

Enhancing Security through the use of Load-Balancing Stochastic Algorithms

Rodrigo Fonseca Pires

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Daniel de Matos Silvestre
Profa. Rita Maria Mendes de Almeida Correia da Cunha

Examination Committee

Chairperson: Nuno João Neves Mamede
Supervisor: Prof. Daniel de Matos Silvestre
Member of the Committee: João Pedro Castilho Pereira Santos Gomes

October 2022

Acknowledgments

I have finally made it to end of my academic journey (at least for now). I have spent the last 5 years studying at Instituto Superior Técnico and, although it was not an easy task, I really enjoyed my time here. Throughout the course of both my Bachelor's and Master's degree I was able to acquire not only a lot of technical knowledge in the computer science field but also friendships and connections that I will cherish for life. For allowing me to achieve all of this, I would like to thank the following people.

I would like to thank my family for the support they gave me my whole life and especially the past 5 years. A special thanks to my parents who made it possible for me to move to Lisbon and attend university, although it being a massive financial and mental burden, they never made me worry about those problems and allowed me to focus on my studies.

Thank you to all my friends who accompanied me in this journey, we joined IST in the same year and remained close until the end. I would also like to thank all the teachers that helped me grow in all this years and gave me the knowledge for me to be where I am at today.

To each and every one of you – Thank you.

This work was partially supported by the Portuguese Fundação para a Ciência e a Tecnologia (FCT) through Institute for Systems and Robotics (ISR), under Laboratory for Robotics and Engineering Systems (LARSyS) project UIDB/50009/2020, through project PCIF/MPG/0156/2019 FirePuma and through COPELABS, University Lusófona project UIDB/04111/2020.

Abstract

Computer networks are growing everyday, as are the demands on performance and reliability. People use wireless ad hoc networks everywhere and perform tasks that deal with sensitive data and require fast responses. The wide spread use of wireless networks makes for an increase in the concern of attacks like the Man-In-The-Middle attack or Denial of Service (DoS). These attacks are very successful when routing protocols aim to increase performance, by prioritizing the use of optimal paths with minimal latency, making the interception of packets easier for an attacker by being trivial to predict. Many security solutions revolve around authentication methods and cryptography to protect intercepted data. What we propose here is the use of stochastic algorithms with reinforcement learning that reduces the chances of interception in the first place.

We approached this topic with a game theory perspective, defining the problem as a game and a learning algorithm that computes the equilibrium for the current scenarios. In order to assess the different algorithms, we built a framework to run simulations of our game to prove the results experimentally. Finally, we compared our results to deterministic approaches used in state-of-the-art network protocols to show increases in security and minimal impact on the performance. With these results we were able to conclude that learning algorithms are effective at protecting routers from targeted attacks, while keeping the network average performance similar to deterministic routing algorithms or even improving performance through indirect load balancing.

Keywords

Network protocols; Network routing; Stochastic algorithms; Security; Game theory.

Resumo

As redes de computadores estão em constante crescimento, assim como a necessidade de terem alta performance e confiabilidade. As pessoas utilizam redes "wireless" em todo o lado e realizam tarefas que lidam com dados sensíveis e precisam de respostas rápidas. O uso abundante destas redes leva a um aumento da preocupação com ataques do tipo "Man-In-The-Middle" e "Denial of Service" (DoS). Estes ataques têm grande sucesso quando os protocolos de roteamento dão prioridade à performance da rede, utilizando caminhos ótimos com o mínimo de latência, facilitando a interceção de pacotes por serem triviais de prever. Muitas soluções de segurança revolvem à volta de métodos de autenticação e encriptação para proteger os dados intercetados. Nós propomos a utilização de algoritmos estocásticos com aprendizagem para reduzir as chances de interceção de pacotes na rede. Atacamos o problema com uma perspectiva de teoria dos jogos, definindo o problema como um jogo e um algoritmo de aprendizagem para calcular o equilíbrio em vários cenários. Para avaliar diferentes algoritmos, desenvolvemos uma "framework" para correr as simulações do nosso jogo para provar os resultados experimentalmente. Finalmente, comparamos os resultados com soluções determinísticas usadas em protocolos de redes do estado da arte para mostrar melhorias na segurança e impactos mínimos na performance. Com estes resultados conseguimos concluir que algoritmos de aprendizagem são eficazes a proteger routers de ataques direcionados, enquanto mantém a performance média da rede semelhante à de algoritmos de roteamento determinísticos e até, em alguns casos, melhorar a performance através de "load balancing" indireto.

Palavras Chave

Protocolos de Rede; Roteamento em redes; Algoritmos estocásticos; Segurança; Teoria de jogos

Contents

1	Introduction	1
1.1	Main Objectives	4
1.2	Terms and definitions	4
1.2.1	Networks and Routing	4
1.2.2	Stochastic Learning	6
1.3	Related Work	7
1.3.1	Network Routing Protocols	8
1.3.2	Reinforcement Learning	9
1.3.2.A	Learning algorithms	9
1.3.2.B	Artificial Barriers	11
1.3.2.C	Ant optimization	11
2	Resilient Learning in Routing Games	13
2.1	Problem Statement	15
2.1.1	Scenario Overview	15
2.1.2	Environment	16
2.1.2.A	Network Node	16
2.1.2.B	Network Link	16
2.1.2.C	Network Packet	17
2.1.3	Agents	17
2.1.3.A	Defender	17
2.1.3.B	Attacker	17
2.1.3.C	Normal user	18
2.1.4	Game	18
2.1.4.A	Round	18
2.1.4.B	Preparation phase	18
2.1.4.C	Attack phase	19
2.1.5	Scoring	19

2.1.6	Formal solution	19
2.1.6.A	Defender	20
2.1.6.B	Attacker	20
2.1.6.C	Game example	21
2.2	Stochastic routing game development	22
2.2.1	Software solution overview	22
2.2.1.A	Used Technologies	22
2.2.1.B	Architecture Design	23
2.2.2	User interface	24
2.2.2.A	MainWindow	24
2.2.2.B	PlotViewer	28
2.2.3	Backend services	29
2.2.3.A	NetworkGenerator	29
2.2.3.B	NetworkGameBackend	30
2.2.3.C	NetworkGameDataCollector	31
2.2.4	Libraries	31
2.2.4.A	Network project	31
2.2.4.B	NetworkUtils	34
2.2.4.C	PythonIntegration	34
2.2.5	Algorithm Implementations	35
2.2.5.A	Routing Strategies	35
2.2.5.B	Packet Creation Strategies	38
2.2.5.C	Packet Picking Strategies	39
2.2.5.D	Route Discovery	39
3	Simulations and Results	45
3.1	Random routing strategy	53
3.1.1	No route discovery	54
3.1.2	Best path only	55
3.2	Linear reward penalty routing strategy	56
3.2.1	No route discovery & low probability of creation	56
3.2.2	No route discovery & medium probability of creation	57
3.2.3	No route discovery & medium probability of creation & high penalty rate	59
3.2.4	No route discovery & high probability of creation	60

4 Conclusion	63
4.1 Summary	65
4.2 Overview	65
4.3 Future work	67
4.3.1 Improving the application	67
4.3.2 Further investigation	68
Bibliography	69
A Network images	73

List of Figures

2.1	Scenario example	15
2.2	Game example	22
2.3	Visual Studio Solution	23
2.4	Architectural design	24
2.5	Main window UI	25
2.6	NetworkViewer controls	26
2.7	MainWindow Topbar	26
2.8	MainWindow controls	27
2.9	PlotViewer	28
2.10	Network file example	30
2.11	User configurable properties example	34
2.12	Generate network properties example	35
2.13	Update value example with minimum barrier probability = 0.1	37
2.14	Update value example with minimum barrier probability = 0.1	37
2.15	Link probabilities to send packets with destination router 0 using no route discovery	40
2.16	Link probabilities to send packets with destination router 0 using best route only discovery	41
2.17	Link probabilities to send packets with destination router 0 using breadth first route discovery	42
2.18	Link probabilities to send packets with destination router 0 using dijkstra route discovery	43
3.1	Average variance line chart example	48
3.2	Average packet queue time line chart example	49
3.3	Average packet delivery time normalized example	50
3.4	Router created packets line chart example	51
3.5	Router created packets percentage line chart example	52
3.6	Defender created packets percentage line chart example	53

A.1	Network with 20 nodes and probability of each pair of nodes having a link equal to 0.1; Longest path = 39, Average Path Length ~ 19.78 , $TTL = 59$	75
A.2	Network A.1; random routing; no discovery; packet creation probability 0.1	76
A.3	Network A.1; random routing; no discovery; packet creation probability 0.3	77
A.4	Network A.1; random routing; no discovery; packet creation probability 0.5	78
A.5	Network A.1; random routing; best path only; packet creation probability 0.1	79
A.6	Network A.1; random routing; best path only; packet creation probability 0.3	80
A.7	Network A.1; random routing; best path only; packet creation probability 0.5	81
A.8	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; low learning rate	82
A.9	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; medium learning rate	83
A.10	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; high learning rate	84
A.11	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; low learning rate	85
A.12	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; medium learning rate	86
A.13	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; high learning rate	87
A.14	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; low reward rate and high penalty rate	88
A.15	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; medium reward rate and high penalty rate	89
A.16	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; high reward rate and high penalty rate	90
A.17	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; random packet picking	91
A.18	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; FIFO packet picking	92
A.19	Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; random remove unreachable packet picking	93

List of Tables

2.1	Routing table for node V_1	22
2.2	Routing table for node X	36
2.3	Routing table for node X	37
3.1	Comparison for different packet creation probabilities on network shown in Figure A.1 with random routing and no route discovery	54
3.2	Comparison for different packet creation probabilities on network shown in Figure A.1 with random routing and best path only discovery	56
3.3	Comparison for different learning rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery and $P_c = 0.1$	57
3.4	Comparison for different learning rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery and $P_c = 0.3$	59
3.5	Comparison for different reward rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery, $P_c = 0.3$, high penalty rate and artificial barriers ($MinP = 0.02$)	60
3.6	Comparison for different packet picking strategies (see Section 2.2.5.C) on network shown in Figure A.1 with linear reward penalty routing, no route discovery, $P_c = 0.5$, $R_r = 1$, $R_p = 1$ and artificial barriers ($MinP = 0.05$)	62

Listings

2.1 .network file example	29
-------------------------------------	----

Acronyms

AODV	Ad-hoc On-demand Distance Vector
A-SAODV	Adaptive Secure Ad-hoc On-demand Distance Vector
DSR	Dynamic Source Routing
DoS	Denial of Service
DSDV	Destination Sequenced Distance Vector
MANET	Mobile Ad-Hoc NETworks
MAODV	Multicast Ad-hoc On-demand Distance Vector
OS	Operating System
SAODV	Security-aware Ad-hoc On-demand Distance Vector
TTL	Time to Live
UI	User Interface
WPF	Windows Presentation Foundation

1

Introduction

Contents

1.1 Main Objectives	4
1.2 Terms and definitions	4
1.3 Related Work	7

Throughout the years computer networks have been growing exponentially [31, 32] in size and complexity while, at the same time, the demands in performance and reliability increase as well. This constant growth puts pressure on researchers to continually work on network routing to allow the networks to keep expanding without losing their qualities. As of late, computer networks also have the necessity to be mobile and adaptable, as mobile devices become increasingly more popular [33]. This mobility is assured by wireless networks, which allow and are prepared for constant changes in their topology, with nodes connecting/disconnecting and moving around.

Traditional wired networks have a constant topology, that only occasionally changes when new nodes are added or some are removed/fail. This persistence allows routing protocols to compute optimal paths between nodes and store them as, they either never change, or do so at such slow rate that recalculations are scarce. However, when talking about wireless networks, adaptability is key because optimal paths computed at the present moment may become sub-optimal or even broken due to the volatility of this type of networks. Thus, wireless networks rely on cooperation between nodes to constantly keep every node up to date, but this cooperation with unknown participants brings major security concerns, as they open the doors to attacks such as packet interception, which can be used to either stop packets from reaching their destination or to "snoop" information from them. These attacks can even be more effective if the malicious node is strategically positioned in a optimal path between nodes, as the majority, if not all of the packets transmitted will pass through them.

Nowadays networks have a lot of built-in security measures to prevent malicious activity, like eavesdropping attacks. These measures are usually related with authentication and authorization methods, using techniques like cryptography to encrypt messages. Although this does not prevent the packets from being compromised, it secures the information in them. What we propose here is a way to enhance security by routing means, complementing existing security measures in place, to prevent the packets from being intercepted in the first place.

The approach we propose here is to use stochastic routing techniques to make use of multiple paths between source and destination, in order to spread the packets unpredictably through the network. By using a probability distribution to choose the next hop for a packet instead of precomputed optimal paths, we make it impossible for an attacker to accurately predict which nodes the packets will pass through. On top of this, we also introduce reinforcement learning into the routing algorithm, to constantly update the probabilities and adapt to the current state of the network and possible attackers. By making the nodes constantly update their routing tables we can quickly adjust to changes in the network and guarantee close to optimal performance when the network is being compromised as well as when everything is fine and there are no malicious nodes.

To work on the proposed routing algorithm, we took a game theory approach to the problem. We defined a game that emulates a scenario where there is a network being used by multiple players and an attacker is introduced. This malicious player will target one of the users in the network, the defender, and try its best to intercept packets while the network learns to adjust to the malicious activity. By defining the problem as a stochastic game we will be able to compute its Nash Equilibrium, which allows us to compare multiple algorithms in different conditions. When the game converges we are able to see the worst case scenario for that specific situation and consequently analyse which approaches better respond to an attacker intervention.

1.1 Main Objectives

The main goal of our work is to propose a way to enhance security in computer networks through making use of multiple routes between source and destination, without negatively impacting the performance of the network. We have set some objectives we want to work on in order to achieve this goal:

- Develop a framework where it is possible to generate different network graphs with customizable topology
- Develop the simulation for the game (described in Section 2.1)
- Gather data from the simulations for statistical analysis
- Analyse and compare various strategies and algorithms through simulations to find the best suitable to increase security in computer networks (presented in Chapter 3)
- Understand the benefits and limitations of using stochastic routing algorithms
- Present possible ways of continuing further development for future researchers on the topic

1.2 Terms and definitions

In this subsection we define most of the technical terms used throughout the proposal with the aim of being as objective as possible. How they are connected and used in our game is explained further in Section 2.1.

1.2.1 Networks and Routing

Computer network: A computer network is a set of computers that are connected either directly (neighbors) or indirectly (through hopping between neighbors) and can exchange information or share

resources with each other. This structure is composed mainly by network nodes, network links and network packets. It will be represented as a graph $G = (V, E)$.

Network node: In the context of computer networks, a node is either a redistribution point (like modems, switches, routers, etc.) and/or a communication endpoint (like servers, personal computers, phones, etc.). A network node is capable of emitting or redirecting network packets through the network. The nodes in our game will be represented as the vertices V of graph G and will work both as redistribution points and communication endpoints.

Network packet: Network packets are the unit of data that are transmitted through the network. They are composed of two types of data: control information and the payload. Control information refers to the data that the network needs to successfully deliver the packet to its destination. The payload is the user data to be transmitted from the source node to the destination node. We will ignore the payload in our simulations because the user data does not influence the routing.

Network link: Links are the medium that connects the various nodes to form a computer network. Network links can be physical (like electric cables or optical fiber) or wireless (using electromagnetic waves to communicate). Links will be represented as the edges E of graph G and will represent the wireless connections between the nodes.

Routing: Routing is the process of selecting a route for a network packet to follow across a computer network with the objective of reaching its destination node in the network. In our case, we will only be doing next hop routing because each node only has direct information about its neighbours and cannot plan a completed route to the destination.

Route: A route in a computer network is a set of routers that a packet can follow to go from a source node to a destination node. Due to the fact that we will only be performing next hop routing, we can only know the complete route for a packet when it reaches the destination.

Routing table: The routing process is usually performed with the use of routing tables that maintain in memory paths to different destinations in the network and possibly some other metrics to help decide the next hop. Using the routing tables, the node can choose which link the packets will be sent through. This tables may be specified by the network administrator and/or modified over time using routing protocols. In our game, the next hop decision will be made as a function of some probability distribution dynamically computed by a learning algorithm.

Routing protocol: Routing protocols specify how nodes communicate with each other in order to exchange information that can be used in routing algorithms to compute the routes packets can follow to reach their destination. In our experiments, we will perform this communication "magically", as our focus is in the user packets.

Routing algorithm: Routing algorithms aim to compute a route for packets so they can reach their destinations. This process is normally done by trying to predict which of the possible routes will be the best according to a set of metrics. Most routing algorithms use only the best path that was computed, although it is possible to make use of more than one route.

Multipath routing algorithm: This type of routing algorithm makes use of more than one route simultaneously to deliver its packets to the destination. This type of technique usually leads to an array of benefits, such as increased bandwidth, better fault tolerance and improved security, as we aim to prove with our game theory approach.

1.2.2 Stochastic Learning

Reinforcement learning: It's an area of machine learning that explores how intelligent agents make decisions in an environment with the aim of maximizing/minimizing some form of reward/cost. This type of learning is especially useful in situations where the agent has a limited amount of information about the environment.

Repeated game: In game theory, a repeated game is a game that consists of a number of repetitions of a base game. This type of games come from the idea that past playthroughs of the base game will have an influence on the current and future games being played.

Stochastic game: Is an extension of a repeated game where each action available for each player in a certain state has an associated probability of being played. This inserts a certain amount of randomness into the game and makes it possible to explore more combinations and actions when compared to a game where you always choose the "best action". Then, depending on the reward the player obtains in a "round" of the base game, it will adjust the probabilities of choosing his actions. We will simulate our network as a stochastic game so that it is able to adapt to the disruption of the attacker.

Zero-Sum game: Describes a situation in a game where the aggregated gains of all players is equivalent to the aggregated losses of all players, meaning that the net change in the game total "wealth" is always zero.

Matrix game: Is a two player zero-sum game in which the players have finite strategy sets. In our game, each node has a finite number of actions, equal to its number of links.

