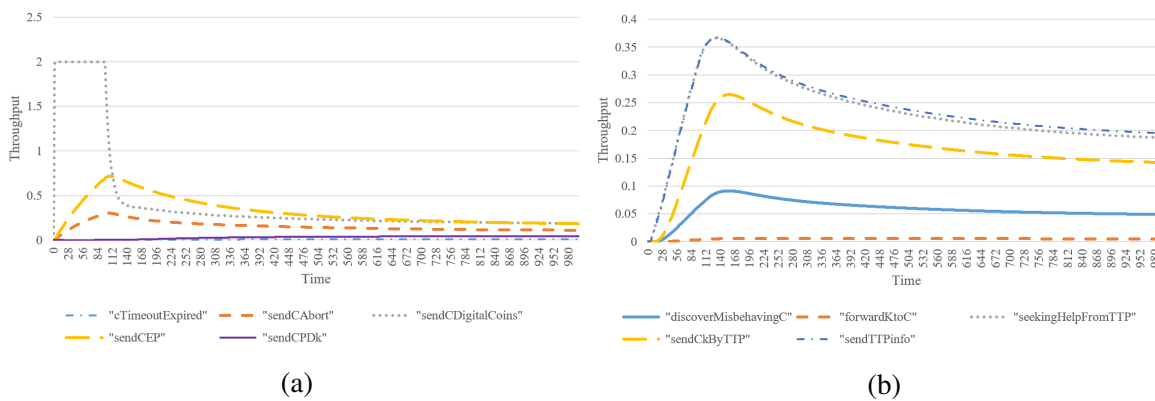


Fig. 4.43 The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=20$.

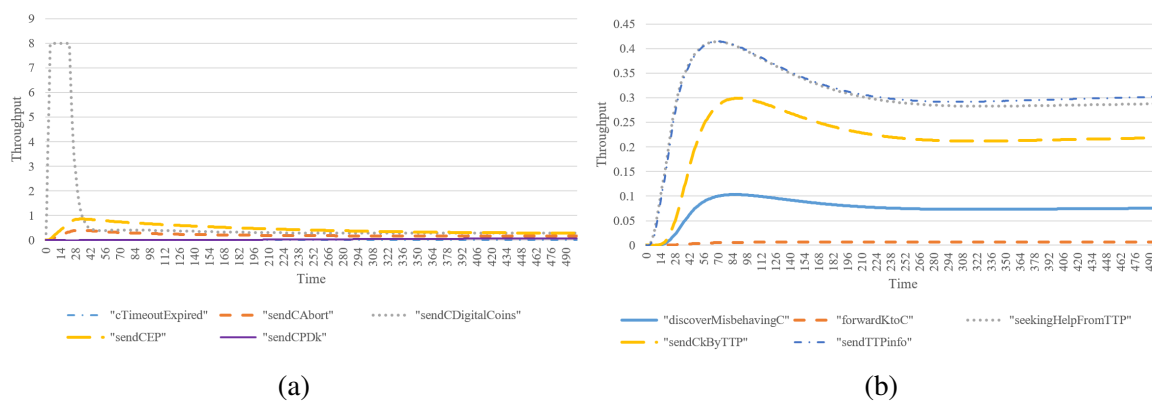


Fig. 4.44 The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=40$.

Second experiment now the rate values of the actions *sendCAbort*, *sendMAbort* and *cTimeoutExpired* are increased and be larger than the other action rates in the same states.

Therefore, the rate of aborting the customer request (for action *sendCAbort*) is increased to be faster than the actions of the merchant to send the customer an encrypted product and to send the customer the product encryption key. The rate of *sendCAbort* action is calculated based on the number of customers interacting with the merchant. It is calculated as follows:

$$\begin{aligned} r_{sendCAbort1} &= 2, r_{sendCAbort} = 2/N, \\ r_{sendCAbort} &= 2/200 = 0.01 \end{aligned}$$

Where N is the number of customer and merchant copies.

Moreover, we assumed that it is more likely that the merchant does not send the customer the valid encrypted product, so the customer will send him an abort message (*sendMAbort* action). So the rate of a customer to abort is increased to 2.

Furthermore, we assumed that it is more likely that the customer's timeout expired before receiving a response from the merchant, so the rate of *cTimeoutExpired* action should be larger than the other optional actions in state C_6 . Therefore, we assigned 0.009 to the rate of *cTimeoutExpired* action.

The following figures show the population level analysis for C_4 and C_6 for having a service from M, and C_9 and C_{10} for interacting and having a service from TTP for solving the dispute. Figure 4.45 shows the fluctuation that the system experienced on the number of copies of all the states (C_4 , C_6 , C_9 and C_{10}) when the rates of *sendCAbort*, *sendMAbort* and *cTimeoutExpired* are increased. However, these fluctuations are disappeared when the number of TTP and B involving in the system increased, Figure 4.46 and 4.47. Moreover, the C_{10} copies increase in all figures compared to the C_{10} copies in Figures 4.39, 4.40 and 4.41 when the rates of these actions are lower. This indicates that many disputes occurred. Further, C_6 copies are less in Figures 4.45, 4.46 and 4.47 compared to C_6 copies in 4.39, 4.40 and 4.41, respectively. This means larger number of customers contacted TTP to solve the disputes.

Also when we increased the number of both B and TTP, the system reached the steady-state at 904.277 time units, Figure 4.47. However, the system reached the steady-state in Figure 4.45 at 1922.812 time units and in Figure 4.46 at 2976.602 time units. The TTP also needs to contact B to solve the disputes between C and M. Therefore, increasing the number of both B and TTP will make the system reaches the steady-state faster.

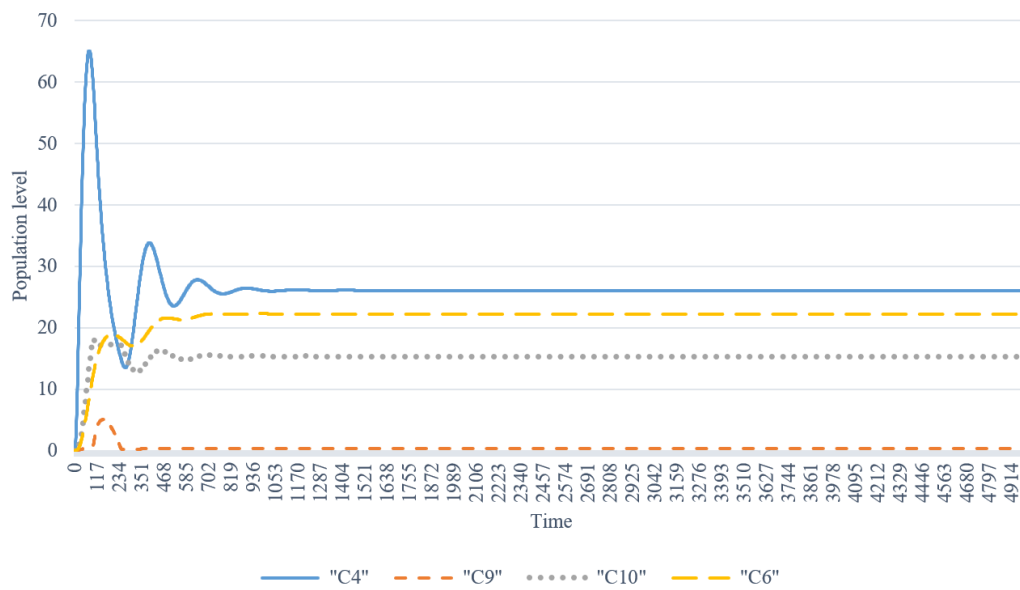


Fig. 4.45 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$.

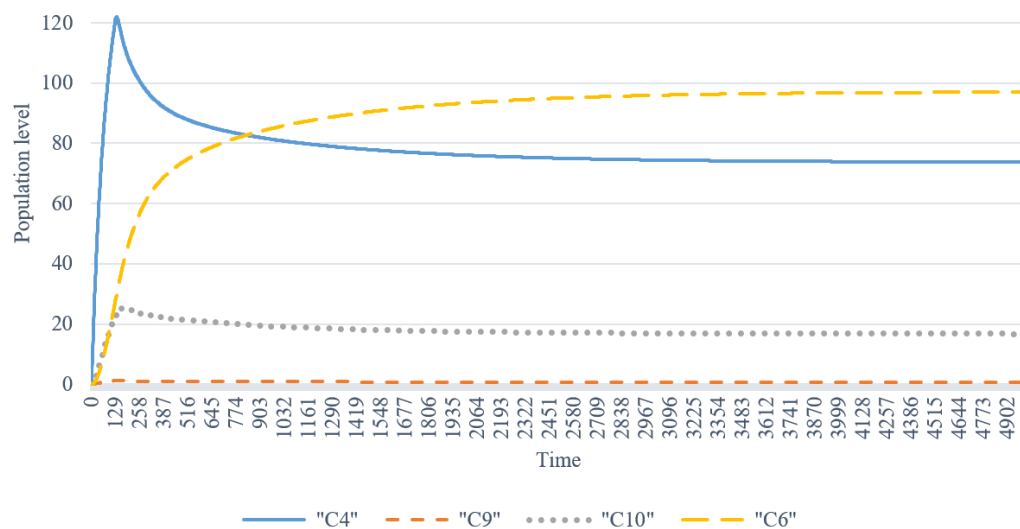


Fig. 4.46 The population level analysis using ODE with $K=60$, $N=200$ and $S=20$.

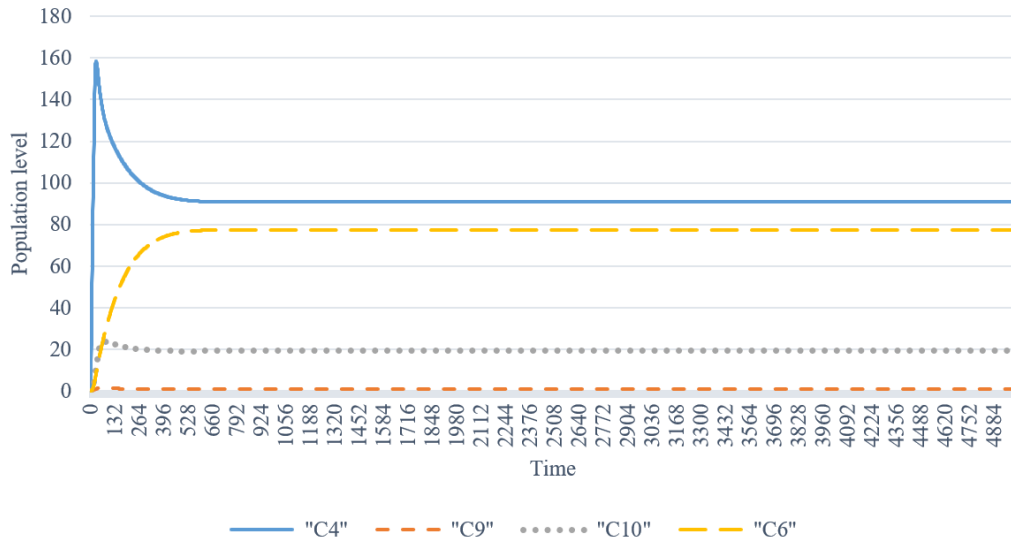


Fig. 4.47 The population level analysis using ODE with $K=60$, $N=200$ and $S=40$.

The following figures are the throughput analysis of some main actions related to the customer (C) when gets a service and interacts with M, B and TTP. *cTimeoutExpired*, *sendCAbort*, *sendCEP*, *sendCPDk* are the actions when the customer interacts with M. *sendCDigitalCoins* is the action when the customer gets digital coins from B. *discoverMisbehavingC*, *forwardKtoC*, *seekingHelpFromTTP*, *sendCkByTTP*, *sendTTPinfo* are actions when the customer gets services from TTP to solve the disputes between C and M.

Figures 4.48 to 4.50 show a clear improvement in the actions throughput of a service that customers get from TTP, M and B and the faster settlement of the system when the number of TTP and B are increased. Figure 4.48 show a clear fluctuation that the system experienced on the throughput of the actions. Moreover, the throughput of *sendCAbort* action is larger than *sendCEP* in Figures 4.48(a), 4.49(a) and 4.50(a). Furthermore, Figures 4.48(b), 4.49(b) and 4.50(b) shows how increasing the number of both TTP and B have a clear increase on the throughput of the TTP actions when response to Cs requests to solve the disputes.

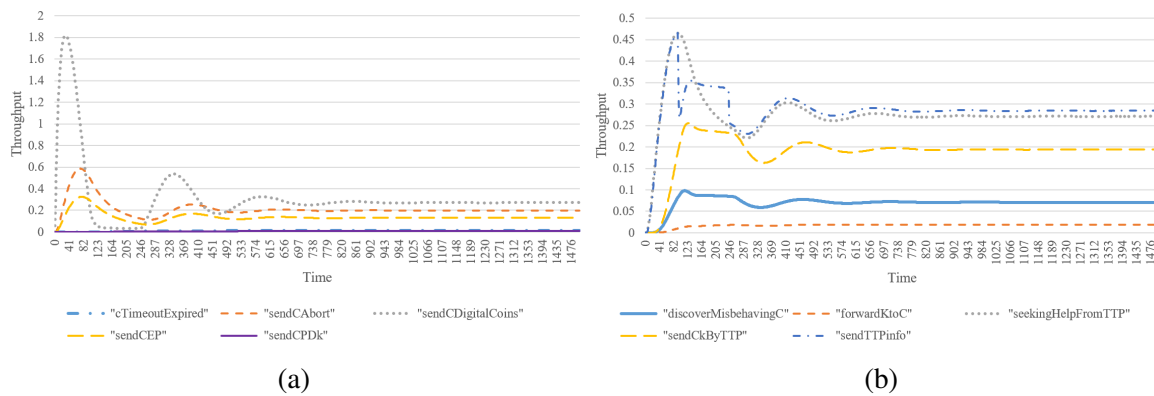


Fig. 4.48 The throughput analysis of the actions using ODE with $K=20$, $N=200$ and $S=20$.

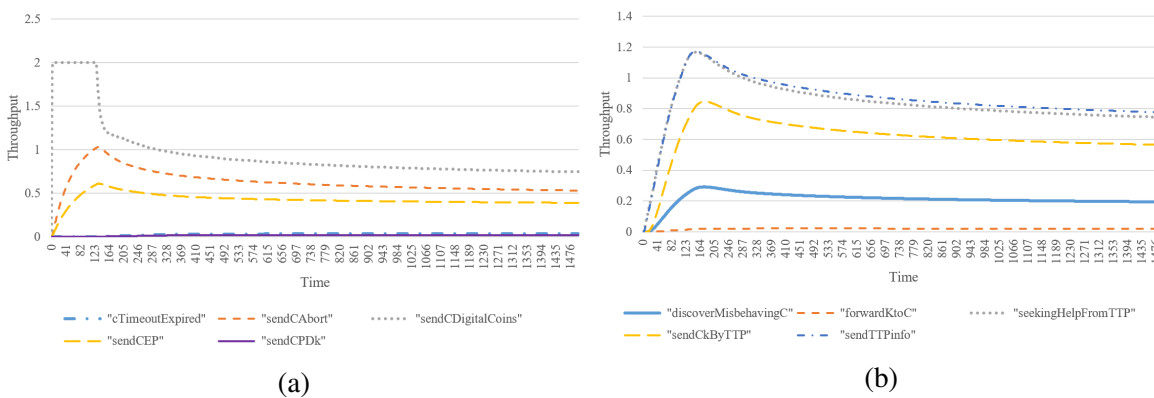


Fig. 4.49 The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=20$.

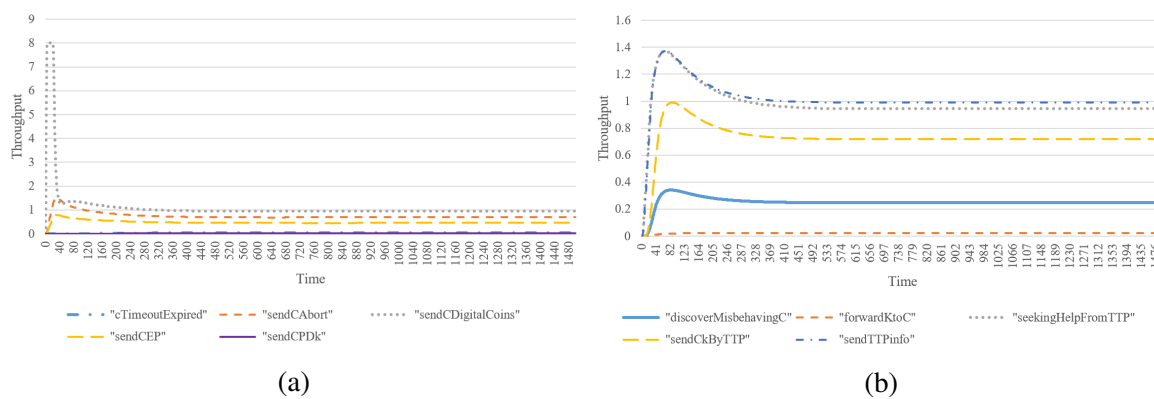


Fig. 4.50 The throughput analysis of the actions using ODE with $K=60$, $N=200$ and $S=40$.

The extended optimistic anonymous protocol with probabilities distribution of misbehaviour

We are interested in investigating the population level analysis of C_4 , C_6 , C_9 and C_{10} (C_4 and C_6 for having a service from M and C_9 and C_{10} for interacting and having a service from TTP) and throughput analysis of some main actions that provide service to C. Both analyses are studied in relation to different probabilities of M to be honest and the population number of Cs and M's copies. We assigned 1 as a value for all rates. The main actions of M are calculated based on the number of customers in the system and the main actions of TTP are calculated based on the number of customers and TTP in the system, as mentioned in Subsection 4.3.4.

In Figures 4.51 and 4.52, the probabilities for M to be honest are changed from $p = 0.1$ to $p = 0.9$, the number of TTP is 20, the number of C and M's copies is 200 and the number of banks is 20. Figure 4.51 shows how increasing the probabilities of M to be honest has a clear effect on decreasing the number of C_{10} copies. So less customers seek dispute resolution. However, the number of C in C_4 and C_6 does not experience any change but there are more Cs in C_6 waiting to get a decryption key for the encrypted product than in C_4 to get an encrypted product. Figure 4.52 illustrates the significant decrease in the throughput of the TTP's actions that provide the dispute resolution when the probabilities of M to be honest is increasing. We believe this is because fewer customers are seeking help from TTP.

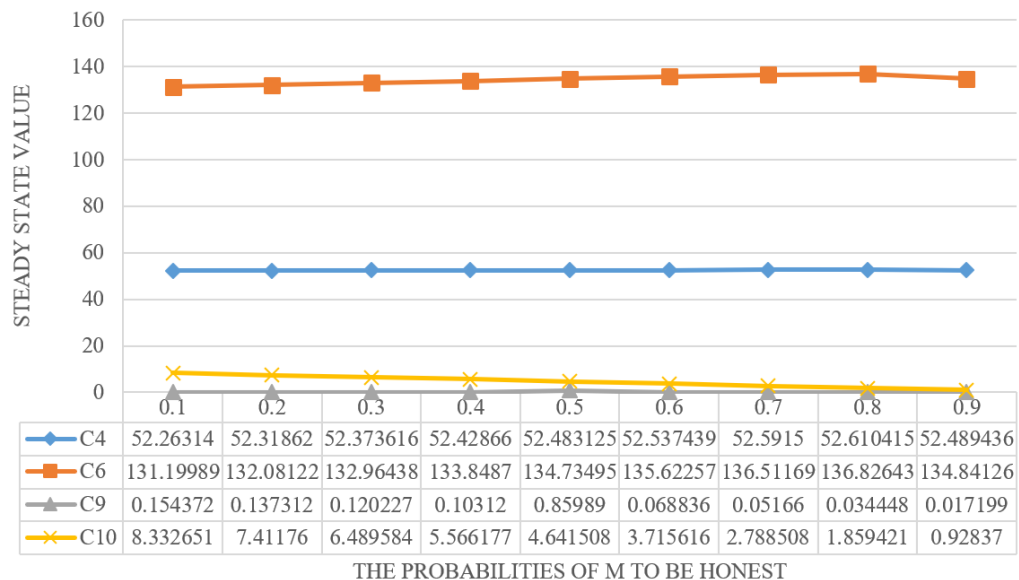


Fig. 4.51 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ in relation to different probabilities of M to be honest.

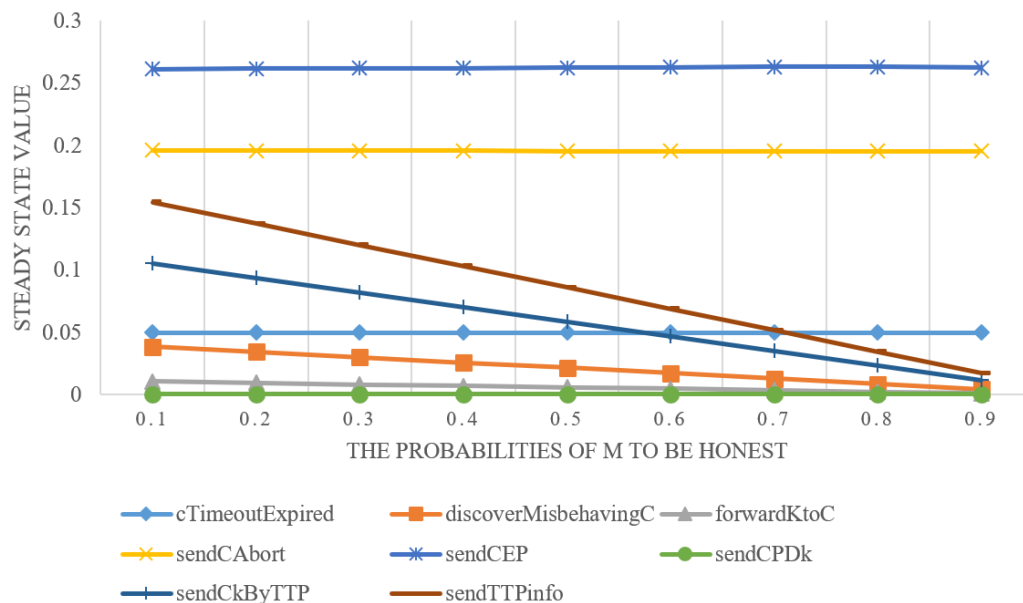


Fig. 4.52 The throughput analysis of actions using ODE with $K=20$, $N=200$ and $S=20$ in relation to different probabilities of M to be honest.

In Figures 4.53 and 4.54, the number of C and M’s copies (N) is increased to 600. Increasing the number of Cs in a system has a clear impact on the population level and the actions’ throughputs. Just like Figure 4.51, Figure 4.53 shows how increasing the probabilities of M to be honest has a clear effect on decreasing the average number of C_{10} as fewer customers seek a dispute resolution. Also, the number of Cs in C_4 and C_6 does not experience any change, but more Cs in C_6 waiting to get a decryption key for the encrypted product than in C_4 to get an encrypted product. However, increasing the number of Cs in the system does not significantly impact C_9 and C_{10} . The impact is clearly on C_4 and C_6 , compared to C_4 and C_6 in Figure 4.51.

Moreover, Figure 4.54 shows the decrease in the throughput of the TTP’s actions that provide the dispute resolution when the probabilities of M to be honest is increasing. We believe this is because fewer customers are seeking help from TTP. Moreover, all actions have a clear reduction on their throughput when we increased the number of C. The throughput of the TTP’s actions are less than the throughput of the TTP’s actions in Figure 4.52 when the the number of Cs in a system is 200. Therefore, more customers in the system will have an affect on the TTP’s responses.

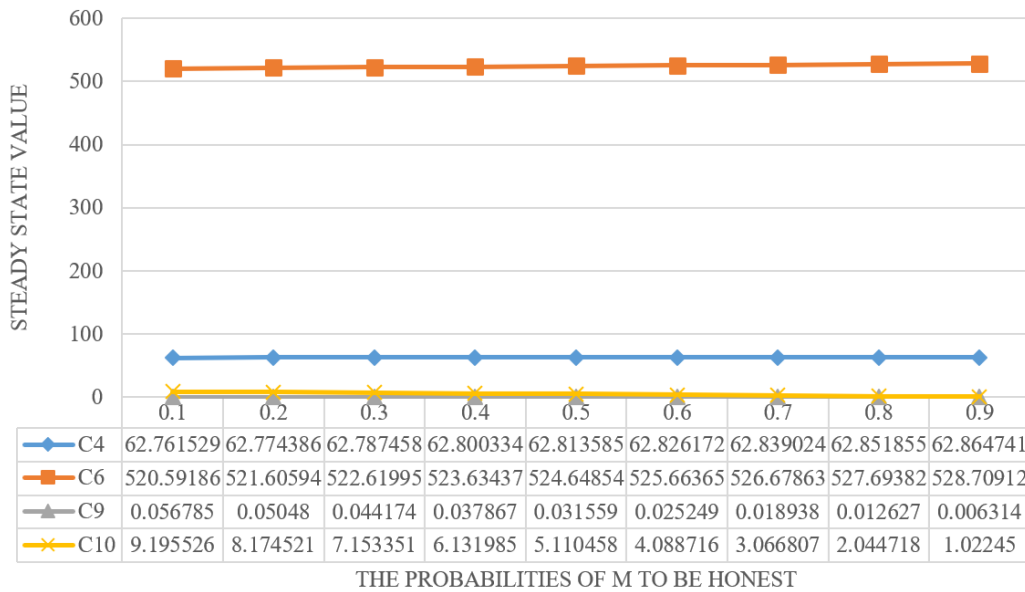


Fig. 4.53 The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ in relation to different probabilities of M to be honest.

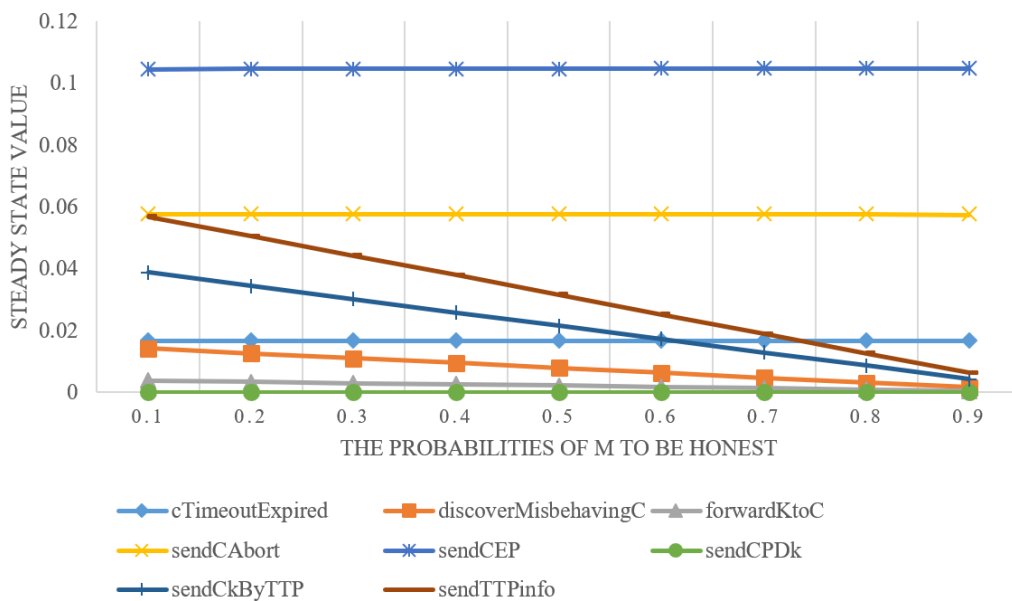


Fig. 4.54 The throughput analysis of actions using ODE with $K=20$, $N=600$ and $S=20$ in relation to different probabilities of M to be honest.

Figure 4.55 shows how faster the system settled in relation to the population number of Cs and M's copies and the probabilities of M to be honest. The larger the population number

is and the less probability of M to be honest is, the longer time will be taken for the system to settle.

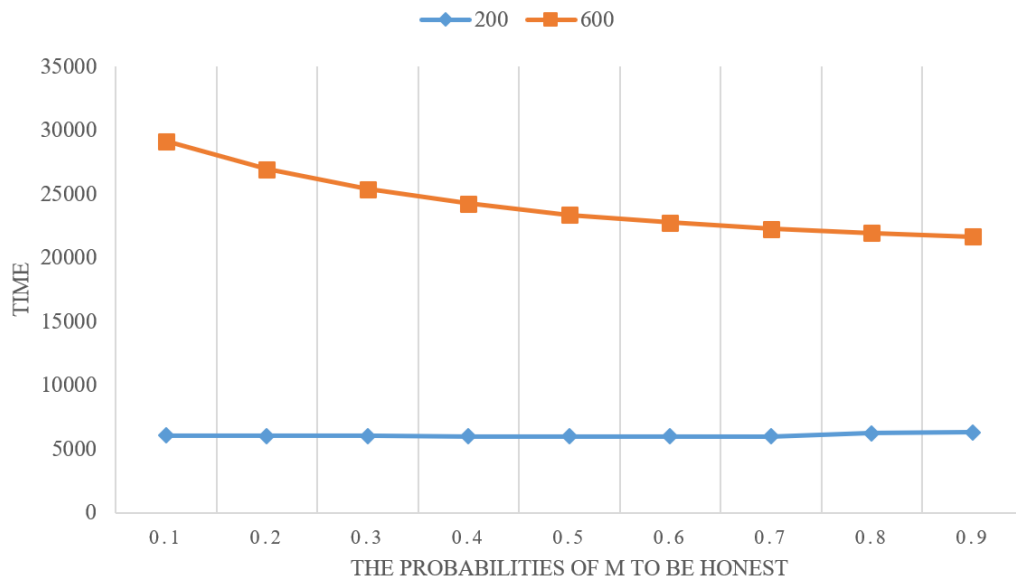


Fig. 4.55 The steady-state detection time in relation to the population number ($N=200$ and $N=600$).

Now we interested in changing the rates of the shared actions between M and B. The shared actions between M and B are $rsendBCoinByM$, $rsendMyes$ and $rsendMno$. We increased and decreased the rates to show how these would have an impact on the system performance. First, the shared actions rates between M and B decreased to $r=0.2$ and $r=0.5$. Then they increased to $r=2$ and $r=4$. The rates are calculated depend on the number of N and S involved in the system, as follows:

$$\begin{aligned}
 rsendBCoinByM &= r/N \\
 rsendMyes &= (r/N) * S \\
 rsendMno &= (r/N) * S
 \end{aligned}$$

Where N is the number of Cs and M's copies and S is the number of banks.

The following figures show the population level analysis of C_4 and C_6 for having a service from M and C_9 and C_{10} for interacting and having a service from TTP. The probability for M to be honest is changed from $p = 0.1$ to $p = 0.9$. Figures 4.56 and 4.57 illustrate that when the shared actions rates between M and B to check the C's digital coin's validity are slow, C experiences a big delay in receiving a product decryption key. There are large waiting Cs in C_6 . In Figures 4.56 and 4.57, you can notice that the population levels of C_9 and C_{10} are slightly decreased. We believe this is because more Cs waiting in C_4 and C_6 to be served

before moving to C_9 and C_{10} . So the faster the rate, the less delay would be, as shown in Figure 4.58 and 4.59. Moreover, Figures 4.58 and 4.59 show a significant increase in the population levels of C_{10} .

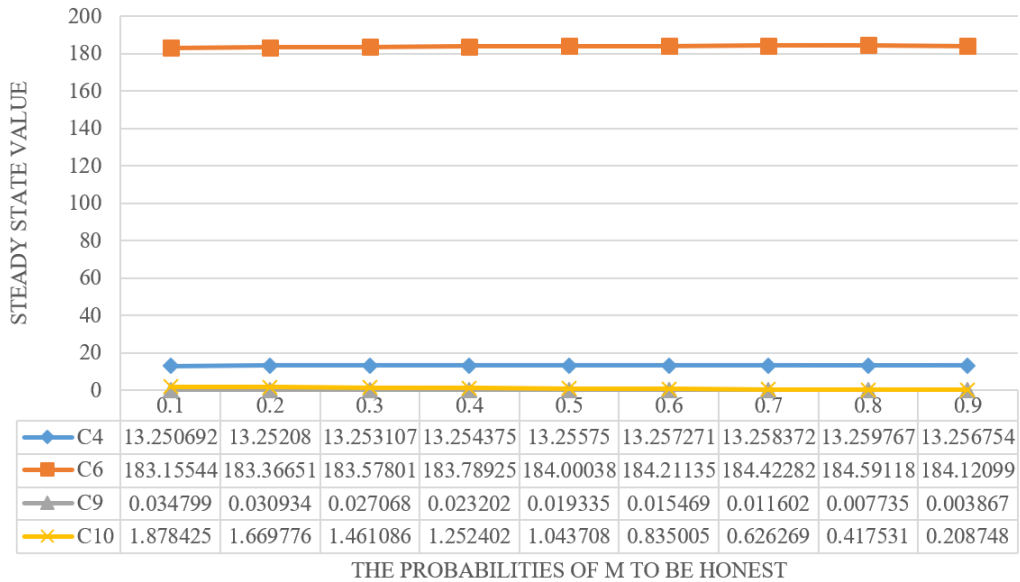


Fig. 4.56 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=0.2$.

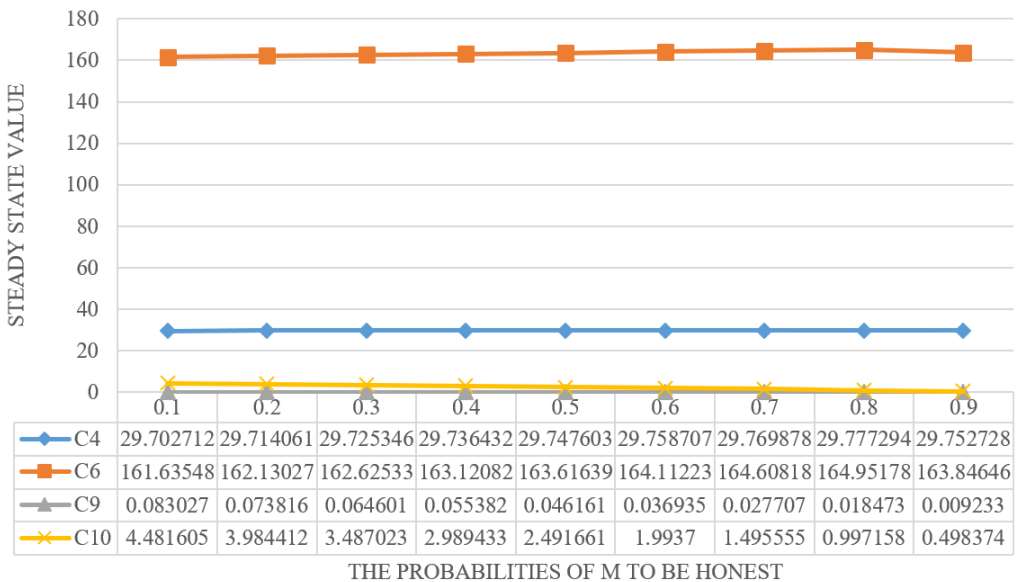


Fig. 4.57 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=0.5$.

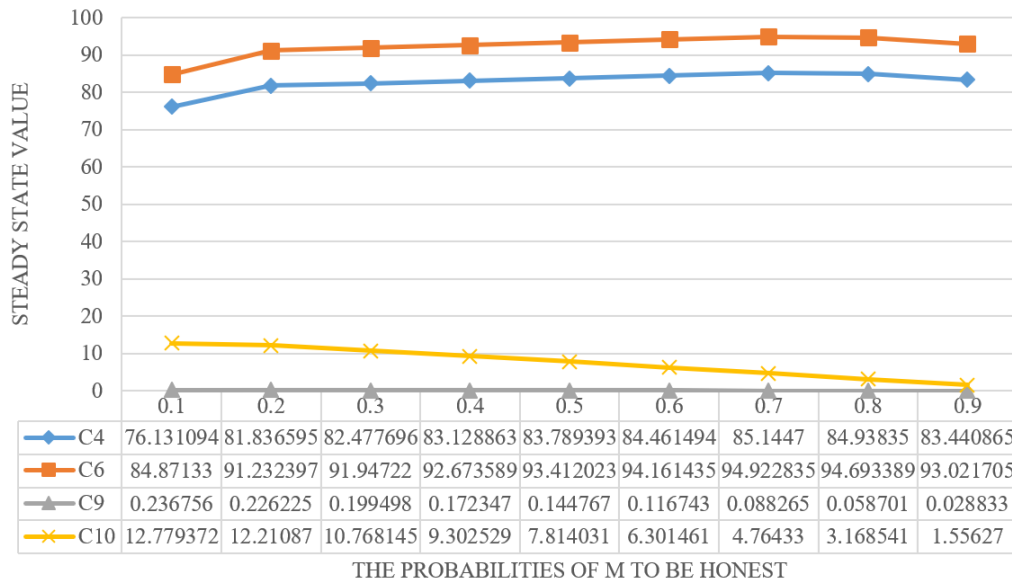


Fig. 4.58 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r=2$.

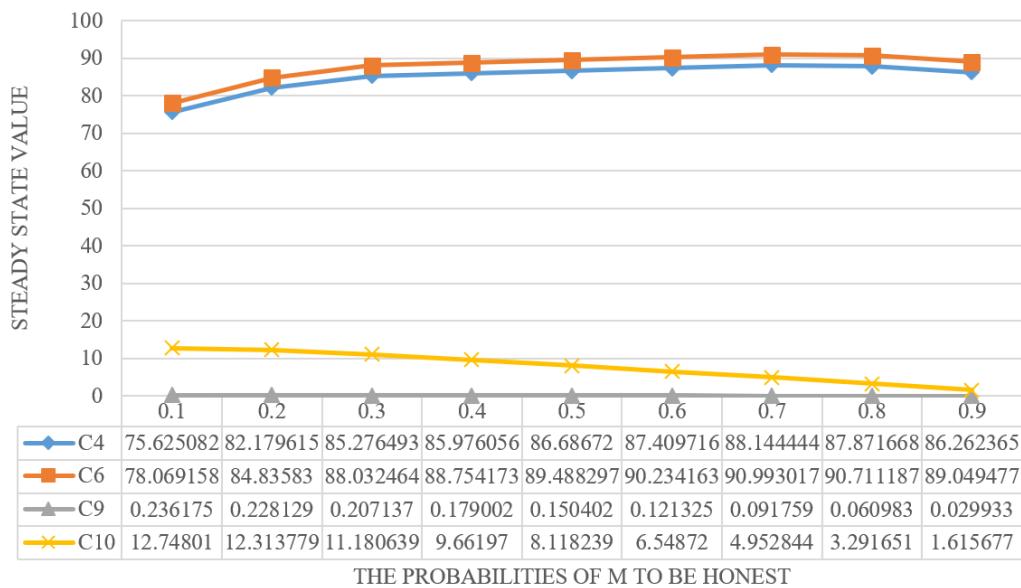


Fig. 4.59 The population level analysis using ODE with $K=20$, $N=200$ and $S=20$ when $r = 4$.

In Figures 4.60, 4.61, 4.62 and 4.63, we change the number of Cs and M's copies to 600. This creates even more delay that Cs need to wait to receive a response from M during state C_6 , compared to when $N=200$ in Figure 4.56, 4.57, 4.58 and 4.59. Moreover, increasing these actions' rates mitigates the negative influence to some points till it loses its control when the probability of M to be honest increases, as illustrated in Figures 4.62 and 4.63.

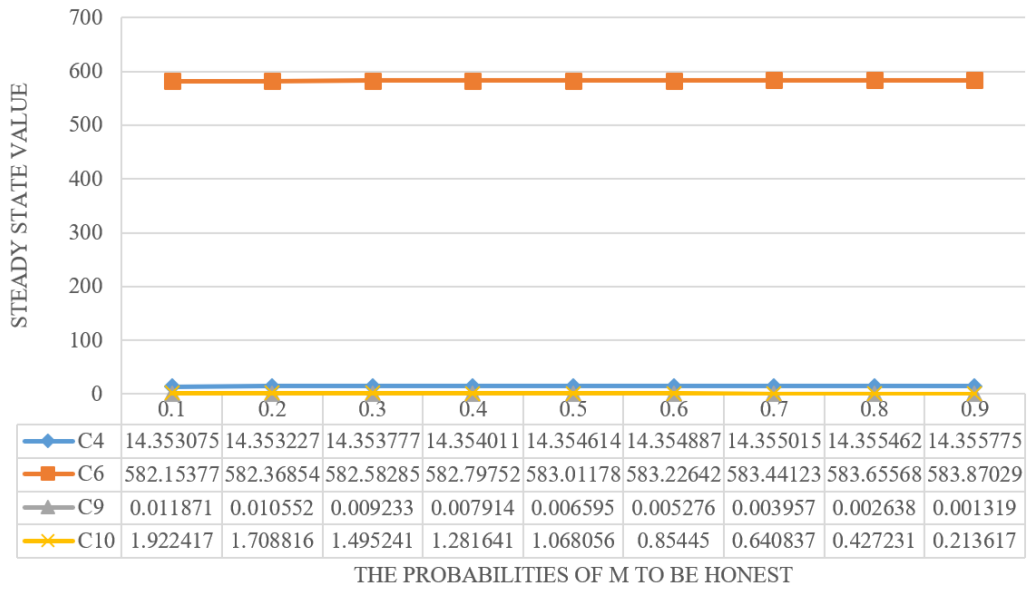


Fig. 4.60 The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=0.2$.

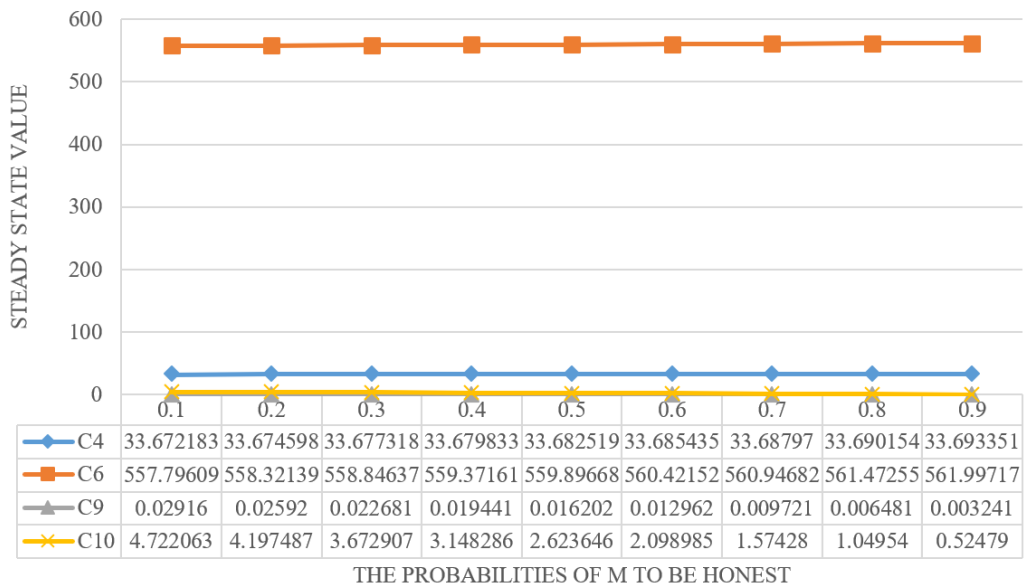


Fig. 4.61 The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=0.5$.

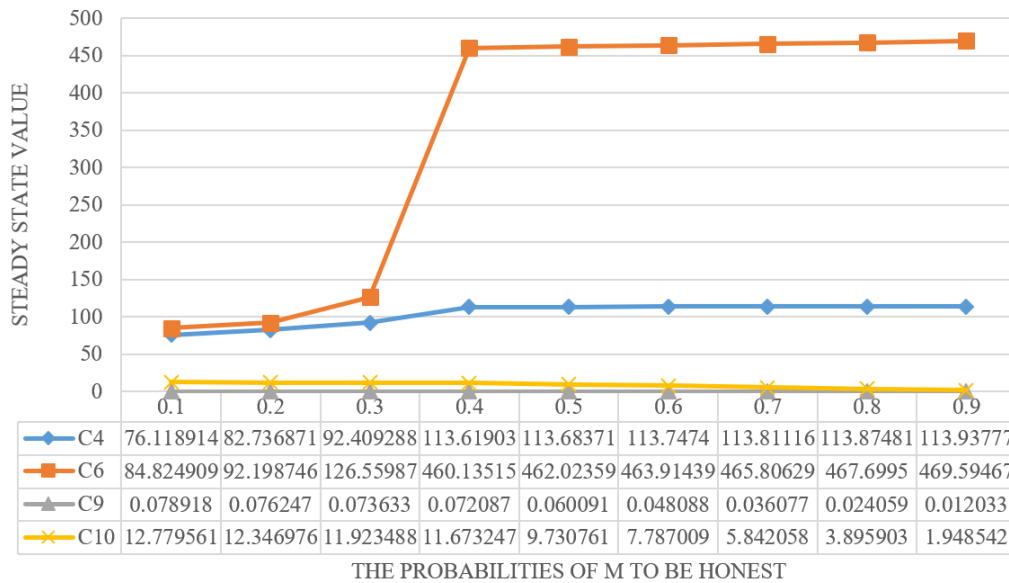


Fig. 4.62 The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=2$.

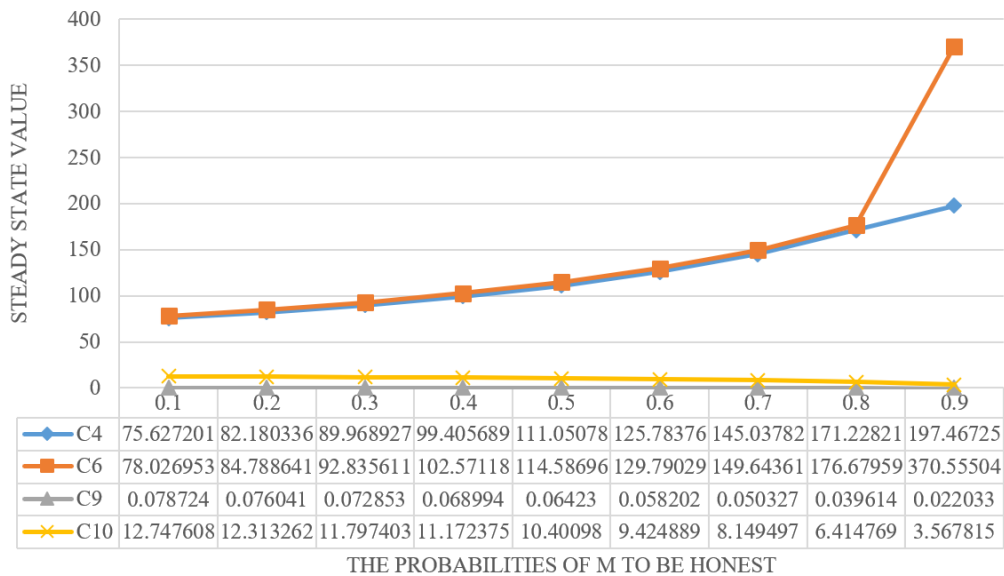


Fig. 4.63 The population level analysis using ODE with $K=20$, $N=600$ and $S=20$ when $r=4$.

Figure 4.64 shows the throughput of some main actions to serve a customer. These actions are *discoverMisbehavingC*, *forwardKtoC*, *sendCkByTTP* and *sendTTPinfo* to interact and seek a help form TTP and *sendCAabort*, *sendCEP* and *sendCPDk* to get a service form M and *cTimeoutExpired*. In Figure 4.64, the probabilities for M to be honest are changed from $p = 0.1$ to $p = 0.9$, the number of TTP is 20, the number of C and M's copies

is 200 and the number of banks is 20. The shared actions rates between M and B decreased to $r = 0.2$ and $r = 0.5$ and then increased to $r = 2$ and $r = 4$. The rates are calculated based on the number of N and S involved in the system, as mentioned in this subsection.

Figure 4.64 illustrates a significant decrease in the throughput of the TTP's actions that provide the dispute resolution when M's probabilities to be honest are increasing in all Sub figures. The TTP's actions are *discoverMisbehavingC*, *forwardKtoC*, *sendCkByTTP* and *sendTTPinfo*. We believe this is because fewer customers are seeking help from TTP when M's probabilities to be honest are higher. Moreover, there is a considerable improvement in all actions' throughputs when the shared actions rates between M and B to check the C's digital coin's validity are higher.

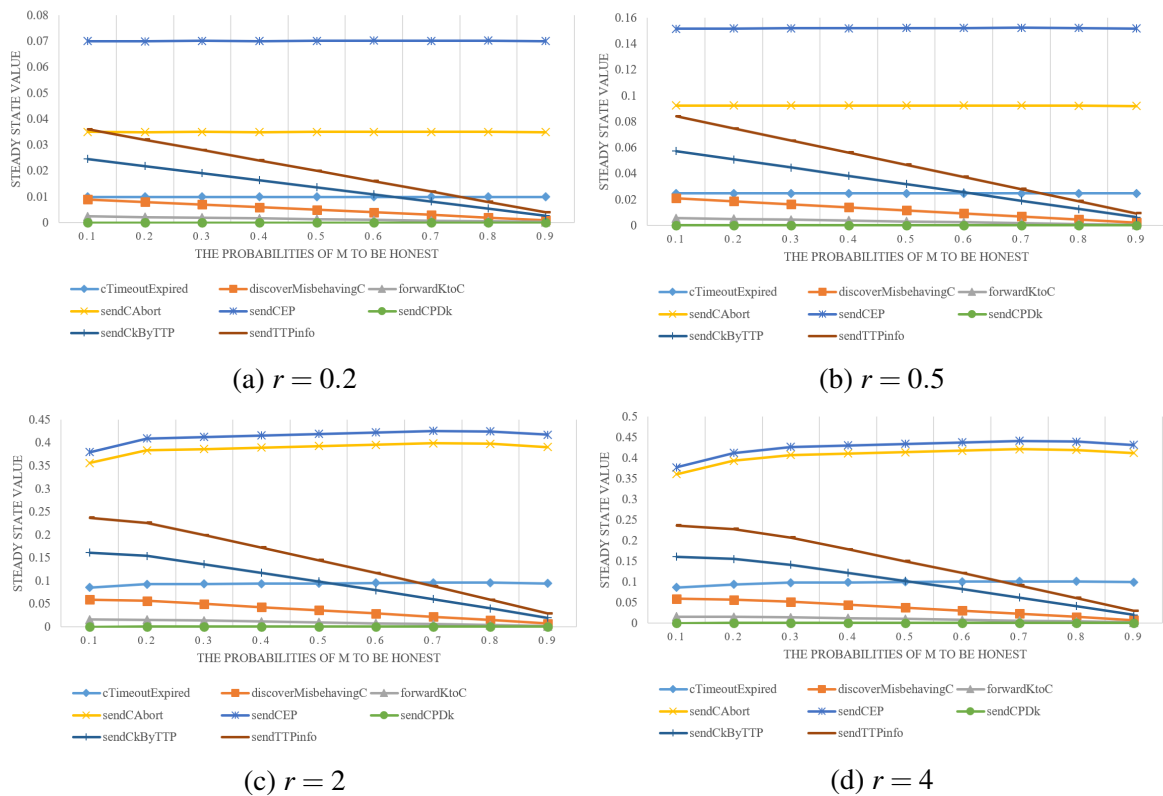


Fig. 4.64 The throughput analysis using ODE with $K=20$, $N=200$ and $S=20$ and with different rates for the shared actions between M and B.

The extended optimistic anonymous protocol when honest Ms interacts with two types of Cs

In this subsection, we explore our third proposed PEPA model presented in Subsection 4.3.4. In this model, the honest Ms copies interact with two types of customers: honest and

misbehaving. Thus we evaluated the model providing a different population of honest and misbehaving customers to study how this would impact the system performance. We are interested in investigating the population level and the throughput analyses as performance measures. We assigned 1 as a value for all action rates. The main actions of M are calculated based on the number of customers in the system, and the main actions of TTP are calculated based on the number of customers and TTP in the system, as mentioned in Section 4.3.4.

We are interested in the population level analysis of C_4 , C_6 , C_{04} , C_{06} , C_9 and C_{10} states. C_4 and C_6 are the states of misbehaving customer waiting to be served by M. C_{04} and C_{06} are the states of the honest customer to have a service from M. C_9 and C_{10} are the states when C sends TTP a request to solve the dispute and receives a response from TTP, respectively. Further, we provide the throughput analysis of some actions of M and TTP to serve the customers when the system deals with different populations of honest and misbehaving customers.

First, the number of TTP and Bs in the system are assigned to 20 for both. Figure 4.65 shows the population level analysis using ODE when the number of customers is 100 and 10 of them are misbehaving customers. Figure 4.66 shows the population level analysis using ODE when the number of customers is 100 and 90 of them are misbehaving customers. Comparing the results of the two figures, when the number of misbehaving customers increased, this would trigger the load on TTP to increase, causing the waiting customer on C_{10} to increase. Also, the number of honest customers waiting on C_{06} to receive the product decryption key is larger in Figure 4.65 compared to the number of misbehaving customers waiting on C_6 in Figure 4.66 to receive a response from M. This is because misbehaving customers would seek a dispute resolution from TTP.

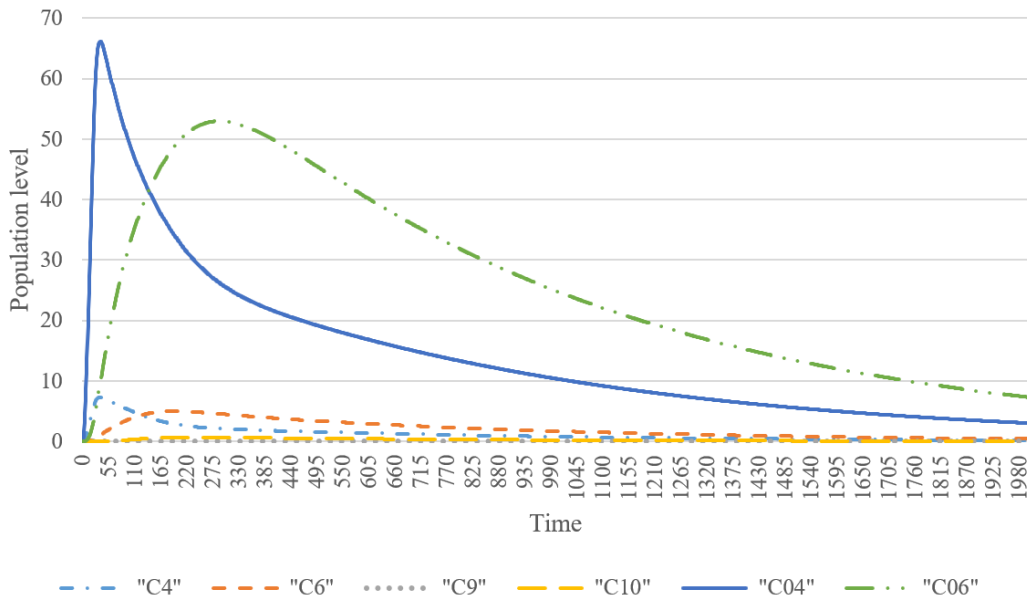


Fig. 4.65 The population level analysis using ODE with $K=20$, $N=100$ and $S=20$ (10 misbehaving customers).

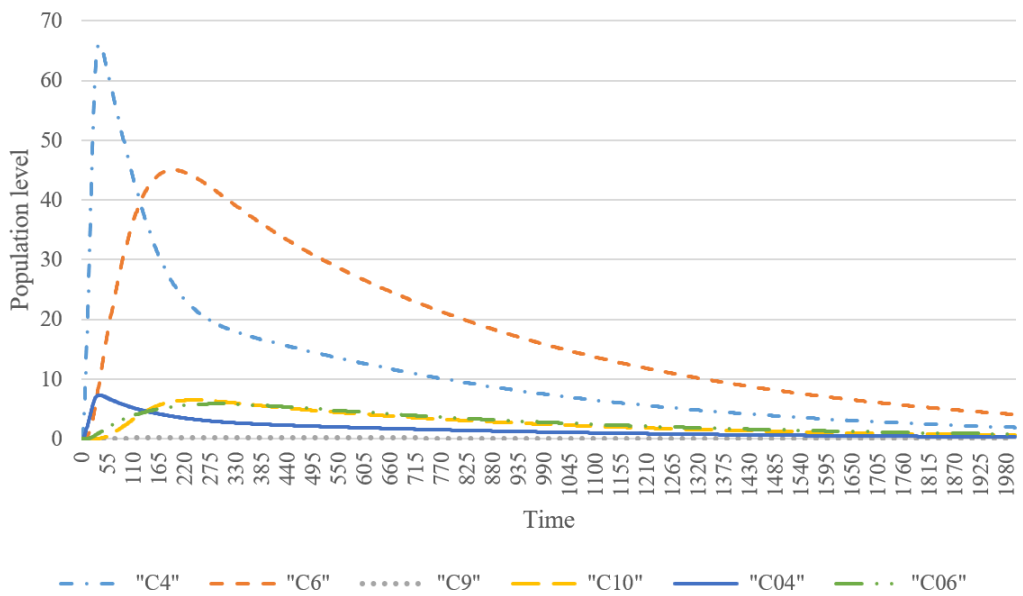


Fig. 4.66 The population level analysis using ODE with $K=20$, $N=100$ and $S=20$ (90 misbehaving customers).

Figures 4.67 and 4.68 present the throughput analysis of some actions to serve the customers from M and TTP when the number of misbehaving customers in the system

is 10 and 90, respectively. In Figure 4.67, when the number of honest customers larger in the system, the throughput of action *sendCPDk* is larger than the throughput of action *sendCPDk* in Figure 4.68 when we have just 10 honest customers in the system. This means more honesty customers are waiting to receive the product decryption key from the honest Merchant. Further, you could see clearly that the throughput values of the actions *cTimeoutExpired*, *sendTTPinfo*, *cspendTheCoinToTTP*, and *discoverMisbehavingC* are larger in Figure 4.68 when we have a larger number of misbehaving customer in the system, compared to Figure 4.67.

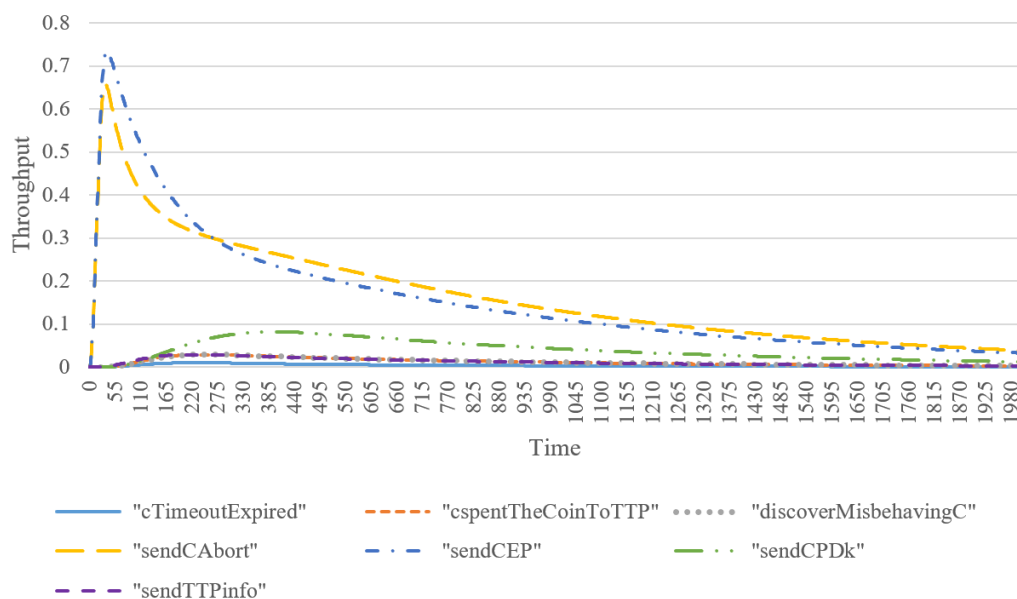


Fig. 4.67 The throughput analysis of actions using ODE with $K=20$, $N=100$ and $S=20$ (10 misbehaving customers).

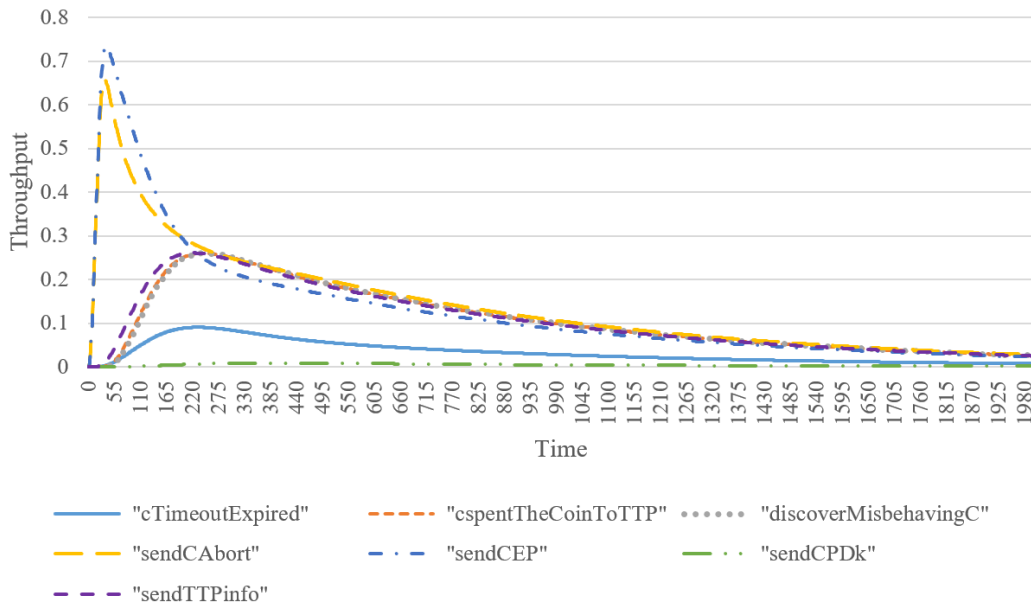


Fig. 4.68 The throughput analysis of actions using ODE with $K=20$, $N=100$ and $S=20$ (90 misbehaving customers).

Second, we increased the number of the customer population to 500 and kept the number of TTP and Bs in the system to 20 to see how having a relatively small number of resources would influence the system performance. Figure 4.69 shows the population level when the number of misbehaving customers is 50, which is 10% of the total population. Compared to Figure 4.65, when the number of customers is 100 and 10% of them misbehaving, there are a large variance between C_4 and C_6 , and C_{04} and C_{06} for a much longer time. Furthermore, increasing the population to 500 causes the system to take a much longer time to settle and reached a steady state. Figure 4.70 shows the population level when the number of misbehaving customers is 450, which is 90% of the total population. Having a larger number of misbehaving customer would cause the system to fluctuate for quite a long time, compared to Figure 4.69.

Figures 4.71 and 4.72 present the throughput analysis of some actions to serve the customers from M and TTP when the number of misbehaving customers in the system is 50 and 450, respectively. Figure 4.72 shows the fluctuation in the throughput values of *sendCAbort*, *sendCEP*, *sendTTPinfo*, *cspendTheCoinToTTP*, and *discoverMisbehavingC* actions when the population number is 500 and 90% of them are misbehaving, compared to Figure 4.71 when just 10% of the population are misbehaving. Further, It shows the system takes a longer time to settle.

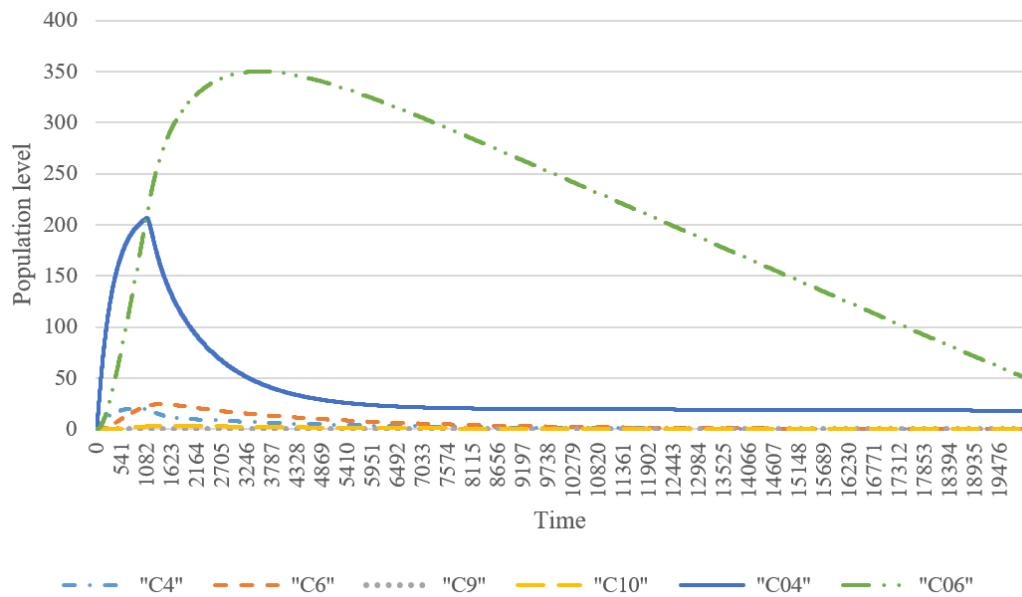


Fig. 4.69 The population level analysis using ODE with $K=20$, $N=500$ and $S=20$ (50 misbehaving customers).

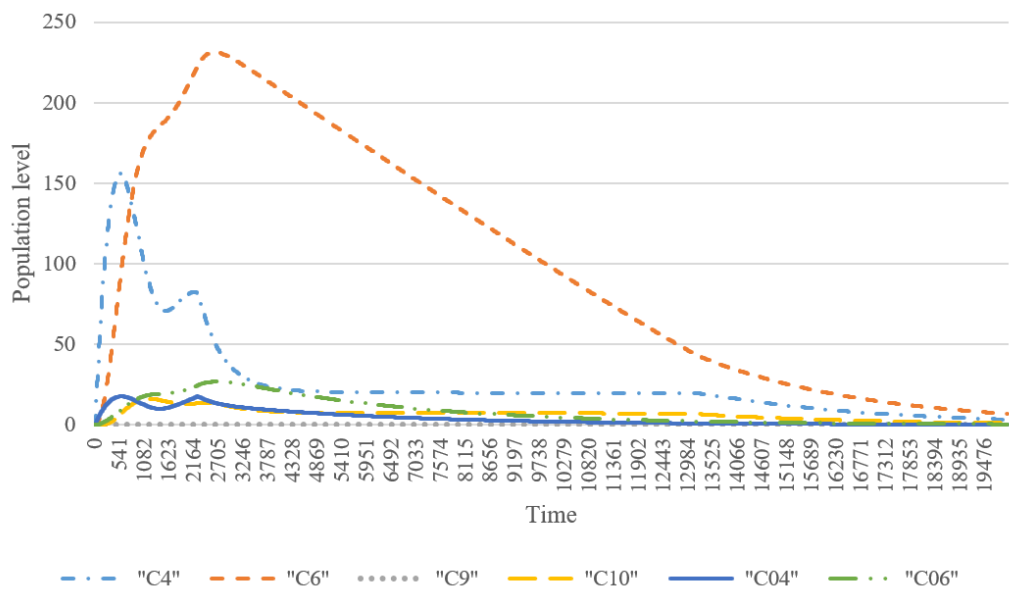


Fig. 4.70 The population level analysis using ODE with $K=20$, $N=500$ and $S=20$ (450 misbehaving customers).

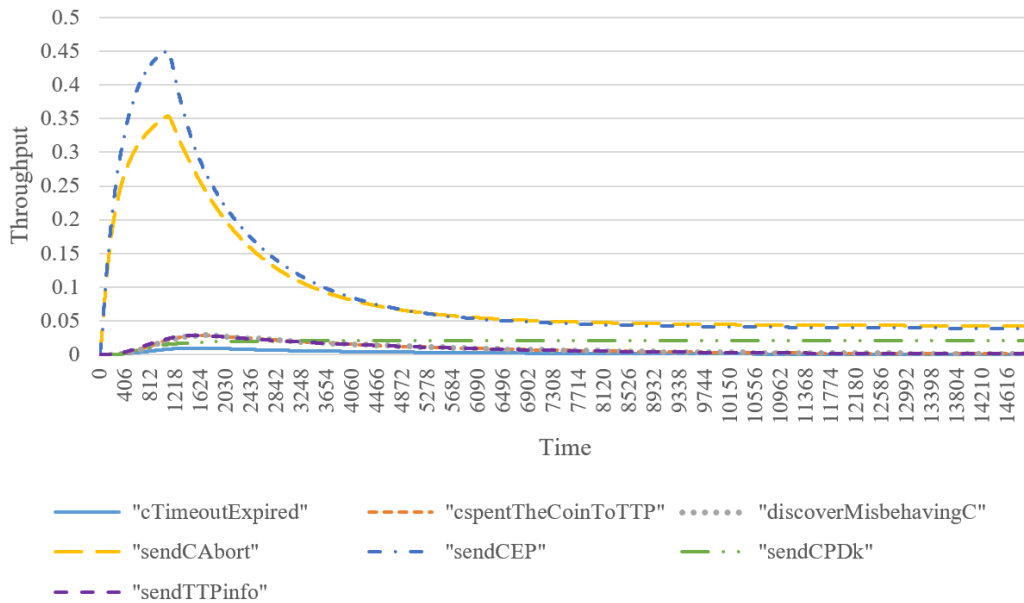


Fig. 4.71 The throughput analysis of actions using ODE with $K=20$, $N=500$ and $S=20$ (50 misbehaving customers).

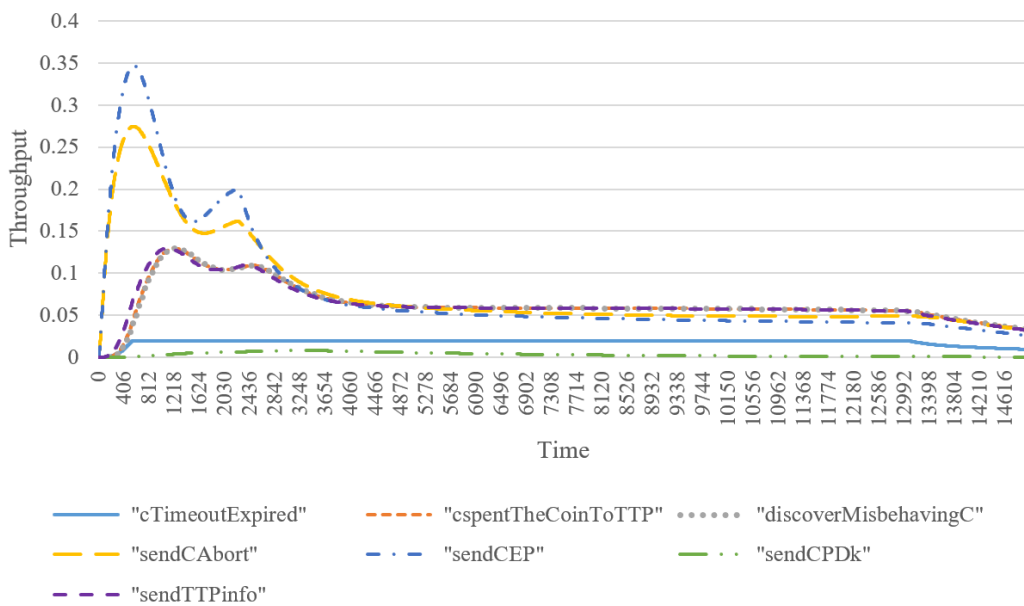


Fig. 4.72 The throughput analysis of actions using ODE with $K=20$, $N=500$ and $S=20$ (450 misbehaving customers).