Nash-Equilibrium: In game theory, NA is the usual way of defining a solution for a non-cooperative game and it represents a set of strategies (one for each player) in which no player, knowing the strategies all others choose, can unilaterally change its own strategy and get better rewards. This means that, in a state of NA, no player has any incentive to pick another strategy, thus the game is said to be in equilibrium.

Saddle point: A saddle point in a Matrix game is a pair of strategies that is both the minimum of its row and maximum of its column. Also, if there is a NA in a Matrix game, it is necessarily a Saddle point.

Strategy: In game theory, a strategy is an action that a player chooses in a certain state, taking into account, not only its own set of possible actions but also the other players' possible moves after the player did its own.

Pure strategy: A Pure strategy determines the action a player will perform at every possible state the game can be on. This means that knowing the player and the game state is enough to know for sure what action he would make. Obviously, a pure strategy does not make sense in our game, as it would mean always taking the same path, defeating the purpose of a stochastic routing algorithm.

Mixed strategy: A mixed strategy consists in assigning a probability to each possible action a player can execute in a given state. The player then randomly selects one move based on the probability distribution. These probabilities are often computed proportionally to some "expected value" of performing the said move. This is the type of strategy we want our nodes to employ in order to maximise the chances of their packets not being intercepted by the attacker.

1.3 Related Work

In this section we will present some work developed in the two main areas of research relevant to our proposal: Network routing (Section 1.3.1) and Reinforcement learning (Section 1.3.2). We will make a brief summary of some existing routing protocols as well as some learning algorithms and discuss some of their shortcomings and flaws to lay the groundwork for our proposal.

1.3.1 Network Routing Protocols

We will first be looking at the evolution of Mobile Ad-Hoc NETworks (MANET) through some examples that have been compiled in [11]. With this analysis we will see that some of the challenges and limitations of MANET have been haunting researchers in the area of network routing since they first started getting developed.

MANET have a lot of advantages when compared with more traditional networks, their infrastructure is easier and cheaper to set up; they offer more fault tolerance due to their decentralized nature; the usage of multiple hops reduces the risk of bottlenecks and, obviously, they offer the mobility that wired networks cannot.

However, it is not all advantages as MANET still lacks in reliability due to its ever changing nodes and connections and, because each router needs to perform routing tasks, they require more memory and computational power, which contradicts their mobility advantage, as we want mobile devices to be small and lightweight, but that limits their computational resources and battery power. MANETs are also hard to scale, because as we increase the number of nodes in the network, the amount of processing power and memory necessary to keep the routing efficient also increases, putting more work in each individual node.

There are some other factors that negatively affect the performance of MANETs like the limited range of wireless transmission and the constant flux of in and out of nodes in the network due to its mobile characteristics, but here our main focus is security. Due to the mobile and wireless characteristics of MANETs, malicious nodes can enter the network at any time and launch attacks such as man-in-the-middle and Denial of Service (DoS). The security of the network is usually maintained by security layers, like authentication [12] and cryptography [13], but what we propose is a way to increase security through routing methods.

Some of first MANET algorithms were proposed in the 90's like the Destination Sequenced Distance Vector (DSDV) [14] and the Dynamic Source Routing (DSR) [15]. Both algorithms worked with precalculated routes, which means that when a packet is sent, the whole route is already determined. In order to compute and maintain paths from and to multiple nodes, these algorithms used a lot of memory (to keep the routes cached in routing tables) and needed a lot of bandwidth every time recalculations of paths were necessary. The main difference between the two is in the way that they compute new routes when a change happens in the network topology, DSDV [14] being proactive by constantly exchanging updates between nodes and DSR [15] being reactive by making use of acknowledgements and triggering rediscovery of paths when a lot of error messages are received.

After learning from the previous algorithms researchers came up with better performing protocols, one of which was the Ad-hoc On-demand Distance Vector (AODV) protocol [16], improving upon the DSDV protocol [14] and taking the route discovery through using on-demand route requests from DSR [15]. But it still lacked a major feature, multicast support, that would be later implemented on a protocol that was an extension of AODV, the Multicast Ad-hoc On-demand Distance Vector (MAODV) [17]. This new functionality improves the protocol by enhancing communication with multiple nodes and increasing routing knowledge while also reducing control traffic overheads [18].

After all these developments made in order to improve the reliability and performance of MANETs there was still a major issue to be addressed, which was security. Wireless networks are more vulnerable to a wide variety of network attacks than wired networks due to its non physical way of transmission. The original AODV protocol had no security measures in place and as such, was extremely vulnerable to malicious activity, like tempering with the control headers that were used by the nodes to exchange network knowledge. To try and mitigate this problems, researchers developed numerous security and authentication methods for MANETs [12] as well as continuing to iterate over previous protocols, such as the Security-aware Ad-hoc On-demand Distance Vector (SAODV) [19] and Adaptive Secure Ad-hoc On-demand Distance Vector (A-SAODV) [20], both designed after the AODV protocol, but with security in mind.

Although a lot of progress has been made to improve security in wireless networks [21], the solutions mostly revolve around authentication and authorization methods, that prevent attackers from forging or changing messages and inject them into the network. What we want to propose here is a way to improve security through routing methods, making it harder for a malicious node to have the chance of intercepting packets in the network or having to expend a lot more resources to do so.

1.3.2 Reinforcement Learning

Now we will take a look at works developed in the machine learning area, more specifically, reinforcement learning methods and techniques. Agents using this form of learning rely on getting rewards (or penalties) for their actions, resulting in them adjusting their strategies to try to maximize some sort of score or minimize some cost [22].

1.3.2.A Learning algorithms

Xiaosong Lu and Howard M. Schwartz designed and presented a decentralized learning algorithm [4] that makes use of the Lagging Anchor algorithm [5] [6] to converge to a Nash-Equilibrium in two-player zero-sum matrix games, be it a Pure Strategy or a Mixed one. They also proved that the algorithm converges into NA with only the knowledge of the action and its reward. To achieve this, the authors

studied other algorithms that served as inspiration for the development of their own and divided them into two groups. Of the four presented here, the first two are based on learning automata, which have the objective of learning the optimal strategy by updating its action probability distribution based on the environment response [4]. The latter two are based in gradient ascent learning methods that consist in updating the player strategy in the direction of the current gradient with some small step size [8].

(1) Linear Reward-Inaction Algorithm The Linear Reward-Inaction algorithm is defined in [7] and consists on a reinforcement learning method that attributes a probability to each action of a player and after each play, if the reward was positive, the probability of choosing the same action in the future increases, otherwise nothing changes. It is also proven in [7] that if all players in a matrix game use this algorithm, then it "guarantees the convergence to a Nash Equilibrium under the assumption that the game only has strict Nash equilibria in pure strategies" [4].

(2) Linear Reward-Penalty Algorithm The Linear Reward-Penalty algorithm is similar to the previous Linear Reward-Inaction algorithm in the sense that it gives each of the actions of the player some probability of being executed and after each play, if the reward is positive, it will increase the respective probability of that action to be picked. However, if the action fails to give a positive reward, instead of doing nothing, we decrease the probability of using the same action in the future. This algorithm can only be applied to two-player zero-sum games that have Nash Equilibrium in fully mixed strategies [7]. It is also important to note that the Linear Reward-Penalty algorithm "can guarantee the convergence to the Nash equilibrium in the sense of expected value, but not the player's strategy itself" [4].

(3) WoLF-IGA Algorithm Win or learn fast-infinitesimal gradient ascent algorithm [9] can be applied in two-player two-action matrix games. The algorithm constantly updates the player strategy based on the current gradient and some variable learning rate. This learning rate is smaller when the player is winning when compared with the value when the player is losing. This algorithm "can guarantee the convergence to a Nash equilibrium in fully mixed or pure strategies for two-player two-action matrix games" [4], but is not decentralized because the player needs to know its own reward matrix and the action selected by the opponent.

(4) The Lagging Anchor Algorithm Introduced by Dahl [5] [6], the Lagging Anchor algorithm, like the previous one, updates the player strategy according to the gradient but, unlike the WoLF-IGA algorithm, in this one "the limitation on each player's actions to be two actions is removed and the convergence to the Nash equilibrium in fully mixed strategies is guaranteed" [4], but it still requires the knowledge of the reward matrices of the players, thus this algorithm is also not decentralized.

After studying the previous algorithms, the authors in [4] designed the L_{R-I} Lagging Anchor algorithm, focusing on "both the player's current strategy and the long-term average of the player's previous strategies at the same time" [4]. They basically took the way in which the player strategy is updated in the Linear Reward-Inaction algorithm and added the lagging anchor term from the Lagging Anchor algorithm [4]. They also proved that this new algorithm guarantees the convergence to NA in both pure and fully mixed strategies in two-player two-action zero-sum games.

What makes this algorithm especially useful is the fact that it is decentralized, which is the type of learning we want to apply to our network so that each node can be independent from all others. Also, because we will be working with an incomplete information game, only needing to know the action and consequent reward is a really good reason to try to make use of this algorithm.

1.3.2.B Artificial Barriers

Anis Yazidi, Daniel Silvestre and B. John Oommen discuss and propose the use of Artificial Barriers in Learning Automata to solve Two-Person Zero-sum Stochastic Games [1]. LA had previously been used [2] to compute the Nash-Equilibrium of this type of games under limited information. However, they noted that these existing algorithms often focused in the cases where the Saddle Point existed in a Pure Strategy. This can be a problem, especially on the game we are trying to solve in our research, because if the game strategy converges into a single action, it becomes easy to predict the other player's move. So, they proposed a solution that makes it possible to converge into an optimal mixed Nash-Equilibrium even if the game has no Saddle Point when a Pure Strategy is invoked. This was accomplished with the use of Artificial Barriers that prevent the game from being absorbed into a Pure Strategy. Although a similar method had already been proposed by Lakshmivarahan and Narendra [3] 40 years ago, this new approach is, as described by the three researchers, "more elegant and required less parameter tuning".

In section "IV.Simulations D.Real-life Application Scenarios" [1] they suggest the use of their leaning algorithm in security games and describe a very similar, although simpler, version of the game we are trying to investigate in our proposal. The main reason for this suggestion is that a mixed Nash-Equilibrium is preferred over Pure Strategies in security games, because it makes it a lot harder to predict what the other player will do.

1.3.2.C Ant optimization

Ant optimization is a technique used to find good paths in a network. Initially proposed by Marco Dorigo in 1992 [23] [24], the idea was to find an optimal path in a graph by emulating the ants' behavior when seeking a path between their colony and a source of food.

Real ants start by wondering randomly looking for food. When a ant finds a source of food, it starts walking back to the colony and leaving a pheromone trail where it passes. Other ants will be attracted to that pheromone and are more likely to start following that path. When they eventually find food as well, they will reinforce the trail with more pheromones, creating a positive feedback loop where more ants following a path will attract even more ants. This pheromone evaporates over time and if no ants follow a path it will eventually be lost. This fact is very relevant, as longer paths will be more likely to disappear, as they take more time to transverse, thus letting the pheromone evaporate. Thanks to this fact, we avoid converging into a local optimal solution, as shorter paths will be more likely to retain higher concentration of pheromone [25].

This type of algorithms have numerous applications in computer networks [26] [27] and we took great interest in them because of their decentralized nature as well as being fast to adapt to changes and having a low amount of overhead [25]. These characteristics make this approach a good candidate for building a stochastic routing algorithm, as it will be useful to set the initial probabilities in the nodes' routing tables to take advantage of multiple paths between source and destination.

2

Resilient Learning in Routing Games

Contents

2.1 Problem Statement	15
2.2 Stochastic routing game development	22

In this chapter we will go over the problem at hand, describing it and identifying the elements and rules that compose it. Then we will present the software solution that was developed to help us study the problem.

2.1 Problem Statement

In this section we will describe the game that we will be implementing and studying to analyse the different strategies and algorithms to make and optimize routing decisions. We will start by making a more informal and general description of the solution to give an overview of the game and make it easier to understand. Then we will formalize it with a Game Theory approach.

2.1.1 Scenario Overview

First, we will present an example scenario to help illustrate the problem. As shown below, in Figure 2.1, our problem consists in a computer network where there are multiple users exchanging packets. At some point, an Attacker infects a router in the network. This malicious user has the objective of intercepting packets from a specific user, we call it the Defender, from reaching their destination, the Target User.

What we want to study in this scenario is how can the routing algorithm adapt to help mitigate the negative impact that the Attacker creates on the Defender; without needing to know who the Attacker and Defender are, or even if there is a malicious user in the network.

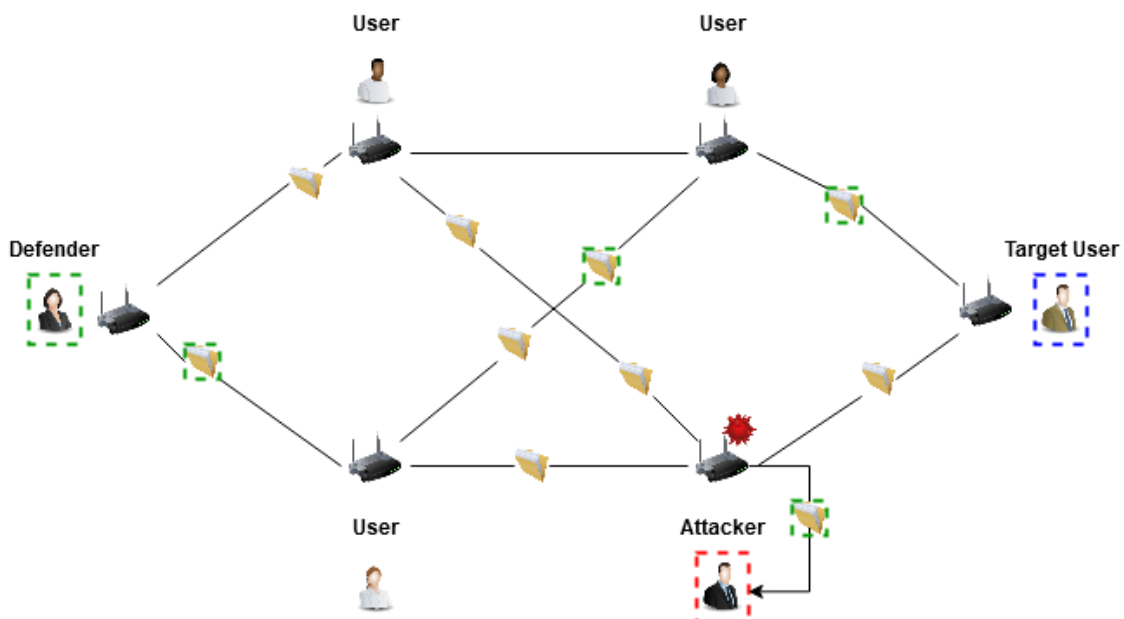


Figure 2.1: Scenario example

2.1.2 Environment

Now let us describe the game environment in which the players will compete. Since we are trying to simulate what would happen to network packets P in a computer wireless network, what makes the most sense is representing the game world as a graph $G = (V, E)$, in which the vertex V_i represents the network node number i and the edge $E_{i,j}$ represents the network link connecting nodes i and j . Although this is a simulation, we want to apply some restrictions on our game network components to closer reassemble a real life scenario, thus making the game more realistic and the obtained results more reliable.

2.1.2.A Network Node

As stated before, a network node is represented by the vertex V_i of the graph G and works both as a redistribution point and a communication endpoint. This means that all vertices are capable of redirecting incoming packets as well as generating new ones and injecting them into the network.

A node can only send one packet per each unit of time and has a waiting queue that can have a limited capacity. If a node has a full waiting queue it will discard some packets following some policy, for example:

- Discards the latest packets that arrived at the node (gives priority to the packets that arrived first)
- Discards the oldest packets (the ones with lower Time to Live (TTL), meaning that they have been circulating for more time)
- Discards the packets that are further away from their destination node (because they could be more likely no not make it)

Aside from this discard policy, each node will also have a routing table that will determine which link are the packets sent to. We will talk more about this in Section 2.2.5.A.

2.1.2.B Network Link

A network link that connects node i and node j is represented by the edge $E_{i,j}$ and it works as the medium in which packets can be transmitted from one node to another. Every link has a positive integer length l that represents the number of units of time that a packet takes to be transmitted from node i to node j and vice-versa.

2.1.2.C Network Packet

Network packets P are the units of data that transverse our network and carry the data to be transmitted from some node V_i to some node V_j and metadata that takes part in helping to deliver the packet to its destination.¹

Metadata that packets contain:

- Source/Destination node number
- TTL
- Route taken thus far

2.1.3 Agents

Our game is a competitive game, where two agents are trying to maximize their own score. On one hand we have what we will call the defender and it represents a "normal" user of the network that has access to a node and wants to send packets to some other node; on the other hand we have the attacker which represents a "malicious" user of the network that is trying to sabotage the communication of the defender.

2.1.3.A Defender

The defender has access to some node $V_{defender}$ in the network where he can inject packets. In each unit of time, the defender can either send one packet or do nothing. The defender has no idea that is being attacked and as such considers everything that happens to its packets a consequence of the environment.

2.1.3.B Attacker

The attacker infects one node in the network that is neither the node that the defender has access to or the node that the defender wants to send packets to (as that would make the attacker have an overwhelming advantage). The attacker has control over the infected node and can:

- Generate packets and inject them in any link connected to the infected node
- Control the waiting queue of the node and the next packet to be sent
- Drop packets from the defender

¹The data part of the packet is irrelevant in our simulations; we will assume that every packet has "valid" data, because the point here is to increase security through routing means and not data verification.

2.1.3.C Normal user

Normal users perform the same actions as the defender, but are not being targeted by the attacker. Their role is to send packets to populate the network and make for a more realistic and complex scenario. These agents could be seen as a part of the environment.

2.1.4 Game

A game has two phases: the "preparation phase" (see Section 2.1.4.B) and the "attack phase" (see Section 2.1.4.C). A game is separated in multiple "rounds" and each round represents what happens in a unit of time.