Moreover, we seek to investigate how increasing the number of resources such as TTP and Bs would mitigate the performance cost. Figure 4.73 shows the population level when the number of misbehaving customers is 450, which is 90% of the total population, and the

number of TTP in the system is increased to 100. The fluctuation that the system experienced has disappeared, compared to Figure 4.70. However, increasing the number of Bs to 40 has a significant improvement in the population level, causing them to decrease much quicker, as seen in Figure 4.74. Therefore having a reasonable number of resources in relation to the number of customers in the system could mitigate the performance cost introduced by the protocol behaviours.

Further, Figures 4.75 and 4.76 show the throughput analysis of the same actions as in Figure 4.72. They show how increasing the resources would improve the throughput values of the actions and cause the system to settle faster.

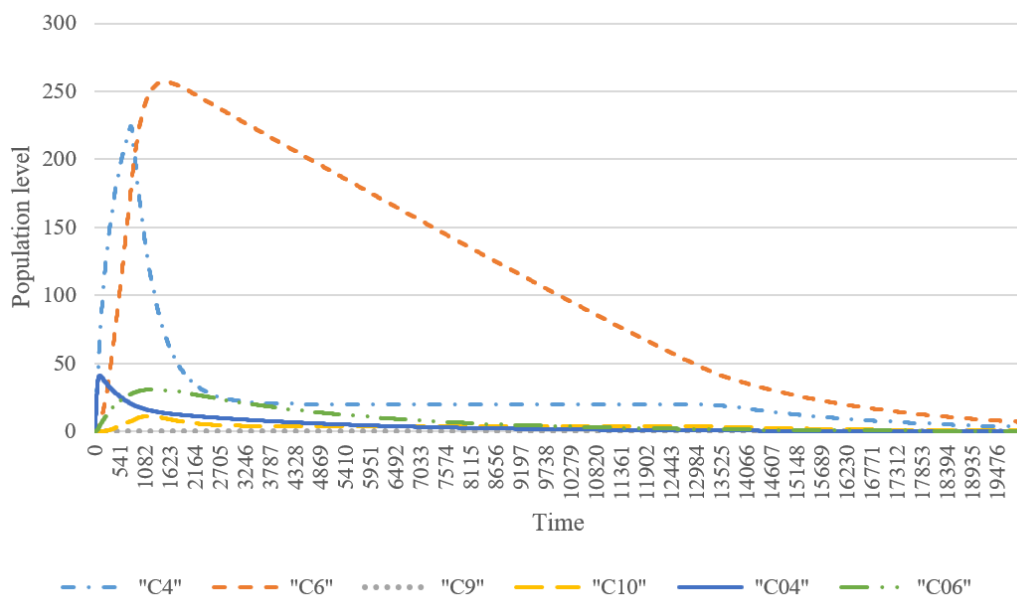


Fig. 4.73 The population level analysis using ODE with $K=100$, $N=500$ and $S=20$ (450 misbehaving customers).

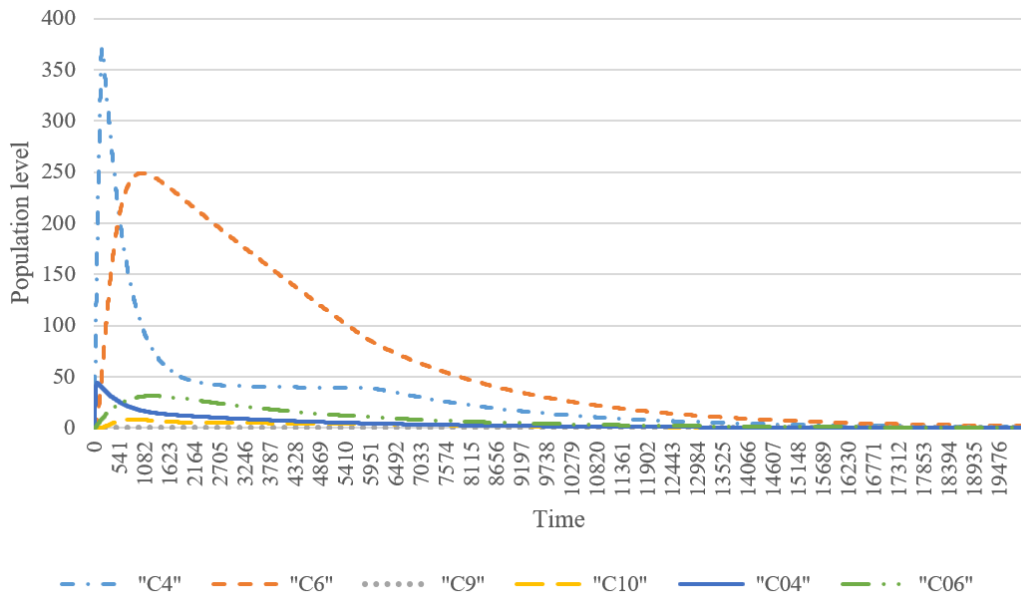


Fig. 4.74 The population level analysis using ODE with $K=100$, $N=500$ and $S=40$ (450 misbehaving customers).

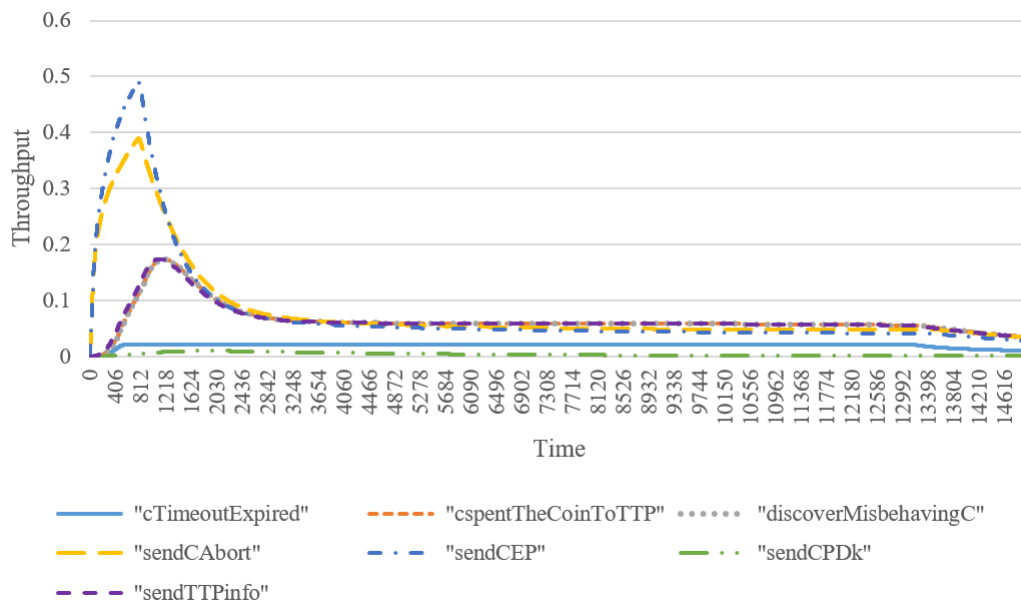


Fig. 4.75 The throughput analysis of actions using ODE with $K=100$, $N=500$ and $S=20$ (450 misbehaving customers).

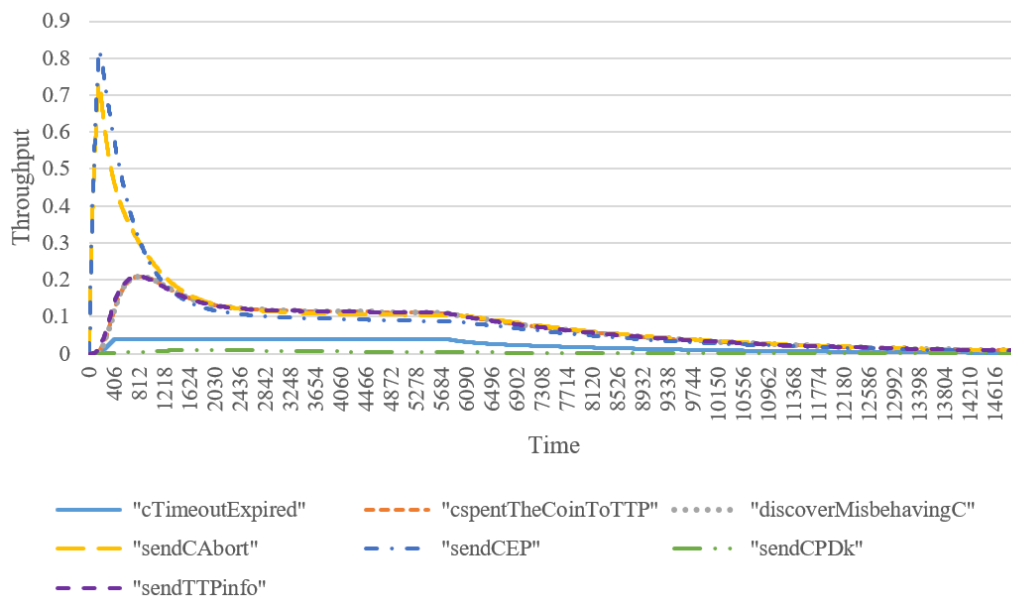


Fig. 4.76 The throughput analysis of actions using ODE with $K=100$, $N=500$ and $S=40$ (450 misbehaving customers).

4.5 Conclusion

This investigation considers a type of e-commerce fair-exchange protocol called an anonymous and failure resilient fair-exchange e-commerce protocol [62], which is a type of a non-repudiation security protocol implemented during e-commerce transactions. In an electronic commerce environment, two or more parties interact with each other to exchange products. These types of security protocols have been developed to ensure fair exchange between participants and that no party can take advantage over the other party. We proposed PEPA models for the anonymous and failure resilient fair-exchange e-commerce protocol which is proposed by Ray *et al.* [62]. We formulated the PEPA models in different ways based on the description provided by Ray *et al.* to have a complete understanding of the protocol's behaviours. The models are then evaluated based on different scenarios, which help achieve an in-depth understanding of protocol behaviour and the associated performance costs.

The results indicated that the basic failure protocol without an anonymity feature introduced lower performance cost than when the protocol preserved the anonymity of customer which introduced extra performance cost. Furthermore, when there is a dispute between participants so that TTP involvement is active, this would introduce extra load on TTP and/or Bs which in turn would influence the system performance. Moreover, we showed

how scaling up TTP and/or B resources to handle escalating misbehaving party mitigates a negative impact on the system performance.

4.6 Chapter Summary

This chapter explores the performance cost introduced by a security protocol known as an anonymous and failure resilient fair-exchange e-commerce protocol. Following the description provided by Ray *et al.* in [62], the PEPA models were formulated in two different ways: with and without an anonymity feature. Moreover, both protocol versions were also modelled in two ways: as a basic protocol with no misbehaviour of any parties whereby it does not require the active involvement of TTP, and as an extended protocol whereby the involvement of TTP is essential to resolving disputes between participants. These enable understanding the protocol's behaviour, evaluating the performance cost it introduces and showing how adding more security features, such as anonymity, introduces extra performance overhead. We evaluated and studied some aspects of the system performance using the ordinary differential equations (ODEs). Understanding the behaviour of these protocols and the performance cost they introduce could enable the development of lightweight ecommerce protocols. This study uses a PEPA Eclipse plug-in to support the creation and evaluation of the proposed PEPA models.

In this chapter, we have begun to model how a user of the system can misbehave. We extend this idea in the following chapter by introducing explicit models of the misbehaving person/attacker. We investigate systems under attack by developing a PEPA model for an attacker behaviour based on an attack graph. We provide two methods for automating the generation of PEPA models based on an existing attack graph specification. We investigate the attacker's behaviour by evaluating the PEPA models. We perform path analysis, sensitivity analysis, and estimating the time to compromise a system which can assist a defender in developing an effective defence strategy.

Chapter 5

Performance modelling of attack graphs

5.1 Introduction

Defending a system or a network and keeping it secure is not trivial. A defender must keep a system secure by preventing or early detecting the attacker's intent in order to react and recover in good time. This can be accomplished by identifying and analysing all possible attack paths that an attacker could use to attack the system, and then implementing appropriate protection measures. An attack graph is a popular graph-based method proposed by Swiler *et al.* [70]. It can support the defender to understand the attacker's behaviour and then work to protect a system [46]. Attack graphs present the different attack paths that attacker can follow to exploit multiple vulnerabilities in a system in order to achieve its final goal [42, 46]. In an attack graph, each attack path is a sequence of nodes representing the vulnerabilities or exploits in a system or host [46]. It shows an attacker's behaviour in exploiting vulnerabilities in a system or host from an initial node [42].

Doing a risk assessment based on the attack graph of a system allows a defender to see the system from an attacker's perspective. The defender can perform path analysis to identify the most significant threatening path that the attacker can exploit to compromise a system [70]. Moreover, the defender can then perform a sensitivity analysis to show how hardening the security of some nodes would affect the security status of the system. Many studies have been conducted in the field of risk assessment using attack graphs [1, 2, 40, 60, 69, 86].

Building a Performance Evaluation Process Algebra (PEPA) model version of the attack graph can help to understand attacker behaviour and identify critical threats, estimate time to compromise a system, and then implement suitable countermeasures in timely to keep a system secure. The time to compromise a system is the time it takes an attacker to successfully compromise a system [54]. It is an important factor in determining how much safe time a system has before it is compromised. In this chapter, we develop methods to build a PEPA

model of an existing attack graph to represent the attacker's behaviour and interaction with a system with time-variable aspects. This allow us to perform path analysis, sensitivity analysis and to estimate the time to compromise a system. Our proposed PEPA models of an attack graph can be analysed via a continuous-time Markov chain with rates to support estimating the time to compromise the system and the time it takes the attacker to get to a particular vulnerability on a system.

This chapter presents two methods to create PEPA model for a pre-constructed attack graph with its vulnerabilities that already assigned attack probability from the international standard CVSS. The first method generates PEPA model that comprises of a system and an attacker as a coupled component. The second method generates a PEPA model that comprises of separate system and attacker components. The first method is simpler, whereas the second method has more potential. The second method allows us to construct a PEPA model of an attack graph that includes an attacker component, allowing us to explicitly implement different attacker skills into the model. In this chapter, we show through a case study how we can use the generated PEPA models to perform path analysis and sensitivity analysis, and to estimate the attacker's time to compromise the system.

5.2 Methods for creating a PEPA model for a given attack graph

5.2.1 Basic PEPA model

The outcome of the algorithm

We propose an algorithm which derives an equivalence PEPA model of a pre-existing attack graph specification. The PEPA model comprises of a coupled component of system and attacker. For each node in the attack graph, a vulnerability name is an action name. The action rate is based on the probability of the vulnerability being breached and the transition probability. The probability of the vulnerability being breached is defined by the complexity of the attack in the Common Vulnerability Scoring System (CVSS). A *failed* action and rate for each node are introduced in the model. This action occurs when the attacker fails to exploit the vulnerability in that node. The *failed* action rate is based on the probability that an attacker is unable to exploit the vulnerability. In this method, we assume that when an attacker fails to exploit a vulnerability, the attacker returns to the attack graph's root node, as this can be a possible consequence. While it is possible that a failed attack would give the attacker more knowledge of the system, we do not model that learning aspect in this

version of the model. In Chapter 6, we will consider different models of attacker expertise and the potential for learning. The resultant PEPA model can be used to perform attack graph analysis, such as path analysis and sensitivity analysis.

The starting point of the algorithm

The input to the algorithm is an attack graph specification, which includes information about the relationship between nodes in the attack graph, the vulnerability name, and the probability of the vulnerability being breached for each node. The transition probability from one node to the next node is calculated by Markov chain as explained in [69]. For example, Figure 5.1 illustrates a simple example of an attack graph. The attack graph comprises four nodes. Node Start is where the attack begins, and H1, H2 and H3 are hosts in a system. The breach probability of the vulnerability in node H1 is 0.4 ($bH1 = 0.4$), and the breach probability of the vulnerability in node H2 is 0.8 ($bH2 = 0.8$). We calculate the transition probability from the Start node to node H1 (P_{SH1}) based on the following equation:

$$P_{SH1} = \frac{bH1}{bH1 + bH2}$$

Where P_{SH1} is the transition probability from the Start node to node H1, $bH1$ is the breach probability of the vulnerability in H1 and $bH2$ is the breach probability of the vulnerability in H2.

$$P_{SH1} = \frac{0.4}{0.4 + 0.8} = 0.33$$

The transition probability from the Start node to node H1 is 0.33 ($P_{SH1} = 0.33$).

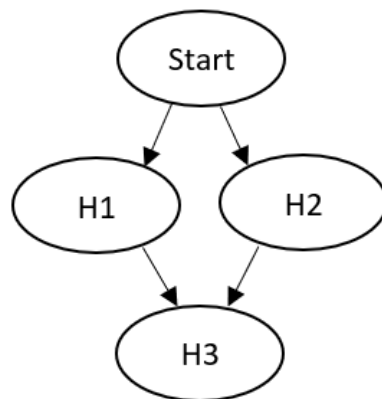


Fig. 5.1 An example of an attack graph that comprises four nodes: Start, host 1 (H1), host 2 (H2), and host 3 (H3).

The steps

Algorithm 1 Create the system and attacker component based on the attack graph given

```

1: print "Start=(start,1).SystemAttacker"+first node's name";"
2: for all node  $\in$  attackGraph do
3:   if #directNeighbournode > 0 & node  $\neq$  goalnode then
4:     print "SystemAttacker"+node name+"="
5:     s  $\leftarrow$  #directNeighbournode
6:     for all neigh  $\in$  directNeighbournode do
7:       s  $\leftarrow$  s - 1
8:       convert at least first letter of neigh's vulnerability name to lower case
9:       print "("+neigh's vulnerability name+","
           +node's BreachProbability*neigh's TransitionProbability
           +").SystemAttacker" +neigh's name
10:    if s = 0 then
11:      if this node  $\neq$  first node in attackGraph then
12:        print "(failed,"+ (1 - node's BreachProbability)+").System"
           +attackGraph's first node name ";" +new line
13:      else
14:        print ";" +new line
15:      end if
16:    else
17:      print "+"

```

```

18:     end if
19:   end for
20: else
21:   print "SystemAttacker"+node's name+"=(compromised, "
        +node's BreachProbability +").Completed+(failed,"
        +(1 - node's BreachProbability)+").SystemAttacker"
        + attackGraph's first node name";" + new line
22:   print "Completed=(completed,1).Start;"
23: end if
24: end for

```

Example

In this subsection, we illustrate an example of a valid PEPA model resulting from implementing and running the method using Java. This model can be evaluated by using the PEPA Eclipse plug-in. Figure 5.2 illustrates the attack graph. The probability of a vulnerability to be breached for each host is presented in Table 5.1.

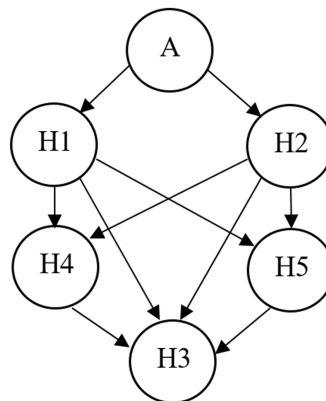


Fig. 5.2 Attack graph consists of six nodes: node A is the starting node, and the H1, H2, H3, H4, and H5 nodes are the hosts in a system [69].

Table 5.1 The probabilities of vulnerabilities to be breached [69].

Host	H1	H2	H3	H4	H5
The vulnerability action	servU5	telnet	sql	rpc	remoteLog
The breach probability	0.4	0.8	0.6	0.5	0.9

The PEPA model for the attack graph is as follows:

The sequential component of an attacker and a system

$$\begin{aligned}
Start &= (start, 1).SystemAttackerA; \\
SystemAttackerA &= (servU5, 0.33).SystemAttackerH1 \\
&+ (telnet, 0.67).SystemAttackerH2; \\
SystemAttackerH1 &= (sql, 0.12).SystemAttackerH3 + (rpc, 0.1).SystemAttackerH4 \\
&+ (remoteLogin, 0.18).SystemAttackerH5 \\
&+ (failed, 0.6).SystemAttackerA; \\
SystemAttackerH2 &= (sql, 0.24).SystemAttackerH3 + (rpc, 0.2).SystemAttackerH4 \\
&+ (remoteLogin, 0.36).SystemAttackerH5 \\
&+ (failed, 0.2).SystemAttackerA; \\
SystemAttackerH3 &= (compromised, 0.6).Completed + (failed, 0.4).SystemAttackerA; \\
Completed &= (completed, 1).Start; \\
SystemAttackerH4 &= (sql, 0.5).SystemAttackerH3 + (failed, 0.5).SystemAttackerA; \\
SystemAttackerH5 &= (sql, 0.9).SystemAttackerH3 + (failed, 0.1).SystemAttackerA;
\end{aligned}$$

The above component presents a coupled component of the attacker and system. It has eight behaviours. The behaviours *SystemAttackerA*, *SystemAttackerH1*, *SystemAttackerH2*, *SystemAttackerH3*, *SystemAttackerH4*, *SystemAttackerH5* represent the nodes A, H1, H2, H3, H4 and H5 in the attack graph, respectively. *Start* and *Completed* represent that the attacker starts the attack and the attacker successfully completes the attack, respectively. The actions *servU5*, *telnet*, *sql*, *rpc* and *remoteLogin* are the vulnerabilities in the attack graph and the action rates are based on the probability of the vulnerability being breached and the transition probability. The *failed* action is when the attacker fails to exploit the vulnerability and its rate is based on the probability of the attacker failing to breach the vulnerability. The *compromised* action is performed by the attacker when the attacker successfully reaches its final goal in the attack graph.

5.2.2 Multiple component PEPA model

The outcome of the algorithm

We propose an algorithm which derives an equivalent PEPA model of a pre-constructed attack graph. The PEPA model comprises two sequential components, representing system and attacker, and an equation for the model with the cooperation set. For each node in the attack graph, a vulnerability name is an action name. The action rate is based on the probability of

the vulnerability being breached and the transition probability. In this method, we considered an alternative consequence that can happen when an attacker fails to exploit the vulnerability, which is different from the outcome assumed in the first method presented in Section 5.2.1. In this method, we assumed that when an attacker fails to exploit the vulnerability, the attacker remains on the current node, as this can also be a possible outcome.

The starting point of the algorithm

The inputs are:

- The attack graph: the relationship between nodes in the attack graph, the vulnerability name, and the probability of the vulnerability being breached for each node. The transition probability from one node to the next node is calculated by Markov chain as explained in [69] and described in Section 5.2.1.
- The abilities/steps of an attacker that the attacker follows to compromise a system: The relationship between the attacker's abilities, the attacker's different abilities name, and rate are inputs to the algorithm. An example of the attacker's abilities/steps to compromise a system and the relation between them is presented in the following graph:

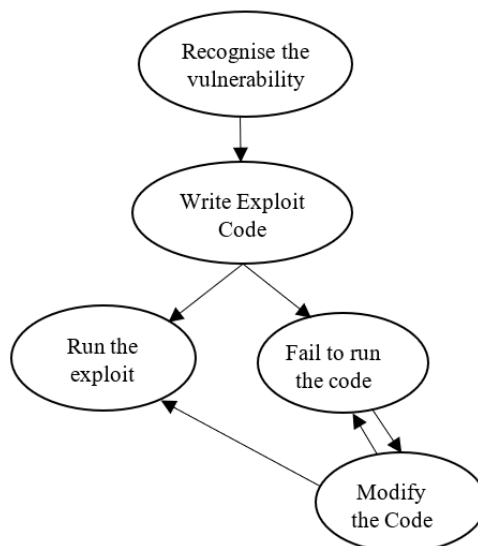


Fig. 5.3 An example of the attacker's abilities/steps to compromise a system.

The steps

Algorithm 2 Create a system component based on the attack graph given

```

1: print "Start=(start,1).System"+first node's name";"
2: for all node  $\in$  attackGraph do
3:   if #directNeighbournode > 0 & node  $\neq$  goalnode then
4:     print "System"+node name+"="
5:      $s \leftarrow$  #directNeighbournode
6:     for all neigh  $\in$  directNeighbournode do
7:        $s \leftarrow s - 1$ 
8:       convert at least first letter of neigh's vulnerability name to lower case
9:       print "("+neigh's vulnerability name+", "
           +node's BreachProbability*neigh's TransitionProbability
           +").System" +neigh's name
10:      if  $s = 0$  then
11:        if this node  $\neq$  first node in attackGraph then
12:          print "(failed,"+ (1 - node's BreachProbability)+").System"
              +node' name+";"+new line
13:        else
14:          print ";" +new line
15:        end if
16:      else
17:        print "+"
18:      end if
19:    end for
20:  else
21:    print "System"+node's name+"=(compromised, "
           +node's BreachProbability +").Completed+(failed,"
           +(1 - node's BreachProbability)+").System"
           +node' name";" + new line
22:    print "Completed=(completed,1).Start;"
23:  end if
24: end for

```

Algorithm 3 Create an attacker component based on the given attacker's steps to attack a system and the attack graph

```

1: convert at least first letter of attacker's name to upper case
2: print attacker's name+"Start=(start,1)."+ attacker's name+ first node's name+";"

```

```

3: for all node ∈ attackGraph do
4:   if node ≠ first node in attackGraph then
5:     for all step ∈ attackerSteps do
6:       print attacker's name+step's number +node's name + "="
7:       if #directNextStepsstep > 0 then
8:         r ← #directNextStepsstep
9:         for all nextStep ∈ directNextStepsstep do
10:          r ← r − 1
11:          convert at least first letter of step's name to lower case
12:          print "("+ step's name+", "+ step's rate +")."+attacker's name
              + nextStep's number+node's name
13:          if r = 0 then
14:            print ";" + new line
15:          else
16:            print "+"
17:          end if
18:        end for
19:      else
20:        convert at least first letter of step's name to lower case
21:        print "("+ step's name +", "+ step's rate +")."+attacker's name + node's name
              + ";" + new line
22:      end if
23:    end for
24:  end if
25:  if #directNeighbournode > 0 & node ≠ goalnode then
26:    print attacker's name + node's name + "="
27:    s ← #directNeighbournode
28:    for all neigh ∈ directNeighbournode do
29:      s ← s − 1
30:      convert at least first letter of neigh's vulnerability name to lower case
31:      print "("+neigh's vulnerability name+", "
              +node's BreachProbability*neigh's TransitionProbability
              +")." +attacker's name+attacker's first step's number+neigh's name
32:      if s = 0 then
33:        if this node ≠ first node in attackGraph then
34:          print "(failed,"+ (1 − node's BreachProbability) +")."

```

```

    +attacker's name+first step's number +node's name+";"+new line
35:     else
36:         print ";" +new line
37:     end if
38:     else
39:         print "+"
40:     end if
41: end for
42: else
43:     print attacker's name+node's name+"=(compromised, "
        +node's BreachProbability +").Completed+(failed,"
        +(1 - node's BreachProbability)+")."
        +attacker's name+first step's number +node's name+";"
44:     print "Completed=(completed,1)."+ attacker's name+"Start; "
45: end if
46: end for

```

Algorithm 4 Create a system equation for the model

```

1: print attacker's name+ "Start<start, failed, compromised, completed"
2: for all node ∈ attackGraph do
3:     if node's vulnerability's name != null then
4:         if node != last node in attackGraph then
5:             print node's vulnerability' name+", "
6:         else
7:             print node's vulnerability' name
8:         end if
9:     end if
10: end for
11: print ">"+ "Start"

```

Example

The following is a valid PEPA model that comprises of an attacker component, a system component and a system equation. This model was generated by implementing and running the method using Java. This model can be evaluated using the PEPA Eclipse plug-in. The attack graph is the same as the attack graph in Figure 5.2, and the probability of a

vulnerability being breached for each host is given in Table 5.1. Figure 5.4 shows the attacker's abilities/steps to compromise a system. The PEPA model for the attack graph is as follows:

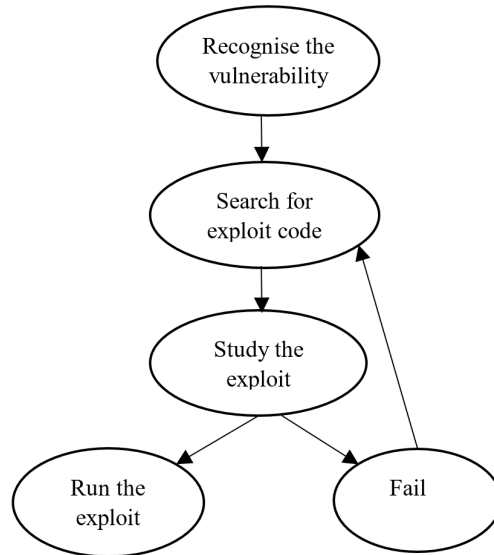


Fig. 5.4 An example of the attacker's abilities to compromise a system.

System component

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemA}; \\
 \textit{SystemA} &= (\textit{servU5}, 0.33).\textit{SystemH1} + (\textit{telnet}, 0.67).\textit{SystemH2}; \\
 \textit{SystemH1} &= (\textit{sql}, 0.12).\textit{SystemH3} + (\textit{rpc}, 0.1).\textit{SystemH4} \\
 &\quad + (\textit{remoteLogin}, 0.18).\textit{SystemH5} \\
 &\quad + (\textit{failed}, 0.6).\textit{SystemH1}; \\
 \textit{SystemH2} &= (\textit{sql}, 0.24).\textit{SystemH3} + (\textit{rpc}, 0.2).\textit{SystemH4} \\
 &\quad + (\textit{remoteLogin}, 0.36).\textit{SystemH5} \\
 &\quad + (\textit{failed}, 0.2).\textit{SystemH2}; \\
 \textit{SystemH3} &= (\textit{compromised}, 0.6).\textit{Completed} + (\textit{failed}, 0.4).\textit{SystemH3}; \\
 \textit{Completed} &= (\textit{completed}, 1).\textit{Start}; \\
 \textit{SystemH4} &= (\textit{sql}, 0.5).\textit{SystemH3} + (\textit{failed}, 0.5).\textit{SystemH4}; \\
 \textit{SystemH5} &= (\textit{sql}, 0.9).\textit{SystemH3} + (\textit{failed}, 0.1).\textit{SystemH5};
 \end{aligned}$$

The system is depicted by the component above. The description of this component is similar to the component described in the example in Section 5.2.1.

Attacker component

AttackerStart = (*start*, 1).*AttackerA*;
AttackerA = (*servU5*, 0.33).*Attacker0H1* + (*telnet*, 0.67).*Attacker0H2*;
Attacker0H1 = (*recogniseVuln*, 1).*Attacker1H1*;
Attacker1H1 = (*searchCode*, 1).*Attacker2H1*;
Attacker2H1 = (*studyExploit*, 1).*Attacker3H1*
+ (*studyExploit*, 1).*Attacker4H1*;
Attacker3H1 = (*runExploit*, 1).*AttackerH1*;
Attacker4H1 = (*fail*, 1).*Attacker1H1*;
AttackerH1 = (*sql*, 0.12).*Attacker0H3* + (*rpc*, 0.1).*Attacker0H4*
+ (*remoteLogin*, 0.18).*Attacker0H5*
+ (*failed*, 0.6).*Attacker0H1*;
Attacker0H2 = (*recogniseVuln*, 1).*Attacker1H2*;
Attacker1H2 = (*searchCode*, 1).*Attacker2H2*;
Attacker2H2 = (*studyExploit*, 1).*Attacker3H2*
+ (*studyExploit*, 1).*Attacker4H2*;
Attacker3H2 = (*runExploit*, 1).*AttackerH2*;
Attacker4H2 = (*fail*, 1).*Attacker1H2*;
AttackerH2 = (*sql*, 0.24).*Attacker0H3* + (*rpc*, 0.2).*Attacker0H4*
+ (*remoteLogin*, 0.36).*Attacker0H5*
+ (*failed*, 0.2).*Attacker0H2*;
Attacker0H3 = (*recogniseVuln*, 1).*Attacker1H3*;
Attacker1H3 = (*searchCode*, 1).*Attacker2H3*;
Attacker2H3 = (*studyExploit*, 1).*Attacker3H3*
+ (*studyExploit*, 1).*Attacker4H3*;
Attacker3H3 = (*runExploit*, 1).*AttackerH3*;
Attacker4H3 = (*fail*, 1).*Attacker1H3*;
AttackerH3 = (*compromised*, 0.6).*AttackerCompleted*
+ (*failed*, 0.4).*Attacker0H3*;
AttackerCompleted = (*completed*, 1).*AttackerStart*;
Attacker0H4 = (*recogniseVuln*, 1).*Attacker1H4*;
Attacker1H4 = (*searchCode*, 1).*Attacker2H4*;
Attacker2H4 = (*studyExploit*, 1).*Attacker3H4*
+ (*studyExploit*, 1).*Attacker4H4*;
Attacker3H4 = (*runExploit*, 1).*AttackerH4*;
Attacker4H4 = (*fail*, 1).*Attacker1H4*;
AttackerH4 = (*sql*, 0.5).*Attacker0H3* + (*failed*, 0.5).*Attacker0H4*;

$$\begin{aligned}
Attacker0H5 &= (recogniseVuln, 1).Attacker1H5; \\
Attacker1H5 &= (searchCode, 1).Attacker2H5; \\
Attacker2H5 &= (studyExploit, 1).Attacker3H5 \\
&\quad + (studyExploit, 1).Attacker4H5; \\
Attacker3H5 &= (runExploit, 1).AttackerH5; \\
Attacker4H5 &= (fail, 1).Attacker1H5; \\
AttackerH5 &= (sql, 0.9).Attacker0H3 + (failed, 0.1).Attacker0H5;
\end{aligned}$$

This component represents the attacker's behaviours. The attacker moves sequentially from state *AttackerStart* to *AttackerCompleted* to successfully attack the system and reach its final goal. *AttackerA*, *AttackerH1*, *AttackerH2*, *AttackerH3*, *AttackerH4* and *AttackerH5* reflect A, H1, H2, H3, H4 and H5 in the attack graph, respectively. The actions *servU5*, *telnet*, *sql*, *rpc* and *remoteLogin* are the vulnerabilities of the hosts in the attack graph. Their rates are based on the probability of the vulnerability being breached and the transition probability. The *failed* action reflects the attacker failure to exploit the vulnerability and its rate is the probability of the attacker failing to breach the vulnerability. The behaviours between *AttackerA* and *AttackerH1* represent the attacker's steps/abilities to exploit the vulnerability in the host, as it is shown in Figure 5.4. They are *Attacker0H1*, *Attacker1H1*, *Attacker2H1*, *Attacker3H1* and *Attacker4H1*. They are the same behaviours formulated between each connected hosts in the attack graph to present the attacker's steps to exploit the hosts' vulnerabilities. The actions *recogniseVuln*, *searchCode*, *studyExploit*, *runExploit* and *fail* are the behaviour's action of *Attacker0H1*, *Attacker1H1*, *Attacker2H1*, *Attacker3H1* and *Attacker4H1*, respectively. The attacker follows the steps suggested in Figure 5.4 when it moves from host to host to exploit their vulnerabilities.

System equation

$$\begin{aligned}
AttackerStart < start, failed, compromised, completed, \\
servU5, telnet, sql, rpc, remoteLogin > Start
\end{aligned}$$

This is the system equation. The components are initially in their first state, *AttackerStart* and *Start*. The actions between the components are shared actions.

5.3 Case study 1: Basic PEPA model for attack graph

5.3.1 System specification

The attack graph is taken from Sun *et al.* [69]. Sun *et al.* [69] propose Network Security Risk Assessment Model (NSRAM) based on attack graph and Markov chain to provide the optimal attack path. Their proposed model generates the attack graph, and then each vulnerability is assigned attack probability from the international standard CVSS. Then they use a Markov chain method to calculate the transition probability from node to node. The enterprise network security risk level and the attack probability of each path are provided using their proposed model. Their analysis is based on counting the steps till the system is compromised. However, estimating the actual time to compromise the system is critical to indicate how much a safe time the system has before it is compromised. Building a stochastic model of an attack graph enables to estimate the time is taken the attacker to compromise a system.

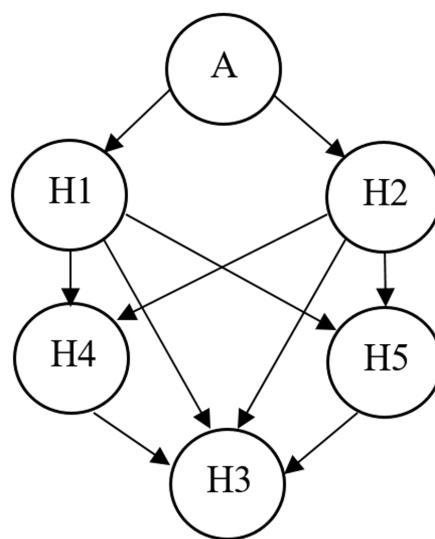


Fig. 5.5 Attack graph [69].

As in [69], the attack graph comprises of six nodes, as shown in Figure 5.5. Node A is the starting node, and node H3 is the target node. H1, H2, H3, H4 and H5 are hosts in the system that attacker tries to exploit their vulnerabilities in order to reach the final goal. Each host has a vulnerability name and probability value that the attacker could breach the host through it as in the following table, Table 5.2.

Table 5.2 The probabilities of vulnerabilities to be breached [69].

Host	H1	H2	H3	H4	H5
The vulnerability	servU5	telnet	sql	rpc	remoteLog
The breach probability	0.4	0.8	0.6	0.5	0.9

5.3.2 PEPA model

The following is a valid PEPA model that is generated by implementing Algorithm 1. In the PEPA model, there is one sequential component that represents the system and attacker moving sequentially from the different behaviours based on the activities specified in the model. The model is formulated as follows:

The sequential component of an attacker and a system

$$\begin{aligned}
Start &= (start, 1).SystemAttackerA; \\
SystemAttackerA &= (servU5, 0.33).SystemAttackerH1 \\
&\quad + (telnet, 0.67).SystemAttackerH2; \\
SystemAttackerH1 &= (sql, 0.12).SystemAttackerH3 + (rpc, 0.1).SystemAttackerH4 \\
&\quad + (remoteLogin, 0.18).SystemAttackerH5 \\
&\quad + (failed, 0.6).SystemAttackerA; \\
SystemAttackerH2 &= (sql, 0.24).SystemAttackerH3 + (rpc, 0.2).SystemAttackerH4 \\
&\quad + (remoteLogin, 0.36).SystemAttackerH5 \\
&\quad + (failed, 0.2).SystemAttackerA; \\
SystemAttackerH3 &= (compromised, 0.6).Completed + (failed, 0.4).SystemAttackerA; \\
Completed &= (completed, 1).Start; \\
SystemAttackerH4 &= (sql, 0.5).SystemAttackerH3 + (failed, 0.5).SystemAttackerA; \\
SystemAttackerH5 &= (sql, 0.9).SystemAttackerH3 + (failed, 0.1).SystemAttackerA;
\end{aligned}$$

The first state in our model is *Start* state. It is a starting point for the attacker to attack the system in our model. The attacker performs action *start* at rate 1 leading to state *SystemAttackerA*, which represents the first node in the attack graph. Then, the attacker can perform either action *servU5* at rate 0.33, which is the transition probability to H1, leading to *SystemAttackerH1* or *telnet* action at rate 0.67, which is the transition probability to H2, leading to *SystemAttackerH2*. *SystemAttackerH1* and *SystemAttackerH2* represent H1 and H2 in the attack graph, respectively.

When the attacker in state *SystemAttackerH1*, there is one of four actions could happen either *sql* at rate 0.12, which is the product of the probability of *servU5* being breached and the transition probability to H3, leading to *SystemAttackerH3*, *rpc* at rate 0.1, which is the product of the probability of *servU5* being breached and the transition probability to H4, leading to *SystemAttackerH4*, *remoteLogin* at rate 0.18, which is the product of the probability of *servU5* being breached and the transition probability to H5, leading to *SystemAttackerH5* or *failed* at rate 0.6, which is the result of (1-the probability of *servU5* being breached), leading to *SystemAttackerA*. *SystemAttackerH3*, *SystemAttackerH4*, *SystemAttackerH5* and *SystemAttackerA* represent H3, H4, H5 and A in the attack graph, respectively.

In state *SystemAttackerH2*, there is also one of four actions that could be performed either *sql* at rate 0.24, which is the product of the probability of *telnet* being breached and the transition probability to H3, leading to *SystemAttackerH3*, *rpc* at rate 0.2, which is the product of the probability of *telnet* being breached and the transition probability to H4, leading to *SystemAttackerH4*, *remoteLogin* at rate 0.36, which is the product of the probability of *telnet* being breached and the transition probability to H5, leading to *SystemAttackerH5* or *failed* at rate 0.2, which is the result of (1-the probability of *telnet* to be breached), leading to *SystemAttackerA*.

When the attacker in state *SystemAttackerH4*, it can perform either *sql* at rate 0.5, which is the product of the probability of *rpc* being breached and the transition probability to H3, leading to *SystemAttackerH3* or *failed* at rate 0.5, which is the result of (1- the probability of *rpc* being breached), leading back to *SystemAttackerA*. In state *SystemAttackerH5*, one of two actions can happen either *sql* at rate 0.9, which is the product of the probability of *remoteLogin* being breached and the transition probability to H3, leading to *SystemAttackerH3* or *failed* at rate 0.1, which is the result of (1-the probability of *remoteLog* being breached), leading back to *SystemAttackerA*.

In state *SystemAttackerH3* which represents the attacker starts to attack the target node by performing either action *compromised* at rate 0.6, which is the probability of *sql* being breached, leading to *Completed* state or action *failed* at rate 0.4, which is the result of (1-the probability of *sql* to be breached) leading back to *SystemAttackerA*. *Completed* is the state that represents the system is successfully attacked. In the *Completed* state, the attacker performs action *complete* at rate 1 returning back to *Start* state so that the model becomes cyclic and steady-state measures can be obtained.

System equation

Start

This is the system equation. The component is initially in the first state.

5.3.3 Performance evaluation

Path analysis

There are six possible paths that the attacker can follow to compromise the system [69]. We are interested in estimating the time to compromise which is the time that the attacker takes to compromise the system. The time to compromise is calculated for each path in the attack graph. Therefore, we perform the passage-time analysis for each path. In this model, when the attacker fails to exploit the vulnerability of any node, the attacker goes back to the first node in the attack graph.

The following graph, Figure 5.6, shows the passage-time analysis for each path in the attack graph from the first vulnerability action in a path to *compromised* action. If the attacker fails to exploit the vulnerability, it returns to *SystemAttackerA* state.

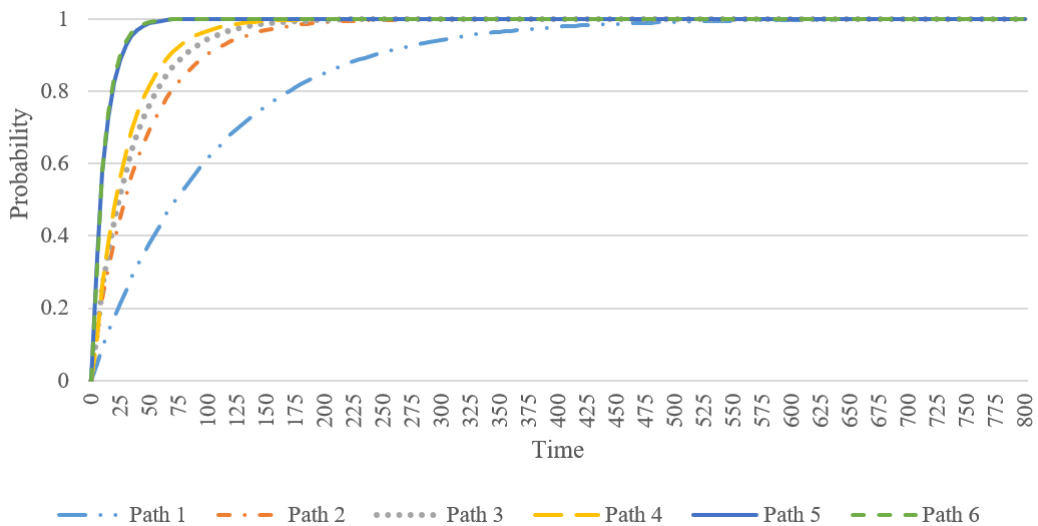


Fig. 5.6 Passage-time analysis for each path in the attack graph.

Table 5.3 Attack Paths and probabilities, Sun *et al.* [69].

Path number	Starting node	Target node	The path	Attack probability
1	A	H3	A→H1→H4→H3	0.0099
2	A	H3	A→H1→H3	0.02376
3	A	H3	A→H1→H5→H3	0.032076
4	A	H3	A→H2→H4→H3	0.0402
5	A	H3	A→H2→H3	0.09684
6	A	H3	A→H2→H5→H3	0.130248

Table 5.3 is taken from Sun *et al.* [69]. It shows the attack path and the attack probability of each path. The lowest attack probability is for path 1, whereas the highest attack probability is for path 6. Figure 5.6 shows our evaluation results which are the time until the system is compromised for each path in the attack graph. The fastest path is path 6, which has the highest attack probability, as in Table 5.3. The slowest path is path 1, which has the lowest attack probability, as in Table 5.3.

Table 5.4 The path order from our results and Sun *et al.* [69].

The fastest path form PEPA model	The highest attack probability from [69]
Path 6	Path 6
Path 5	Path 5
Path 4	Path 4
Path 3	Path 3
Path 2	Path 2
Path 1	Path 1
The slowest path	The lowest attack probability

Table 5.4 presents the fastest to slowest paths from our PEPA model evaluation result and the highest to lowest attack probabilities from Sun *et al.*'s work. Our evaluation result has the same path order as in Sun *et al.*'s work. It shows clearly the most threatening path to the lowest threatening path in the attack graph. Moreover, It shows the safe time the system has before it is compromised.

Moreover, to attack the system, the attacker first needs to exploit either *servU5* vulnerability in H1 or *telnet* vulnerability in H2. The following figure, Figure 5.7, shows when the attacker chooses to exploit *telnet* first, the attacker's time to compromise the system is less than the time when the attacker starts to exploit *servU5*.

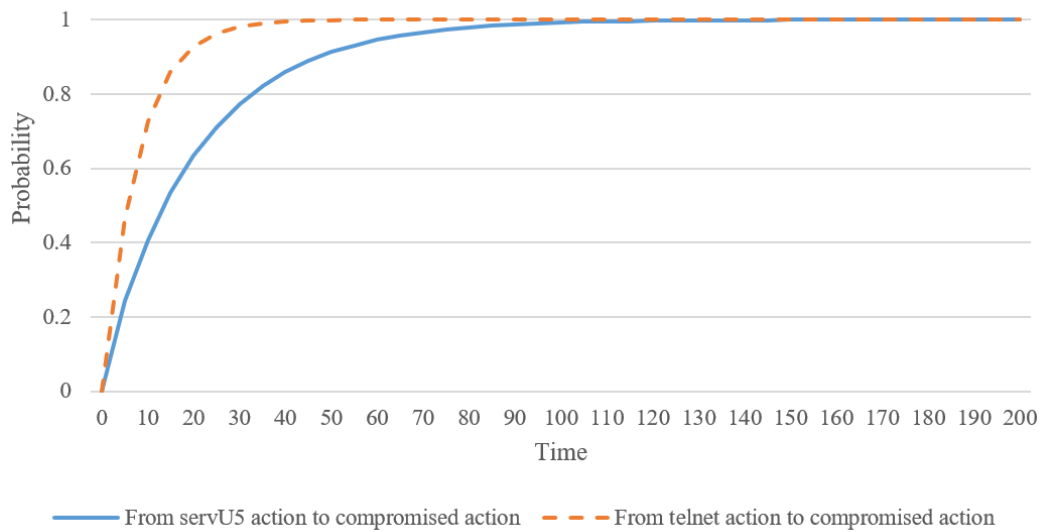


Fig. 5.7 Passage-time analysis for exploiting *servU5* and *telnet* to compromised action.

Sensitivity analysis

We are also interested in performing sensitivity analysis on the proposed PEPA model to evaluate the sensitivity of each path that the attacker can follow to compromise the system. We are going to change the probability of the vulnerability to be breached of *telnet* action and then perform the passage-time analysis for each path to see how sensitive each path is and how the time to compromise for each path will be affected. We change the *telnet* action rate to 0.5 and then to 0.1 and keep all other actions unchanged. Figures 5.8 and 5.9 show when we reduce the probability of *telnet* to be breached, the time takes the attacker to successfully reach its final goal increases for path 6, 5 and 4.

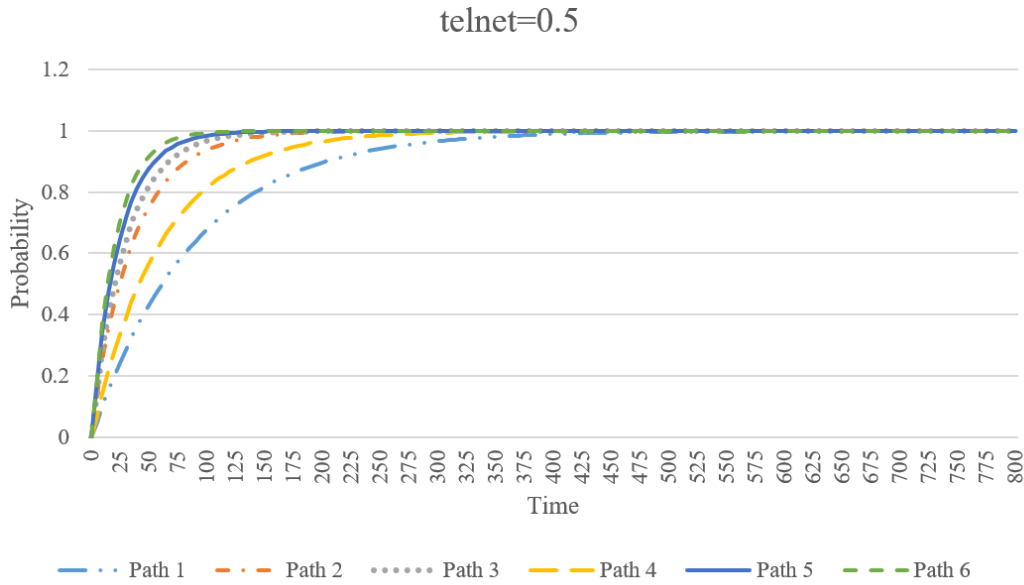


Fig. 5.8 Passage-time analysis for all paths when *telnet* action rate=0.5.

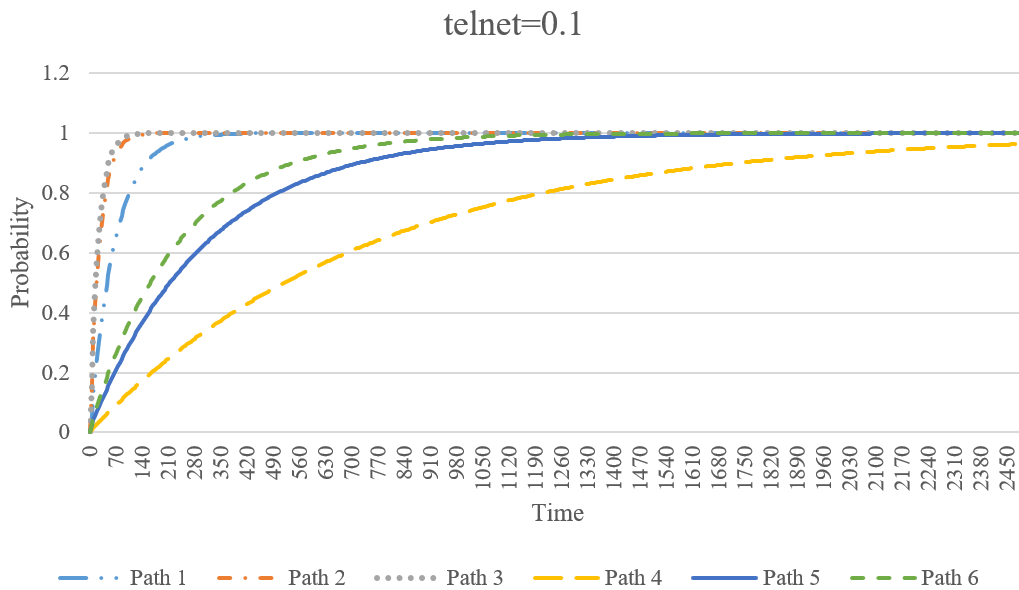


Fig. 5.9 Passage-time analysis for all paths when *telnet* action rate=0.1.

Following the same way of Sun *et al.* [69] to calculate the attack probability of each path, the new attack probability after changing the value of *telnet* is presented in Table 5.5. In Table 5.5, the attack probability of each path when the probability of *telnet* to be breached is 0.8 is taken from [69]. The attack probability of each path when the probability of *telnet* to be breached is changed to 0.5 and 0.1 are calculated by following the same approach that

Sun *et al.* follow. As you can see from Table 5.5, there is a significant decrease in the attack probability of path 4, 5 and 6 when the probability of *telnet* to be breached is reduced. This is shown in our evaluation results, Figures 5.8 and 5.9, as an increase in the attacker's time to compromise. The attack probability of path 1, 2 and 3 slightly increased.

Table 5.5 The attack probability of each path when the probability of *telnet* to be breach is equal to 0.8, 0.5 and 0.1.

Attack path	telnet=0.8	telnet=0.5	telnet=0.1
Path 1	0.0099	0.0132	0.024
Path 2	0.02376	0.03168	0.0576
Path 3	0.032076	0.042768	0.07776
Path 4	0.0402	0.020625	0.0015
Path 5	0.09684	0.0495	0.0036
Path 6	0.130248	0.066825	0.00486

In Figure 5.8, when *telnet* is 0.5, the fastest path is path 6, which has the highest attack probability. Whereas the slowest path is path 1, which has the lowest attack probability as in Table 5.5. However, path 6 in Figure 5.6 when *telnet* is 0.8 is faster than path 6 in Figure 5.8. When the attacker follows path 6 to compromise the system, the attacker's time to compromise is 155 time units when *telnet* is 0.8 and 290 time units when *telnet* is 0.5. In Figure 5.9, when *telnet* is reduced to 0.1, the fastest path becomes path 3, which has the highest attack probability. Whereas the slowest path is path 4, which has the lowest attack probability as it is shown in Table 5.5. That shows that our evaluation result reflects the change in the probability of *telnet* to be breached.

Furthermore, to compromise the system, the attacker has to first attack either H1 by exploiting *servU5* or H2 by exploiting *telnet*. We change the *telnet* action rate to 0.9, 0.5 and 0.1 and keep all other actions unchanged to see the impact of the change on the attacker's time to compromise.

Changing the probability of *telnet* to be breached could significantly impact the attacker's time to compromise the system. The probability could be reduced by implementing some protecting tools or security measures in host H2 to make it difficult for the attacker to exploit the vulnerability. The following figures, Figure 5.10, 5.11 and 5.12, illustrate how reducing the probability rate of *telnet* action to be breached causes the attacker's time to compromise the system to increase when the attacker attacks the system by exploiting *telnet*.

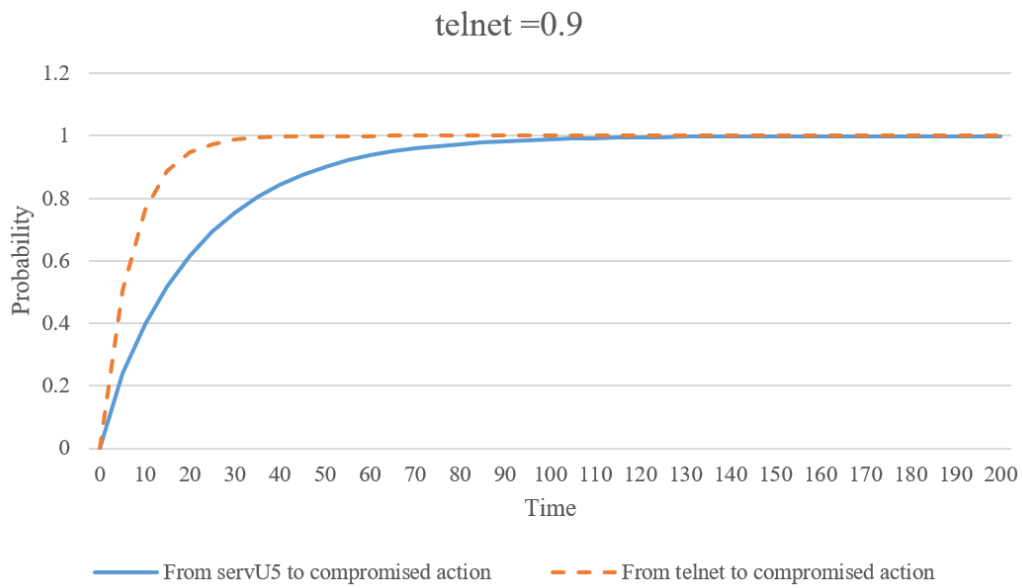


Fig. 5.10 Passage-time analysis when *telnet* action rate =0.9.

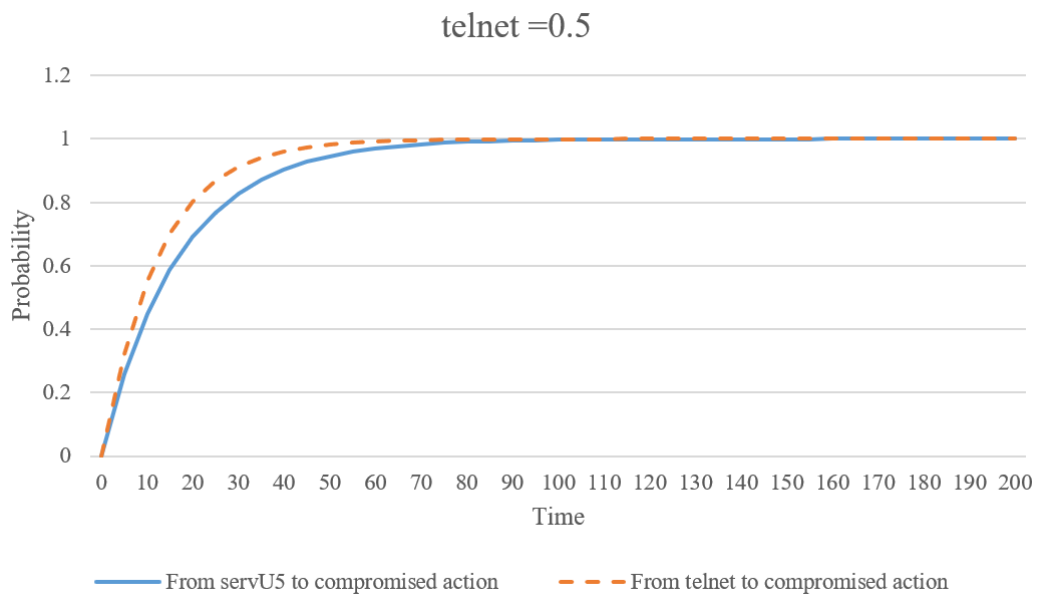


Fig. 5.11 Passage-time analysis when *telnet* action rate =0.5.

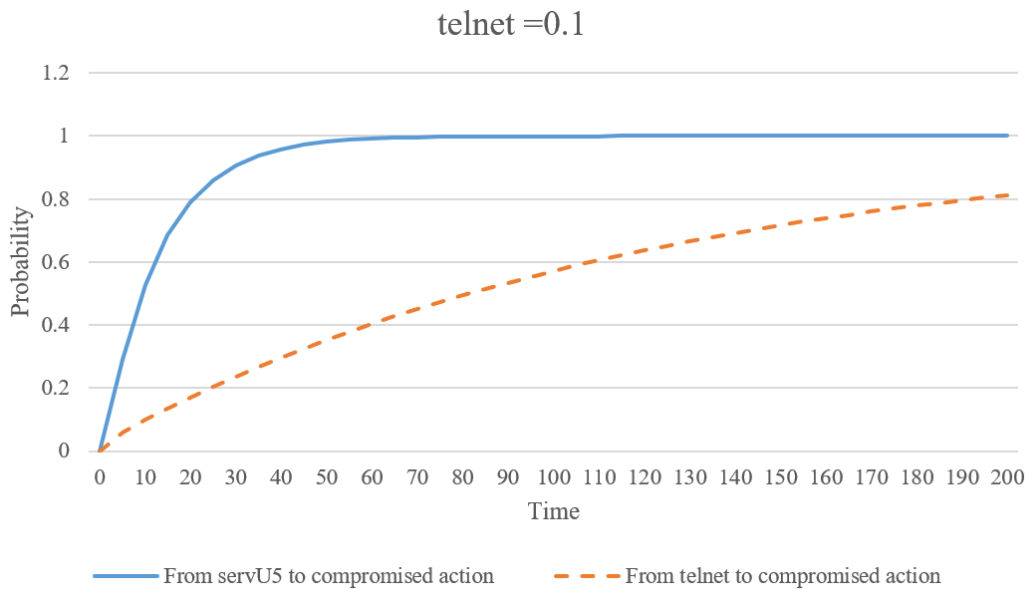


Fig. 5.12 Passage-time analysis when *telnet* action rate = 0.1.

5.3.4 Alternative PEPA model

In this alternative model, when the attacker fails to exploit the host's vulnerability, he returns to the previous state instead of returning to the state of the first node in the attack graph as suggested in the previous PEPA model, shown in Section 5.3.2. This could be a possible consequence. For clarity, we model the attack graph path separately. We could model them as a single model, but this would make interpretation more challenging. The following are the PEPA models for each path in the attack graph.

Path 1

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerA} &= (\textit{servU5}, 0.33).\textit{SystemAttackerH1}; \\
 \textit{SystemAttackerH1} &= (\textit{rpc}, 0.1).\textit{SystemAttackerH4} \\
 &\quad + (\textit{failed}, 0.6).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerH3} &= (\textit{compromised}, 0.6).\textit{Completed} \\
 &\quad + (\textit{failed}, 0.4).\textit{SystemAttackerH4}; \\
 \textit{Completed} &= (\textit{completed}, 1).\textit{Start}; \\
 \textit{SystemAttackerH4} &= (\textit{sql}, 0.5).\textit{SystemAttackerH3} \\
 &\quad + (\textit{failed}, 0.5).\textit{SystemAttackerH1};
 \end{aligned}$$

Path 2

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerA} &= (\textit{servU5}, 0.33).\textit{SystemAttackerH1}; \\
 \textit{SystemAttackerH1} &= (\textit{sql}, 0.12).\textit{SystemAttackerH3} \\
 &\quad + (\textit{failed}, 0.6).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerH3} &= (\textit{compromised}, 0.6).\textit{Completed} \\
 &\quad + (\textit{failed}, 0.4).\textit{SystemAttackerH1}; \\
 \textit{Completed} &= (\textit{completed}, 1).\textit{Start};
 \end{aligned}$$
Path 3

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerA} &= (\textit{servU5}, 0.33).\textit{SystemAttackerH1}; \\
 \textit{SystemAttackerH1} &= (\textit{remoteLogin}, 0.18).\textit{SystemAttackerH5} \\
 &\quad + (\textit{failed}, 0.6).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerH3} &= (\textit{compromised}, 0.6).\textit{Completed} \\
 &\quad + (\textit{failed}, 0.4).\textit{SystemAttackerH5}; \\
 \textit{Completed} &= (\textit{completed}, 1).\textit{Start}; \\
 \textit{SystemAttackerH5} &= (\textit{sql}, 0.9).\textit{SystemAttackerH3} \\
 &\quad + (\textit{failed}, 0.1).\textit{SystemAttackerH1};
 \end{aligned}$$
Path 4

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerA} &= (\textit{telnet}, 0.67).\textit{SystemAttackerH2}; \\
 \textit{SystemAttackerH2} &= (\textit{rpc}, 0.2).\textit{SystemAttackerH4} \\
 &\quad + (\textit{failed}, 0.2).\textit{SystemAttackerA}; \\
 \textit{SystemAttackerH3} &= (\textit{compromised}, 0.6).\textit{Completed} \\
 &\quad + (\textit{failed}, 0.4).\textit{SystemAttackerH4}; \\
 \textit{Completed} &= (\textit{completed}, 1).\textit{Start}; \\
 \textit{SystemAttackerH4} &= (\textit{sql}, 0.5).\textit{SystemAttackerH3} \\
 &\quad + (\textit{failed}, 0.5).\textit{SystemAttackerH2};
 \end{aligned}$$

Path 5

$$\begin{aligned}
Start &= (start, 1).SystemAttackerA; \\
SystemAttackerA &= (telnet, 0.67).SystemAttackerH2; \\
SystemAttackerH2 &= (sql, 0.24).SystemAttackerH3 \\
&+ (failed, 0.2).SystemAttackerA; \\
SystemAttackerH3 &= (compromised, 0.6).Completed \\
&+ (failed, 0.4).SystemAttackerH2; \\
Completed &= (completed, 1).Start;
\end{aligned}$$

Path 6

$$\begin{aligned}
Start &= (start, 1).SystemAttackerA; \\
SystemAttackerA &= (telnet, 0.67).SystemAttackerH2; \\
SystemAttackerH2 &= (remoteLogin, 0.36).SystemAttackerH5 \\
&+ (failed, 0.2).SystemAttackerA; \\
SystemAttackerH3 &= (compromised, 0.6).Completed \\
&+ (failed, 0.4).SystemAttackerH5; \\
Completed &= (completed, 1).Start; \\
SystemAttackerH5 &= (sql, 0.9).SystemAttackerH3 \\
&+ (failed, 0.1).SystemAttackerH2;
\end{aligned}$$

5.3.5 Performance evaluation of alternative PEPA model

Now, we perform the passage-time analysis for each path in the alternative PEPA models of the attack graph to calculate the attacker's time to compromise from the first vulnerability action reflecting the vulnerability of the first host in the attack graph to *compromised* action, which reflects that the attacker reaches its goal.

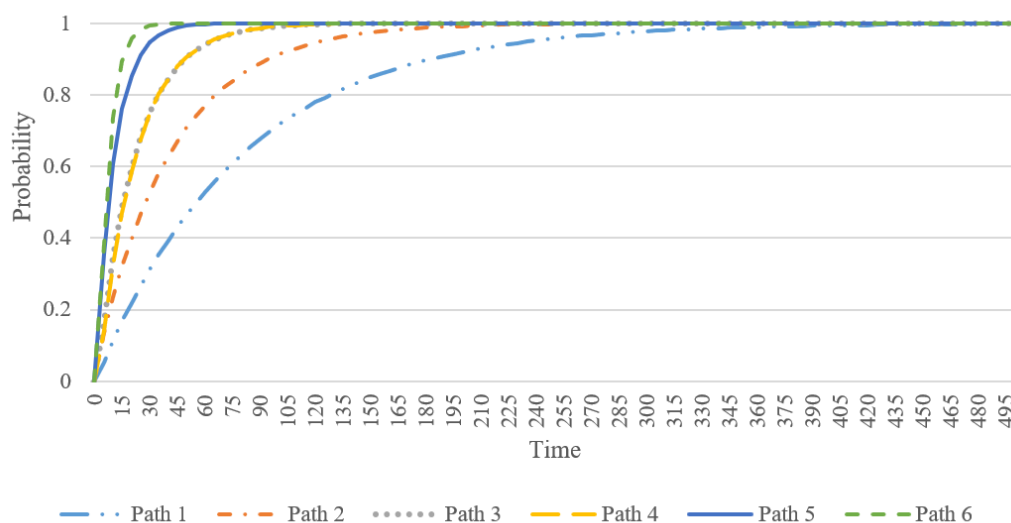


Fig. 5.13 Passage-time analysis for each path in the alternative PEPA models.

Figure 5.13 shows that the evaluation has the same path order as in previous PEPA model, Figure 5.6. However, the attacker's time to compromise the system for each path when the attacker returns to the previous state after it fails to exploit a vulnerability in one host is less than the times when the attacker returns to the first node after failure as in the previous PEPA model, Figure 5.6. In Figure 5.13, the time to compromise for path 1, path 2, path 3, path 4, path 5 and path 6 are 1150, 610, 305, 300, 150 and 85 time units, respectively. Whereas in Figure 5.6, the attacker's time to compromise for path 1, path 2, path 3, path 4, path 5 and path 6 are 1530, 635, 510, 425, 165 and 155 time units, respectively.

5.4 Case study 2: Multiple component PEPA model for attack graph

5.4.1 System specification

Now, we introduce an attacker component as a separate sequential component to our PEPA model. We assign some steps/abilities to the attacker. The proposed steps/abilities that the attacker follows to support compromising the system are presented in the following graph, Figure 5.14. In this case study, the attack graph is the same as the attack graph in Section 5.3.1.

In Figure 5.14, the attacker first recognises the vulnerability, then he can either search for existing suitable exploit code or write an exploit code. If the attacker finds a suitable exploit

code, he runs it in order to exploit the vulnerability. If the attacker does not find suitable code, he goes back to search again. If the attacker writes an exploit code, he goes to run action to exploit the vulnerability. Each node in the attacker abilities/steps graph is an action with a rate in our model.

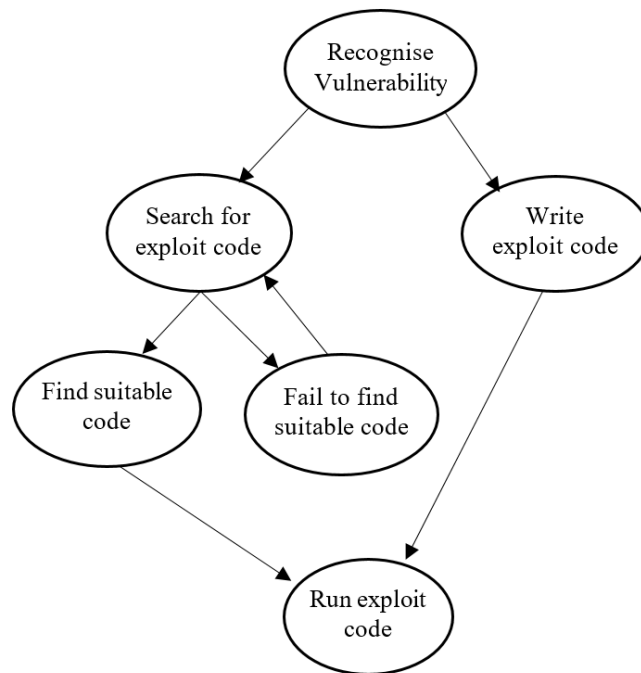


Fig. 5.14 The attacker's abilities/steps to compromise the system.