2.1.4.A Round

A round represents everything that happens in one unit of time. In a round:

- Every network user (including the defender and the attacker) can send one packet through one of its links
- Every packet that is currently on a link moves one step forward
- Every packet that either reaches its destination or gets dropped (by running out of TTL or, in the case of the packets sent by the defender packets, gets attacked) sends an acknowledgment² to all the nodes it passed through

2.1.4.B Preparation phase

At the start, there is no attacker and the nodes have empty routing tables. In this phase the goal is to populate these tables and establish the best routes in a normal situation. This is done by generating packets in every node and send them to random destinations.

In this case, without the disruption of the attacker we expect the routing tables to tend to favor the shortest paths if the number of packets sent is not too high. Otherwise, if the network is crowded with packets we expect the routing tables to tend to favor splitting them between multiple links to take advantage of multiple paths.

After some arbitrary number of iterations the tables will reach some equilibrium and won't change much anymore (if the rate at which packets are generated is constant). When this equilibrium is achieved we can introduce the attacker.

²We are not going to simulate acknowledgment packets (might be something to experiment on in the future)

2.1.4.C Attack phase

In this phase the attacker starts by choosing a node to infect (that is not the source/destination of the packets sent by the defender). Then the attacker starts redirecting packets that reach the infected node and dropping packets that come from the defender.

In a similar fashion to the previous phase, the network will eventually reach an equilibrium in which it won't change its routing tables much anymore, we can end the game at this point and collect relevant data to compute "scores" for both players.

2.1.5 Scoring

At the end of a game we want to attribute a score to both the attacker and the network (represented by the defender). This will let us compare different strategies and algorithms across different games. On one hand, the attacker will be rewarded by how much he was able to disrupt the packets sent by the defender, on the other hand the network will be rewarded by how much it was able to "protect" the defender from the disruption caused by the attacker.

To attribute these scores we will use multiple metrics, like the packet loss experienced by the defender (both from direct attacks and timeouts) and the time, paths and number of hops that packets took to arrive to the destination (minimum/maximum, average, standard deviation, etc.).

2.1.6 Formal solution

Now we will formally define the proposed solution as a game. To define a game we need to specify the Players, their available Actions, the Payoffs for each action and the Information that they have at each point, known as PAPI (players, actions, payoffs, and information) [10]. We will be referring to the proposed game as "Stochastic Routing Game". The game only truly starts when we have both the attacker and the defender in the network, which we refer to as the "Attack phase" described in Section 2.1.4.C.

In the Routing Game we will define the players as attacker and defender. Both of the players will occupy and play as a router (node in the graph), but there are many more routers in the network. These other routers will have the same actions and information as the defender, but will not be targeted by the attacker and will not have the objective of maximising some score/utility. As such, for simplicity of the model, we will treat all other routers as the "Nature" [10]. They will be simply following the routing algorithm controlled by the defender and inject packets into the network randomly.

2.1.6.A Defender

The defender action set will be dependent on how many links the node he is occupying is connected to, let us call it $V_{defender}$. If the node $V_{defender}$ has n links, then the defender action set is defined as $A_{defender} = \{a_i\}$, ($i = 1, \dots, n$), where each action a_i represents sending a packet through the i th link of node $V_{defender}$. We also define the target node for the packets sent by the defender as $V_{target} \neq V_{defender}$.

Each action a_i will also have some probability p_i of being executed by the defender and the set of probabilities is defined as $P_{defender} = \{p_i\}$, ($i = 1, \dots, n$), where each probability p_i represents the likelihood of sending the packet through link i of node $V_{defender}$.

As for the payoff of each action, we need to first make the distinction between the expected payoff and the actual payoff. Not only the defender cannot know the exact payoff some action will give when it is played out, he also does not know right after executing the action. This is due to the fact that we cannot know how successful or unsuccessful the network was at delivering a packet before it either reaches the destination or is dropped. Because of this delay in receiving the reward, the defender needs to have a set of expected payoffs for each action a_i , defined as $E_{defender} = \{e_i\}$, ($i = 1, \dots, n$), where each expected payoff e_i represents an estimate of how successful the network will be at delivering the packet through link i of node $V_{defender}$, which directly correlates with the probability p_i .

Regarding the information the defender (or any other node that is not the attacker) has available, he only knows the final reward (score) of every packet he sent. This means that when a packet sent by the defender reaches its end, successfully or not, the defender gets an acknowledgement³ with the reward and associates it with the action a_i that was executed to send the respective packet. This also means that the order in which the defender gets the rewards for the actions he executes is not guaranteed to be the same as the actions were played out. Because of this limited amount of information, we can even consider the attacker as part of the "Nature" from the perspective of the defender, as he is unaware that he is being targeted.

2.1.6.B Attacker

The attacker has a decision to make before the start of the game that will influence the game until the end. After the "Preparation phase", described in Section 2.1.4.B, the attacker has to pick a node to occupy, (that is neither the defender or the target of the packets sent by the defender), let us call it $V_{attacker} \neq V_{defender} \neq V_{target}$.

After the making this initial decision, the game begins (start of the "Attack phase", see Section 2.1.4.C) and the attacker has one action set dependent on how many links the node $V_{attacker}$ is connected to. If

³As explained in note 1 in Section 2.1.4.A, this acknowledgements are made "magically" in our game for the sake of simplicity.

the node $V_{attacker}$ has m links, then the attacker action set is defined as $A_{attacker} = \{b_i\}, (i = 1, \dots, m)$, where each action b_i represents sending a packet through the i th link of node $V_{attacker}$.

Each action b_i will have some probability q_i or r_i of being executed by the attacker depending on the origin of the packet at play.

If the packet has as its source the node $V_{defender}$, the set of probabilities is defined as $P_{attackerD} = \{q_i\}, (i = 1, \dots, m)$, where each probability q_i represents the likelihood of sending the packet through link i of node $V_{attacker}$.

If the packet has as its source some node $V_i \neq V_{defender}$, the set of probabilities is defined as $P_{attackerN} = \{r_i\}, (i = 1, \dots, m)$, where each probability r_i represents the likelihood of sending the packet through link i of node $V_{attacker}$. (This separation gives the attacker means to strategize in a way that he can maximize the damage caused to the defender while "doing its job" as a normal router so that the other routers do not start avoiding sending packets through $V_{attacker}$.)

Similarly to the defender, the attacker has to make his decisions based on an expected payoff, because he can only know the actual payoff of executing an action after the affected packet reaches its end. As such, the attacker needs to have two sets of expected payoffs for each action b_i , dependent on the origin of the packet at play.

If the packet has as its source the node $V_{defender}$, the set of expected payoffs is defined as $E_{attackerD} = \{f_i\}, (i = 1, \dots, m)$, where each expected payoff f_i represents an estimate of how unsuccessful the network will be at delivering the packet through link i of node $V_{attacker}$.

If the packet has as its source some node $V_i \neq V_{defender}$, the set of expected payoffs is defined as $E_{attackerN} = \{g_i\}, (i = 1, \dots, m)$, where each expected payoff g_i represents an estimate of how successful the network will be at delivering the packet through link i of node $V_{attacker}$.

In regards to the information available to the attacker, he only has knowledge of the final reward of every packet he sent (same as all other nodes). This means that he has the same amount of information has the defender, except for the fact that he knows who the defender is (otherwise he would not be able to target him) and can differentiate between packets sent from the defender and all other nodes.

2.1.6.C Game example

Let us now present an example of the described game. In Figure 2.2 we can observe a game in the Attack Phase (Section 2.1.4.C) that has been running for some time. The network has 6 routers, where V_D represents the defender, V_T the destination for the packets created by the defender, V_A the attacker and V_1, V_2, V_3 are normal users. The values in each extremity of a link represent the probability that the node closest to the value has to send a packet through the link, when the destination of said packet is V_T . As an example, the routing table for V_1 is shown in Table 2.1 for this moment in the game.

Table 2.1: Routing table for node V_1

	Link to V_D	Link to V_A	Link to V_2
Router ?
Router V_T	0.1	0.3	0.6
Router ?

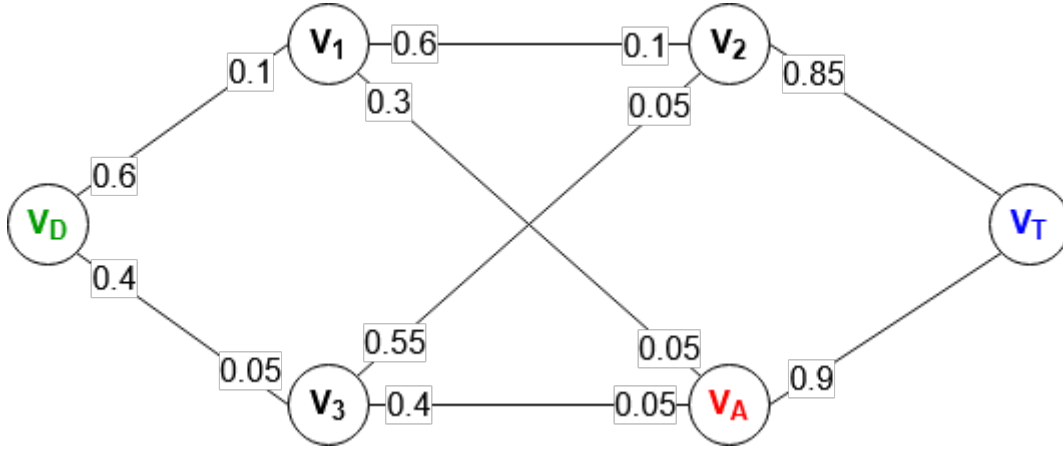


Figure 2.2: Game example

2.2 Stochastic routing game development

In this section we will present the software project developed⁴ to facilitate the study of different network and agent configurations to test and evaluate the proposed stochastic routing game. We will be going through what was developed and how it works, as well as suggestions for improving the project and further development ideas.

2.2.1 Software solution overview

2.2.1.A Used Technologies

First let us look at what technologies were used to develop the solution. The solution is written for the most part in C# and developed in a visual studio environment. The program interface is a Windows Presentation Foundation (WPF) project, which can only be ran on Windows systems, but all other supporting projects are written in .NET Standard 2.0 which can be compiled to run in any system. This means that if you want to run the solution in another Operating System (OS), you will need to implement a new interface for it.

⁴You can view the code at <https://github.com/RodrigoPires190776/NetworkGame>

Other than standard Microsoft libraries, the WPF project also makes use of the ScottPlot.WPF [28] library to display all the graphs used to present the simulations' data.

There is also an optional feature that makes use of the Python3 library NetworkX [29] to generate graphs given some parameters. This feature is integrated in the main solution, but you will need Python3 installed on the machine where you are running the program.

2.2.1.B Architecture Design

Let us analyse the overall architecture design of the solution.

The solution is composed of seven projects and can be divided in three major parts:

- User interface(Section 2.2.2): NetworkGameFrontend
- Backend services(Section 2.2.3): NetworkGameBackend, NetworkGameDataCollector and NetworkGenerator
- Libraries(Section 2.2.4): Network, NetworkUtils and PythonIntegration

The user interface contains the executable part of the solution, where the user can see, interact and control the application; the backend services have the logic to execute the main functionalities of the application and the libraries consists of data structures and useful functions used across all backend services.

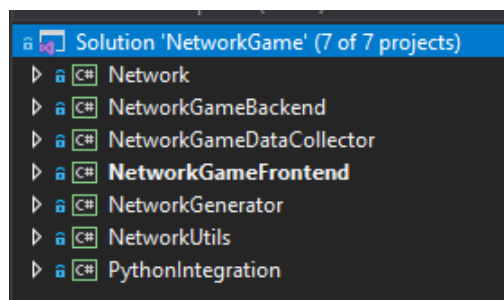


Figure 2.3: Visual Studio Solution

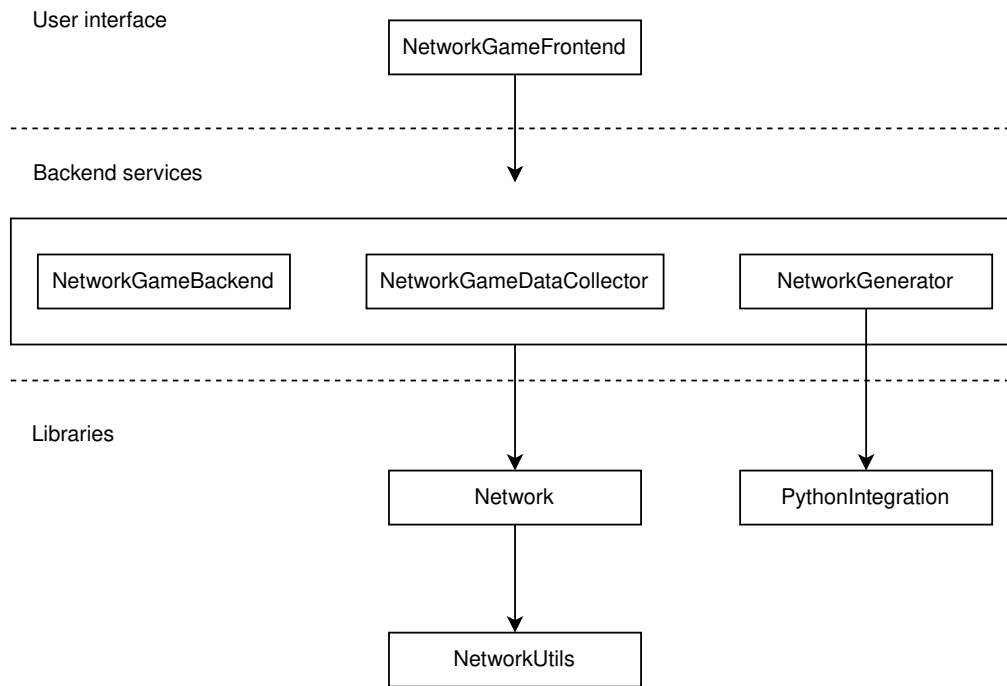


Figure 2.4: Architectural design

2.2.2 User interface

First let us present the User Interface (UI) for the NetworkGame solution. There are 3 types of windows that you can interact in the application, the MainWindow (Section 2.2.2.A) where you can see and configure the NetworkGame simulations; the PlotViewer (Section 2.2.2.B) where various plots are rendered to preset data from the simulations and miscellaneous windows that allow the application to prompt the user for inputs.

2.2.2.A MainWindow

There are two major parts on the MainWindow UI, the Network Viewer (Figure 2.5 left) and the controls (Figure 2.5 right and top).

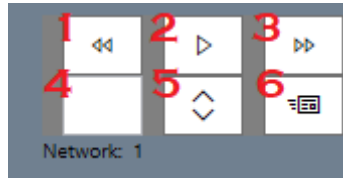


Figure 2.6: NetworkViewer controls

B – Controls The application controls include the top bar and the right side panel. They allow the user to control and configure the simulations.

The top bar lets us:

- File/Import: Import (Figure 2.7(a) 1) a network file into the application
- File/Generate: Generate (Figure 2.7(a) 2) a random connected network with some configurable parameters
- Network/Load Network: Load (Figure 2.7(b) 1) a network that was either imported or generated into the NetworkViewer
- Network/Export Network: Export (Figure 2.7(b) 2) a network into a file that the user can later import back into the application

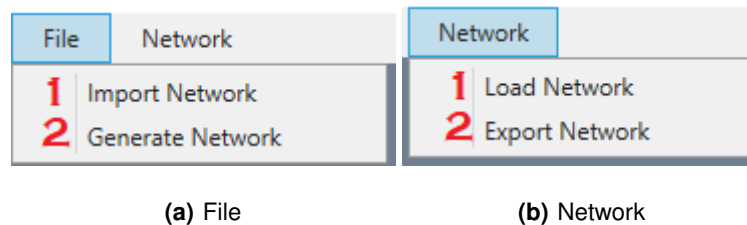


Figure 2.7: MainWindow Topbar

On the right side of the MainWindow we have the NetworkControls, where we can customize the simulations we want to run, see live data from the simulations and create graphs to display data from the NetworkGames. Using the NetworkControls we can:

- Choose (Figure 2.8 1) and configure (Figure 2.8 2) strategies for:
 - Routing (Figure 2.8 1)
 - Picking the next packet that a router will send (Figure 2.8 3)
 - Generating new packets (Figure 2.8 4)
 - Initialize the routing tables (Figure 2.8 5)

- Define the number of games (Figure 2.8 6) to run in parallel; the packet TTL (Figure 2.8 7) and choose if the runtime data should be saved (Figure 2.8 9)
- Start the simulation (Figure 2.8 8)
- Observe data from the selected router (Figure 2.8 10)
- Introduce the attacker into the network (Figure 2.8 11)
- Generate (Figure 2.8 13) and configure (Figure 2.8 12) plots
- Generate the default plots (Figure 2.8 14) (this are the graphs that we most used during our testing)
- Observe information (Figure 2.8 15) from the network

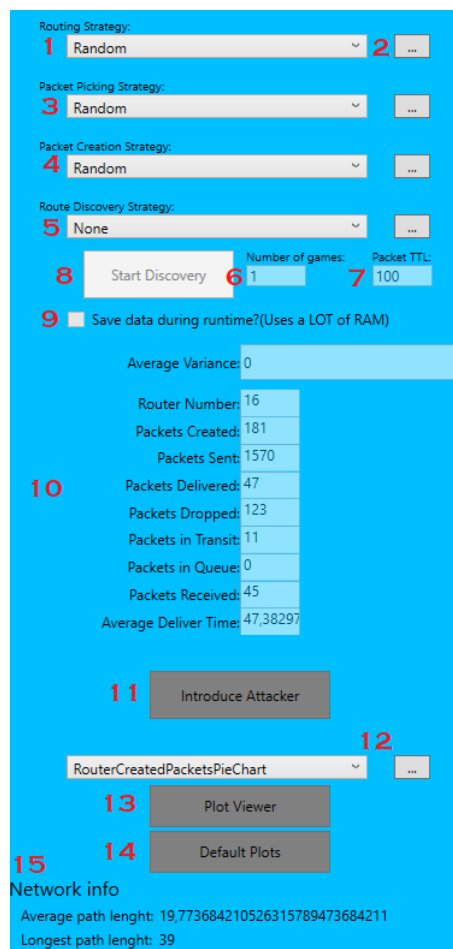


Figure 2.8: MainWindow controls

2.2.2.B PlotViewer

A PlotViewer Figure 2.9 opens every time the user wants to render a graph. There is no limit to how many PlotViewer windows can be open at the same time and they are all independent from each other. In this window has functionalities such as:

- Panning: Holding the left mouse button and moving the cursor allows the user to pan over the graph
- Zooming: Scrolling the mouse wheel lets the user zoom in and out on the graph
- Show Plot Information: Such as; if the data presented is an average of all running games or a specific game (Figure 2.9 1) and how many game cycles does it take to update the network (Figure 2.9 2) (how many cycles a unit on the x-axis represents)
- Reset View: Restores (Figure 2.9 3) the view of the graph to show the whole graph
- Save: Prompts (Figure 2.9 4) the user to choose a name and location to save a high resolution image of the graph

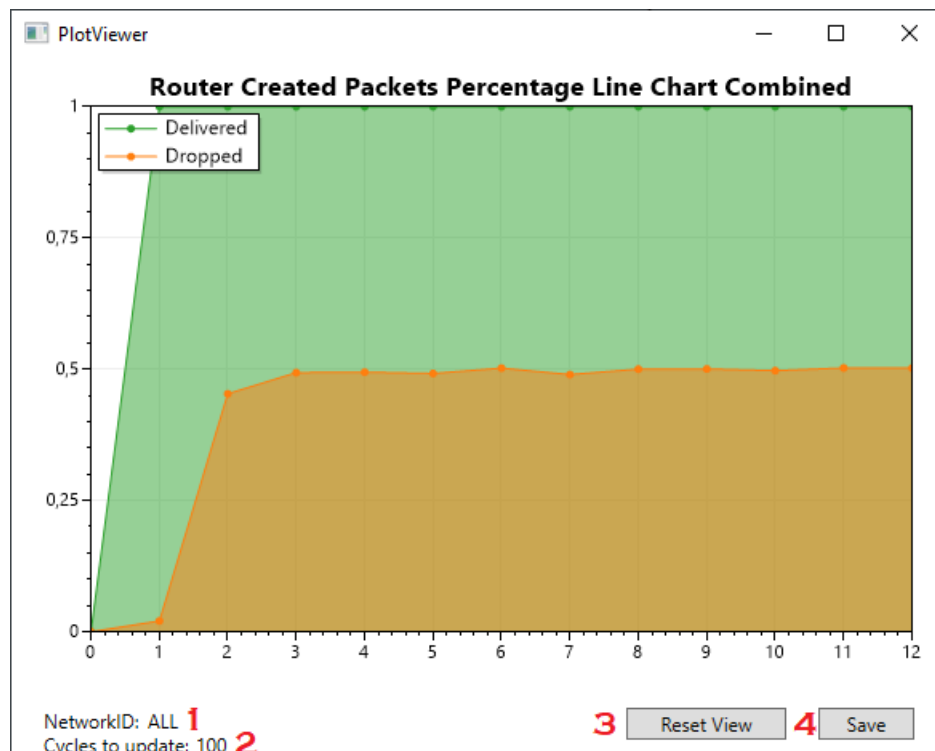


Figure 2.9: PlotViewer

2.2.3 Backend services

In this section we will present the Backend services that implement all the logic required to import/export/generate networks (Section 2.2.3.A); run the simulations (Section 2.2.3.B) and gather data to be presented in graphs (Section 2.2.3.C).

2.2.3.A NetworkGenerator

The NetworkGenerator project allows us to import/export networks in and out of the application and generate networks using a python library.

A – .network file extension With the objective of being able to use the same network across multiple simulations, we needed a way to store the network structure and load it into the application. For that, we defined a structure that represents the network and allows the program to import and export networks.

The .network file has the following structure:

Listing 2.1: .network file example

```
1 6                                # Number of routers in the network
2 0.1,0.2                          # -----
3 0.7,0.8                          #
4 0.3,0.9                          # Positions of each router on a
5 0.8,0.4                          # square with side length 1
6 0.85,0.5                         #
7 0.2,0.4                          #
8 1-2                              # -----
9 0-3                              #
10 0-1                             #
11 0-2                             # Links in the network, where the routers
12 3-4                             # are identified by the order in which
13 1-4                             # they appear in the "Positions" section
14 0-5                             #
15 1-5                             #
```

This example file Listing 2.1 will generate the following network when loaded into the application:

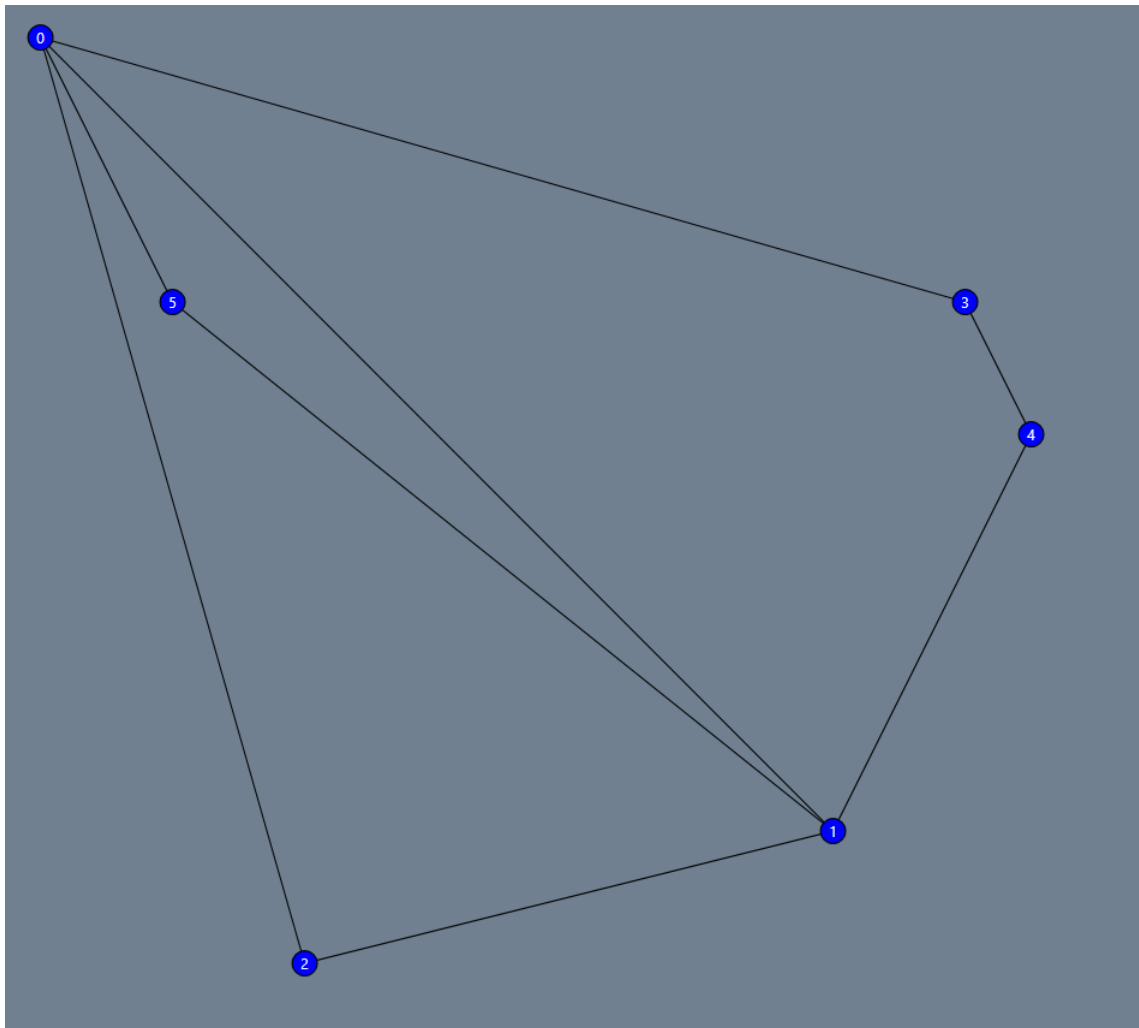


Figure 2.10: Network file example

B – Generator The NetworkGenerator project makes use of the Python library networkX to generate random connected graphs and export them into a .network file that we can use in simulations. It takes in 3 parameters: number of nodes, probability of 2 nodes having a link and the shortest length a link can have.

2.2.3.B NetworkGameBackend

In the NetworkGameBackend project we have only 2 classes: Game and GameMaster. The main objective of this project is to synchronize the multiple simulations that we can run at the same time and to expose controls of the games to the NetworkFrontend project.

A – Game A Game contains one network. It is responsible to run the simulation on the network and fire an event every time the network completes a cycle. This event will then be used to coordinate the information shown on the application as well as the data gathered in the plots.

B – GameMaster The GameMaster class controls all the Games that are being ran at the time, providing:

- Load balancing: When we have a lot of Games running at the same time, the GameMaster distributes the load across multiple threads to improve the performance of the simulations
- Synchronization: The GameMaster makes sure that all the games are in the same game cycle; allowing the plotting of data from multiple games in real time
- Expose controls: This class also exposes functions that allow to control the simulations, such as "Run/Pause" and "Introducing the Attacker" into the networks

2.2.3.C NetworkGameDataCollector

The NetworkGameDataCollector is composed of 4 data structures that hold data about the various entities of a network (routers, links, packets and the network itself). The main class on this project, NetworkDataCollector, serves as a middleman to access the simulation data in the rest of the solution.

2.2.4 Libraries

In this section we will take a look at the supporting libraries that contain data structures and functions that are used across the entire solution. There are 3 libraries: PythonIntegration (Section 2.2.4.C) and NetworkUtils (Section 2.2.4.B) that provide useful functions to other projects and the main library of the solution, the Network project (Section 2.2.4.A).

2.2.4.A Network project

The Network project contains the logic of how the network works and how it should "behave" during a game. Its main class, NetworkMaster, holds all the networks loaded in the application and makes them available in the whole solution. The classes that define the main Components in the games are also implemented on this project, containing their data structures and logic, as well as, the Strategies that change the behavior of the components and Route Discovery that allow us to set different initial conditions.

A – Components The components in a game are the network, the routers, the links and the packets. Every cycle of the game, every component performs a step, updating itself following the rules of the game. Going from the simplest to the most complex, we have:

Packet:

- Increments the number of steps

Link:

- Advances all the packets in the link one position forward (depending on the direction they are going)
- If a packet reaches a router, removes the packet from the link
- If a packet reaches the TTL, it is removed from the network

Router:

- Checks if there are packets in the queue that have reached the TTL, if so, removes them
- Tries to create a packet according to its Packet Creation Strategy
- If there are any packets in the queue:
 - It chooses one to send following its Packet Picking Strategy
 - Chooses which link to send the packet through, according to its Routing Strategy
 - Sends the packet and removes it from the queue

Network:

- Steps all the packets
- Steps all the links
- Steps all the routers
- Created the update objects that contain the changes in the network for that cycle

After every cycle, the network returns an UpdateState object that has the information about the changes that happened in that cycle. The UpdateState object corresponds to the network, in the same way that the UpdateRouter, UpdateLink and UpdatePackets objects correspond to the Router, Link and Packet components, respectively. This type of objects are called UpdateObjects and contain the following information:

UpdatePacket:

- Number of cycles the packet has been alive for
- Whether it has reached its destination or not
- Whether it has been dropped or not

UpdateLink:

- List of the packets currently in transit in the link
- List of packets that were in transit in the link, but reached a router in this cycle
- List of packets that were in transit in the link, but were dropped in this cycle

UpdateRouter:

- Number of packets in queue
- If a packet was created this cycle
- If a packet was sent this cycle

UpdateState:

- List of UpdatePackets
- List of UpdateLinks
- List of UpdateRouters
- Average variance
- Number of total cycles that the network has been running for

These objects are then used to update the visual interface of the application, as well as providing information to plot the simulation data in graphs.

B – Strategies The Strategies in the game define the behavior of the components and allow us to test different combinations of configurations for the network. There are 3 types of strategies: Routing Strategies(Section 2.2.5.A), Packet Creation Strategies(Section 2.2.5.B) and Packet Picking Strategies(Section 2.2.5.C).

C – Route Discovery By default, every link has the same probability of being chosen to send a packet through at the start of the game. The Route Discovery option allows us to set the initial probabilities in some other way, in order to test how the network adapts with different initial conditions.

2.2.4.B NetworkUtils

This library serves as a base for the entire solution, where we can define useful classes and functions that we want to make available to all projects. Currently there is only one class implemented here, the Property class; which lets us more easily implement user defined properties into the code.

For example, we can attribute some properties to a Strategy in the project, then when the user wants to configure that Strategy during runtime, a configuration window, as seen in Figure 2.11, appears with the properties dynamically loaded in, without needing a specific implementation for that Strategy.

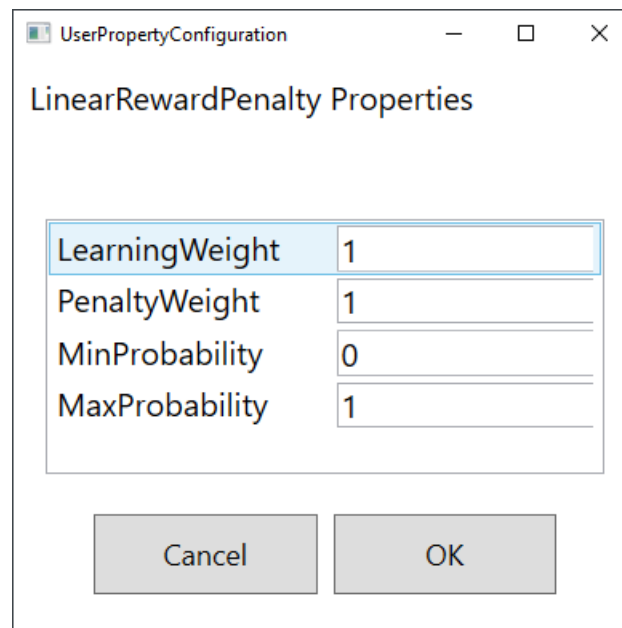


Figure 2.11: User configurable properties example

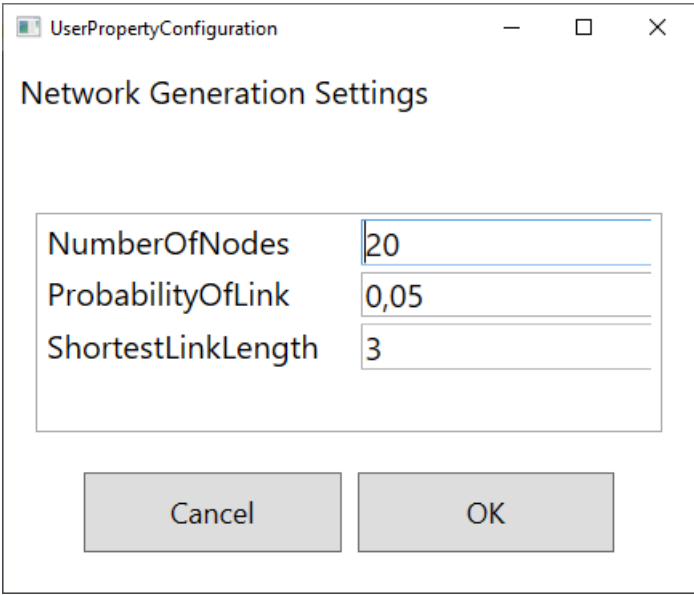
2.2.4.C PythonIntegration

The PythonIntegration project allows us to run Python scripts through the application. It makes use of the Property class to let the user input parameters to call the scripts and has a return object to allow the application to receive values from the script. This project is being used to generate networks directly in the application, using the Python library NetworkX [29].

It asks the user for inputs, as shown in Figure 2.12:

- Number of nodes in the network
- Probability of each pair of nodes having a link
- Length of the shortest link (that is used to compute the length of all other links)

And returns a string with the data to load the network in the .network file format.



The image shows a Windows-style dialog box titled "UserPropertyConfiguration" with a subtitle "Network Generation Settings". Inside the dialog, there is a table-like structure with three rows of settings:

NumberOfNodes	20
ProbabilityOfLink	0,05
ShortestLinkLength	3

Below the settings table, there are two buttons: "Cancel" and "OK".

Figure 2.12: Generate network properties example

2.2.5 Algorithm Implementations

In this section we will describe how and which behaviors are implemented. All strategies and route discovery algorithms were developed using the Strategy Design Pattern, that easily allows the creation of more behaviors, only needing to implement a class that inherits the base strategy.

2.2.5.A Routing Strategies

The Routing Strategies are the main focus of the work. They define how the router makes the decision to send the packets through the links and how it updates its routing table.

All routing strategies extend the "RoutingStrategy" class, that implements the logic for the Routing table and has two properties: minimum and maximum probabilities that every link can have of being chosen, implementing the artificial barriers described in [1].

First we will take a look at the routing table and then present the routing strategies that were implemented..

A – Routing table The routing table holds all the information necessary for a router to choose a link to send a packet through. Every router has a table and the table has an entry for every possible destination (every other router in the network); each entry has an entry for every link in the router. So

every router has a matrix with $N - 1$ rows and L columns, where N is the number of router in the network and L the number of links the router in question has.⁵

There are some special rules in place when updating values in the routing table in order to make it more consistent:

The update value is adjusted in proportion to the probability of the link to being chosen: When updating a value V_{new} in the routing table, the input value V_{in} is balanced according to the current probability P_{ij} of that link being chosen (as is evident in 2.1). This is to counter-balance the fact that a link with higher probability of being chosen, will have more opportunities to "learn", this is, to be updated.

When the input value is positive (reward), the value is reduced for links with high probability of being chosen. When the input value is negative (penalty), the value is reduced for links with low probability of being chosen:

$$V_{new} = \begin{cases} V_{in}(1 - P_{ij}) & \text{if } value > 0 \\ V_{in}P_{ij} & \text{if } value < 0 \end{cases} \quad (2.1)$$

This way of balancing the values allows for links that have a low probability of being chosen to be more rewarded when they have a good result and links with high probability of being chosen to be more punished when they have a bad result.

Updating a value updates all others in the same row: When updating a value, all other values in the same row are equally updated to maintain the sum of all probabilities equal to 1.