5.4.2 PEPA model

The following is a valid PEPA model that is generated by implementing and executing Algorithm 2 for the system component, Algorithm 3 for the attacker component, and Algorithm 4 for the system equation using Java. In this PEPA model, there are two types of components: system and attacker. The model comprises of two parts, one for each component. The system and the attacker move sequentially from their different behaviours based on the activities specified in the model. The model is formulated as follows:

A system component

$$\begin{aligned}
Start &= (start, 1).SystemA; \\
SystemA &= (servU5, 0.33).SystemH1 + (telnet, 0.67).SystemH2; \\
SystemH1 &= (sql, 0.12).SystemH3 + (rpc, 0.1).SystemH4 \\
&+ (remoteLogin, 0.18).SystemH5 \\
&+ (failed, 0.6).SystemH1; \\
SystemH2 &= (sql, 0.24).SystemH3 + (rpc, 0.2).SystemH4 \\
&+ (remoteLogin, 0.36).SystemH5 \\
&+ (failed, 0.2).SystemH2; \\
SystemH3 &= (compromised, 0.6).Completed + (failed, 0.4).SystemH3; \\
Completed &= (completed, 1).Start; \\
SystemH4 &= (sql, 0.5).SystemH3 + (failed, 0.5).SystemH4; \\
SystemH5 &= (sql, 0.9).SystemH3 + (failed, 0.1).SystemH5;
\end{aligned}$$

The above model component specifies the System's different behaviours, moving from *Start* to *Completed*. The description of this component is similar to the description of the component in Section 5.3.2, except that when a *failed* action happens, the system remains in the same state.

Attacker component

$$\begin{aligned}
AttackerStart &= (start, 1).AttackerA; \\
AttackerA &= (servU5, 0.33).Attacker0H1 + (telnet, 0.67).Attacker0H2; \\
Attacker0H1 &= (recogniseVuln, 1).Attacker1H1 \\
&+ (recogniseVuln, 1).Attacker5H1; \\
Attacker1H1 &= (searchCode, 1).Attacker2H1 + (searchCode, 1).Attacker3H1; \\
Attacker2H1 &= (findSuitableCode, 1).Attacker4H1; \\
Attacker3H1 &= (failToFindSuitableCode, 1).Attacker1H1; \\
Attacker4H1 &= (runExploitCode, 1).AttackerH1; \\
Attacker5H1 &= (writeExploitCode, 1).Attacker4H1; \\
AttackerH1 &= (sql, 0.12).Attacker0H3 + (rpc, 0.1).Attacker0H4 \\
&+ (remoteLogin, 0.18).Attacker0H5 \\
&+ (failed, 0.6).Attacker0H1; \\
Attacker0H2 &= (recogniseVuln, 1).Attacker1H2 \\
&+ (recogniseVuln, 1).Attacker5H2; \\
Attacker1H2 &= (searchCode, 1).Attacker2H2 + (searchCode, 1).Attacker3H2; \\
Attacker2H2 &= (findSuitableCode, 1).Attacker4H2;
\end{aligned}$$

$$\begin{aligned}
Attacker3H2 &= (failToFindSuitableCode, 1).Attacker1H2; \\
Attacker4H2 &= (runExploitCode, 1).AttackerH2; \\
Attacker5H2 &= (writeExploitCode, 1).Attacker4H2; \\
AttackerH2 &= (sql, 0.24).Attacker0H3 + (rpc, 0.2).Attacker0H4 \\
&\quad + (remoteLogin, 0.36).Attacker0H5 \\
&\quad + (failed, 0.2).Attacker0H2; \\
Attacker0H3 &= (recogniseVuln, 1).Attacker1H3 \\
&\quad + (recogniseVuln, 1).Attacker5H3; \\
Attacker1H3 &= (searchCode, 1).Attacker2H3 + (searchCode, 1).Attacker3H3; \\
Attacker2H3 &= (findSuitableCode, 1).Attacker4H3; \\
Attacker3H3 &= (failToFindSuitableCode, 1).Attacker1H3; \\
Attacker4H3 &= (runExploitCode, 1).AttackerH3; \\
Attacker5H3 &= (writeExploitCode, 1).Attacker4H3; \\
AttackerH3 &= (compromised, 0.6).AttackerCompleted \\
&\quad + (failed, 0.4).Attacker0H3; \\
AttackerCompleted &= (completed, 1).AttackerStart; \\
Attacker0H4 &= (recogniseVuln, 1).Attacker1H4 \\
&\quad + (recogniseVuln, 1).Attacker5H4; \\
Attacker1H4 &= (searchCode, 1).Attacker2H4 + (searchCode, 1).Attacker3H4; \\
Attacker2H4 &= (findSuitableCode, 1).Attacker4H4; \\
Attacker3H4 &= (failToFindSuitableCode, 1).Attacker1H4; \\
Attacker4H4 &= (runExploitCode, 1).AttackerH4; \\
Attacker5H4 &= (writeExploitCode, 1).Attacker4H4; \\
AttackerH4 &= (sql, 0.5).Attacker0H3 + (failed, 0.5).Attacker0H4; \\
Attacker0H5 &= (recogniseVuln, 1).Attacker1H5 \\
&\quad + (recogniseVuln, 1).Attacker5H5; \\
Attacker1H5 &= (searchCode, 1).Attacker2H5 + (searchCode, 1).Attacker3H5; \\
Attacker2H5 &= (findSuitableCode, 1).Attacker4H5; \\
Attacker3H5 &= (failToFindSuitableCode, 1).Attacker1H5; \\
Attacker4H5 &= (runExploitCode, 1).AttackerH5; \\
Attacker5H5 &= (writeExploitCode, 1).Attacker4H5; \\
AttackerH5 &= (sql, 0.9).Attacker0H3 + (failed, 0.1).Attacker0H5;
\end{aligned}$$

The above part of the model represents the attacker's different behaviours, moving from *AttackerStart* to *AttackerCompleted* to compromise the system. The attacker first performs

action *start* at rate 1 leading to *AttackerA* which is the first node in the attack graph. Then, the attacker can perform either *servU5* at rate 0.33 leading to *Attacker0H1* or *telnet* at rate 0.67 leading to *Attacker0H2*. In state *Attacker0H1*, one of two actions could happen either *recogniseVuln* at rate 1 leading to *Attacker1H1* or *recogniseVuln* at rate 1 leading to *Attacker5H1*. This reflects the first node in the attacker ability/steps graph, as shown in Figure 5.14.

When the attacker is in the state *Attacker1H1*, he can perform either action *searchCode* at rate 1 leading to *Attacker2H1* or action *searchCode* at rate 1 leading to *Attacker3H1*. *Attacker1H1* represents when an attacker is in the ‘search for exploit code’ node in the attacker’s ability/steps graph. In state *Attacker2H1*, the only action could happen is *findSuitableCode* at rate 1 leading to *Attacker4H1*. *Attacker2H1* reflects ‘Find suitable code’ node in the attacker’s ability/steps graph.

When the attacker is in the state *Attacker4H1*, he can perform *runExploitCode* at rate 1 leading to *AttackerH1* which means the attacker is in the H1 node trying to exploit the vulnerability to be able to move to the next host in the attack graph. In state *Attacker3H1*, the attacker performs *failToFindCode* action at rate 1 leading it back to state *Attacker1H1*. In state *Attacker5H1*, the only action could happen is *writeExploitCode* at rate 1 leading to state *Attacker4H1*.

In state *AttackerH1*, one of four actions could be performed either *failed* action at rate 0.6 leading back to *Attacker0H1* which is the first state of the attacker’s ability/steps graph, *sql* action at rate 0.12 leading to *Attacker0H3*, *rpc* action at rate 0.1 leading to *Attacker0H4* or *remoteLogin* action at rate 0.18 leading to *Attacker0H5*. State *Attacker0H3*, *Attacker0H4* and *Attacker0H5* reflect when the attacker is in the first ability’s node in the attacker’s ability/steps graph trying to attack H3, H4 and H5 host in the attack graph, respectively. Then moving from state to state repeats until reaching the last state which is state *AttackerCompleted*.

System equation

The system equation and complete specification are given by

$$\begin{aligned} \text{AttackerStart} < \text{start}, \text{failed}, \text{compromised}, \text{completed}, \\ \text{servU5}, \text{telnet}, \text{sql}, \text{rpc}, \text{remoteLogin} > \text{Start} \end{aligned}$$

The two components are initially in state *AttackerStart* and *Start*. The actions between them are shared actions between the two components.

5.4.3 Performance evaluation

We are again interested in calculating the time to compromise a system for each path that attacker can follow on the attack graph. There are six possible paths that the attacker can follow to compromise the system [69]. We performed the passage-time analysis for each path in order to estimate the time that the attacker takes to compromise the system. The action rate for each attacker' abilities/steps is 1.

The following graph, Figure 5.15, shows the passage-time analysis for each path in the attack graph that the attacker can follow to reach its final target. The passage-time analysis is calculated from the first vulnerability action in a path to *completed* action. If the attacker fails to exploit the vulnerability, it returns to the first step in the attacker's ability/steps graph in order to try again to find way to exploit the vulnerability in the host. Figure 5.15 shows that the fastest path is path 6, which has the highest attack probability as in [69] and the slowest path is path 1, which has the lowest attack probability as in [69]. Our evaluation result has the same path order as in Sun *et al.* [69]. The evaluation results clearly illustrate the most threatening attack paths, the least threatening attack path, and the time it takes the attacker to compromise the system for each path in the attack graph. The time to compromise for each attack path that is resulted from evaluating the model can rank the risk of all attack paths. It can also provide a safe time that the system has before the attacker successfully attacks the system/network.

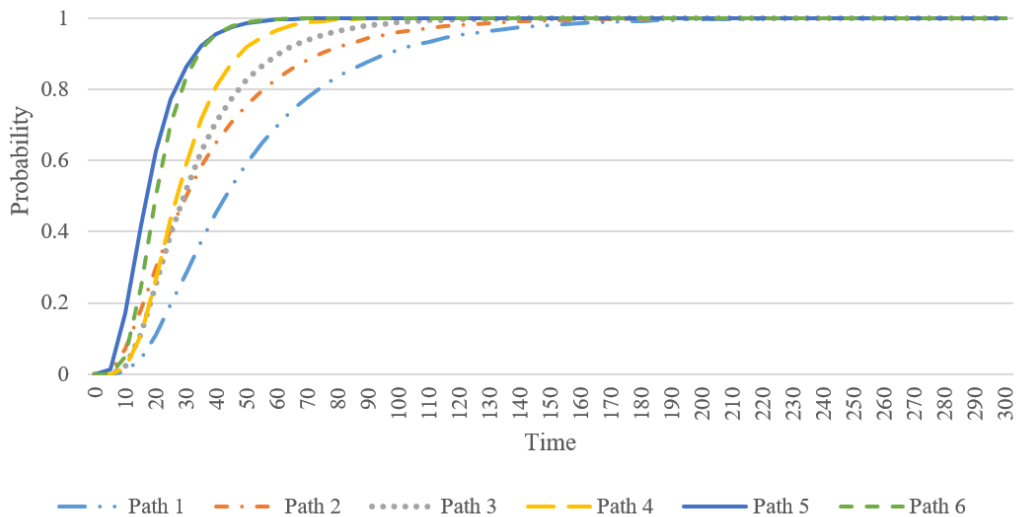


Fig. 5.15 Passage-time analysis of each path in the attack graph.

5.5 Conclusion

In this chapter, we provided two methods to automate the generation of PEPA model based on a pre-existing attack graph specification. The first method is more straightforward. It generates a PEPA model that comprises one sequential component and system equation. This component represents the system and attacker coupled together. The second method has greater potential. It generates a PEPA model that comprises two sequential components and the system equation. One component represents a system or network, and the other component represents an attacker. The attacker component enables us to explicitly implement different attacker skills. Furthermore, when we developed the PEPA models for the attack graph, we considered the alternative consequences that might happen when an attacker fails to exploit the vulnerabilities. When an attacker fails, he may return to the initial node, return to the previous node, or remain at the current node.

Furthermore, we demonstrated through the case study how we used the generated PEPA models to deduce the most and the least system security threatening paths and time to compromise, which can be used by the defender as an indicator of how much a safe time the system has before it is compromised. In addition, the time it takes to compromise a system for each attack path can be used to rank the risk of all attack paths.

Moreover, we performed a sensitivity analysis by changing the probability of the vulnerability being breached of some actions and then did the passage-time analysis for all paths. The evolution of the model clearly shows the sensitivity of each path to the change and the effect on the attacker's time to compromise the system of each path. This can help the defender to prioritise the countermeasures. This study used a PEPA Eclipse plug-in to support the evaluation of the PEPA model.

5.6 Chapter Summary

An attack graph is one of the popular graph-based methods that can support a defender in understanding the attacker's behaviour [46]. It also supports the defender to detect a possible threat and then work to defend the system or network. It shows the possible attack paths that an attacker can follow to exploit multiple vulnerabilities in a system to successfully reach its final goal [42, 46]. Therefore, building a PEPA model version of an attack graph could help to enhance the understanding of the attacker's behaviour and assist in identifying critical threats.

This chapter represents an initial study into using Performance Evaluation Process Algebra (PEPA) to model and analyse the attack graphs. Such an approach adds timing

information into the model and therefore extends the range of available analysis techniques. In this chapter, we proposed methods to generate PEPA model based on an existing attack graph. PEPA models can support the defender to enhance its understanding of the system's security current state. By using the PEPA models, the defender can deduce the most and the least threatening attack path in the system. The time takes the attacker to compromise the system (time to compromise) can also be estimated. The PEPA model is analysed via a continuous-time Markov chain with rates to estimate the time to compromise the system and the time it takes the attacker to get to a particular vulnerability on a system. Knowledge of time to compromise the system can also support prioritising implementing the countermeasures.

It is also important to consider critical factors such as the attackers' skills and the availability of the exploit code when developing a PEPA model version of the attack graph to help gain a better understanding of the attack and assist in identifying critical threats. In the next chapter, we extended our study by modelling three types of attackers and employing the probability of the exploit code availability in the proposed PEPA models. Also, we enhance the adaptability of our proposed PEPA models by incorporating learning behaviour for the attacker and defender and then show the impact on the attacker's time to compromise the system.

Chapter 6

Advanced models of attacker behaviour

6.1 Introduction

There are a number of factors that can be used to estimate how fast the attackers can compromise a system, such as the different attackers' capabilities [28, 50] and the availability of exploit code for a vulnerability [28]. Different types of attackers can attack and compromise a system in different ways. The attackers can be classified based on their capabilities into beginner, intermediate and expert [50]. The beginner attacker can attack a system by using existing exploit code, the intermediate attacker can use and/or modify existing exploit code and an expert attacker can use, modify and create exploit code [50].

Furthermore, the attacker's and defender's learning abilities can influence how long it takes the attacker to compromise the system. After the failed attack attempts, the attacker can learn more about the target system and gain knowledge from previous attack attempts [59]. The defender can also learn about the attack activity by monitoring the system and looking for any attack activity indicators such as system alerts and anomaly detection in a log file. The defender can then work to secure the system by implementing additional security measures that make it more difficult for the attacker to compromise the system.

In this chapter, we propose the attacker's steps/abilities graph for three different types of attackers: beginners, intermediates, and experts. Then, for each attacker, we generate the PEPA models using the second method proposed in Chapter 5. The time required until the attacker compromises the system is then estimated for each path using the vulnerabilities in the attack graph and taking into account the probability of exploit code availability and the attacker's skill. In addition, we include learning behaviours in the model for both the attacker and the defender. Then, we show how learning behaviours affect the attacker's time to compromise the system.

6.2 PEPA models of attacker skill

In this section, we introduce different components of the attacker to the attack graph PEPA model in Section 5.4. Each component represents a different attacker capability to compromise the system based on the attacker's skills. They are created based on Algorithm 3 in Chapter 5.

Moreover, to introduce an exploit code availability factor in our model, we introduce a rate of p , which represents the probability of an exploit code availability. The rate of *searchCode* actions in all attacker's states is modelled by having a probabilistic choice p to go to the state *Attacker2H1* when the suitable code is found or to the state *Attacker3H1* when the suitable exploit code is not found.

In the following, we used the same case study in Chapter 5 in Section 5.4. The system component is kept unchanged. It is the same as the system component in subsection 5.4.2. The attacker component is changed based on its skill and capabilities.

6.2.1 Beginner attacker component

A beginner attacker can only exploit a known vulnerability by utilising pre-existing exploit code. Figure 6.1 shows the proposed beginner attacker's ability/steps graph.

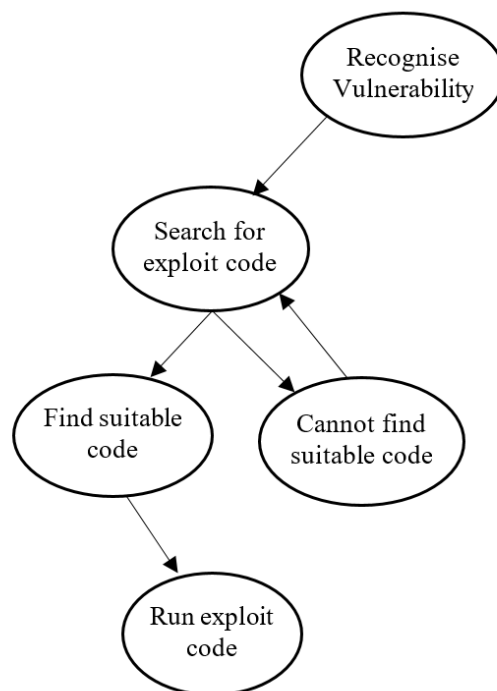


Fig. 6.1 The beginner attacker's capabilities to compromise the system.

The PEPA sequential component for the beginner attacker

This part of the model represents the beginner attacker's different behaviours, moving from *AttackerStart* to *AttackerCompleted* to compromise the system.

$$\begin{aligned}
 \textit{AttackerStart} &= (\textit{start}, 1).\textit{AttackerA}; \\
 \textit{AttackerA} &= (\textit{servU5}, 0.33).\textit{Attacker0H1} + (\textit{telnet}, 0.67).\textit{Attacker0H2}; \\
 \textit{Attacker0H1} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H1}; \\
 \textit{Attacker1H1} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H1} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H1}; \\
 \textit{Attacker2H1} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H1}; \\
 \textit{Attacker3H1} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H1}; \\
 \textit{Attacker4H1} &= (\textit{runExploitCode}, 1).\textit{AttackerH1}; \\
 \textit{AttackerH1} &= (\textit{sql}, 0.12).\textit{Attacker0H3} + (\textit{rpc}, 0.1).\textit{Attacker0H4} \\
 &\quad + (\textit{remoteLogin}, 0.18).\textit{Attacker0H5} \\
 &\quad + (\textit{failed}, 0.6).\textit{Attacker0H1}; \\
 \textit{Attacker0H2} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H2}; \\
 \textit{Attacker1H2} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H2} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H2}; \\
 \textit{Attacker2H2} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H2}; \\
 \textit{Attacker3H2} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H2}; \\
 \textit{Attacker4H2} &= (\textit{runExploitCode}, 1).\textit{AttackerH2}; \\
 \textit{AttackerH2} &= (\textit{sql}, 0.24).\textit{Attacker0H3} + (\textit{rpc}, 0.2).\textit{Attacker0H4} \\
 &\quad + (\textit{remoteLogin}, 0.36).\textit{Attacker0H5} \\
 &\quad + (\textit{failed}, 0.2).\textit{Attacker0H2}; \\
 \textit{Attacker0H3} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H3}; \\
 \textit{Attacker1H3} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H3} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H3}; \\
 \textit{Attacker2H3} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H3}; \\
 \textit{Attacker3H3} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H3}; \\
 \textit{Attacker4H3} &= (\textit{runExploitCode}, 1).\textit{AttackerH3}; \\
 \textit{AttackerH3} &= (\textit{compromised}, 0.6).\textit{AttackerCompleted} \\
 &\quad + (\textit{failed}, 0.4).\textit{Attacker0H3}; \\
 \textit{AttackerCompleted} &= (\textit{completed}, 1).\textit{AttackerStart}; \\
 \textit{Attacker0H4} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H4}; \\
 \textit{Attacker1H4} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H4} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H4}; \\
 \textit{Attacker2H4} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H4}; \\
 \textit{Attacker3H4} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H4};
 \end{aligned}$$

$$\begin{aligned}
 \textit{Attacker4H4} &= (\textit{runExploitCode}, 1).\textit{AttackerH4}; \\
 \textit{AttackerH4} &= (\textit{sql}, 0.5).\textit{Attacker0H3} + (\textit{failed}, 0.5).\textit{Attacker0H4}; \\
 \textit{Attacker0H5} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H5}; \\
 \textit{Attacker1H5} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H5} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H5}; \\
 \textit{Attacker2H5} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H5}; \\
 \textit{Attacker3H5} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H5}; \\
 \textit{Attacker4H5} &= (\textit{runExploitCode}, 1).\textit{AttackerH5}; \\
 \textit{AttackerH5} &= (\textit{sql}, 0.9).\textit{Attacker0H3} + (\textit{failed}, 0.1).\textit{Attacker0H5};
 \end{aligned}$$

6.2.2 Intermediate attacker component

The intermediate attacker has the capability of utilising and/or modifying pre-existing exploit code. Figure 6.2 shows the proposed intermediate attacker's ability/steps graph.

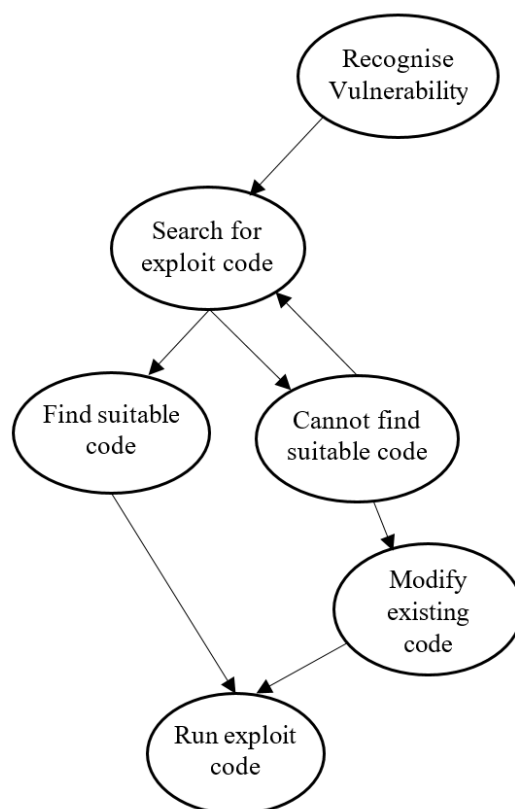


Fig. 6.2 The intermediate attacker's capabilities to compromise the system.

The PEPA sequential component for the intermediate attacker

This part of the model represents the intermediate attacker's different behaviours, moving from *AttackerStart* to *AttackerCompleted* to compromise the system.

$$\begin{aligned}
 \textit{AttackerStart} &= (\textit{start}, 1).\textit{AttackerA}; \\
 \textit{AttackerA} &= (\textit{servU5}, 0.33).\textit{Attacker0H1} + (\textit{telnet}, 0.67).\textit{Attacker0H2}; \\
 \textit{Attacker0H1} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H1}; \\
 \textit{Attacker1H1} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H1} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H1}; \\
 \textit{Attacker2H1} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H1}; \\
 \textit{Attacker3H1} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H1} \\
 &\quad + (\textit{failToFindSuitableCode}, 1).\textit{Attacker5H1}; \\
 \textit{Attacker4H1} &= (\textit{runExploitCode}, 1).\textit{AttackerH1}; \\
 \textit{Attacker5H1} &= (\textit{modifyExploitCode}, 1).\textit{Attacker4H1}; \\
 \textit{AttackerH1} &= (\textit{sql}, 0.12).\textit{Attacker0H3} + (\textit{rpc}, 0.1).\textit{Attacker0H4} \\
 &\quad + (\textit{remoteLogin}, 0.18).\textit{Attacker0H5} \\
 &\quad + (\textit{failed}, 0.6).\textit{Attacker0H1}; \\
 \textit{Attacker0H2} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H2}; \\
 \textit{Attacker1H2} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H2} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H2}; \\
 \textit{Attacker2H2} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H2}; \\
 \textit{Attacker3H2} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H2} \\
 &\quad + (\textit{failToFindSuitableCode}, 1).\textit{Attacker5H2}; \\
 \textit{Attacker4H2} &= (\textit{runExploitCode}, 1).\textit{AttackerH2}; \\
 \textit{Attacker5H2} &= (\textit{modifyExploitCode}, 1).\textit{Attacker4H2}; \\
 \textit{AttackerH2} &= (\textit{sql}, 0.24).\textit{Attacker0H3} + (\textit{rpc}, 0.2).\textit{Attacker0H4} \\
 &\quad + (\textit{remoteLogin}, 0.36).\textit{Attacker0H5} \\
 &\quad + (\textit{failed}, 0.2).\textit{Attacker0H2}; \\
 \textit{Attacker0H3} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H3}; \\
 \textit{Attacker1H3} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H3} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H3}; \\
 \textit{Attacker2H3} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H3}; \\
 \textit{Attacker3H3} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H3} \\
 &\quad + (\textit{failToFindSuitableCode}, 1).\textit{Attacker5H3};
 \end{aligned}$$

$$\begin{aligned}
Attacker4H3 &= (runExploitCode, 1).AttackerH3; \\
Attacker5H3 &= (modifyExploitCode, 1).Attacker4H3; \\
AttackerH3 &= (compromised, 0.6).AttackerCompleted \\
&+ (failed, 0.4).Attacker0H3; \\
AttackerCompleted &= (completed, 1).AttackerStart; \\
Attacker0H4 &= (recogniseVuln, 1).Attacker1H4; \\
Attacker1H4 &= (searchCode, 1 * p).Attacker2H4 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H4; \\
Attacker2H4 &= (findSuitableCode, 1).Attacker4H4; \\
Attacker3H4 &= (failToFindSuitableCode, 1).Attacker1H4 \\
&+ (failToFindSuitableCode, 1).Attacker5H4; \\
Attacker4H4 &= (runExploitCode, 1).AttackerH4; \\
Attacker5H4 &= (modifyExploitCode, 1).Attacker4H4; \\
AttackerH4 &= (sql, 0.5).Attacker0H3 + (failed, 0.5).Attacker0H4; \\
Attacker0H5 &= (recogniseVuln, 1).Attacker1H5; \\
Attacker1H5 &= (searchCode, 1 * p).Attacker2H5 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H5; \\
Attacker2H5 &= (findSuitableCode, 1).Attacker4H5; \\
Attacker3H5 &= (failToFindSuitableCode, 1).Attacker1H5 \\
&+ (failToFindSuitableCode, 1).Attacker5H5; \\
Attacker4H5 &= (runExploitCode, 1).AttackerH5; \\
Attacker5H5 &= (modifyExploitCode, 1).Attacker4H5; \\
AttackerH5 &= (sql, 0.9).Attacker0H3 + (failed, 0.1).Attacker0H5;
\end{aligned}$$

6.2.3 Expert attacker component

An expert attacker is able to use and/or modify a pre-existing exploit code or write a new exploit code. Figure 6.3 shows the proposed expert attacker's ability/steps graph.

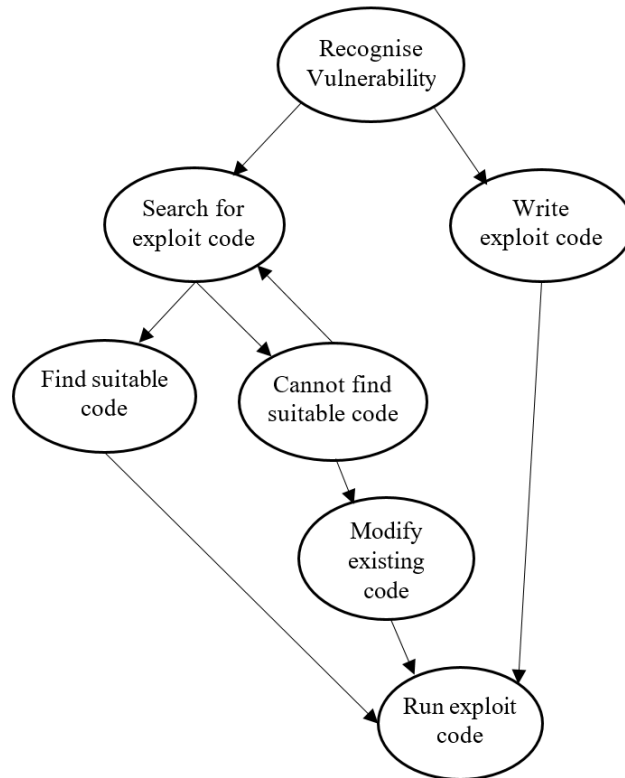


Fig. 6.3 The expert attacker's capabilities to compromise the system.

The PEPA sequential component for the expert attacker

This part of the model represents the expert attacker's different behaviours, moving from *AttackerStart* to *AttackerCompleted* to compromise the system.

$$\begin{aligned}
 \textit{AttackerStart} &= (\textit{start}, 1).\textit{AttackerA}; \\
 \textit{AttackerA} &= (\textit{servU5}, 0.33).\textit{Attacker0H1} + (\textit{telnet}, 0.67).\textit{Attacker0H2}; \\
 \textit{Attacker0H1} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H1} \\
 &\quad + (\textit{recogniseVuln}, 1).\textit{Attacker6H1}; \\
 \textit{Attacker1H1} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H1} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H1}; \\
 \textit{Attacker2H1} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H1}; \\
 \textit{Attacker3H1} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H1} \\
 &\quad + (\textit{failToFindSuitableCode}, 1).\textit{Attacker5H1}; \\
 \textit{Attacker4H1} &= (\textit{runExploitCode}, 1).\textit{AttackerH1}; \\
 \textit{Attacker5H1} &= (\textit{modifyExploitCode}, 1).\textit{Attacker4H1};
 \end{aligned}$$

Attacker6H1 = (*writeExploitCode*, 1).*Attacker4H1*;
AttackerH1 = (*sql*, 0.12).*Attacker0H3* + (*rpc*, 0.1).*Attacker0H4*
+ (*remoteLogin*, 0.18).*Attacker0H5*
+ (*failed*, 0.6).*Attacker0H1*;
Attacker0H2 = (*recogniseVuln*, 1).*Attacker1H2*
+ (*recogniseVuln*, 1).*Attacker6H2*;
Attacker1H2 = (*searchCode*, $1 * p$).*Attacker2H2*
+ (*searchCode*, $1 * (1 - p)$).*Attacker3H2*;
Attacker2H2 = (*findSuitableCode*, 1).*Attacker4H2*;
Attacker3H2 = (*failToFindSuitableCode*, 1).*Attacker1H2*
+ (*failToFindSuitableCode*, 1).*Attacker5H2*;
Attacker4H2 = (*runExploitCode*, 1).*AttackerH2*;
Attacker5H2 = (*modifyExploitCode*, 1).*Attacker4H2*;
Attacker6H2 = (*writeExploitCode*, 1).*Attacker4H2*;
AttackerH2 = (*sql*, 0.24).*Attacker0H3* + (*rpc*, 0.2).*Attacker0H4*
+ (*remoteLogin*, 0.36).*Attacker0H5*
+ (*failed*, 0.2).*Attacker0H2*;
Attacker0H3 = (*recogniseVuln*, 1).*Attacker1H3*
+ (*recogniseVuln*, 1).*Attacker6H3*;
Attacker1H3 = (*searchCode*, $1 * p$).*Attacker2H3*
+ (*searchCode*, $1 * (1 - p)$).*Attacker3H3*;
Attacker2H3 = (*findSuitableCode*, 1).*Attacker4H3*;
Attacker3H3 = (*failToFindSuitableCode*, 1).*Attacker1H3*
+ (*failToFindSuitableCode*, 1).*Attacker5H3*;
Attacker4H3 = (*runExploitCode*, 1).*AttackerH3*;
Attacker5H3 = (*modifyExploitCode*, 1).*Attacker4H3*;
Attacker6H3 = (*writeExploitCode*, 1).*Attacker4H3*;
AttackerH3 = (*compromised*, 0.6).*AttackerCompleted*
+ (*failed*, 0.4).*Attacker0H3*;
AttackerCompleted = (*completed*, 1).*AttackerStart*;
Attacker0H4 = (*recogniseVuln*, 1).*Attacker1H4*
+ (*recogniseVuln*, 1).*Attacker6H4*;
Attacker1H4 = (*searchCode*, $1 * p$).*Attacker2H4*
+ (*searchCode*, $1 * (1 - p)$).*Attacker3H4*;
Attacker2H4 = (*findSuitableCode*, 1).*Attacker4H4*;
Attacker3H4 = (*failToFindSuitableCode*, 1).*Attacker1H4*
+ (*failToFindSuitableCode*, 1).*Attacker5H4*;
Attacker4H4 = (*runExploitCode*, 1).*AttackerH4*;

$$\begin{aligned}
Attacker5H4 &= (modifyExploitCode, 1).Attacker4H4; \\
Attacker6H4 &= (writeExploitCode, 1).Attacker4H4; \\
AttackerH4 &= (sql, 0.5).Attacker0H3 + (failed, 0.5).Attacker0H4; \\
Attacker0H5 &= (recogniseVuln, 1).Attacker1H5 \\
&+ (recogniseVuln, 1).Attacker6H5; \\
Attacker1H5 &= (searchCode, 1 * p).Attacker2H5 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H5; \\
Attacker2H5 &= (findSuitableCode, 1).Attacker4H5; \\
Attacker3H5 &= (failToFindSuitableCode, 1).Attacker1H5 \\
&+ (failToFindSuitableCode, 1).Attacker5H5; \\
Attacker4H5 &= (runExploitCode, 1).AttackerH5; \\
Attacker5H5 &= (modifyExploitCode, 1).Attacker4H5; \\
Attacker6H5 &= (writeExploitCode, 1).Attacker4H5; \\
AttackerH5 &= (sql, 0.9).Attacker0H3 + (failed, 0.1).Attacker0H5;
\end{aligned}$$

6.3 Performance evaluation of alternative PEPA models

We are interested in estimating the time to compromise for each path that each attacker can follow. Thus, we evaluate the PEPA models by performing passage-time analysis. We set the probability of exploit code availability to 0.2 ($p = 0.2$) to show the impact of the lack of exploit code on each attacker. The time to compromise for each attack path in the attack graph for the different attackers when the probability of exploit code availability is 0.2 are shown in Figures 6.4, 6.5 and 6.6. The time to compromise of paths 1, 2, 3, 4, 5 and 6 for the expert attacker is less than paths 1, 2, 3, 4, 5 and 6 for other attackers, respectively. The fastest path in our PEPA model for the attack graph is path 6 which has the highest attack probability in [69]. The time to compromise the system for path 6 for the beginner attacker is 430 time units which is larger compared to the time to compromise the system for path 6 for the intermediate and the expert attackers. The time to compromise the system for path 6 for the intermediate and the expert attackers are 190 and 150 time units, respectively. Moreover, the figures clearly show the significant effect of lack of exploit code on the beginner attacker. The time to compromise for each path is larger for beginner attacker compared to each attack path for the other attackers.

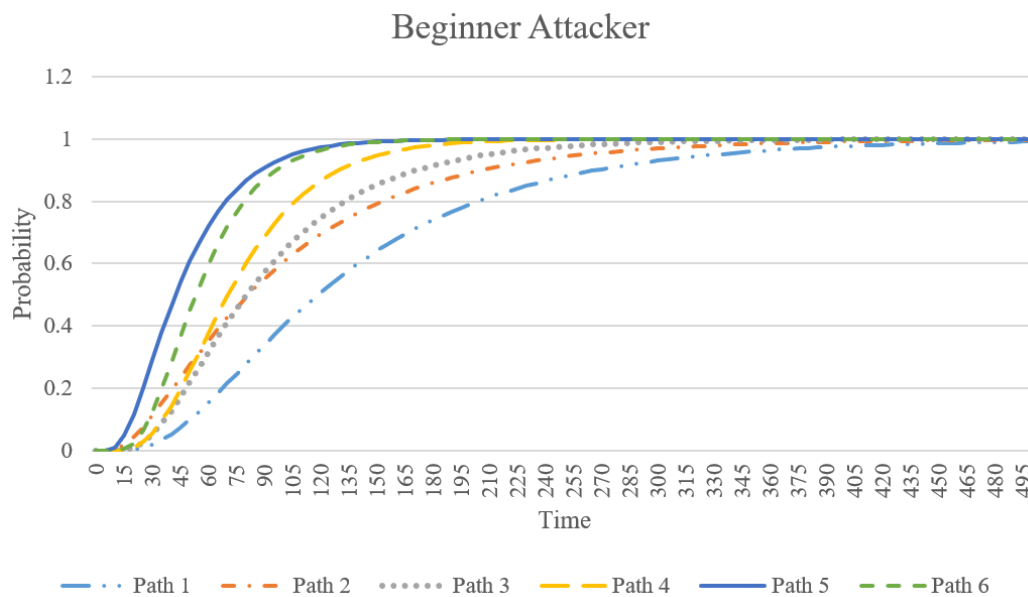


Fig. 6.4 Passage-time analysis of each path in the attack graph for the beginner attacker when the probability of exploit code availability is 0.2.

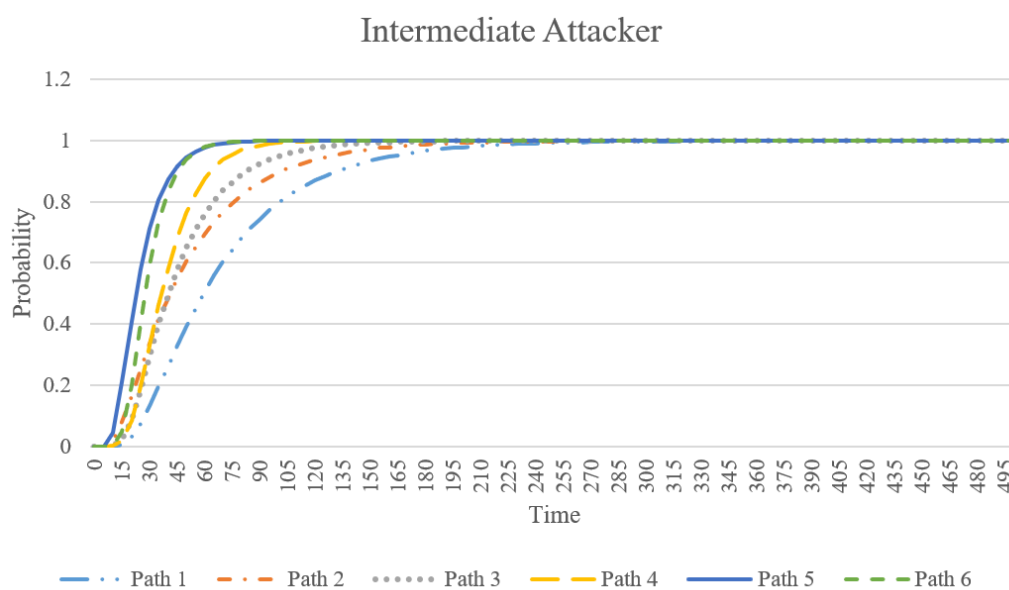


Fig. 6.5 Passage-time analysis of each path in the attack graph for the intermediate attacker when the probability of exploit code availability is 0.2.

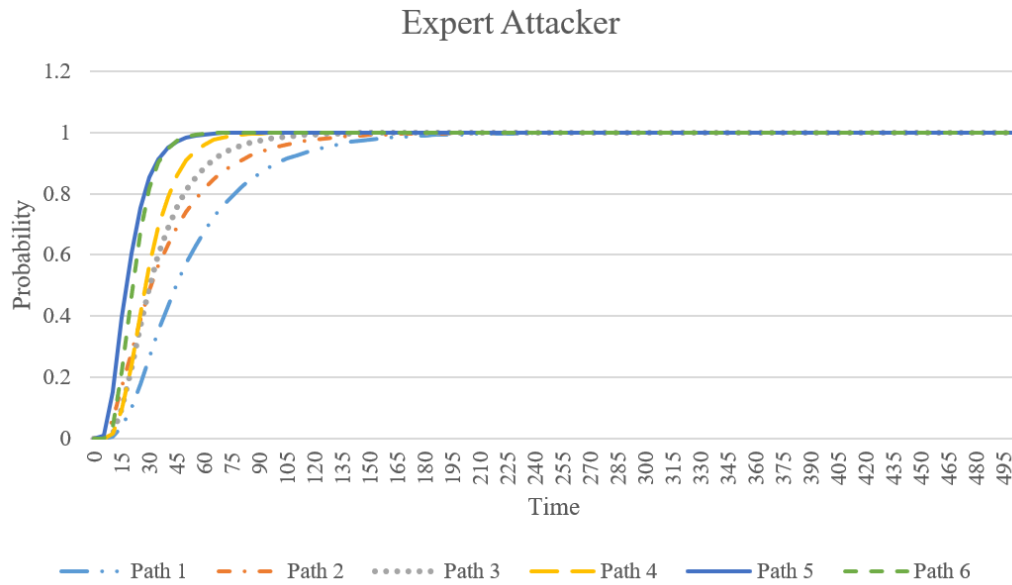


Fig. 6.6 Passage-time analysis of each path in the attack graph for the expert attacker when the probability of exploit code availability is 0.2.

Now we set the probability of exploit code availability p to 0.8 ($p = 0.8$). Figures 6.7, 6.8 and 6.9 illustrate the impact of the exploit code availability on the attacker's time to compromise of path 6 for each attacker based on two probability values of exploit code availability: when $p = 0.2$ and when $p = 0.8$. Path 6 is the fastest path and the most threatening path in our PEPA model for the attack graph. The impact of exploit code availability on the beginner attacker's time to compromise is significant as this attacker can only be capable of using an existing exploit code, Figure 6.7. Whereas, the expert attacker has minimal impact as this attacker is capable of modifying and creating exploit code when the suitable exploit code is not available, as shown in Figure 6.9.

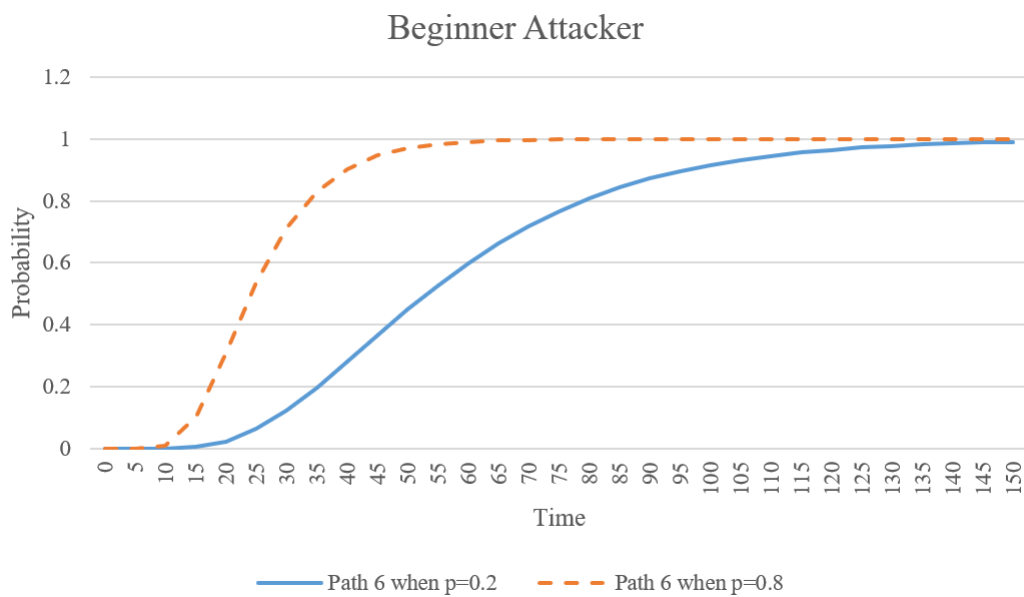


Fig. 6.7 Path 6 in the attack graph for beginner attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).

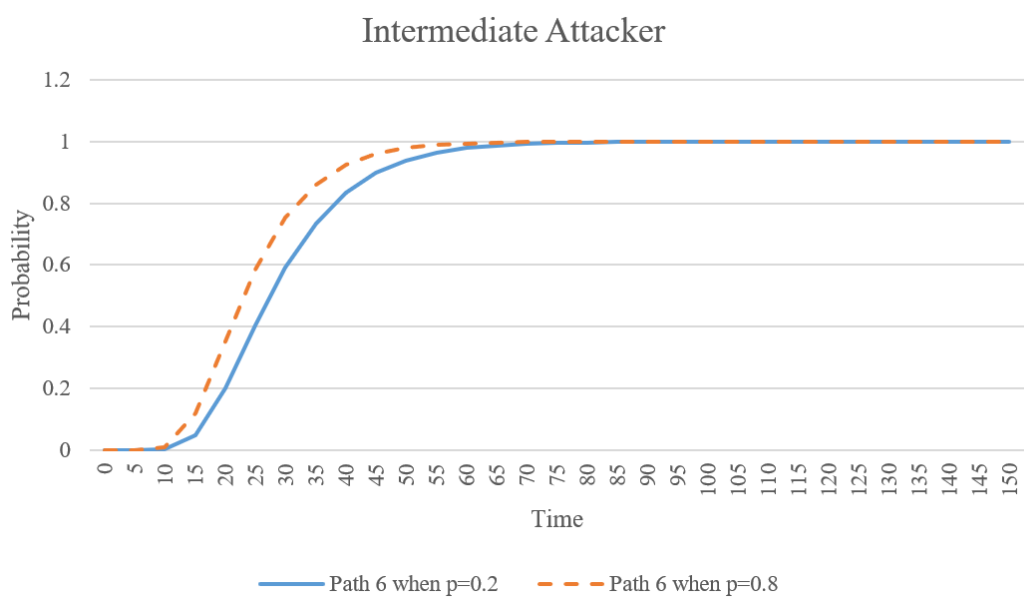


Fig. 6.8 Path 6 in the attack graph for intermediate attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).

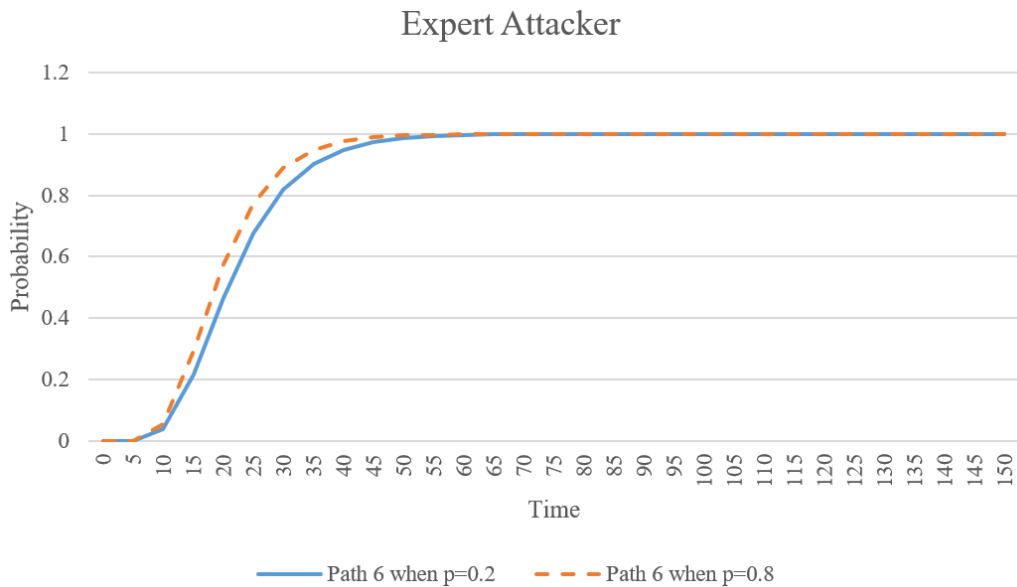


Fig. 6.9 Path 6 in the attack graph for expert attacker with two probabilities value of exploit code availability ($p=0.2$ and 0.8).

Moreover, as shown in the attack graph and our PEPA model for the attackers, the attacker first needs to exploit either *servU5* vulnerability in H1 or *telnet* vulnerability in H2 to start to attack the system. We perform passage-time analysis for beginner and expert attackers when they start by exploiting *servU5* in H1 and when they start by exploiting *telnet* in H2. Figures 6.10 and 6.11 show when the beginner and expert attackers choose to exploit *telnet* first, the attacker's time to compromise the system is less than the time when the attacker starts to exploit *servU5*. This is because the probability of *talnet* being breached is 0.8, whereas the probability of *servU5* to be breached is 0.4. It is much harder to exploit *servU5* than *talnet*. However, the figures show that the expert attacker is faster than beginner attacker. In Figures 6.10 and 6.11, the probability of exploit code availability for all vulnerabilities set to 0.2. Then, we set the probability of exploit code availability for all vulnerabilities to 0.8. Table 6.1 shows the significant impact of increasing the probability of exploit code availability for all vulnerabilities from 0.2 ($p = 0.2$) to 0.8 ($p = 0.8$) on the time to compromise the system for the beginner attacker. Whereas the expert attacker has just a slight impact, as shown in Table 6.2.

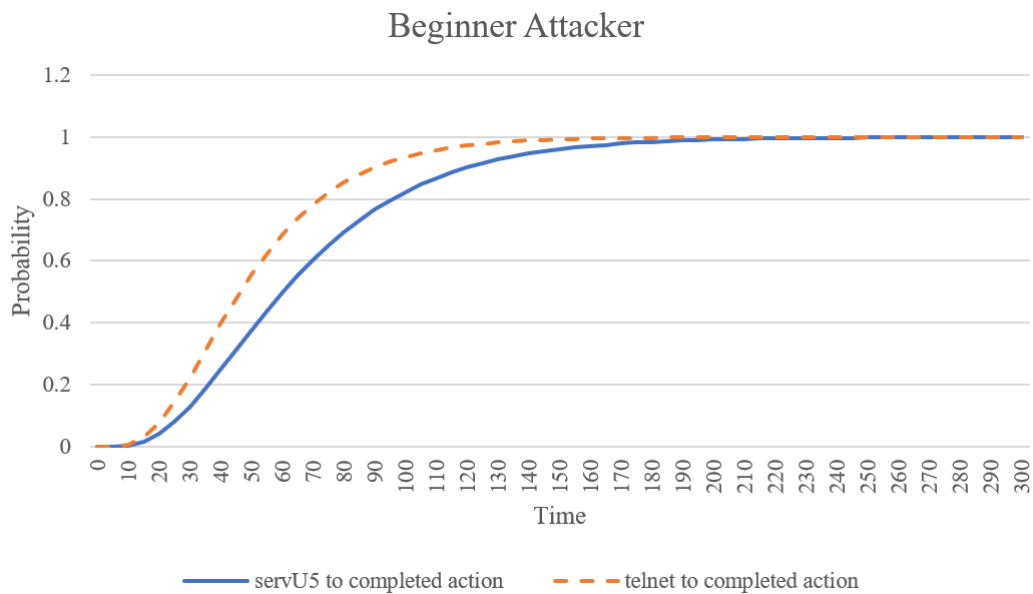


Fig. 6.10 Passage-time analysis for the beginner attacker when $p = 0.2$.

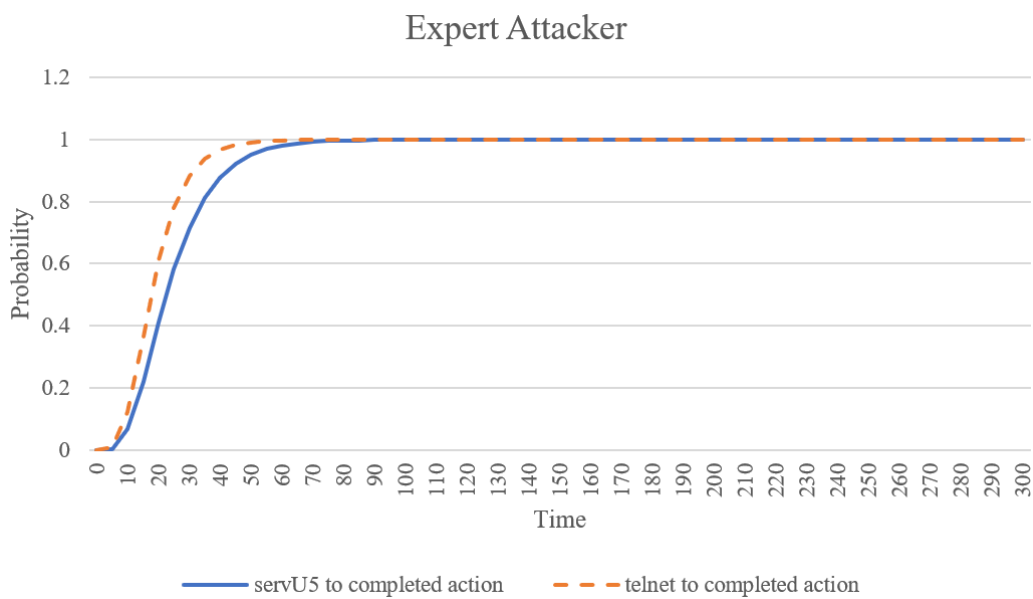


Fig. 6.11 Passage-time analysis for the expert attacker when $p = 0.2$.

Table 6.1 The average time to compromise the system for beginner attacker.

The probability of exploit code availability	From <i>servU5</i> to <i>completed</i> action	From <i>telnet</i> to <i>completed</i> action
$p = 0.2$	602 time units	465 time units
$p = 0.8$	237 time units	178 time units

Table 6.2 The average time to compromise the system for expert attacker.

The probability of exploit code availability	From <i>servU5</i> to <i>completed</i> action	From <i>telnet</i> to <i>completed</i> action
$p = 0.2$	205 time units	156 time units
$p = 0.8$	180 time units	136 time units

Furthermore, Figures 6.4, 6.5 and 6.6 show the same risk order of the attack paths as in the first case study, as shown in Figure 5.6. However, in this case study, we considered and implemented two important factors that impact the time to compromise the system. These factors are the probability of exploit code availability and attacker skill. Our evaluations of this case study clearly show the impact of these factors on the attacker time to compromise the system, as shown in Figures 6.4, 6.5, 6.6, 6.7, 6.8 and 6.9. In this case study, If attacker fails to exploit the vulnerability, it returns to the first step in the attacker's steps graph to try again to attack the same host. Whereas in the first case study, when the attacker fails to exploit the vulnerability in any host, the attacker returns to the root node in the attack graph. This is clearly shown as an increase in the attacker time to compromise the system in the first case study, as illustrated in Figure 5.6.

6.4 Attacker learning behaviour

Now, we introduce a learning behaviour to the PEPA model of the attack graph. The learning behaviour is implemented for the attacker component by adding a counter component to the model. The counter will change the rate of the *failed* action of the vulnerability. The more the attacker fails to exploit the vulnerability, the more the attacker learns and gains knowledge from previous attempts and the less the probability to fail. The counter will slow the rate of the *failed* action. The change here is on the *failed* action's rate of the vulnerability. The probability of the vulnerability to be breach kept unchanged.

6.4.1 Attack graph PEPA model with attacker learning behaviour

We apply the learning behaviour to the beginner attacker in this case study. We add the learning counters to the PEPA model for the beginner attacker. The following are the beginner attacker and counter components. The system component is the same as the system component in Subsection 5.4.2.

The beginner attacker component

This component represents the beginner attacker's different behaviours, moving from *AttackerStart* to *AttackerCompleted* to compromise the system. The description of this component is the same as the beginner attacker PEPA component in Subsection 6.2 except we introduce 4 states to allow us to implement the learning counters. These states are *AttackerNextH3*, *AttackerNextH4*, *AttackerNextH5* and *AttackerNextCompleted*.

$$\begin{aligned}
 \textit{AttackerStart} &= (\textit{start}, 1).\textit{AttackerA}; \\
 \textit{AttackerA} &= (\textit{servU5}, 0.33).\textit{Attacker0H1} + (\textit{telnet}, 0.67).\textit{Attacker0H2}; \\
 \textit{Attacker0H1} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H1}; \\
 \textit{Attacker1H1} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H1} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H1}; \\
 \textit{Attacker2H1} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H1}; \\
 \textit{Attacker3H1} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H1}; \\
 \textit{Attacker4H1} &= (\textit{runExploitCode}, 1).\textit{AttackerH1}; \\
 \textit{AttackerH1} &= (\textit{sql}, 0.12).\textit{AttackerNextH3} + (\textit{rpc}, 0.1).\textit{AttackerNextH4} \\
 &\quad + (\textit{remoteLogin}, 0.18).\textit{AttackerNextH5} \\
 &\quad + (\textit{failed}, T).\textit{Attacker0H1}; \\
 \textit{Attacker0H2} &= (\textit{recogniseVuln}, 1).\textit{Attacker1H2}; \\
 \textit{Attacker1H2} &= (\textit{searchCode}, 1 * p).\textit{Attacker2H2} \\
 &\quad + (\textit{searchCode}, 1 * (1 - p)).\textit{Attacker3H2}; \\
 \textit{Attacker2H2} &= (\textit{findSuitableCode}, 1).\textit{Attacker4H2}; \\
 \textit{Attacker3H2} &= (\textit{failToFindSuitableCode}, 1).\textit{Attacker1H2}; \\
 \textit{Attacker4H2} &= (\textit{runExploitCode}, 1).\textit{AttackerH2}; \\
 \textit{AttackerH2} &= (\textit{sql}, 0.24).\textit{AttackerNextH3} + (\textit{rpc}, 0.2).\textit{AttackerNextH4} \\
 &\quad + (\textit{remoteLogin}, 0.36).\textit{AttackerNextH5} \\
 &\quad + (\textit{failed}, T).\textit{Attacker0H2};
 \end{aligned}$$

$$\begin{aligned}
AttackerNextH3 &= (nextH1, nextRate).Attacker0H3 \\
&+ (nextH2, nextRate).Attacker0H3 \\
&+ (nextH4, nextRate).Attacker0H3 \\
&+ (nextH5, nextRate).Attacker0H3; \\
Attacker0H3 &= (recogniseVuln, 1).Attacker1H3; \\
Attacker1H3 &= (searchCode, 1 * p).Attacker2H3 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H3; \\
Attacker2H3 &= (findSuitableCode, 1).Attacker4H3; \\
Attacker3H3 &= (failToFindSuitableCode, 1).Attacker1H3; \\
Attacker4H3 &= (runExploitCode, 1).AttackerH3; \\
AttackerH3 &= (compromised, 0.6).AttackerNextCompleted \\
&+ (failed, T).Attacker0H3; \\
AttackerNextCompleted &= (next, nextRate).AttackerCompleted; \\
AttackerCompleted &= (completed, 1).AttackerStart; \\
AttackerNextH4 &= (nextH1, nextRate).Attacker0H4 \\
&+ (nextH2, nextRate).Attacker0H4; \\
Attacker0H4 &= (recogniseVuln, 1).Attacker1H4; \\
Attacker1H4 &= (searchCode, 1 * p).Attacker2H4 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H4; \\
Attacker2H4 &= (findSuitableCode, 1).Attacker4H4; \\
Attacker3H4 &= (failToFindSuitableCode, 1).Attacker1H4; \\
Attacker4H4 &= (runExploitCode, 1).AttackerH4; \\
AttackerH4 &= (sql, 0.5).AttackerNextH3 + (failed, T).Attacker0H4; \\
AttackerNextH5 &= (nextH1, nextRate).Attacker0H5 \\
&+ (nextH2, nextRate).Attacker0H5; \\
Attacker0H5 &= (recogniseVuln, 1).Attacker1H5; \\
Attacker1H5 &= (searchCode, 1 * p).Attacker2H5 \\
&+ (searchCode, 1 * (1 - p)).Attacker3H5; \\
Attacker2H5 &= (findSuitableCode, 1).Attacker4H5; \\
Attacker3H5 &= (failToFindSuitableCode, 1).Attacker1H5; \\
Attacker4H5 &= (runExploitCode, 1).AttackerH5; \\
AttackerH5 &= (sql, 0.9).AttackerNextH3 + (failed, T).Attacker0H5;
\end{aligned}$$

The counter components

The attack graph has five vulnerabilities. As the *failed* action rate for each vulnerability is different, we have to implement a counter for each vulnerability. The following are five counter components, one for each vulnerability.

servU5's *failed* action rate counter

$$\begin{aligned}
 CounterS &= (servU5, T).CounterS0; \\
 CounterS0 &= (gainKnowledge, a1).CounterS1 \\
 &+ (failed, failedServU5).CounterS0 \\
 &+ (nextH1, nextRate).CounterS; \\
 CounterS1 &= (gainKnowledge, a1).CounterS2 \\
 &+ (failed, failedServU5 * 0.1).CounterS1 \\
 &+ (nextH1, nextRate).CounterS; \\
 CounterS2 &= (gainKnowledge, a1).CounterS3 \\
 &+ (failed, failedServU5 * 0.09).CounterS2 \\
 &+ (nextH1, nextRate).CounterS; \\
 CounterS3 &= (gainKnowledge, a1).CounterS4 \\
 &+ (failed, failedServU5 * 0.07).CounterS3 \\
 &+ (nextH1, nextRate).CounterS; \\
 CounterS4 &= (gainKnowledge, a1).CounterS5 \\
 &+ (failed, failedServU5 * 0.05).CounterS4 \\
 &+ (nextH1, nextRate).CounterS; \\
 CounterS5 &= (failed, failedServU5 * 0.03).CounterS5 \\
 &+ (nextH1, nextRate).CounterS;
 \end{aligned}$$

The above part of the model is for a counter component which is the part that slows the *failed* action's rate each time the attacker learns and gains knowledge after a failure happened. This counter is for *servU5* vulnerability in H1 in the attack graph. It has 7 behaviours starting from *CounterS*. In state *CounterS*, the only action that can happen is *servU5* leading to *CounterS0*. In state *CounterS0*, one of three actions can happen either *gainKnowledge* at rate *a1* leading to next state *CounterS1*, *failed* at rate *failedServU5* and then stay in current state *CounterS0* or *nextH1* at rate *nextRate* leading it back to the first state *CounterS* which means that the attacker exploits this vulnerability. This will be repeated for *CounterS1*, *CounterS2*, *CounterS3* and *CounterS4*. The only changes in these states are the *failed* action's rate value. While the counter sequentially moves to *CounterS1*, *CounterS2*,

CounterS3, *CounterS4* and then to *CounterS5*, the *failed* action's rate is multiplied by 0.1, 0.09, 0.07, 0.05 and then 0.03, respectively. This slows the *failed* action's rate of the vulnerability to indicate the leaning behaviour of the attacker. The more the attacker fails to exploit the vulnerability the more the attacker learns and gains knowledge and the less the probability to fail.

In the last state *CounterS5*, one of two actions can be performed either *failed* at rate $failedServU5 * 0.03$ and then stay in current state *CounterS5* or *nextH1* at rate *nextRate* leading back to the first state in this component.

The following are the counter components for *telnet*, *sql*, *rpc* and *remoteLogin* vulnerabilities. They follow the same approach that *servU5*'s counter follows.

telnet's *failed* action rate counter

$$\begin{aligned}
 CounterT &= (telnet, T).CounterT0; \\
 CounterT0 &= (gainKnowledge, a1).CounterT1 \\
 &+ (failed, failedTelnet).CounterT0 \\
 &+ (nextH2, nextRate).CounterT; \\
 CounterT1 &= (gainKnowledge, a1).CounterT2 \\
 &+ (failed, failedTelnet * 0.1).CounterT1 \\
 &+ (nextH2, nextRate).CounterT; \\
 CounterT2 &= (gainKnowledge, a1).CounterT3 \\
 &+ (failed, failedTelnet * 0.09).CounterT2 \\
 &+ (nextH2, nextRate).CounterT; \\
 CounterT3 &= (gainKnowledge, a1).CounterT4 \\
 &+ (failed, failedTelnet * 0.07).CounterT3 \\
 &+ (nextH2, nextRate).CounterT; \\
 CounterT4 &= (gainKnowledge, a1).CounterT5 \\
 &+ (failed, failedTelnet * 0.05).CounterT4 \\
 &+ (nextH2, nextRate).CounterT; \\
 CounterT5 &= (failed, failedTelnet * 0.03).CounterT5 \\
 &+ (nextH2, nextRate).CounterT;
 \end{aligned}$$

sql's failed action rate counter

$$\begin{aligned}
CounterSQ &= (sql, T).CounterSQ0; \\
CounterSQ0 &= (gainKnowledge, a1).CounterSQ1 \\
&+ (failed, failedSql).CounterSQ0 \\
&+ (next, nextRate).CounterSQ; \\
CounterSQ1 &= (gainKnowledge, a1).CounterSQ2 \\
&+ (failed, failedSql * 0.1).CounterSQ1 \\
&+ (next, nextRate).CounterSQ; \\
CounterSQ2 &= (gainKnowledge, a1).CounterSQ3 \\
&+ (failed, failedSql * 0.09).CounterSQ2 \\
&+ (next, nextRate).CounterSQ; \\
CounterSQ3 &= (gainKnowledge, a1).CounterSQ4 \\
&+ (failed, failedSql * 0.07).CounterSQ3 \\
&+ (next, nextRate).CounterSQ; \\
CounterSQ4 &= (gainKnowledge, a1).CounterSQ5 \\
&+ (failed, failedSql * 0.05).CounterSQ4 \\
&+ (next, nextRate).CounterSQ; \\
CounterSQ5 &= (failed, failedTelnet * 0.03).CounterSQ5 \\
&+ (next, nextRate).CounterSQ;
\end{aligned}$$

rpc's failed action rate counter

$$\begin{aligned}
CounterR &= (rpc, T).CounterR0; \\
CounterR0 &= (gainKnowledge, a1).CounterR1 \\
&+ (failed, failRpc).CounterR0 \\
&+ (nextH4, nextRate).CounterR; \\
CounterR1 &= (gainKnowledge, a1).CounterR2 \\
&+ (failed, failRpc * 0.1).CounterR1 \\
&+ (nextH4, nextRate).CounterR; \\
CounterR2 &= (gainKnowledge, a1).CounterR3 \\
&+ (failed, failRpc * 0.09).CounterR2 \\
&+ (nextH4, nextRate).CounterR; \\
CounterR3 &= (gainKnowledge, a1).CounterR4 \\
&+ (failed, failRpc * 0.07).CounterR3 \\
&+ (nextH4, nextRate).CounterR;
\end{aligned}$$

$$\begin{aligned}
CounterR4 &= (gainKnowledge, a1).CounterR5 \\
&+ (failed, failRpc * 0.05).CounterR4 \\
&+ (nextH4, nextRate).CounterR; \\
CounterR5 &= (failed, failRpc * 0.03).CounterR5 \\
&+ (nextH4, nextRate).CounterR;
\end{aligned}$$

remoteLogin's failed action rate counter

$$\begin{aligned}
CounterRO &= (remoteLogin, T).CounterRO0; \\
CounterRO0 &= (gainKnowledge, a1).CounterRO1 \\
&+ (failed, failedRemoteLog).CounterRO0 \\
&+ (nextH5, nextRate).CounterRO; \\
CounterRO1 &= (gainKnowledge, a1).CounterRO2 \\
&+ (failed, failedRemoteLog * 0.1).CounterRO1 \\
&+ (nextH5, nextRate).CounterRO; \\
CounterRO2 &= (gainKnowledge, a1).CounterRO3 \\
&+ (failed, failedRemoteLog * 0.09).CounterRO2 \\
&+ (nextH5, nextRate).CounterRO; \\
CounterRO3 &= (gainKnowledge, a1).CounterRO4 \\
&+ (failed, failedRemoteLog * 0.07).CounterRO3 \\
&+ (nextH5, nextRate).CounterRO; \\
CounterRO4 &= (gainKnowledge, a1).CounterRO5 \\
&+ (failed, failedRemoteLog * 0.05).CounterRO4 \\
&+ (nextH5, nextRate).CounterRO; \\
CounterRO5 &= (failed, failedRemoteLog * 0.03).CounterRO5 \\
&+ (nextH5, nextRate).CounterRO;
\end{aligned}$$

The system equation

The system equation and complete specification are given by

$$\begin{aligned}
System &\stackrel{def}{=} (CounterS || CounterSQ || CounterR || CounterRO \\
&|| CounterT) \bowtie_F AttackerStart \bowtie_N Start
\end{aligned}$$

Where $F = \{failed, next, nextH1, nextH2, nextH4, nextH5, servU5, telnet, sql, rpc, remoteLogin\}$ and $N = \{start, failed, compromised, completed, servU5, telnet, sql, rpc, remoteLogin\}$. Any action in the list F and N is a shared action between the components specified in system equa-

tion. The components are initially in the states *CounterS*, *CounterSQ*, *CounterR*, *CounterRO*, *CounterT*, *AttackerStart* and *Start*.

6.4.2 Performance evaluation of the PEPA model with attacker learning behaviour

We are interested in performing passage-time analysis for each attack path to estimate the time it takes the beginner attacker's to compromise the system for each path in the attack graph. We assign 1 to *nextRate*, 0.2 to *p* and 0.01 to *a1* which is the rate of *gainKnowledge* action. The action rates for *failedServU5*, *failedTelnet*, *failedSql*, *failedRpc* and *failedRemoteLog* are 0.6, 0.2, 0.4, 0.5 and 0.1, respectively.

Figure 6.12 shows the passage-time analysis for each path when *gainKnowledge* action rate is 0.01 ($a1 = 0.01$) for the beginner attacker. Figure 6.12 shows decrease in the time to compromise for each path compared to the result of the model without learning behaviour, as shown in Figure 6.4. In Figure 6.12, for example, the time it takes the attacker to compromise the system for path 6, which is the fastest path, is 365 time units. Whereas, in Figure 6.4, the attacker's time for path 6 is 430 time units.