Consider *nodeX* with the following routing table (Table 2.2). Imagine now that the router wants to update the probability for choosing *linkC* when the destination of a packet is *routerY* by -0.1 . When the value for choosing *linkC* is decreased the difference has to be equally distributed across the other links, as we can see in Figure 2.13.

Table 2.2: Routing table for node X

	Link A	Link B	Link C
Router ?
Router Y	0.3	0.45	0.25
Router ?

⁵We are aware of possible memory problems that can arise from this configuration, as the memory necessary to hold all the tables grows exponentially with the number of routers in the network as well as how many links the average router has. We address this concerns in Section 4.3



Figure 2.13: Update value example with minimum barrier probability = 0.1

Updating a value would go over/under an artificial barrier: When updating a value would make it go over a maximum or below a minimum (1 and 0 by default respectively), the "excess" value is distributed across the other link probabilities.

For example, let's say we are running a game where the minimum probability a link can have to be chosen is 0.1. Imagine now that there is a *nodeX* with the following routing table (Table 2.3). Consider that the router wants to update the probability for choosing *linkC* when the destination of a packet is *routerY* by -0.1 . However, because of the artificial barrier in place, we can't set that probability to 0.05. So the "remainder" of the update has to be equally distributed across the other links, as we can see in Figure 2.14.

Table 2.3: Routing table for node X

	Link A	Link B	Link C
Router ?
Router Y	0.5	0.35	0.15
Router ?



Figure 2.14: Update value example with minimum barrier probability = 0.1

This rule is recursive, that is, when the routing table updates the other values, if any of those would also go over/under an artificial barrier, the difference is once again distributed across the remaining links that can still be updated.

B – Random routing strategy The Random routing strategy is the simplest of all the routing strategies. It consists on never updating the routing table. This means that whatever happens to the packets, it will not change the probability of a link being chosen.

By default, all links start with the same probability of being chosen ($1/L$, where L is the number of links the router has). By using the Random routing strategy, these probabilities never change, thus the routing is totally random and unpredictable.

We can combine this strategy with a route discovery algorithm to test some scenarios, for example, what would happen if the router always chose the link that will lead to the shortest path, using the Best Route only discovery, as presented in Section 3.1.2.

C – Linear Reward Inaction routing strategy The Linear Reward Inaction algorithm described in Section 1.3.2.A (1) only updates the values in the routing table when a packet reaches its destination. When a packet reaches its destination, all the routers where it passed through will update the probability of choosing the link that they picked to send the packet with a positive value V_{in} , computed using the Reward rate R_r and the number of steps that the packet took to reach the destination S_i :

$$V_{in} = \frac{R_r}{S_i^2} \quad (2.2)$$

D – Linear Reward Penalty routing strategy The Linear Reward Penalty algorithm described in Section 1.3.2.A (2) updates the values in the routing table whenever a packet either reaches its destination or its dropped (due to reaching the TTL or being caught by the Attacker). All the routers where the packet passed through will update the link that they chose to send the packet with a value V_{in} , computed using the Reward rate R_r , Penalty rate R_p and the number of steps that the packet took to reach the destination S_i :

$$V_{in} = \begin{cases} \frac{R_r}{S_i^2} & \text{if reached the destination} \\ -\frac{R_p}{S_{TTL}} & \text{if otherwise} \end{cases} \quad (2.3)$$

2.2.5.B Packet Creation Strategies

The Packet Creation Strategies define when a packet should create a new packet and add it to its queue.

A – Random creation strategy The Random creation strategy attempts to create a packet every cycle, with some user defined probability of actually creating a packet and adding it to the waiting queue. If a packet is to be created, its destination is also randomly selected from all routers, except for the one

creating the packet. Each router has a $1/(N - 1)$, where N is the number of routers in the network, probability of being the destination for the created packet.

2.2.5.C Packet Picking Strategies

The packet picking strategies define how a router chooses the next router to send from its waiting queue.

A – Random picking strategy The Random picking strategy will send a packet every cycle, when there are packets in the waiting queue. It gives every packet in the queue an equal chance of being chosen. The probability of each packet of being picked is $1/P$, where P is the number of packets currently in the queue.

B – FIFO picking strategy The FIFO picking strategy will send a packet every cycle, when there are packets in the waiting queue. The packets are sent in the same order they were received in a First In First Out fashion.

C – Random remove unreachable picking strategy The Random remove unreachable picking strategy will send a packet every cycle, when there are packets in the waiting queue. It gives every packet in the queue that can still reach its destination an equal chance of being chosen. We say a packet can still reach its destination if the difference between the TTL and the number of steps done by the packet (the number of cycles before it expires) is more than the length of the shortest path between the current router and the destination router. The probability of each packet of being picked is $1/P$, where P is the number of packets currently in the queue that fit the condition.

2.2.5.D Route Discovery

The Route discovery algorithms allow the user to set different initial values in the routing table to be able to test a variety of scenarios with distinct starting conditions.

A – No route discovery When the user does not set a route discovery algorithm, the default values in the routing table are applied. In this case, every link has the same chance of being chosen by a router (Figure 2.15). Each router sets the probability of choosing each of its links at $1/L$, where L is the number of links the router has.

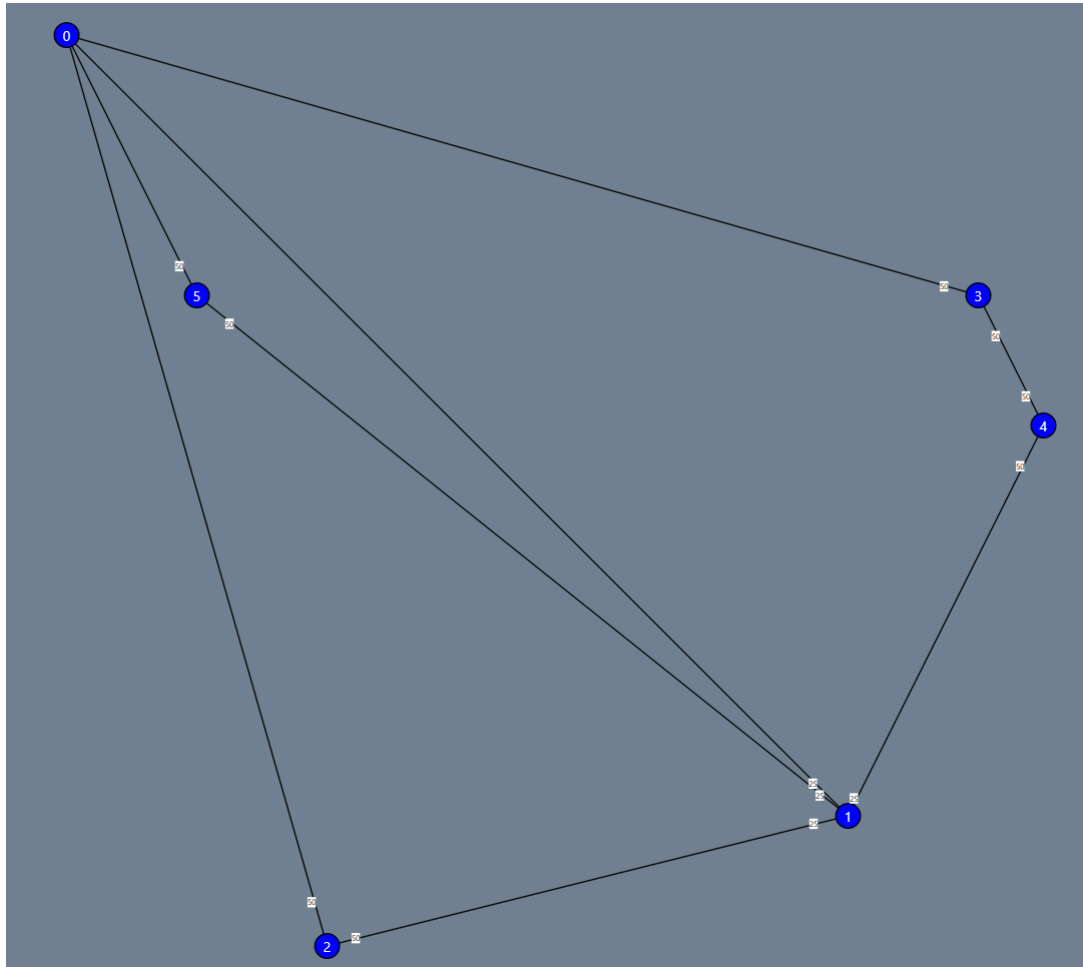


Figure 2.15: Link probabilities to send packets with destination router 0 using no route discovery

B – Best Route Only discovery The Best route only discovery performs the Dijkstra algorithm on the network to find the shortest routes between every pair of nodes. It then sets the probability of choosing the links in the shortest routes to 1 and all others to 0, so that at the start of a game only the shortest paths are chosen (Figure 2.16).

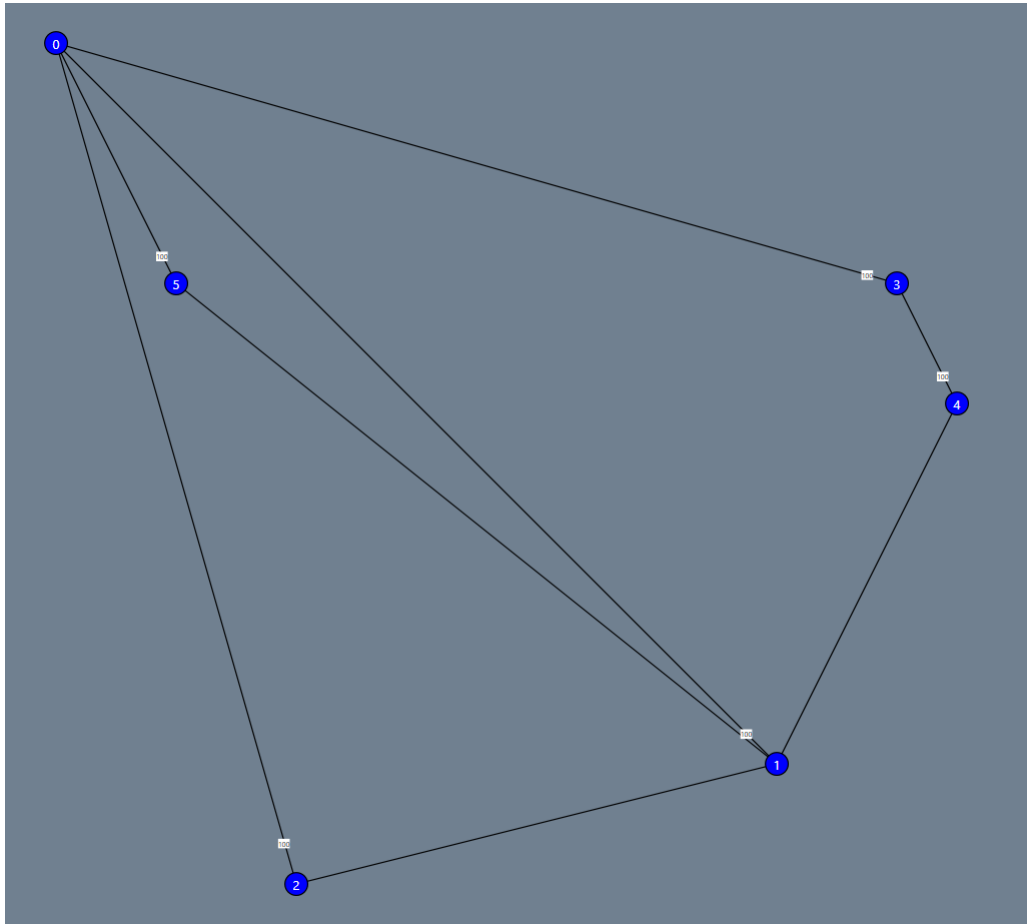


Figure 2.16: Link probabilities to send packets with destination router 0 using best route only discovery

C – Breadth First Route discovery The Breadth first route discovery performs a BFS for each node to compute the minimum number of hops necessary to reach each other router in the network. Then it gives each link a probability of being chosen that is inversely proportional to the minimum number of hops needed to reach the destination after the packet transverses the link (Figure 2.17). The probability of a link being chosen becomes $h_i / \text{Sum}H$ where h_i is the number of hops needed to reach the destination after taking link i and $\text{Sum}H$ is the sum of h for all links in the router.

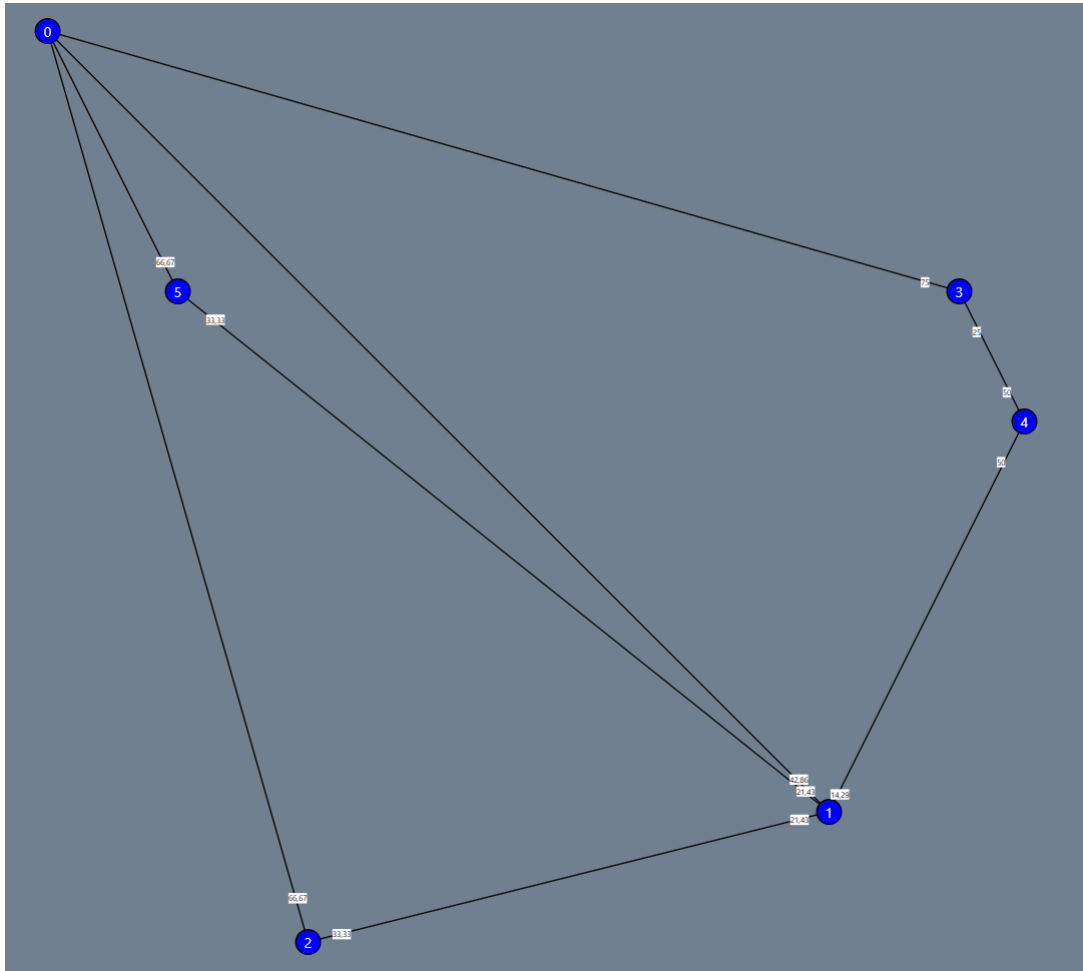


Figure 2.17: Link probabilities to send packets with destination router 0 using breadth first route discovery

D – Dijkstra Route discovery The Dijkstra route discovery performs the Dijkstra algorithm on the network to find the shortest routes between every pair of nodes. Then it gives each link a probability of being chosen that is inversely proportional to the minimum number of cycles (a link with length x takes x cycles to be transversed) needed to reach the destination after the packet transverses the link (Figure 2.18). The probability of a link being chosen becomes $l_i / \text{Sum}L$ where l_i is the number of cycles needed to reach the destination after taking link i and $\text{Sum}L$ is the sum of l for all links in the router.

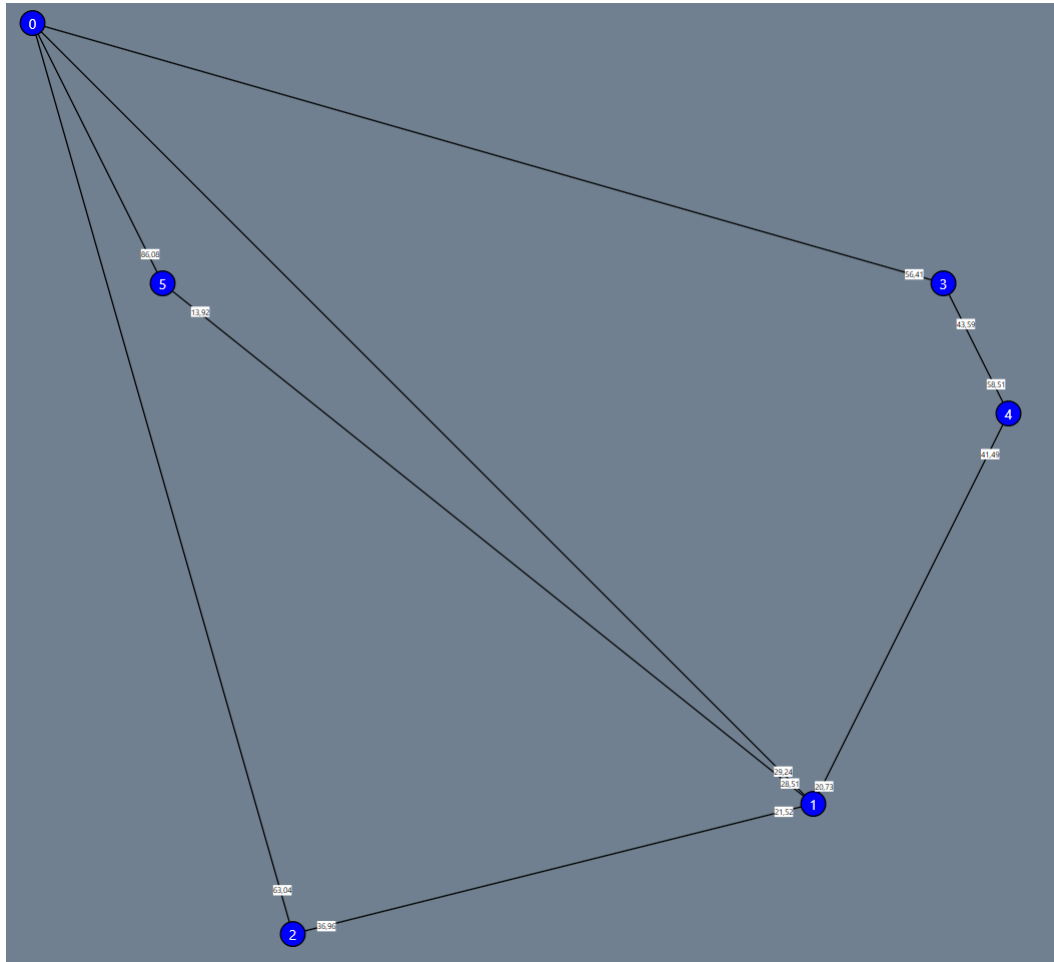


Figure 2.18: Link probabilities to send packets with destination router 0 using dijkstra route discovery

3

Simulations and Results

Contents

3.1 Random routing strategy	53
3.2 Linear reward penalty routing strategy	56

In this chapter we will present the simulations that were done and the results that we got from them. We tested a variety of scenarios to see how different algorithms and initial conditions lead to different results.