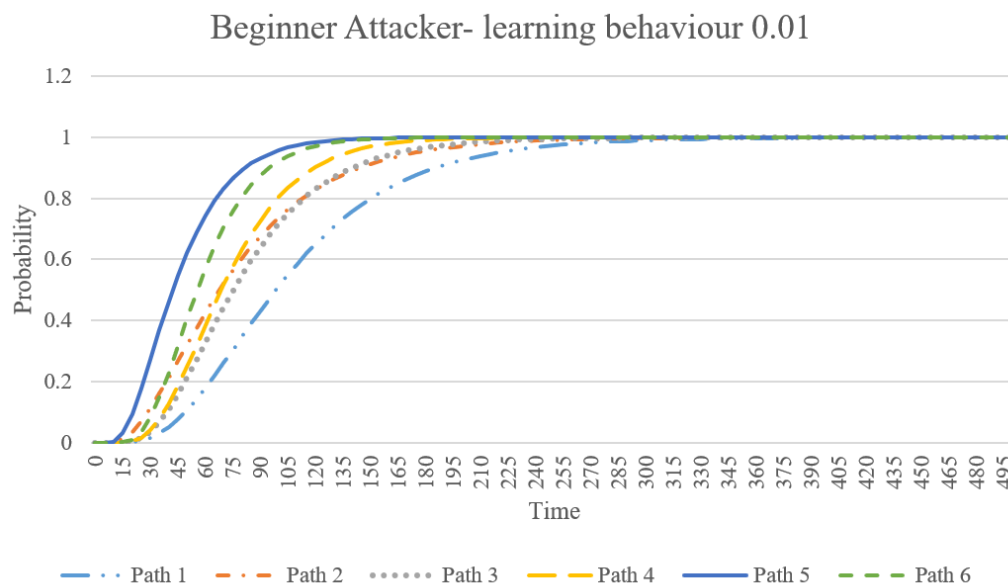


Fig. 6.12 Passage-time analysis for each path when *gainKnowledge* action rate $a1=0.01$

Now, we increase the rate of *gainKnowledge* action to 0.1 to show how increasing the leaning behaviour impacts the time to compromise the system for each path. Figure 6.13 shows the passage-time analysis for each path when *gainKnowledge* action rate $a1=0.1$ for

the beginner attacker. This result also shows a significant decrease on the time to compromise the system for each path when *gainKnowledge* action rate (a_1) is equal to 0.1. For example, the time to compromise the system for path 6 is 260 time units, Figure 6.13. Whereas in Figure 6.12, the attacker's time for path 6 is 365 time units. Thus, the faster the attacker learns, the lesser time it takes the attacker to exploit the vulnerability.

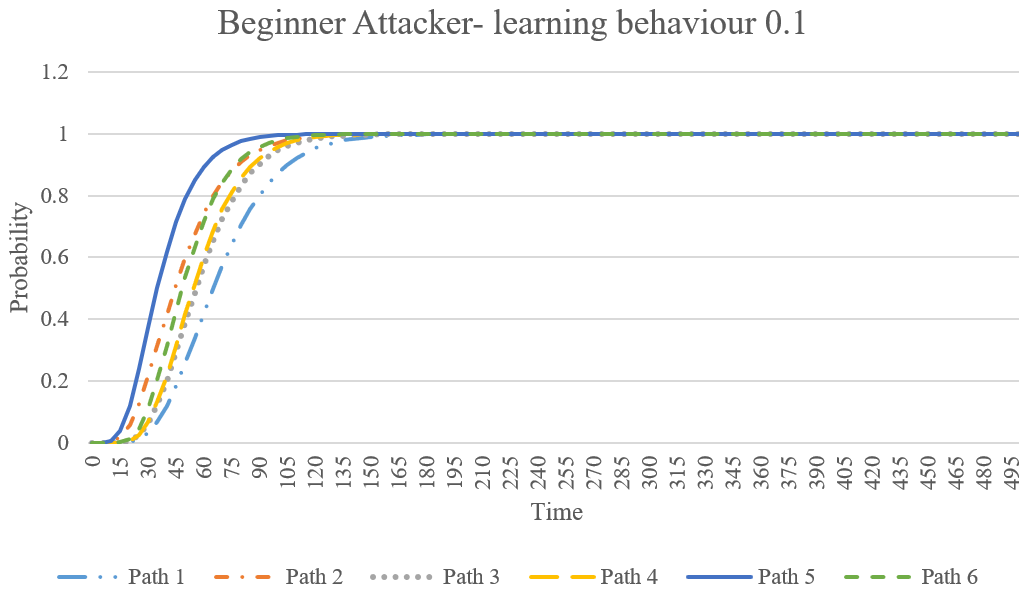


Fig. 6.13 Passage-time analysis for each path when *gainKnowledge* action rate $a_1=0.1$.

The following figures, Figures 6.14 and 6.15, clearly illustrate the impact of implementing the learning behaviour and increasing the rate of *gainKnowledge* action on the time it takes the attacker to compromise the system for path 1 and path 6. Figure 6.14 shows a significant impact on path 1 when we increased the rate a_1 to 0.1 for the *gainKnowledge* action. Whereas, Figure 6.15 shows a slight impact on path 6 when we increased the rate a_1 to 0.1 for the *gainKnowledge* action. This is because path 1, which is the slowest path in the attack graph, has a great impact when we increased the learning behaviour (the *gainKnowledge* action) rate.

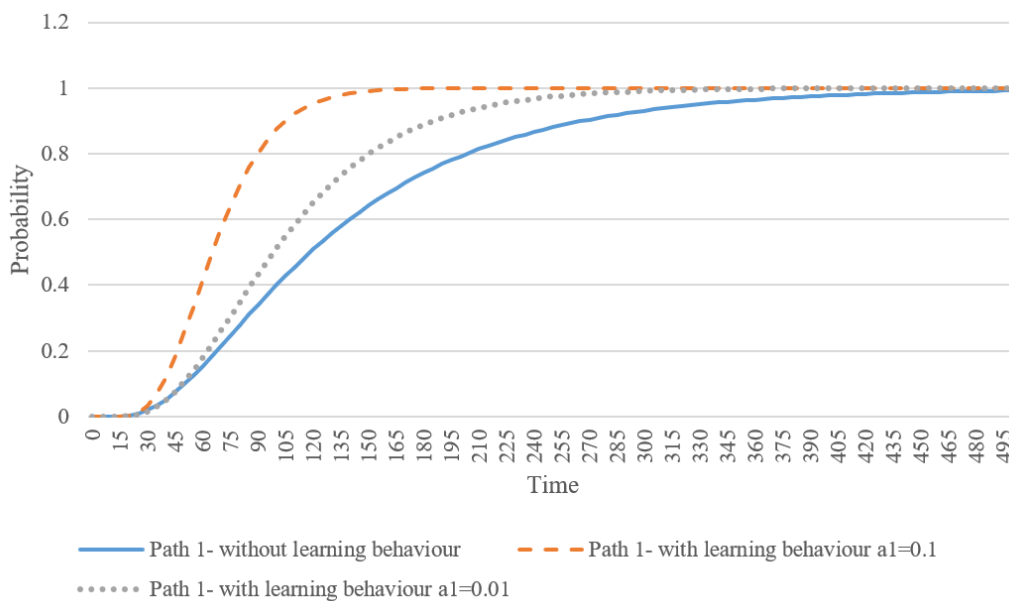


Fig. 6.14 The impact of implementing learning behaviour for the attacker on path 1.

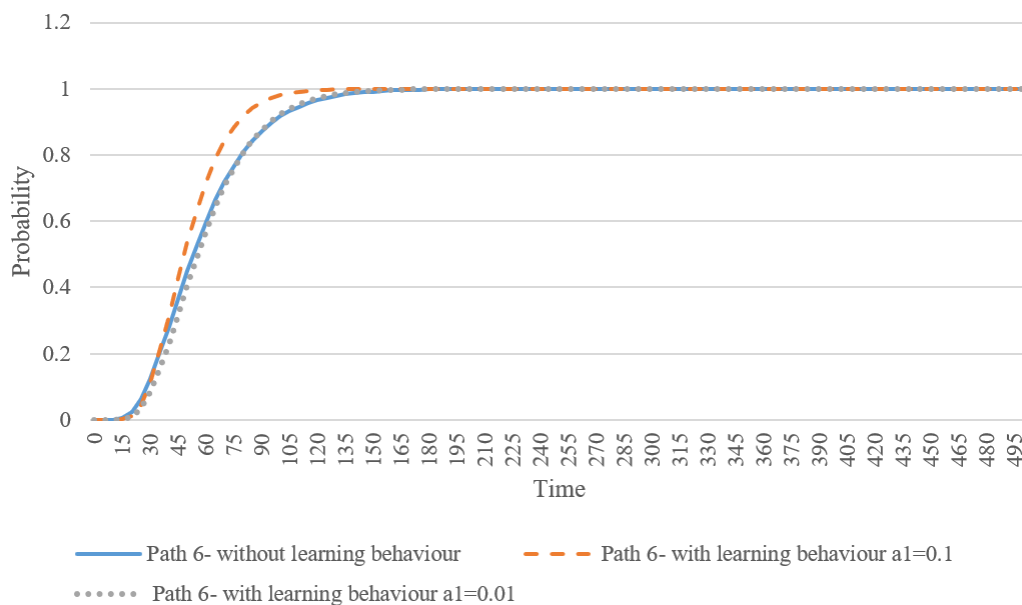


Fig. 6.15 The impact of implementing learning behaviour for the attacker on path 6.

6.5 Defender learning behaviour

In this subsection, we implement the learning behaviour to the system component for defending it. The defender can learn from each attack that targets the system's vulnerabilities.

This can be when the defender notices or detects abnormal behaviour in the system or receives the system alerts. The defender will work to make exploiting the vulnerabilities harder for the attacker by for example implementing more security measures.

Learning is represented in the PEPA model by implementing a counter component. The counter will change the rate of the *failed* action of the vulnerability. The counter will increase the rate of *failed* actions after an unsuccessful attempt to exploit a vulnerability. This means that the *failed* action will be faster, increasing the probability that the *failed* action will be executed. In Section 6.4, however, we implemented the opposite. We developed the counter that decreases the *failed* action rate after an unsuccessful attempt to exploit the vulnerabilities to reflect the attacker's learning behaviour. This indicates that the *failed* action will take longer to finish, decreasing the probability that it will complete before the other actions in the same state in the PEPA model. Further, in both cases, the rates of the other actions in the model kept unchanged.

6.5.1 Attack graph PEPA model with defender learning behaviour

We apply the learning behaviour to the system when interacting with the expert attacker. We add a learning counter to the PEPA model of the expert attacker. The PEPA model comprises of system, expert attacker and counters components.

The system component

This component represents the system's different behaviours, moving from *Start* to *Completed*. The description of this component is the same as the system component in Subsection 5.4.2, except we introduced four states to allow us to implement the learning counter. These states are *AttackerNextH3*, *AttackerNextH4*, *AttackerNextH5* and *AttackerNextCompleted*.

$$\begin{aligned}
 \textit{Start} &= (\textit{start}, 1).\textit{SystemA}; \\
 \textit{SystemA} &= (\textit{servU5}, 0.33).\textit{SystemH1} + (\textit{telnet}, 0.67).\textit{SystemH2}; \\
 \textit{SystemH1} &= (\textit{sql}, 0.12).\textit{SystemNextH3} + (\textit{rpc}, 0.1).\textit{SystemNextH4} \\
 &\quad + (\textit{remoteLogin}, 0.18).\textit{SystemNextH5} \\
 &\quad + (\textit{failed}, T).\textit{SystemH1}; \\
 \textit{SystemH2} &= (\textit{sql}, 0.24).\textit{SystemNextH3} + (\textit{rpc}, 0.2).\textit{SystemNextH4} \\
 &\quad + (\textit{remoteLogin}, 0.36).\textit{SystemNextH5} \\
 &\quad + (\textit{failed}, T).\textit{SystemH2}; \\
 \textit{SystemNextH3} &= (\textit{nextH1}, \textit{nextRate}).\textit{SystemH3} + (\textit{nextH2}, \textit{nextRate}).\textit{SystemH3} \\
 &\quad + (\textit{nextH4}, \textit{nextRate}).\textit{SystemH3} + (\textit{nextH5}, \textit{nextRate}).\textit{SystemH3};
 \end{aligned}$$

$$\begin{aligned}
SystemH3 &= (compromised, 0.6).CompletedNext + (failed, T).SystemH3; \\
CompletedNext &= (next, nextRate).Completed; \\
Completed &= (completed, 1).Start; \\
SystemNextH4 &= (nextH1, nextRate).SystemH4 + (nextH2, nextRate).SystemH4; \\
SystemH4 &= (sql, 0.5).SystemNextH3 + (failed, T).SystemH4; \\
SystemNextH5 &= (nextH1, nextRate).SystemH5 + (nextH2, nextRate).SystemH5; \\
SystemH5 &= (sql, 0.9).SystemNextH3 + (failed, T).SystemH5;
\end{aligned}$$

The expert attacker component

The expert attacker component is the same as the expert attacker in Subsection 6.2.

Counter components

The attack graph has five vulnerabilities. As the *failed* action's rate for each vulnerability is different, we have to implement counter for each vulnerability. The five counters components are the same as the counter components in Subsection 6.4, except the rate of *failed* action's. The *failed* action rate is multiplied by 2.1 after the second fail, 2.3 after the third fail, 2.5 after the fourth fail, 2.7 after the fifth fail and then by 2.9 after the sixth fail to speed the *failed* rate gradually.

The system equation

The system equation and complete specification are given by

$$\begin{aligned}
System &\stackrel{def}{=} AttackerStart \bowtie_N Start \bowtie_F (CounterS || CounterSQ \\
&|| CounterR || CounterRO || CounterT)
\end{aligned}$$

Where $F = \{failed, next, nextH1, nextH2, nextH4, nextH5, servU5, telnet, sql, rpc, remoteLogin\}$ and $N = \{start, failed, compromised, completed, servU5, telnet, sql, rpc, remoteLogin\}$. Any action in the lists F and N is a shared action between the components specified in system equation. The components are initially in the states $CounterS$, $CounterSQ$, $CounterR$, $CounterRO$, $CounterT$, $AttackerStart$ and $Start$.

6.5.2 Performance evaluation of the PEPA model with defender learning behaviour

We are also interested in doing passage-time analysis for paths 1 and 6 to predict the time takes the expert attacker to compromise the system for these paths in the attack graph. Path 6

and path 1 are the fastest and slowest paths in the attack graph, respectively. Thus, we want to show how implementing the learning behaviour will have an impact on these paths and how the learning behaviour rate will have an impact on the time to compromise.

We assign 1 to *nextRate* and 0.2 to p for exploit code availability. The action rates for *failedServU5*, *failedTelnet*, *failedSql*, *failedRpc* and *failedRemoteLog* are 0.6, 0.2, 0.4, 0.5 and 0.1, respectively.

Figures 6.16 and 6.17, clearly illustrate the impact of implementing the learning behaviour for the defender. They show an increase in the time to compromise. Moreover, the time to compromise for path 1 and 6 are calculated when *gainKnowledge* action's rate ($a1$) is 0.1 and then 0.01. The following figures present how increasing the rate of *gainKnowledge* action for the defender will have an affect on the time it takes the attacker to compromise the system for path 1 and path 6. For example, in Figure 6.17, the time to compromise is 175 time units for path 6 when *gainKnowledge* action's rate is 0.01, whereas the time to compromise is 200 time units for path 6 when *gainKnowledge* action's rate is 0.1.

However, path 1 is the slowest path in the attack graph and has the lowest attack probability, see Table 5.3. Our result shows that path 1 has been noticeably impacted when implementing the learning behaviour for the defender compared to path 6.

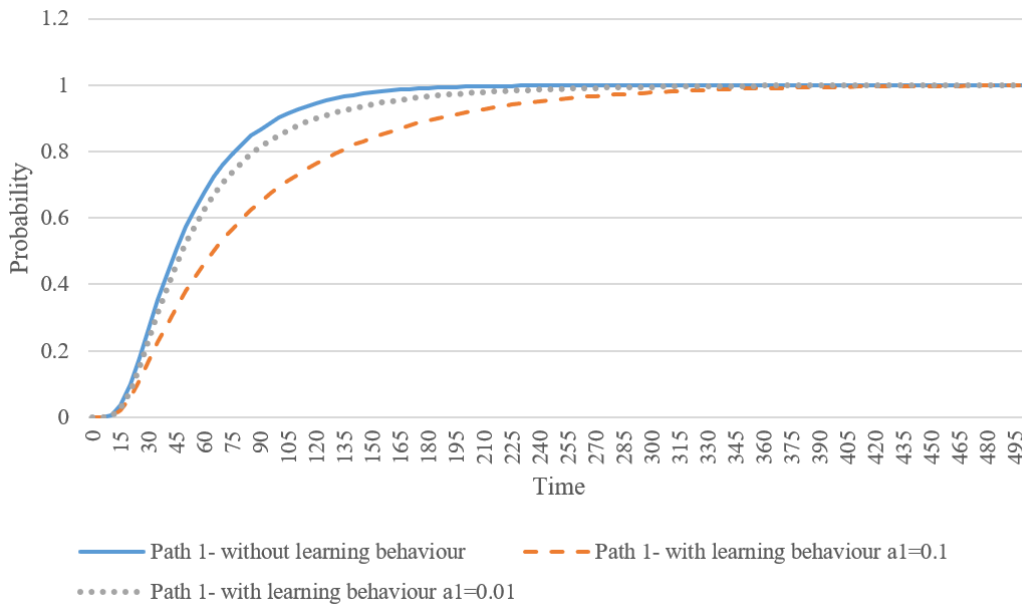


Fig. 6.16 The impact of implementing learning behaviour for the defender on path 1.

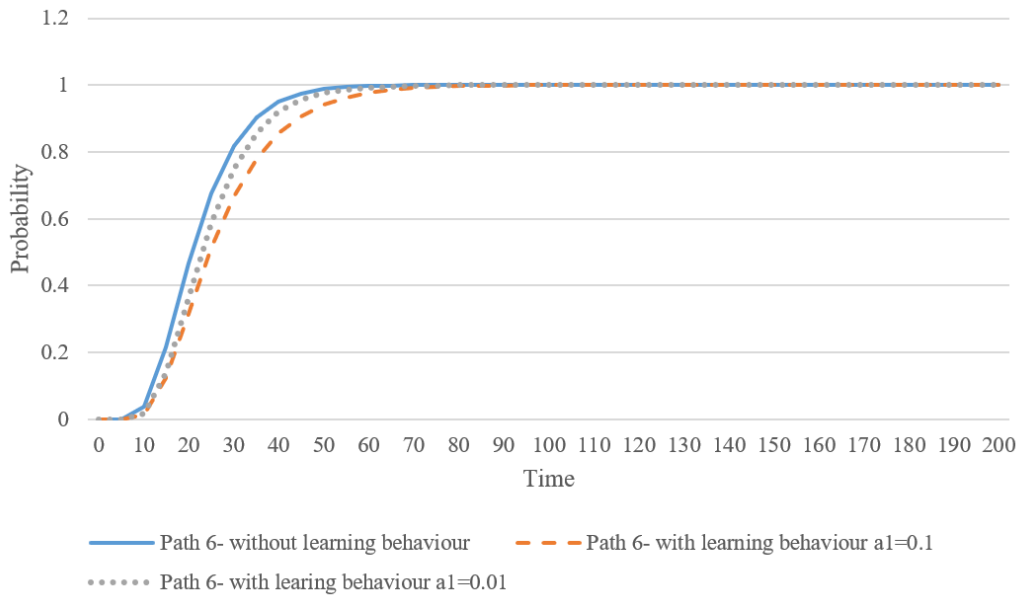


Fig. 6.17 The impact of implementing learning behaviour for the defender on path 6.

6.6 Conclusion

In this chapter, we considered three types of attackers: beginner, intermediate and expert. We proposed the attacker's steps/abilities graph for each attacker type. Then we generated the PEPA attack graph models for each attacker. We considered the attacker skill and the availability of exploit code to estimate the attacker's time to compromise the system. Moreover, we showed how the different attacker skills and the probability of exploit code availability differently impact the attacker's time to compromise the system. We also presented how the lack of exploit code impacts significantly the beginner attacker's time to compromise compared to the other attackers.

Furthermore, we implemented learning behaviours in the model for the attacker and the defender. We illustrated how learning behaviour for both the attacker and the defender would impact the time to compromise the system for the attack path. When the attacker learns about the target system and learns from previous attack attempts, this will help minimise the time to compromise the system. Further, when the defender learns about the attacker's behaviour and previous attack attempts, the defender will implement more security measures to make it difficult to compromise the system and maximise the time to compromise it. This study used the PEPA Eclipse plug-in to support the evaluation of the PEPA model.

6.7 Chapter Summary

This chapter proposes PEPA models for a pre-constructed attack graph taking into account three different skills for the attackers: beginner, intermediate and expert. Each model was evaluated to estimate the attacker's time to compromise the system for each attack path in the attack graph. Moreover, the exploit code availability is introduced as a rate of p for *searchCode* actions in all attacker's components by having a probabilistic choice p to move to next state to run the exploit code when the suitable code is found or go back to search state again. p represents the probability of an exploit code availability. The attacker skills and an exploit code availability have an impact on the time it takes the attacker to compromise a system. Building a PEPA model version of the attack graph by considering the attackers' skills and the availability of the exploit code can enhance the defender understanding and help identify critical threats.

Furthermore, we improve the adaptability of our proposed PEPA models by incorporating learning behaviour for the attacker and defender. The learning behaviours were introduced as sequential components in the PEPA model. The components will change the rate of the *failed* action of the vulnerabilities after each failed attack attempt. The PEPA models are analysed via a continuous-time Markov chain with rates to estimate the time to compromise the system.

Chapter 7

Conclusion

7.1 Summary of the research

In this thesis, we used the Performance Evaluation Process Algebra (PEPA) formalism to model systems under attack and misbehaviour with time-variable aspects in order to investigate the performance-security trade-off. This thesis investigates the performance of a web-based sales system in the presence of cyber-attacks, investigates the performance of a type of e-commerce security protocol, and provides methods for generating PEPA models of the interaction between the attacker and the system based on a pre-constructed attack graph. To begin, in Chapter 3, we investigated the impact and cost that cyber-attacks have on the performance of the web-based sales system. We proposed PEPA models of web-based sales system in two scenarios: with and without denial of service attacks. The proposed PEPA models depict the high-level interaction between the system's components. The models were analysed and the evaluation of the proposed models' throughput and population-level measures demonstrates how the attacks would prevent some or all positive customer orders from being fulfilled and how the delay in selling products would result in product discarding.

Then, Chapter 4 investigates a type of e-commerce fair-exchange protocol called an anonymous and failure resilient fair-exchange e-commerce protocol [62], which is a type of non-repudiation security protocol implemented during e-commerce transactions. This type of security protocol has been developed to ensure fair exchange between participants and that no party can take advantage over the other party. We proposed PEPA models for this protocol proposed by Ray *et al.* [62]. The proposed PEPA models show a high-level interaction between all parties involved in the protocol. We formulated the PEPA models in different ways based on the description provided by Ray *et al.* to have a complete understanding of the protocol's behaviours. The models were formulated in different ways; with and without an anonymity feature and with and without misbehaviour. The models were then evaluated based

on different scenarios, which aids in the development of a comprehensive understanding of protocol behaviour and associated performance costs.

In Chapter 5, we translate a pre-constructed attack graph to a PEPA model to build a stochastic model of an attacker behaviour and show the interaction between the attacker and the system. We proposed two methods to automate the generation of the PEPA model based on a pre-existing attack graph specification. The first method generates a PEPA model that comprises a single sequential component representing the system and attacker. The second method generates PEPA model that comprises two sequential components. The system is represented by one component, while the attacker is represented by the other component. Furthermore, through the case study, we demonstrated how we used the generated PEPA models to perform path analysis and to deduce the most and the least system security threatening paths and the attacker's time to compromise the system, which the defender can use as an indicator of how much safe time the system has before being compromised. In addition, the time taken to compromise a system for each attack path resultant from evaluating the model can rank the risk of all attack paths. Moreover, we performed a sensitivity analysis by changing the probability of the vulnerability to be breached of some action and then did the passage-time analysis for all paths. The evolution of the model clearly shows the sensitivity of each path to the change and the effect on the attacker's time to compromise the system of each path. This can assist the defender in determining the priority of countermeasures.

Chapter 6 proposed PEPA models for the attack graph, taking into account three levels of attacker skill: beginner, intermediate, and expert. We first proposed the attacker's abilities graph for each attacker type. The PEPA models were then created for each attacker based on the attack graph specification and the proposed attacker's abilities graph. Following that, each model was evaluated to estimate the attacker's time to compromise the system for each attack path in the attack graph. Moreover, the exploit code availability factor is integrated in the models as a rate. The attacker skills and exploit code availability have an impact on the time it takes the attacker to compromise a system. We showed how the different levels of attacker skill and the probability of exploit code availability differently impact the attacker's time to compromise the system. We also demonstrated how the lack of exploit code affects the beginner attacker's time to compromise significantly more than the other attackers. Constructing and analysing a PEPA model version of the attack graph while taking into account the attackers' capabilities and availability of the exploit code can help improve the defender's comprehension of the attacker's behaviours and system's security status and detect key risks. Furthermore, we implemented a learning behaviour for the attacker and defender. The learning behaviours were introduced as sequential components in the PEPA models. After each failed attack attempt, the components alter the rate of the *failed* action

of the vulnerabilities. The PEPA models are analysed via a continuous-time Markov chain with rates to estimate the attacker's time to compromise the system. Then, we illustrated how learning behaviour for both the attacker and the defender impact the time to compromise the system. When the attacker learns about the target system and learns from previous attack attempts, this helps minimise the time to compromise the system. Further, as the defender learns about the attacker's behaviour and previous attack attempts, the defender implements additional security measures to make it more difficult to compromise the system and to maximise the time it takes to compromise it. For the evaluation of all our proposed PEPA models, we used the PEPA Eclipse Plug-in, which includes an editor for PEPA model and performance analysis tools using CTMC and ODE methods.

7.2 Summary of contributions

We used PEPA formalism to develop appropriate models for a web-based sales system. The system was modelled in two different scenarios: without an attack and with a denial of service attack. We proposed two PEPA models for the system. In Chapter 3, the models were presented, then analysed and compared based on performance and security trade-offs. We studied the system's performance using the ordinary differential equations (ODEs) method, which is well-suited for large-scale systems with replicated components.

Following that, we studied a more complex system. We developed PEPA models to explore the impact of anonymity and misbehaviour of an anonymous and failure resilient fair-exchange e-commerce protocol. We proposed PEPA models for the protocol based on four main scenarios: with and without an anonymity feature, and with and without the misbehaviour of any parties. We investigated the performance cost imposed by the protocol. In Chapter 4, the models were presented, then analysed by using ordinary differential equations (ODEs) method and compared based on performance and security trade-offs.

Then, we explicitly model the misbehaving person/attacker. We study systems under attack by creating a PEPA model based on an attack graph for an attacker's behaviour. In Chapter 5, we explored the specification of models of attack graphs via two methods to create PEPA models for an attack graph to represent the attacker's behaviour and interaction with a system. Such an approach incorporates timing information into the model. Two methods are proposed to generate a PEPA model for a pre-existing attack graph with known vulnerabilities that assigned attack probability from the international standard CVSS. The first method builds a PEPA model that comprises of system and attacker as a coupled component. The second method generates a PEPA model that comprises separate components of system and attacker and an equation for the model with the cooperation set. Then, we showed through two case

studies how we used the created PEPA models to do path analysis, sensitivity analysis and estimate an attacker's time to compromise a system for each attack path.

Furthermore, we considered the effect of attacker expertise by proposing three different skills for the attackers: beginner, intermediate and expert in Chapter 6. Based on the attack graph specification and the three proposed attackers' skills, we used the second method to generate the PEPA models for each attacker's skill. Then, the models were numerically analysed and compared to show the impact on the time it takes the attacker to compromise a system. Also, in Chapter 6, we considered the effect of the availability of exploit code factor on the attacker's time to compromise a system by introducing the probability of an exploit code's availability into some of the actions rates in the PEPA models. The models were then analysed and compared based on different probability values to demonstrate the effect on the time it takes an attacker to compromise a system.

Moreover, we improved the adaptability of our proposed PEPA models by incorporating learning behaviour for the attacker and defender in Chapter 6. The learning behaviours were introduced as sequential components in the PEPA model. The components change the rate of the *failed* action of the vulnerabilities after each failing attack attempt. In case of an attacker, they decrease the *failed* action rate after a failure to exploit a vulnerability happened. In case of a defender, they increase the *failed* action rate after a failure to exploit a vulnerability happened. The PEPA models were then analysed to show the impact of the learning behaviours for both the attacker and the defender on the attacker's time to compromise the system for the attack path and the security status of the system.

7.3 Limitations of the research

In this thesis, the PEPA modelling formalism is used to model, investigate, and measure the performance and security aspects of the security protocol and attack graph as it allows a compositional, formal, and abstract approach to constructing a model of a complex system. However, PEPA allows just certain types of analysis using CTMC and ODE methods. As PEPA is a Markovian Process Algebra, it can only support actions with negative exponentially distributed rates. The duration of each activity is represented by a random variable with an exponential distribution. Different distributions, particularly those with a higher variance, produce different results. We could approximate these by using multiple actions to create phase-type distributions, but that greatly increases the specification and the number of states in the model. Another limitation is the absence of a real system against which we can validate or derive parameters for our PEPA models.

In Chapter 4, when we evaluate the PEPA models of security protocol, we have not changed most of the rates of the actions from the values that we assigned as stated in Subsections 4.4.1, 4.4.2, 4.4.3 and 4.4.4. We have not explored the protocol based on changing all actions rates. Evaluating the PEPA models based on modifying the different actions' rates in the models can provide valuable insight and help learn more about the protocol. Moreover, in Chapters 5 and 6, time constraints prevent us from consider a temporal metric when modelling attack graph as we just considered the attack complexity metric. A temporal metric is a critical metric that indicates how the probability of a vulnerability being exploited changes over time. Additionally, we have proposed and then implemented three different categories of attackers in the proposed PEPA models (beginner, intermediate, and expert attacker). Various other categories of attackers might be considered when examining the consequences of their behaviour.

7.4 Suggestions for future work

This thesis represents an initial step toward gaining a better knowledge of the impact of attackers and security measures on system performance. In our future work, we are considering modelling and analysing different types of security protocols with PEPA. In Chapter 4, to further investigate the security protocol, we will try to adjust the rates of the proposed PEPA model's actions instead of keeping most of them unchanged. Furthermore, we will model the security protocol and the systems under attack using different modelling formalisms in order to compare the results and gain a comprehensive understanding of the performance cost imposed by the security mechanism and the attacks.

In the case of the attack graph, in Chapter 5, the PEPA models of an attack graph can be expanded by taking into consideration many other factors, such as the different CVSS metrics of the vulnerabilities for assessing system risk. We just considered the complexity of the attack metric for each node to define the probability of the vulnerability being breached. Employing and implementing the temporal metrics in the model and evaluating the PEPA models of the attack graph based on it is also important to acquire a better understanding of the influence on the system and derive some useful information about the system's security status. This metric is critical because it illustrates the evolution of the probability of a vulnerability being exploited over time. It can be used to determine the probability that a system will reach a given state at a specified time. The examples in this context can be to illustrate the probability of the system being breached at a particular point in time and to show the probability of the system not being compromised within a year. This metric can

allow us to provide a time base for the state of the system. Therefore, PEPA models can be extended to include the temporal metrics into the rate.

In this thesis, the attack graph that we studied is relatively small. However, real networked systems have a large number of nodes, each with multiple exploitable vulnerabilities. As a result, a network vulnerability scan generates a large attack graph. Therefore, we will explore the issue of attack graph scale in PEPA model generation and analysis.

Furthermore, sensitivity analysis is an effective method for discovering serious threats. Therefore, we will perform more sensitivity analysis on PEPA models of an attack graph to assess the impact of various threats on a system's security status. Sensitivity analysis allows learning how sensitive the PEPA model evaluation results to the change in one or more vulnerabilities rates and how such sensitivity may affect decision-making related to prioritising countermeasure implementation. We will repeatedly change one or more vulnerability action rates in the PEPA models and keep the other rates unchanged to identify which vulnerability most impacts a system's overall security status.

We will also incorporate some defence mechanisms into our proposed PEPA model of attack graph in order to demonstrate the competition between attacker and defender and the impact on the system's performance and security. Additionally, we are also interested in considering several attackers competing for the opportunity to compromise the system to show the influence on the system's performance and security status. Our ultimate goal is to be able to monitor attacks in progress and develop an adaptive model capable of forecasting future attacker behaviour in real-time in order to provide an effective defence mechanism.

References

- [1] Abraham, S. and Nair, S. (2015). A predictive framework for cyber security analytics using attack graphs. *arXiv preprint arXiv:1502.01240*.
- [2] Abraham, S. M. (2016). Estimating mean time to compromise using non-homogenous continuous-time Markov models. In *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 467–472. IEEE.
- [3] Almasizadeh, J. and Azgomi, M. A. (2013). A stochastic model of attack process for the evaluation of security metrics. *Computer Networks*, 57(10):2159–2180.
- [4] Alrowaithy, M. (2021). *Performance-Efficient Cryptographic Primitives in Constrained Devices*. PhD thesis, Newcastle University.
- [5] Alrowaithy, M. and Thomas, N. (2019). Investigating the performance of C and C++ cryptographic libraries. In *Proceedings of the 12th EAI International Conference on Performance Evaluation Methodologies and Tools*, pages 167–170.
- [6] Alrowaithy, M. and Thomas, N. (2020). Investigating the performance of lightweight cryptographic primitives in constrained devices. In *36th UK Performance Engineering Workshop*.
- [7] Anderson, T., Roscoe, T., and Wetherall, D. (2004). Preventing internet denial-of-service with capabilities. *ACM SIGCOMM Computer Communication Review*, 34(1):39–44.
- [8] Apostolopoulos, G., Peris, V., Pradhan, P., and Saha, D. (2000). Securing electronic commerce: reducing the SSL overhead. *IEEE Network*, 14(4):8–16.
- [9] Apostolopoulos, G., Peris, V., and Saha, D. (1999). Transport layer security: How much does it really cost? In *IEEE INFOCOM'99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No. 99CH36320)*, volume 2, pages 717–725. IEEE.
- [10] Beek, M. H. t., Legay, A., Lafuente, A. L., and Vandin, A. (2020). Variability meets security: quantitative security modeling and analysis of highly customizable attack scenarios. In *Proceedings of the 14th International Working Conference on Variability Modelling of Software-Intensive Systems*, pages 1–9.
- [11] Bernardo, M., Donatiello, L., and Ciancarini, P. (2002). Stochastic process algebra: From an algebraic formalism to an architectural description language. In *IFIP International Symposium on Computer Performance Modeling, Measurement and Evaluation*, pages 236–260. Springer.

- [12] Bossel, H. (2013). *Modeling and simulation*. Springer-Verlag.
- [13] Bradley, J. T., Hayden, R., Knottenbelt, W. J., and Suto, T. (2008). Extracting response times from fluid analysis of performance models. In *SPEC International Performance Evaluation Workshop*, pages 29–43. Springer.
- [14] Buldas, A., Laud, P., Priisalu, J., Saarepera, M., and Willemson, J. (2006). Rational choice of security measures via multi-parameter attack trees. In *International Workshop on Critical Information Infrastructures Security*, pages 235–248. Springer.
- [15] Cerone, A. (2009). Formal analysis of security in interactive systems. In *Handbook of Research on Social and Organizational Liabilities in Information Security*, pages 415–432. IGI Global.
- [16] Cho, J.-H., Chen, R., and Feng, P.-G. (2008). Performance analysis of dynamic group communication systems with intrusion detection integrated with batch rekeying in mobile ad hoc networks. In *22nd International Conference on Advanced Information Networking and Applications-Workshops (AINA workshops 2008)*, pages 644–649. IEEE.
- [17] Ciardo, G., Jones, R. L., Miner, A. S., and Siminiceanu, R. (2003). Logical and stochastic modeling with smart. In *International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 78–97. Springer.
- [18] Ciardo, G., Miner, A. S., and Wan, M. (2009). Advanced features in smart: the stochastic model checking analyzer for reliability and timing. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):58–63.
- [19] Clark, A., Gilmore, S., Hillston, J., and Tribastone, M. (2007). Stochastic process algebras. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 132–179. Springer.
- [20] Courtney, T., Gaonkar, S., Keefe, K., Rozier, E. W., and Sanders, W. H. (2009). Möbius 2.3: An extensible tool for dependability, security, and performance evaluation of large and complex system models. In *2009 IEEE/IFIP International Conference on Dependable Systems & Networks*, pages 353–358. IEEE.
- [21] Deavours, D. D., Clark, G., Courtney, T., Daly, D., Derisavi, S., Doyle, J. M., Sanders, W. H., and Webster, P. G. (2002). The mobius framework and its implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969.
- [22] Dehnert, C., Junges, S., Katoen, J.-P., and Volk, M. (2017). A storm is coming: A modern probabilistic model checker. In *International Conference on Computer Aided Verification*, pages 592–600. Springer.
- [23] Deshmukh, S., Rade, R., Kazi, D., et al. (2019). Attacker behaviour profiling using stochastic ensemble of hidden Markov models. *arXiv preprint arXiv:1905.11824*.
- [24] Dick, S., Thomas, N., and Ball, F. (2006). Performance analysis of PGP. In *Proceedings of 22nd UK Performance Engineering Workshop (UKPEW)*, Bournemouth University.
- [25] Duguid, A., Gilmore, S., Smith, M., and Tribastone, M. (2010). The pepa eclipse plug-in: A modelling, analysis and verification platform for pepa.

- [26] Durkota, K., Lisỳ, V., Bošanskỳ, B., Kiekintveld, C., and Pěchouček, M. (2019). Hardening networks against strategic attackers using attack graph games. *Computers & Security*, 87:101578.
- [27] Frigault, M., Wang, L., Singhal, A., and Jajodia, S. (2008). Measuring network security using dynamic bayesian network. In *Proceedings of the 4th ACM Workshop on Quality of Protection*, pages 23–30.
- [28] Garg, U., Sikka, G., and Aawsthi, L. (2018). A systematic review of attack graph generation and analysis techniques. In *Computer and cyber security: principles, algorithm, applications, and perspectives*, pages 115–146. CRC Press.
- [29] Gelenbe, E. (1991). Product-form queueing networks with negative and positive customers. *Journal of Applied Probability*, 28(3):656–663.
- [30] Gelenbe, E. and Wang, Y. (2019). Modelling the impact of cyber-attacks on web based sales. *Submitted for Publication*.
- [31] Gilmore, S. (2005). Continuous-time and continuous-space process algebra. *Process Algebra and Stochastically Timed Activities (PASTA'05)*.
- [32] Gilmore, S. and Hillston, J. (2003). A survey of the PEPA tools. In *Proc. 2nd PASTA Workshop*, pages 40–49.
- [33] Hillston, J. (2005a). *A compositional approach to performance modelling*. Cambridge University Press.
- [34] Hillston, J. (2005b). Fluid flow approximation of PEPA models. In *Second International Conference on the Quantitative Evaluation of Systems (QEST'05)*, pages 33–42. IEEE.
- [35] Hillston, J. and Ribaudó, M. (1998). Stochastic process algebras: a new approach to performance modeling. *Modeling and Simulation of Advanced Computer Systems. Gordon Breach*.
- [36] Hirschler, B. and Sauter, T. (2016). Performance impact of ipsec in resource-limited smart grid communication. In *2016 IEEE World Conference on Factory Communication Systems (WFCS)*, pages 1–8. IEEE.
- [37] Ibidunmoye, O. E., Alese, B. K., and Ogundele, O. S. (2013). Modeling attacker-defender interaction as a zero-sum stochastic game. *Journal of Computer Sciences and Applications*, 1(2):27–32.
- [38] IEEE (1990). IEEE standard glossary of software engineering terminology. *IEEE Std 610.12-1990*, pages 1–84.
- [39] Jhavar, R., Lounis, K., and Mauw, S. (2016). A stochastic framework for quantitative analysis of attack-defense trees. In *International Workshop on Security and Trust Management*, pages 138–153. Springer.
- [40] Kaluarachchi, P. K., Tsokos, C. P., and Rajasooriya, S. M. (2016). Cybersecurity: a statistical predictive model for the expected path length. *Journal of Information Security*, 7(3):112–128.

- [41] Katipally, R., Yang, L., and Liu, A. (2011). Attacker behavior analysis in multi-stage attack detection system. In *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*, pages 1–1.
- [42] Khaitan, S. and Raheja, S. (2011). Finding optimal attack path using attack graphs: a survey. *International Journal of Soft Computing and Engineering*, 1(3):2231–2307.
- [43] Kottenko, I. and Stepashkin, M. (2006). Analyzing network security using malefactor action graphs. *International Journal of Computer Science and Network Security*, 6(6):226–235.
- [44] Krautsevich, L., Martinelli, F., and Yautsiukhin, A. (2012). Towards modelling adaptive attacker’s behaviour. In *International Symposium on Foundations and Practice of Security*, pages 357–364. Springer.
- [45] Kwiatkowska, M., Norman, G., and Parker, D. (2009). Prism: Probabilistic model checking for performance and reliability analysis. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):40–45.
- [46] Lallie, H. S., Debattista, K., and Bal, J. (2020). A review of attack graph and attack tree visual syntax in cyber security. *Computer Science Review*, 35:100219.
- [47] Lamprecht, C., van Moorsel, A., Tomlinson, P., and Thomas, N. (2006). Investigating the efficiency of cryptographic algorithms in online transactions. *International Journal of Simulation: Systems, Science & Technology*, 7(2):63–75.
- [48] Lazowska, E. D., Zahorjan, J., Graham, G. S., and Sevcik, K. C. (1984). *Quantitative system performance: computer system analysis using queueing network models*. Prentice-Hall, Inc.
- [49] Lenin, A., Willemson, J., and Sari, D. P. (2014). Attacker profiling in quantitative security assessment based on attack trees. In *Nordic Conference on Secure IT Systems*, pages 199–212. Springer.
- [50] Leversage, D. J. and Byres, E. J. (2008). Estimating a system’s mean time-to-compromise. *IEEE Security & Privacy*, 6(1):52–60.
- [51] Lounis, K. (2018). Stochastic-based semantics of attack-defense trees for security assessment. *Electronic Notes in Theoretical Computer Science*, 337:135–154.
- [52] Lounis, K. and Ouchani, S. (2020). Modeling attack-defense trees’ countermeasures using continuous time Markov chains. In *International Conference on Software Engineering and Formal Methods*, pages 30–42. Springer.
- [53] Matthews, I., Mace, J., Soudjani, S., and van Moorsel, A. (2020). Cyclic bayesian attack graphs: a systematic computational approach. In *2020 IEEE 19th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 129–136. IEEE.
- [54] McQueen, M. A., Boyer, W. F., Flynn, M. A., and Beitel, G. A. (2006). Time-to-compromise model for cyber risk reduction estimation. In *Quality of Protection*, pages 49–64. Springer.

- [55] Meng, T., Wolter, K., and Wang, Q. (2015). Security and performance tradeoff analysis of mobile offloading systems under timing attacks. In *European Workshop on Performance Engineering*, pages 32–46. Springer.
- [56] Montecchi, L., Nostro, N., Ceccarelli, A., Vella, G., Caruso, A., and Bondavalli, A. (2015). Model-based evaluation of scalability and security tradeoffs: A case study on a multi-service platform. *Electronic Notes in Theoretical Computer Science*, 310:113–133.
- [57] Okamoto, T. and Ohta, K. (1991). Universal electronic cash. In *Annual International Cryptology Conference*, pages 324–337. Springer.
- [58] Oluwaranti, A. and Adejumo, E. (2013). Performance evaluation of network security protocols on open source and microsoft windows platforms. *Performance Evaluation*, 3(7).
- [59] Outkin, A. V., Eames, B. K., Jones, S. T., Vugrin, E. D., Phillips, C. A., Walsh, S., Hobbs, J. A., Verzi, S. J., and Heersink, B. (2016). A framework for analysis of attacker-defender interaction in cyber systems. Technical report, Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).
- [60] Pokhrel, N. R. and Tsokos, C. P. (2017). Cybersecurity: A stochastic predictive model to determine overall network security risk using Markovian process. *Journal of Information Security*, 8(2):91–105.
- [61] Potlapally, N. R., Ravi, S., Raghunathan, A., and Jha, N. K. (2005). A study of the energy consumption characteristics of cryptographic algorithms and security protocols. *IEEE Transactions on Mobile Computing*, 5(2):128–143.
- [62] Ray, I., Ray, I., and Natarajan, N. (2005). An anonymous and failure resilient fair-exchange e-commerce protocol. *Decision Support Systems*, 39(3):267–292.
- [63] Sadu, A., Stevic, M., Wirtz, N., and Monti, A. (2020). A stochastic assessment of attacks based on continuous-time Markov chains. In *2020 6th IEEE International Energy Conference (ENERGYCon)*, pages 11–16. IEEE.
- [64] Schneier, B. (1999). Attack trees. *Dr. Dobb's journal*, 24(12):21–29.
- [65] Sedaghatbaf, A. and Abdollahi Azgomi, M. (2014). Attack modelling and security evaluation based on stochastic activity networks. *Security and Communication Networks*, 7(4):714–737.
- [66] Serfozo, R. (2009). *Basics of applied stochastic processes*. Springer Science & Business Media.
- [67] Silva, B., Matos, R., Callou, G., Figueiredo, J., Oliveira, D., Ferreira, J., Dantas, J., Lobo, A., Alves, V., and Maciel, P. (2015). Mercury: An integrated environment for performance and dependability evaluation of general systems. In *Proceedings of Industrial Track at 45th Dependable Systems and Networks Conference, DSN*.
- [68] Stallings, W. (2006). *Cryptography and network security, 4/E*. Pearson Education India.

- [69] Sun, F., Pi, J., Lv, J., and Cao, T. (2017). Network security risk assessment system based on attack graph and Markov chain. In *Journal of Physics: Conference Series*, volume 910, page 012005. IOP Publishing.
- [70] Swiler, L. P., Phillips, C., and Gaylor, T. (1998). A graph-based network-vulnerability analysis system. Technical report, Sandia National Labs., Albuquerque, NM (United States).
- [71] Thomas, N. (2009). Using ODEs from PEPA models to derive asymptotic solutions for a class of closed queueing networks. *8th Workshop on Process Algebra and Stochastically Timed Activities, University of Edinburgh*.
- [72] Thomas, N. and Bradley, J. (2001). Terminating processes in pepa. In *Proceedings of the Seventeenth UK Performance Engineering Workshop*, pages 143–154.
- [73] Tidwell, T., Larson, R., Fitch, K., and Hale, J. (2001). Modeling Internet attacks. In *Proceedings of the 2001 IEEE Workshop on Information Assurance and security*, volume 59. United States Military Academy West Point, NY.
- [74] Tribastone, M., Duguid, A., and Gilmore, S. (2009). The PEPA eclipse plugin. *ACM SIGMETRICS Performance Evaluation Review*, 36(4):28–33.
- [75] Wang, C.-H. and Liou, R.-W. (2018). Attack strategy prediction with precisely estimated probability and evidence mapping. *Journal of Advances in Computer Networks*, 6(1).
- [76] Wang, Y., Lin, C., and Li, Q.-L. (2010). Performance analysis of email systems under three types of attacks. *Performance Evaluation*, 67(6):485–499.
- [77] Whitt, W. (2006). Continuous-time Markov chains. *Dept. of Industrial Engineering and Operations Research, Columbia University, New York*.
- [78] Wolter, K. and Reinecke, P. (2010). Performance and security tradeoff. In *International School on Formal Methods for the Design of Computer, Communication and Software Systems*, pages 135–167. Springer.
- [79] Yousefi, M., Mtetwa, N., Zhang, Y., and Tianfield, H. (2017). A novel approach for analysis of attack graph. In *2017 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 7–12. IEEE.
- [80] Zeng, W. and Chow, M.-Y. (2011). A trade-off model for performance and security in secured networked control systems. In *2011 IEEE International Symposium on Industrial Electronics*, pages 1997–2002. IEEE.
- [81] Zhang, Y., Fan, X., Xue, Z., and Xu, H. (2008). Two stochastic models for security evaluation based on attack graph. In *2008 The 9th International Conference for Young Computer Scientists*, pages 2198–2203. IEEE.
- [82] Zhao, Y. and Thomas, N. (2008). Approximate solution of a pepa model of a key distribution centre. In *SPEC International Performance Evaluation Workshop*, pages 44–57. Springer.

-
- [83] Zhao, Y. and Thomas, N. (2009). Efficient analysis of PEPA model of non-repudiation protocols. *School of Computing Science Technical Report Series*.
- [84] Zhao, Y. and Thomas, N. (2010). Efficient solutions of a PEPA model of a key distribution centre. *Performance Evaluation*, 67(8):740–756.
- [85] Zhao, Y. and Thomas, N. (2016). Performance modelling of optimistic fair exchange. In *International Conference on Analytical and Stochastic Modeling Techniques and Applications*, pages 298–313. Springer.
- [86] Zheng, Y., Lv, K., and Hu, C. (2017). A quantitative method for evaluating network security based on attack graph. In *International Conference on Network and System Security*, pages 349–358. Springer.
- [87] Zhu, M. and Martinez, S. (2013). On the performance analysis of resilient networked control systems under replay attacks. *IEEE Transactions on Automatic Control*, 59(3):804–808.
- [88] Zia, T., Zomaya, A., and Ababneh, N. (2007). Evaluation of overheads in security mechanisms in wireless sensor networks. In *2007 International Conference on Sensor Technologies and Applications (SENSORCOMM 2007)*, pages 181–185. IEEE.

Appendix A

The PEPA Plug-in tool features

A.1 Overview

The PEPA Eclipse Plug-in facilitates the development and analysis of performance models. The tool enables us to edit PEPA models, derive the underlying CTMC for steady-state analysis, and conduct time series analysis using SSA or ODE in the Eclipse Platform. Additionally, it allows for the creation of the transition graph, the calculation of the number of states of the model for PEPA models based on the system equation. The tool is used in this study to support the creation of our proposed PEPA models of the systems under investigation and the calculation of the performance measures.

The PEPA Eclipse Plug-in tool interface is shown in Figure A.1. Before you can use the tool and create a PEPA model file, you must first create a project container. To generate a new project, choose **File > New > Project**. Then, choose **General > Project** and click **Next**. Then, write a name for the project and click **Finish**. To create a PEPA model file, right click on the project name and choose **File > New > File**. Following that, choose a name for the file with a *.pepa* extension, which is the file extension for the PEPA model file, and then click **Finish**. Now, you are able to create, edit, and analyse your PEPA model.

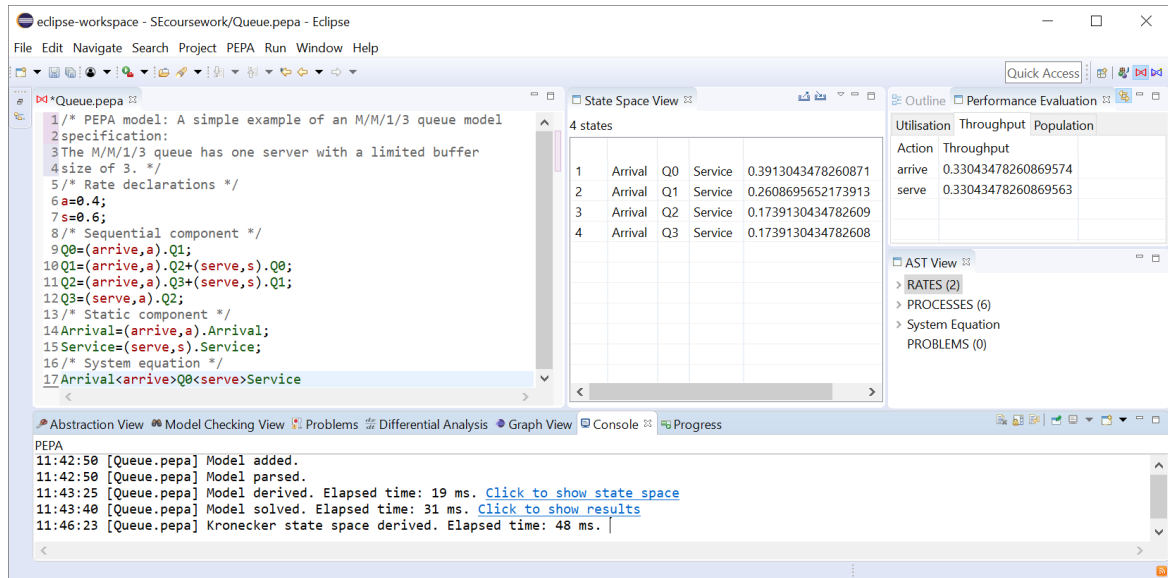


Fig. A.1 The PEPA Eclipse Plug-in interface.

Figure A.1 illustrates the PEPA Eclipse Plug-in interface that shows an editor for the PEPA model with an example of a simple PEPA model, the State Space View displaying the Markov chain's state space, the Throughput tab of the Performance Evaluation view, the Console view showing that the model has been solved for steady-state analysis, and the Abstract Syntax Tree (AST) view.

A.2 Performance evaluation

In this section, we will demonstrate how we used the tool to do performance analysis and to derive population, throughput, and average response time measures.

A.2.1 Population level

To derive the population measure, select **PEPA > Scalable Analysis > Population Level**. Then, from the pop-up box that displays, select the type of analysis, as shown in Figure A.2. We choose ODE analysis. Then, specify appropriate settings in the pop-up box that opens and then select **Analyse**, Figure A.3. You can choose the kind of analysis to be either transient or steady state from the pop-up box. After that, The population level graph is shown on the **Graph View** and you can also export CSV file of the graph, as shown in Figure A.4.

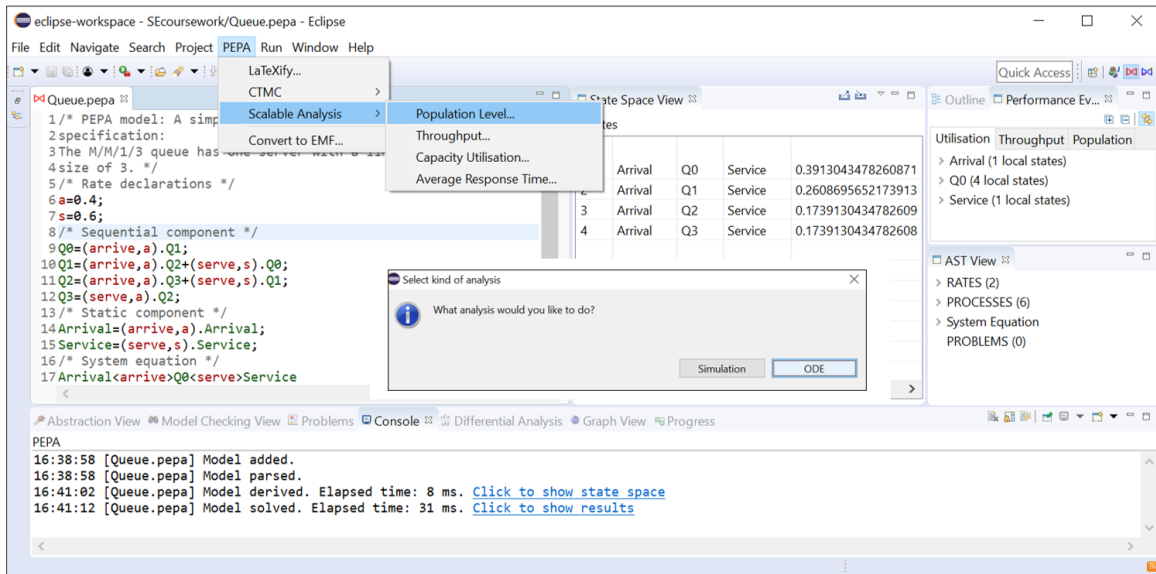


Fig. A.2 The population level analysis.

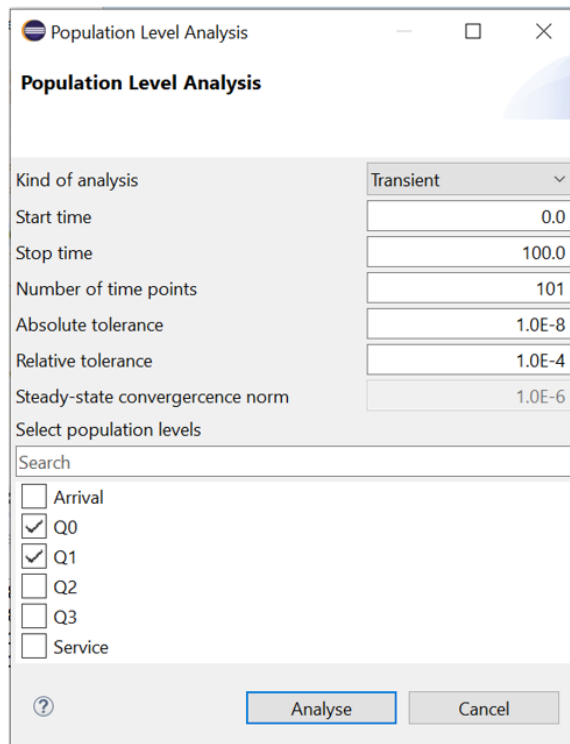


Fig. A.3 The population level analysis pop-up box.

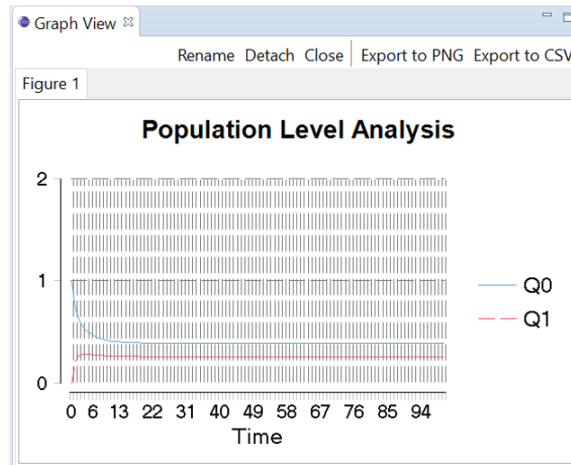


Fig. A.4 The graph view of population level.

A.2.2 Throughput

To derive the throughput measure, select **PEPA > Scalable Analysis > Throughput**. Then, from the pop-up box, choose the type of analysis, as shown in Figure A.5. ODE analysis is chosen. Then, in the pop-up window that appears, enter the proper settings and then click **Analyse**, Figure A.6. The pop-up box allows you to select either transient or steady state analysis. Following that, the throughput graph is displayed in the **Graph View**, and you can also export the graph as a CSV file, as shown in Figure A.7.

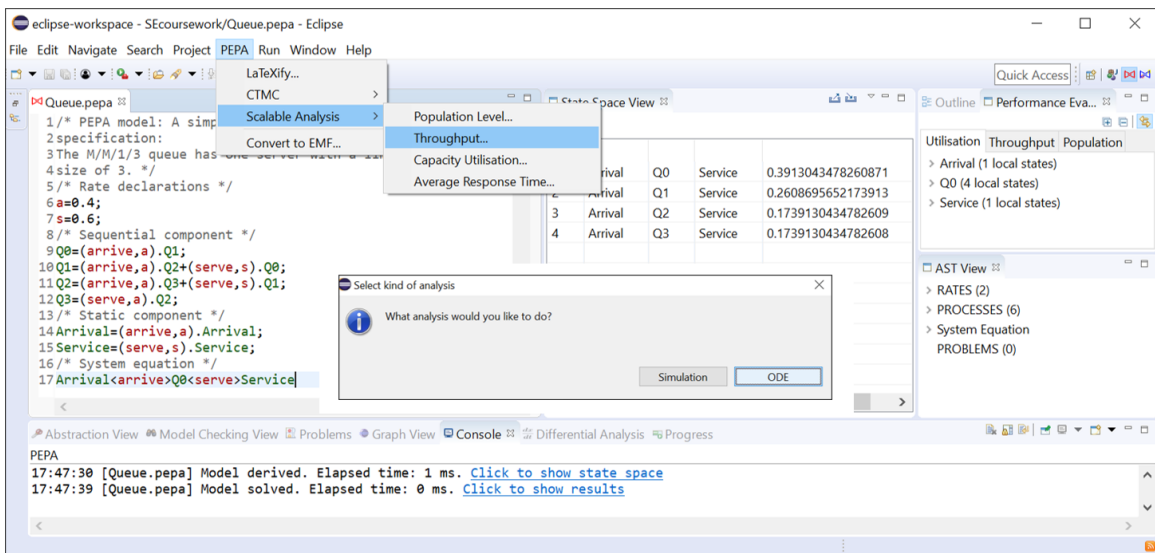


Fig. A.5 The throughput analysis.

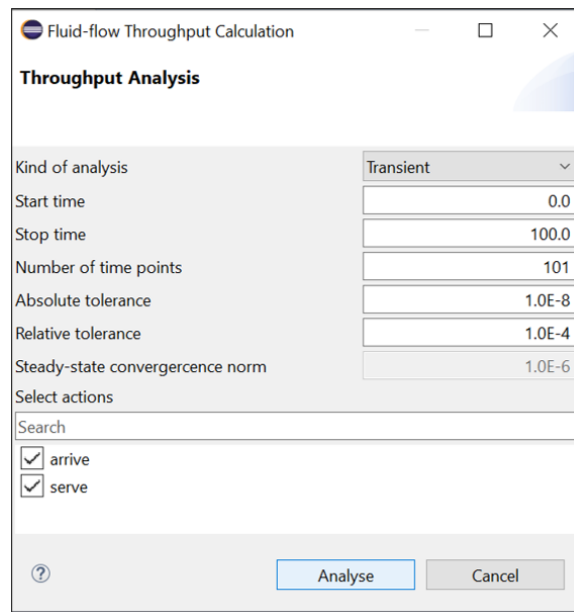


Fig. A.6 The throughput analysis pop-up box.

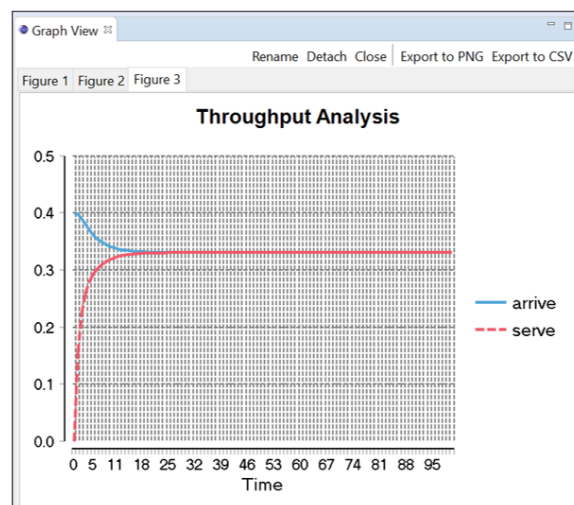


Fig. A.7 The graph view of throughput.

A.2.3 Average response time

To derive the average response time, select **PEPA > Scalable Analysis > Average Response Time**. As with previous measurements, select the type of analysis from the pop-up window, as shown in Figure A.8. The ODE analysis is selected. Then, in the pop-up box, enter the appropriate options and click **Analyse**, Figure A.9. Following that, the result window for the average response is displayed, as shown in Figure A.10.

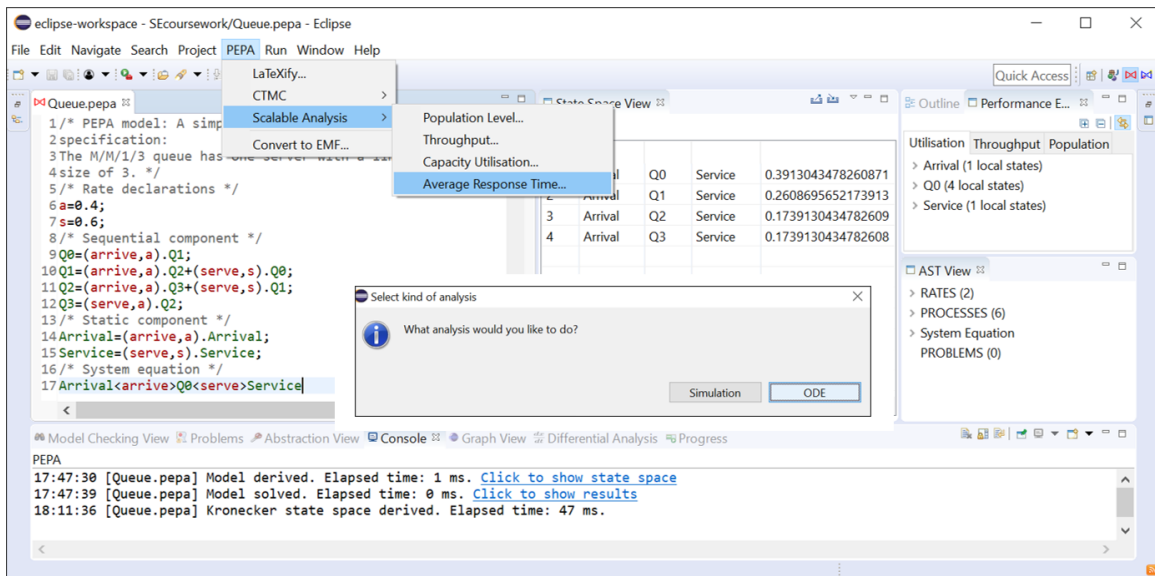


Fig. A.8 The average response time.

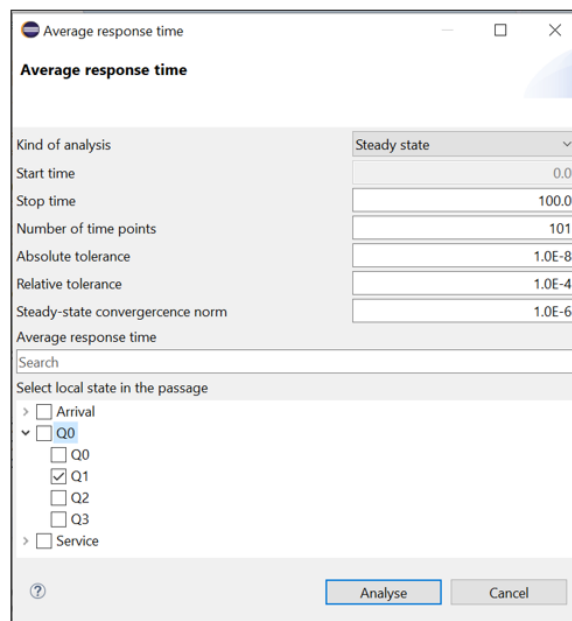


Fig. A.9 The average response pop-up box.

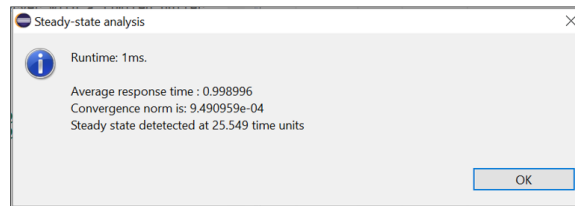


Fig. A.10 The average response's result window.

A.2.4 Passage time analysis

To calculate the passage time measure, select **PEPA > CTMC > Passage-Time Analysis**, as shown in Figure A.11. Then, in the pop-up window that appears, enter the proper settings and then click **Finish**, Figure A.12. The pop-up box allows you to select start time, time step, stop time, source action and target action. Following that, the graph of passage time is displayed in the **Graph View**, and the graph can also be exported as a CSV file, as shown in Figure A.13.

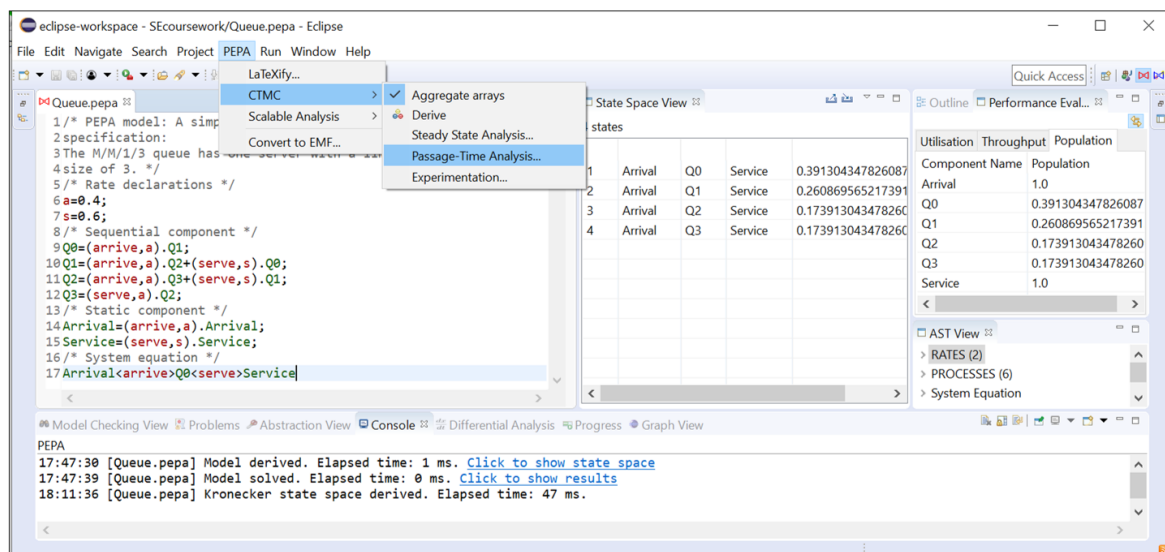
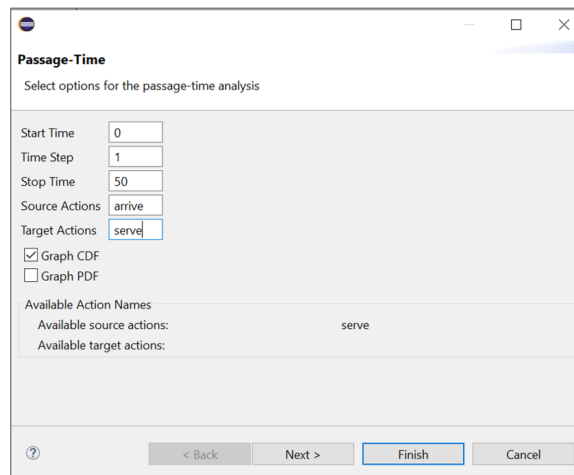


Fig. A.11 The Passage Time Analysis.



Passage-Time
Select options for the passage-time analysis

Start Time:
Time Step:
Stop Time:
Source Actions:
Target Actions:
 Graph CDF
 Graph PDF

Available Action Names
Available source actions: serve
Available target actions:

Fig. A.12 The Passage Time pop-up box.

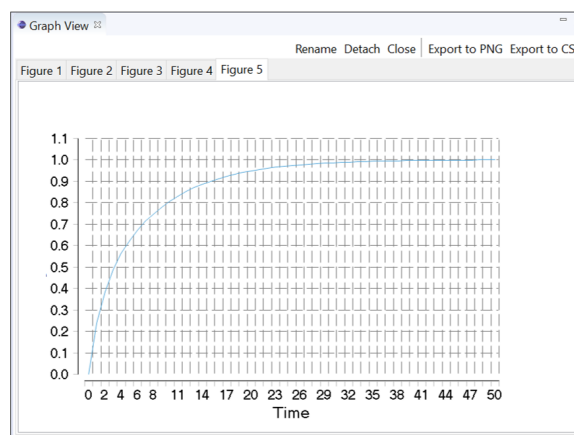


Fig. A.13 The graph view for the passage time result.