The variables that characterize each game are:

- Network size (number of nodes)
- Network average degree (average number of links per router)
- Route discovery (initial values in the routing tables)
- Routing strategy
- Packet creation strategy
- Packet picking strategy
- Artificial barriers threshold

For each simulation we will be looking at data from 6 different types of graphs generated while running the games. In all the graphs presented, the x-axis represents the time, where every unit is equal to 100 cycles in the game. The values shown in the graphs are always an average of the values from all the games running. The agents(described in Section 2.1.3) are randomly selected in each game; first we pick a pair of nodes to be the defender and the target of the packets sent by the defender, then we pick a node that is in the shortest path between the former 2, to be the attacker.

Average variance line chart Shows how much the values in the routing tables are changing on average for the past 100 cycles for all games running. It is computed by recording the values in the routing tables every 100 cycles and calculating the average of the difference between the values at the moment and 100 cycles ago.

Meaning of each line in the chart:

- Green line: Average variance
- Black dotted line: Interpolation of the average variance values to fit a smooth curve
- Blue line: Maximum average variance
- Red line: Minimum average variance
- Red horizontal line: User defined threshold (for reference only); y-axis value is a percentage of the maximum of the average variance

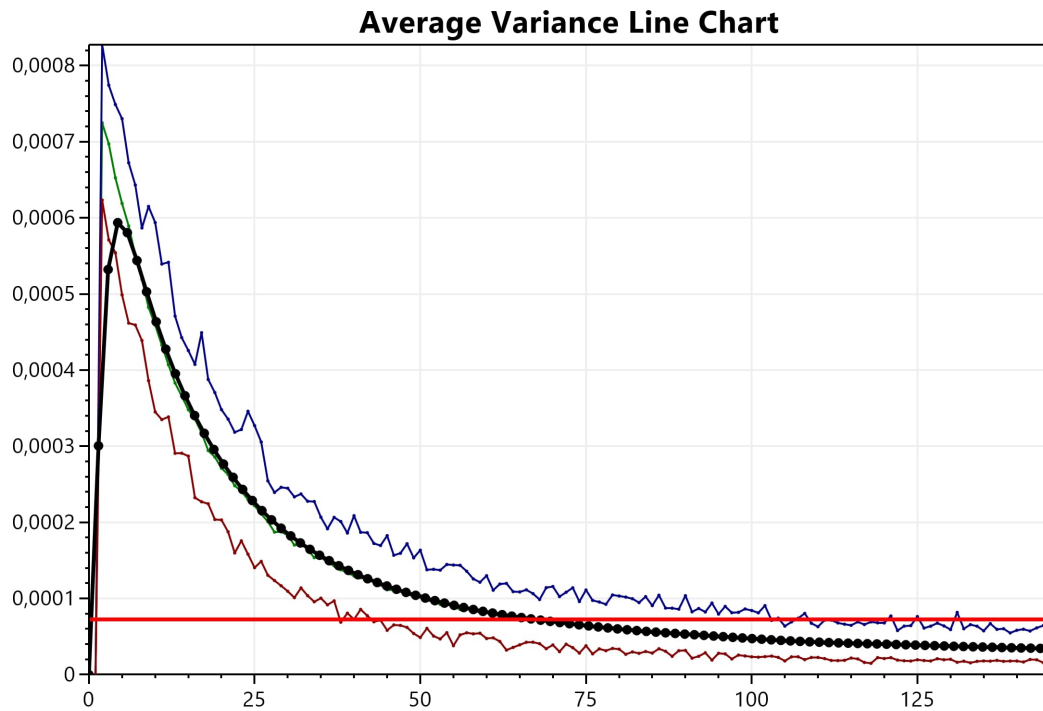


Figure 3.1: Average variance line chart example

Average packet queue time chart Shows the average time (number of cycles) packets are staying in router queues for the past 100 cycles for all games running.

Meaning of each line in the chart:

- Green line: Average packet queue time
- Black dotted line: Interpolation of the average packet queue time values to fit a smooth curve
- Blue line: Maximum average packet queue time (single game)
- Red line: Minimum average packet queue time (single game)

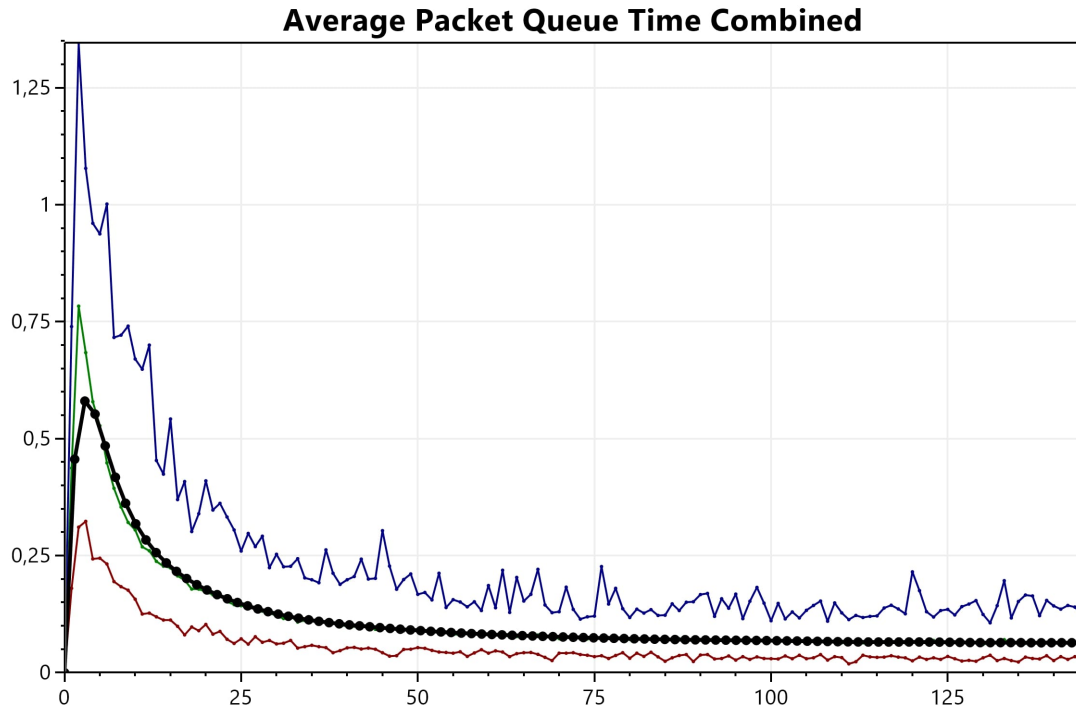


Figure 3.2: Average packet queue time line chart example

Average packet delivery time normalized chart Shows the average time normalized T_n that the packets took from their creation to reaching the destination for the past 100 cycles for all games running. The delivery time is normalized to account for differences in path lengths. The minimum normalized time a packet can take to reach the destination is 1, when the real time T_r is equal to the no delay time T_{nd} (shortest path + no queue times).

$$T_n = \frac{T_r}{T_{nd}} \quad (3.1)$$

Meaning of each line in the chart:

- Green line: Average packet queue time
- Black dotted line: Interpolation of the average packet queue time values to fit a smooth curve
- Blue line: Maximum average packet queue time
- Red line: Minimum average packet queue time

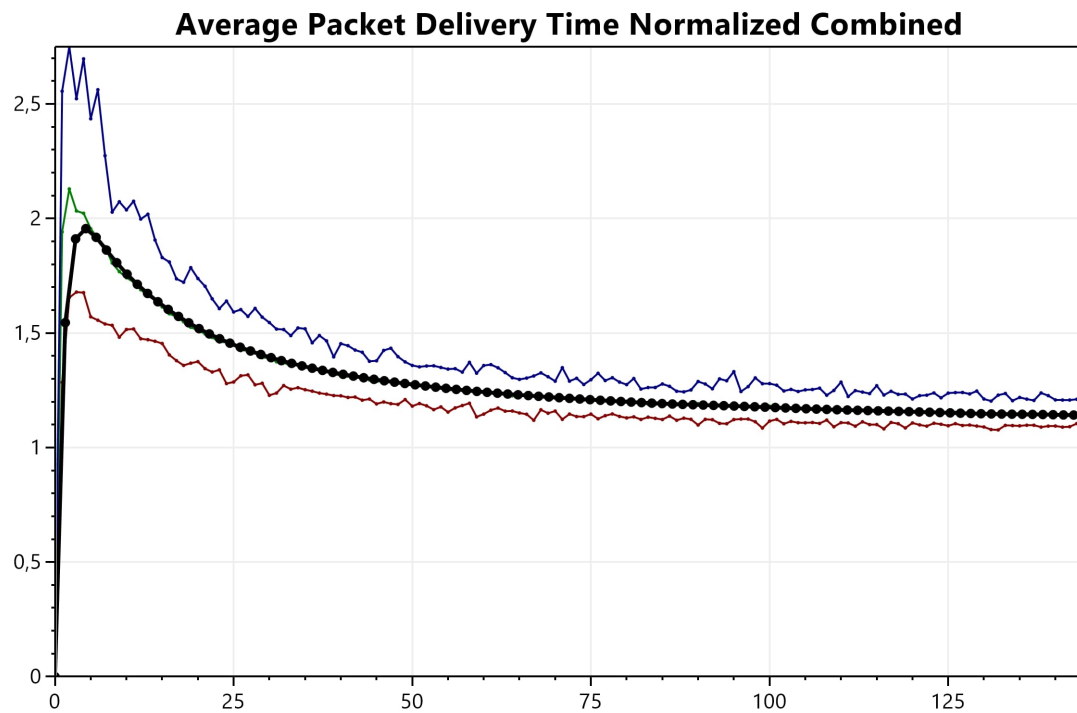


Figure 3.3: Average packet delivery time normalized example

Router created packets line chart Shows the cumulative number of packets that were created and if they were delivered or dropped.

Meaning of each line in the chart:

- Green area: Sum of packets delivered
- Yellow area: Sum of packets dropped

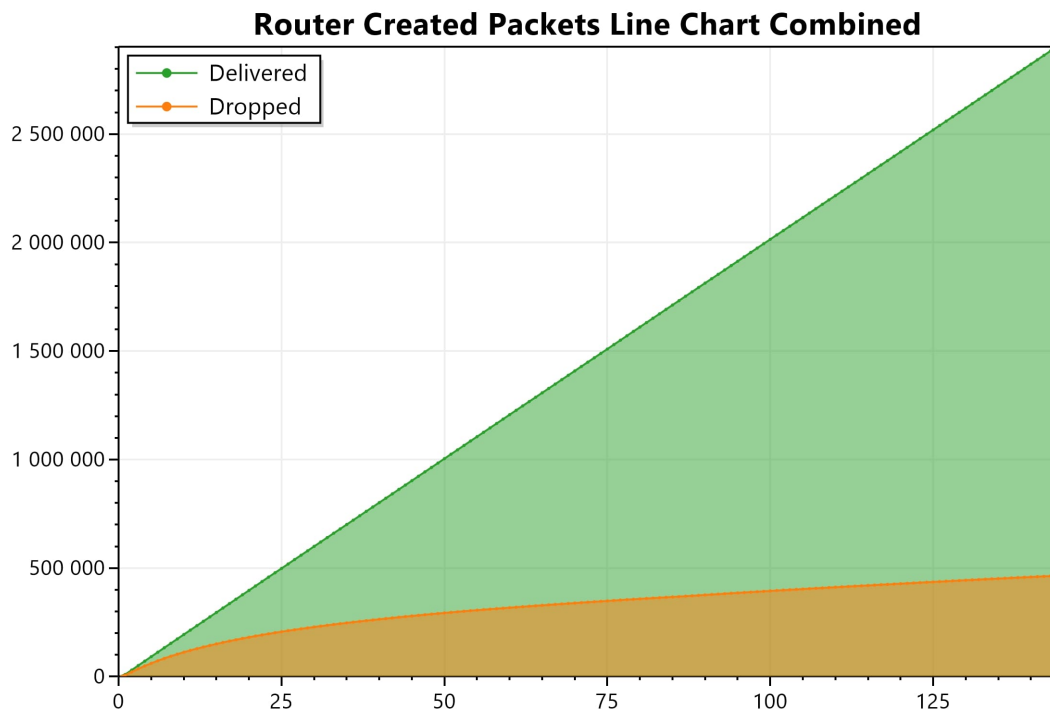


Figure 3.4: Router created packets line chart example

Router created packets percentage line chart Shows the percentage of packets that were delivered or dropped for the past 100 cycles for all games running.

Meaning of each line in the chart:

- Green area: Percentage of packets delivered
- Yellow area: Percentage of packets dropped

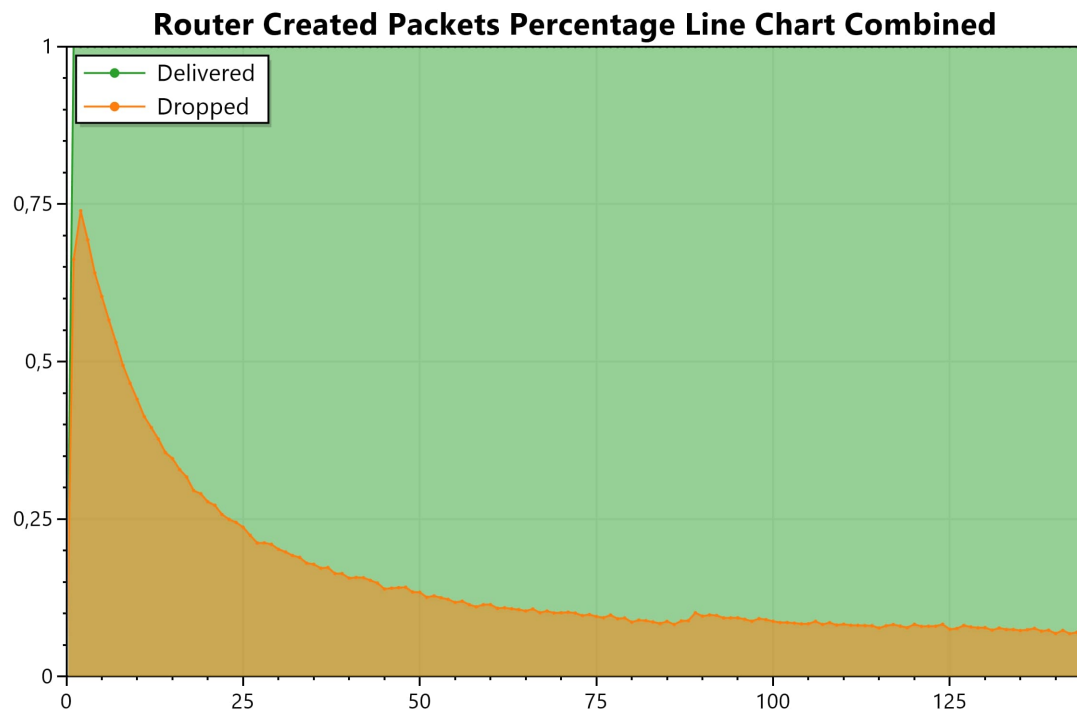


Figure 3.5: Router created packets percentage line chart example

Defender created packets percentage line chart Shows the percentage of packets created by the defender that were delivered or dropped for the past 100 cycles for all games running.

Meaning of each line in the chart:

- Green area: Percentage of packets created by the defender delivered
- Yellow area: Percentage of packets created by the defender dropped

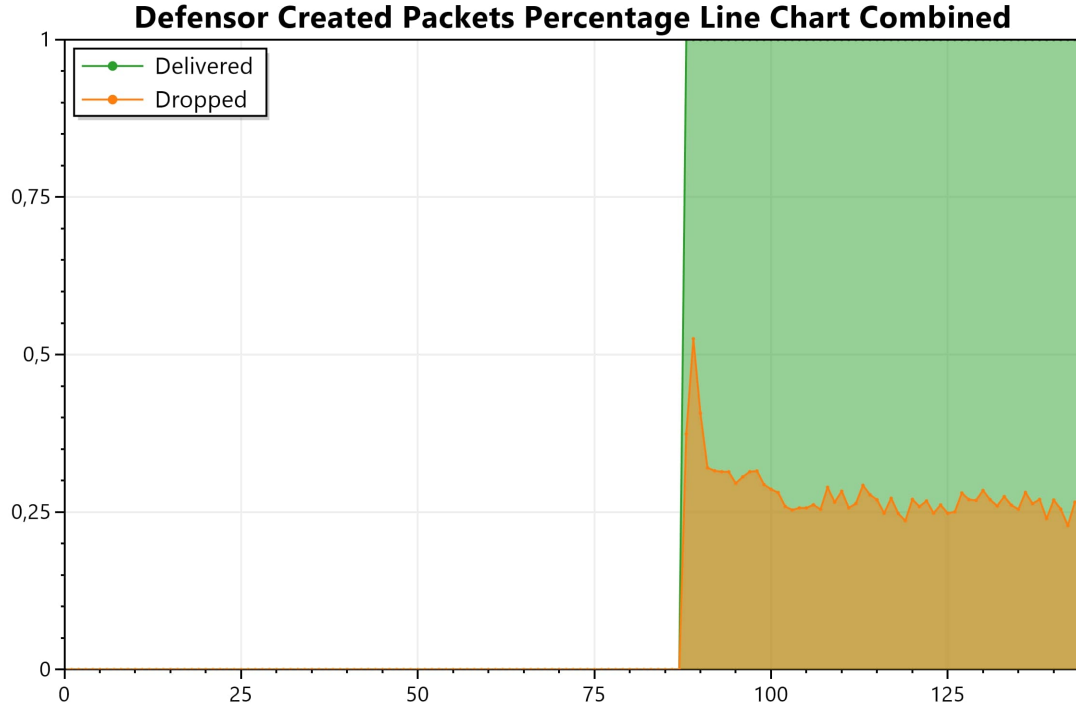


Figure 3.6: Defender created packets percentage line chart example

3.1 Random routing strategy

Let us start by presenting the results of running simulations using the random routing strategy as it will serve as our benchmark to evaluate other routing strategies.

The simulations in this subsection were ran on a medium network with 20 nodes and probability of each pair of nodes having a link equal to 0.1 (see Figure A.1).

For each scenario, we will present a table summarizing the results of the simulations. The full graphs can be viewed in Appendix A. Each column in the following tables has a reference to the corresponding graphs.

Let us also define some variables that we will use to give different properties to each game:

- P_c - Probability of packet creation: Represents the probability that each router has of creating a packet in each round.
- R_r - Reward rate: Weight given to the rewards when using learning algorithms
- R_p - Penalty rate: Weight given to the penalties when using learning algorithms

3.1.1 No route discovery

The simulations in this sub-subsection were ran with no route discovery. Analyzing Table 3.1 we can compare the results from running simulations with different probabilities of creating a new packet each cycle. With these simulations we want to see what happens when the packets are sent in completely random directions with different amounts of packets created in the network.

We can see that the average variance stays equal to 0 throughout the whole simulation as we are using the random routing strategy that never updates the values in the routing table.

The average packet queue time goes up as the network becomes more crowded with packets. The value for the simulations with $P_c = 0.1$ is low, at less than 1 cycle per packet, as the routers never have more than a couple packets in the queue at the same time. However, as we increase the P_c , this value quickly goes up, due to the packets being picked randomly (some packets never get picked and end up being dropped in the waiting queues due to reaching the TTL).

When looking at the average packet delivery time normalized, it actually starts off high even with low P_c , because of the random routing. The value goes up to 2.8 with $P_c = 0.5$ which is close to the maximum it can reach, as the packet TTL in all the simulations presented in this chapter is $TTL = 59$, which is close to 3 times the average path length of the network (see Figure A.1).

Analyzing now the number of packets created we can see that they go up proportionally as we increase P_c and that the percentage of packets delivered goes down rapidly. Because the routing is random, the fact that the packets spend more time in waiting queues makes it so that they have "less opportunities" to reach their destination. Considering now only the packets sent by the defender, we see that they reached their destination roughly 60% of the time when compared with the average percentage of packets delivered for the whole network with the same P_c .

Table 3.1: Comparison for different packet creation probabilities on network shown in Figure A.1 with random routing and no route discovery

	A(Figure A.2)	B(Figure A.3)	C(Figure A.4)
	$P_c = 0.1$	$P_c = 0.3$	$P_c = 0.5$
Average Variance	0	0	0
Average Packet Queue Time	0.9	9	20.5
Average Packet Delivery Time Normalized	2.15	2.45	2.8
Total Packets Created	1 400 000	4 150 000	6 900 000
Total Packets Delivered(%)	280 000(20%)	470 000(11.3%)	500 000(7.2%)
Total Packets Dropped(%)	1 120 000(80%)	3 680 000(88.7%)	6 400 000(92.8%)
Defender % Packets Delivered	12%	8%	4%
Defender % Packets Dropped	88%	92%	96%
Last cycle	7292	7118	7095
Introduce attacker cycle	3451	3259	3404

Note: The presented values are approximations and computed from the average results from 100 games in the same initial

conditions. The agents(described in Section 2.1.3) in each game are randomly selected.

3.1.2 Best path only

The simulations in this sub-subsection were ran with best path only discovery. Analyzing Table 3.2 we can compare the results from running simulations with different probabilities of creating a new packet each cycle. With these simulations we want to see how the network performs when the packets are always sent through the shortest paths with different amounts of packets created in the network. This means that the routing is deterministic instead of stochastic.

We can see that the average variance stays equal to 0 throughout the whole simulation as we are using the random routing strategy that never updates the values in the routing table.

The average packet queue time goes up as the network becomes more crowded with packets. The value for the simulations with $P_c = 0.1$ is almost 0, as the routers almost never have more than 1 packet in the queue. However, as we increase the P_c , this value quickly goes up, due to the packets being picked randomly (some packets never get picked and end up being dropped in the waiting queues due to reaching the TTL). We can see that the queue time values are slightly lower than the values with no route discovery (see Table 3.1) because the packets only follow the best route, thus there are no packets going in random directions cluttering the network.

When looking at the average packet delivery time normalized, it starts at an almost perfect time with low P_c , because the packets follow only the shortest paths and the waiting queues are almost always empty. The value goes up to 1.95 with $P_c = 0.5$ due to the network being more crowded and because central nodes in the network will be part of more "optimal paths", they will have bigger waiting queues. Still, this value is a lot better than the value in the same conditions but with no route discovery (see Table 3.1).

Analyzing now the number of packets created we can see that they go up proportionally as we increase P_c and that the percentage of packets delivered goes down as we increase P_c . Because the packets always take the shortest path, there are nodes that will have bigger waiting queues, as they are more central in the network, thus being part of more "optimal paths". This makes it that, as we increase P_c , this central nodes become more and more busy, dropping packets in the queue. Considering now only the packets sent by the defender, we see that they reached their destination roughly 30% of the time when compared with the average percentage of packets delivered for the whole network with the same P_c . At first thought, we might have expected that this value would always be 0, as the agents (described in Section 2.1.3) are randomly pick and the attacker is always positioned in a router that is the the shortest router between the defender and its packets destination. However, because the agents are randomly selected, there are cases where the defender and its destination router are neighbours,

this is, there are no routers between them in the shortest path. So, this value would indeed be 0 if the defender never sent packets to a neighbour.

Table 3.2: Comparison for different packet creation probabilities on network shown in Figure A.1 with random routing and best path only discovery

	A(Figure A.5)	B(Figure A.6)	C(Figure A.7)
	$P_c = 0.1$	$P_c = 0.3$	$P_c = 0.5$
Average Variance	0	0	0
Average Packet Queue Time	0.07	5.7	15.6
Average Packet Delivery Time Normalized	1.05	1.45	1.95
Total Packets Created	1 490 000	4 150 000	6 900 000
Total Packets Delivered(%)	1 465 000(98.3%)	2 890 000(69.6%)	3 300 000(47.8%)
Total Packets Dropped(%)	25 000(1.7%)	1 260 000(30.4%)	3 600 000(52.2%)
Defender % Packets Delivered	30%	22%	17.5%
Defender % Packets Dropped	70%	78%	82.5%
Last cycle	7502	7099	7137
Introduce attacker cycle	3706	3275	3156

Note: The presented values are approximations and computed from the average results from 100 games in the same initial conditions. The agents(described in Section 2.1.3) in each game are randomly selected.

3.2 Linear reward penalty routing strategy

3.2.1 No route discovery & low probability of creation

The simulations in this sub-section were ran with no route discovery and $P_c = 0.1$. Analyzing Table 3.3 we can compare the results from running simulations with different learning rates. With these simulations we want to see how the network performs with different learning rates when there is a low amount of packets being created.

Looking at the average variance, we can see that the changes from the start of the simulations to the end change proportionally to the learning rates; as we multiply the learning rates by 10, the average variance also increases roughly by the same factor. This makes sense because the values of the rewards and penalties are multiplied by the respective learning rate before updating the values in the routing table. We can also see that having low learning rates makes it so it takes a lot of cycles to get the point where routing tables are stable (consistent variance).

The average queue time starts low across the 3 simulations, as the main factor is how crowded the network is, this is, the value of P_c . However, we can see that, with higher learning rates, the queue times rapidly approach a value close to 0, the same value we see in the simulation results shown in Table 3.2($P_c = 0.1$), as the network learns the optimal paths.

Looking at the average packet delivery time normalized, we can see that in all cases, the starting

value is similar to the value we see in Table 3.1($P_c = 0.1$), as the initial values in the routing tables is the same. With high learning rates, this value quickly approaches the value in Table 3.2($P_c = 0.1$), as the network learns the optimal paths.

Analyzing the number and percentage of packets delivered and dropped we can see that the learning rates have a huge influence on how fast the network learns the best paths and how many packets are dropped in the process. For the simulation with the highest learning rates, we can see that they approach the values in Table 3.2($P_c = 0.1$) when the average variance stabilizes. Considering now only the packets sent by the defender, we can see that when the attacker is introduced, the delivery percentage takes a huge hit when compared to the network average. However, it recovers as the network learns to avoid sending the packets through the router occupied by the attacker, getting close to the network average. Comparing the packet delivery rate after some time to adjust to the attacker with medium to high learning rates with the values in Table 3.2($P_c = 0.1$), we can see a huge improvement, because the network makes use of less optimal paths to avoid the router occupied by the attacker when compared to using only the shortest path where the attacker can intercept the packets.

Table 3.3: Comparison for different learning rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery and $P_c = 0.1$

	A(Figure A.8) $R_r = 0.1$ $R_p = 0.1$	B(Figure A.9) $R_r = 1$ $R_p = 1$	C(Figure A.10) $R_r = 10$ $R_p = 10$
Average Variance	$8 * 10^{-6}$ to $7 * 10^{-6}$	$7.8 * 10^{-5}$ to $1.7 * 10^{-5}$	$6 * 10^{-4}$ to $3 * 10^{-5}$
Average Packet Queue Time	0.9 to 0.5	0.8 to 0.1	0.65 to 0.07
Average Packet Delivery Time Normalized	2.15 to 2.05	2.1 to 1.35	1.95 to 1.15
Total Packets Created	6 100 000	6 000 000	2 900 000
Total Packets Delivered(%)	1 800 000(30%)	4 050 000(67.5%)	2 420 000(83.4%)
Total Packets Dropped(%)	4 300 000(70%)	1 950 000(32.5%)	480 000(16.6%)
Network % Packets Delivered	20% to 37.5%	22% to 86%	26% to 92.5%
Defender % Packets Delivered	15% to 30%	42.5% to 72.5%	47% to 72.5%
Network % Packets Dropped	80% to 62.5%	78% to 14%	74% to 7.5%
Defender % Packets Dropped	85% to 70%	57.5% to 27.5%	53% to 27.5%
Last cycle	31043	30385	14640
Introduce attacker cycle	18956	19344	8906

Note: The presented values are approximations and computed from the average results from 100 games in the same initial conditions. The agents(described in Section 2.1.3) in each game are randomly selected. Values for defender only start being recorded when the attacker is introduced.

3.2.2 No route discovery & medium probability of creation

The simulations in this sub-section were ran with no route discovery and $P_c = 0.3$. Analyzing Table 3.4 we can compare the results from running simulations with different learning rates. With these simulations

we want to see how the network performs with different learning rates when there is a medium amount of packets being created.

Looking at the average variance, we can see that the changes from the start of the simulations to the end change proportionally to the learning rates; as we multiply the learning rates by 10, the average variance also increases roughly by the same factor. This makes sense because the values of the rewards and penalties are multiplied by the respective learning rate before updating the values in the routing table. When comparing with the values in Table 3.3 we observe that the values for the average variance are higher across the board when the value P_c is higher as well due to the fact that having more packets creates more opportunities for the network to learn, thus updating the values more often in the same amount of time (cycles).

The average queue time starts with similar values in the 3 simulations, as the main factor is how crowded the network is, this is, the value of P_c . As the simulations progress we can see that the average queue time goes down in all cases, but the simulation with learning rates equal to 1 seem to perform better than with higher learning rates. We can also see in Figure A.13 that the value goes up after the introduction of the attacker in the simulation with high learning rates and never goes back down. A possible explanation for this phenomenon might be that, because the values in the routing tables are updated so "aggressively" they might never stabilize on the "best values", this is, they never stay in the equilibrium. We can also note that the average queue time for the medium and high learning rates simulations is lower than the one in Table 3.2($P_c = 0.3$); because the network makes use of more routes, the "central" routers are less busy than when the packets only take the shortest paths. As for the simulation with lower learning rates, the average queue time was still going down when the simulation ended, so it is safe to assume that it would reach a value similar to the simulation with medium learning rates, or even a bit lower; however it would take a lot of time (cycles).

Looking at the average packet delivery time normalized, we can see that in all cases, the starting value is similar to the value we see in Table 3.1($P_c = 0.3$), as the initial values in the routing tables is the same. However the values do not go as low as the one in Table 3.2($P_c = 0.3$). The reasoning is the same as to why the average queue time is lower, this is, because the packets take other routes other than the shortest one, the average delivery time will end up higher, as the paths they transverse take more cycles to complete.

Analyzing the number and percentage of packets delivered and dropped we can see a similar trend to the previous metrics, in that the simulation with medium learning rates performed better than the one with high learning rates. For the simulation with learning rates equal to 1, we can see that, by the end of the simulation, it approaches the percentage of packets delivered by the network in Table 3.2($P_c = 0.3$), however the percentage of the packets sent by the defender delivered is doubled (from 22% in Table 3.2($P_c = 0.3$) to 46%) thanks to the network learning to "avoid" the infected router.

Table 3.4: Comparison for different learning rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery and $P_c = 0.3$

	A(Figure A.11)	B(Figure A.12)	C(Figure A.13)
	$R_r = 0.1$ $R_p = 0.1$	$R_r = 1$ $R_p = 1$	$R_r = 10$ $R_p = 10$
Average Variance	$1.18 * 10^{-5}$ to $8.3 * 10^{-6}$	$1.15 * 10^{-4}$ to $3 * 10^{-5}$	$8.5 * 10^{-4}$ to $3 * 10^{-4}$
Average Packet Queue Time	9 to 6.5	9.2 to 4.25	8 to 5.1
Average Packet Delivery Time Normalized	2.45 to 2.05	2.45 to 1.7	2.25 to 1.8
Total Packets Created	54 000 000	33 000 000	8 300 000
Total Packets Delivered(%)	17 000 000(31.5%)	19 000 000(57.6%)	4 700 000(56.6%)
Total Packets Dropped(%)	37 000 000(68.5%)	14 000 000(42.4%)	3 600 000(43.4%)
Network % Packets Delivered	12% to 56%	12.5% to 69%	16% to 60%
Defender % Packets Delivered	20% to 31%	37% to 46%	28% to 28%
Network % Packets Dropped	88% to 44%	87.5% to 31%	84% to 40%
Defender % Packets Dropped	80% to 69%	63% to 54%	72% to 72%
Last cycle	90629	55143	14139
Introduce attacker cycle	68346	42461	8978

Note: The presented values are approximations and computed from the average results from 100 games in the same initial conditions. The agents(described in Section 2.1.3) in each game are randomly selected. Values for defender only start being recorded when the attacker is introduced.

3.2.3 No route discovery & medium probability of creation & high penalty rate

The simulations in this sub-section were ran with no route discovery, $P_c = 0.3$, a high penalty rate ($R_p = 10$) and artificial barriers for the minimum probability (0.02). Analyzing Table 3.5 we can compare the results from running simulations with different reward rates ($R_r = \{0.1, 1, 10\}$). With these simulations we want to see how the network performs when the routers punish bad results disproportionately to how they reward good results from delivering the packets.

Looking at the average variance, we can see that the changes from the start of the simulations to the end are very similar for reward rates equal to 0.1 and 1, as the main factor to how much the average variance changes is the bigger learning rate, in this case the penalty rate. With reward rate equal to the penalty rate, the average variance goes a bit lower, but not by a factor of 10 like when both learning rates go up.

Regarding the average queue times, they are again very similar, with the simulation with high reward rate performing just a little bit better. One thing to note is that, with everything else equal, the network with a barrier for minimum probability of 0.02 performs worse than not having a barrier (Table 3.4(C)), which makes sense because by not allowing links to have 0 probability of being picked, sometimes packets can be sent in the "wrong direction", leading to them being more time in the network and, consequently, more time in router waiting queues.

Looking at the average delivery times, it is once again similar across the board, with simulation C having a small advantage. Comparing simulation C with Table 3.4(C), it performs slightly worse, in the same vein as the average queue time.

Analyzing the number and percentage of packets delivered and dropped we can see that the higher the learning rate, more packets are delivered, both in the network average as well as the defender. Looking at simulation C and comparing it with the results in Table 3.4(C), we see that having the artificial barrier seems to hurt the overall network performance, but might slightly help the defender deliver packets, although the difference is minimal.

Table 3.5: Comparison for different reward rates on network shown in Figure A.1 with linear reward penalty routing, no route discovery, $P_c = 0.3$, high penalty rate and artificial barriers ($MinP = 0.02$)

	A(Figure A.14)	B(Figure A.15)	C(Figure A.16)
	$R_r = 0.1$ $R_p = 10$	$R_r = 1$ $R_p = 10$	$R_r = 10$ $R_p = 10$
Average Variance	$8.5 * 10^{-4}$ to $5.8 * 10^{-4}$	$8.7 * 10^{-4}$ to $5.7 * 10^{-4}$	$8.4 * 10^{-4}$ to $4.8 * 10^{-4}$
Average Packet Queue Time	8.3 to 6.75	8.25 to 6.6	8.1 to 6
Average Packet Delivery Time Normalized	2.35 to 2.05	2.35 to 2.02	2.25 to 1.95
Total Packets Created	6 250 000	5 600 000	8 200 000
Total Packets Delivered(%)	2 450 000(39.2%)	2 250 000(40.2%)	4 050 000(49.4%)
Total Packets Dropped(%)	3 800 000(60.8%)	3 350 000(59.8%)	4 150 000(50.6%)
Network % Packets Delivered	15% to 42%	15% to 43%	17% to 52%
Defender % Packets Delivered	22% to 23%	26% to 27%	27% to 29%
Network % Packets Dropped	85% to 58%	85% to 57%	83% to 48%
Defender % Packets Dropped	78% to 77%	74% to 73%	73% to 71%
Last cycle	10620	9628	13992
Introduce attacker cycle	6291	5775	9009

Note: The presented values are approximations and computed from the average results from 100 games in the same initial conditions. The agents(described in Section 2.1.3) in each game are randomly selected. Values for defender only start being recorded when the attacker is introduced.

3.2.4 No route discovery & high probability of creation

The simulations in this sub-section were ran with no route discovery, $P_c = 0.5$, medium learning rates ($R_r = 1$, $R_p = 1$) and artificial barriers for the minimum probability (0.05). Analyzing Table 3.6 we can compare the results from running simulations with different packet picking strategies (see Section 2.2.5.C): random(A) where all packets in the queue have an equal chance of being picked, FIFO(B) where packets go out the router in the same order they entered and randomRU(C) where the packets that can no longer reach their destination are ignored and the remaining have an equal chance of being picked.

Looking at the average variance, we can observe that the changes from the start of the simulations to the end for random(A) and FIFO(B) are basically the same, but randomRU(C) manages to be a bit more stable in the end.

Regarding the average queue times, random(A) and FIFO(B) are again similar and randomRU(C) performs a lot better than the 2. Even when comparing with the results in Table 3.2(C) the queue time is lower as packets that have no chance of reaching their destination are ignored.

Considering now the average delivery times, as expected FIFO(B) performs the worse out of the 3, because every packet is guaranteed to wait for every other packet that was already in the queue to be sent. Next is random(A), which has a bigger average delivery time when compared to randomRU(C), because even though both randomly pick the next packet to be sent, randomRU(C) has a smaller pool to choose from, as it ignores packets that can no longer reach their destination. All 3 strategies perform worse than the values in Table 3.2(C) in terms of delivery time, but that is also attributed to using stochastic routing instead of just the shortest paths.

Analyzing the number and percentage of packets delivered and dropped we can see that for the network average, randomRU(C) manages to deliver about double the amount that random(A), FIFO(B) and the values in Table 3.2(C) do. However, when considering only the packets sent by the defender, randomRU(C) actually performs worse than random(A) and FIFO(B). One explanation for this phenomenon may be that, because randomRU(C) leans more towards the shortest paths than random(A) and FIFO(B) (as is evident by comparing the average packet delivery times), when the attacker is introduced, as it infects a router in the shortest path between the defender and its destination, it will negatively affect a network using the randomRU(C) strategy more than the first 2 strategies.

Table 3.6: Comparison for different packet picking strategies (see Section 2.2.5.C) on network shown in Figure A.1 with linear reward penalty routing, no route discovery, $P_c = 0.5$, $R_r = 1$, $R_p = 1$ and artificial barriers ($MinP = 0.05$)

	A(Figure A.17)	B(Figure A.18)	C(Figure A.19)
	Random	FIFO	RandomRU
Average Variance	$7.2 * 10^{-5}$ to $6.3 * 10^{-5}$	$7.2 * 10^{-5}$ to $6.4 * 10^{-5}$	$7 * 10^{-5}$ to $5.1 * 10^{-5}$
Average Packet Queue Time	20.3 to 19.6	21 to 19.5	15 to 13.75
Average Packet Delivery Time Normalized	2.75 to 2.6	3.35 to 3.2	2.5 to 2.28
Total Packets Created	26 500 000	26 000 000	26 500 000
Total Packets Delivered(%)	4 000 000(15.1%)	3 500 000(13.5%)	7 000 000(26.4%)
Total Packets Dropped(%)	22 500 000(84.9%)	22 500 000(86.5%)	19 500 000(73.6%)
Network % Packets Delivered	8% to 19%	8% to 17%	11% to 32%
Defender % Packets Delivered	17.5% to 19%	17% to 19%	14% to 14%
Network % Packets Dropped	92% to 81%	92% to 83%	89% to 68%
Defender % Packets Dropped	82.5% to 81%	83% to 81%	86% to 86%
Last cycle	26854	26283	26846
Introduce attacker cycle	20363	20399	20526

Note: The presented values are approximations and computed from the average results from 100 games in the same initial conditions. The agents(described in Section 2.1.3) in each game are randomly selected. Values for defender only start being recorded when the attacker is introduced.

In section Section 4.2 we will go over the main conclusions that we were able to draw from all this experiments.

4

Conclusion

Contents

4.1 Summary	65
4.2 Overview	65
4.3 Future work	67

With computer networks expanding in both size and complexity, as well as being used in more critical areas, like online banking, it is important to keep reinforcing the security of these systems, without taking a toll on the performance. Networks nowadays are very secure against attacks that want to steal sensitive data, with the use of authentication methods and cryptography, but are still vulnerable to attacks that attempt to disrupt their normal operation like packet dropping attacks and denial of service (DoS).

With our work we aim to increase security in computer networks, not by making messages harder to decrypt, but by making it harder to intercept them in the first place. To achieve that, we proposed the use of stochastic routing protocols that make use of multiple paths in the network, decreasing the chance of all packets getting intercepted by an attacker, while trying to maintain or even increase the performance of the network.

4.1 Summary

In summary, we developed an application to simulate computer networks as games and present the data from those simulations. With the results we were able to analyze the impact that different strategies and algorithms have on the performance of the network and the protection that the network is able offer to a node that is being targeted by an attacker.

We saw that, utilizing learning algorithms to dynamically change the probabilities of choosing the next hop significantly increases the amount of packets that a network is able to deliver when a user is being targeted by an attacker that aims to remove its packets from the network; more than doubling it in some cases (see Section 3.2.1).

We also noticed that the impact on the performance varies, mostly depending on how busy the network is; when the network does not have many packets in circulation, deterministic routing prioritizing the shortest routes yields faster deliveries in general; however, in networks with a high amount of traffic, utilizing stochastic routing to make use of different paths that may not be optimal, increases the effectiveness of delivering packets when compared to more deterministic approaches.

With our results we were able conclude that, if security is the main concern in a network, utilizing stochastic routing is a valid option to consider, because it increases the robustness of the network while, in the worst case, having a minimal impact on the performance and, in the best case, even increasing the network performance through the indirect loading balancing that making use of multiple paths provides.

4.2 Overview

In order to investigate our proposition, we developed an application(presented in Section 2.2) that allows us to simulate networks and test the behavior of different algorithms by running multiple simulations as

games. The simulations ran in our application have 3 main characteristics that facilitate the study of our proposition:

- Instead of running a single game at the time, our application allows us to run any number of games in parallel with the same configurations and combine the results, in order to reduce the uncertainty that comes from using stochastic algorithms.
- It is possible to configure multiple behaviors of the network, not only the routing algorithms, but others like how the routers create packets and how the routers choose which packet to send next. It is also easy to expand these behaviors and create more as all the code of the application is accessible.
- Data collection and presentation are deeply integrated in the simulations, which allows us to visualize just about any data we need from the games. This feature is also expandable with new graphs and types of data that we might want to study.

Then we performed multiple simulations using the application, where we tested and analysed various scenarios. We started by doing 2 tests using the random routing strategy(see Section 2.2.5.A) and variant packet creation probability to serve as benchmarks for the other simulations:

- No route discovery(described in Section 3.1.1) : Allows us to analyse how the network would perform in complete randomness.
- Best path only discovery(described in Section 3.1.2) : Lets us see how the network performs using a deterministic approach that only makes use of the shortest paths.

Following these tests, we ran network game simulations using the linear reward penalty routing algorithm(presented Section 2.2.5.A) under different circumstances and compared the results between the different runs and the previous 2 benchmark tests:

- Low packet creation probability(Section 3.2.1) : Shows the differences between learning rates in a network with low amount of traffic.
- Medium packet creation probability(Section 3.2.2) : Shows the differences between learning rates in a network with medium amount of traffic.
- Medium packet creation probability with high penalty rate(Section 3.2.3) : Lets us analyse the differences in changing only the reward rate, while keeping a high penalty rate in a network with medium amount of traffic.
- High packet creation probability(Section 3.2.4) : Allows us to see the differences between packet picking strategies in a network with high amount of traffic.

The simulations produced data presented in the form of graphs by the application we developed. The data collected was then compiled in tables (presented across Chapter 3), with simulations with different configurations side by side for easier comparison. By analyzing this results we were able to draw some conclusions, like:

- The rate at which the network is able to "learn" greatly correlates with how many packets are being sent (how many chances it has to learn) and the value of the learning rates (how much do the values in the routing tables change each time it learns).
- When the network is not crowded (the probability of a router to create a packet is low), the values in the routing tables converge to mainly prefer the shortest paths, becoming almost a form of deterministic routing. However, as the network becomes more crowded, the values in the routing tables tend to be more distributed; because the majority of shortest paths contain a small set of central routers that quickly become overwhelmed if all packets in the network are sent through them, making it a better option to make use of longer paths to avoid spending a lot of time in waiting queues.
- In less crowded networks, the use of stochastic routing tends to perform worse than simply using the shortest paths when there is no attacker. As the network becomes more crowded, the stochastic routing approaches the deterministic routing in performance and even surpasses it, as it provides a form of load balancing, preventing more central routers of being clogged.
- Overall, the use of stochastic routing is a positive for a router that is being targeted by an attacker, even when the network average performance is worse.
- Other than the routing, other decisions, like the order in which the packets are sent from the waiting queues, can greatly affect the performance of the network and its ability to "protect" the defender.

4.3 Future work

In this section we will suggest what could possibly be done in the future to continue the work in this area, with the help of the work that was done.

4.3.1 Improving the application

Concerning the application that was developed, some points that could be worked on are:

- Documentation; not much has been written about the code solution itself, apart from Section 2.2 and some comments on the code.

- Application UI; the UI was developed enough to be usable, but some work can be done to make it more user friendly and a better experience overall to use.
- In Section 2.2.5.A we brought up a possible memory concern regarding the way the routing table is implemented, as its size grows exponentially with the number of nodes in the network. One way to solve this problem is to group nodes that are close together in a region. The routers then have a table where each entry is a router in their region and another table where each entry is a region that is not their own. This way it is possible to massively reduce the number of entries each routing table needs. This solution can also be implemented with any number of levels, meaning that a router can be part of a region, sub-region, sub-sub-region...
- Implementing more strategies and functionalities that future investigators might find useful to study
- Implementing new graphs and collect different types of data to better study the results of changing strategies and configurations
- Make the simulations more realistic; like for example attributing a more realistic amount of time(cycles) to different actions (travel time between routers, time to send a packet, time it takes for the router to make decisions, etc.).

4.3.2 Further investigation

In regards to the matter of the study, the use of stochastic routing algorithms to enhance network security, a lot more could be explored with the help of the implemented solution:

- Simulations with different networks; varying in both size and topology.
- Studying new routing strategies using other learning algorithms; like the Lagging Anchor Algorithm discussed in Section 1.3.2.A.
- Studying new packet creation strategies; like creating packets following some sort of distribution or even a way to simulate dividing and replicating packets so that some packets can be intercepted but the message still gets fully transmitted (see hamming codes [\[30\]](#))
- Studying new packet picking strategies; like some "smart" way of choosing the packets to send in order to minimize the quantity of packets dropped.

The study of all this possibilities could lead to a real world implementation of a stochastic routing algorithm in certain scenarios where the concerns with packet interception are high and security is the highest priority.

Bibliography

- [1] A. Yazidi, D. Silvestre and B. J. Oommen, "Solving Two-Person Zero-Sum Stochastic Games With Incomplete Information Using Learning Automata With Artificial Barriers," in *IEEE Transactions on Neural Networks and Learning Systems*, doi: 10.1109/TNNLS.2021.3099095.
- [2] P. Sastry, V. Phansalkar, and M. Thathachar, "Decentralized learning of nash equilibria in multi-person stochastic games with incomplete information," *IEEE Transactions on systems, man, and cybernetics*, vol. 24, no. 5, pp. 769–777, 1994.
- [3] S. Lakshmivarahan and K. S. Narendra, "Learning algorithms for twoperson zero-sum stochastic games with incomplete information: A unified approach," *SIAM Journal on Control and Optimization*, vol. 20, no. 4, pp. 541–552, 1982
- [4] Lu, Xiaosong and Howard M. Schwartz. "Decentralized learning in two-player zero-sum games: A LR-I lagging anchor algorithm." *Proceedings of the 2011 American Control Conference* (2011): 107-112.
- [5] F. A. Dahl, "The lagging anchor model for game learning — a solution to the crawford puzzle," *Journal of Economic Behavior & Organization*, vol. 57, pp. 287–303, 2005
- [6] F. A. Dahl, "The lagging anchor algorithm: reinforcement learning in two-player zero-sum games with imperfect information," *Machine Learning*, vol. 49, pp. 5–37, 2002.
- [7] M. Thathachar and P. Sastry, *Networks of Learning Automata: Techniques for Online Stochastic Optimization*. Boston, Massachusetts: Kluwer Academic Publishers, 2004.
- [8] S. P. Singh, M. J. Kearns, and Y. Mansour, "Nash convergence of gradient dynamics in general-sum games," in *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, San Francisco, CA, USA, 2000, pp. 541–548.
- [9] M. Bowling and M. Veloso, "Multiagent learning using a variable learning rate," *Artificial Intelligence*, vol. 136, no. 2, pp. 215–250, 2002.

- [10] Rasmusen, Eric (2007). Games and Information (4th ed.). ISBN 978-1-4051-3666-2.
- [11] Alex Hinds, Michael Ngulube, Shaoying Zhu, and Hussain Al-Aqrabi, "A Review of Routing Protocols for Mobile Ad-Hoc NETWORKS (MANET)," International Journal of Information and Education Technology vol. 3, no. 1, pp. 1-5, 2013.
- [12] R. Akbani, T. Korkmaz, and G .V. S. Raju, "HEAP: A packet authentication scheme for mobile ad hoc networks," Ad Hoc Networks, vol. 6, no. 7, pp. 1134–1150, 2008.
- [13] Shyam Nandan Kumar, "Review on Network Security and Cryptography ." International Transaction of Electrical and Computer Engineers System, vol. 3, no. 1 (2015): 1-11. doi: 10.12691/iteces-3-1-1.
- [14] C.Perkins and P.Bhagwat,"Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers," in Proc. of Sigcomm conference on Communications architectures, protocols and applications, London, England, UK, 1994, pp. 234-244
- [15] D. B. Johnson and D. A.Maltz,"Dynamic Source Routing in Ad Hoc Wireless Networks," Mobile Computing, T. Imielinski and H. Korth, Ed. Kluwer Academic Publishers, 1996, vol. 5, pp. 153-181.
- [16] C. E. Perkins and E. M.Royer,"Ad-hoc On-Demand Distance Vector Routing," in Proc. of the 2nd IEEE workshop on mobile computing systems and applications, 1997, pp. 1-11.
- [17] W. A. Mobaideen, H. M. Mimi, F. A. Masoud, and E. Qaddoura, "Performance evaluation of multicast ad hoc on-demand distance vector protocol," Computer Communications, vol. 30, no. 9, pp. 1931–1941, 200
- [18] C. E. Perkins and E. M. Royer,"Multicast operation of the ad-hoc on-demand distance vector routing protocol," in Proc. of 5th annual ACM/IEEE international conference on Mobile computing and networking, Seattle, Washington, USA, August 15-20, pp. 207-218.
- [19] M.G. Zapata, Secure Ad hoc On-Demand Distance Vector Routing, ACM SIGMOBILE Mobile Computing and Communications Review. Jun, (2002), vol, 6(3), pp.106-107.
- [20] D. Cerri and A. Ghioni, "Securing AODV: the A-SAODV secure routing prototype," IEEE Communications Magazine,vol.46 no. 2, pp. 120-125, 2008.
- [21] Di Pietro, R.; Guarino, S.; Verde, N.V.; Domingo-Ferrer, J. (2014). Security in wireless ad-hoc networks – A survey. Computer Communications, 51(), 1–20. doi:10.1016/j.comcom.2014.06.003
- [22] Kaelbling, Leslie P.; Littman, Michael L.; Moore, Andrew W. (1996). "Reinforcement Learning: A Survey". Journal of Artificial Intelligence Research. 4: 237–285. arXiv:cs/9605103. doi:10.1613/jair.301. S2CID 1708582.

- [23] A. Colorni, M. Dorigo et V. Maniezzo, Distributed Optimization by Ant Colonies, actes de la première conférence européenne sur la vie artificielle, Paris, France, Elsevier Publishing, 134-142, 1991.
- [24] M. Dorigo, Optimization, Learning and Natural Algorithms, PhD thesis, Politecnico di Milano, Italy, 1992.
- [25] Kwang Mong Sim, ; Weng Hong Sun, (2002). [IEEE Comput. Soc 2002 International Symposium Cyber Worlds: Theory and Practices. CW2002 - Tokyo, Japan (6-8 Nov. 2002)] First International Symposium on Cyber Worlds, 2002. Proceedings. - Multiple ant-colony optimization for network routing. , (), 277–281. doi:10.1109/cw.2002.1180890
- [26] Zhao, Dongming; Luo, Liang; Zhang, Kai (2009). [IEEE 2009 Fourth International Conference on Bio-Inspired Computing (BIC-TA) - Beijing, China (2009.10.16-2009.10.19)] 2009 Fourth International on Conference on Bio-Inspired Computing - An improved ant colony optimization for communication network routing problem. , (), 1–4. doi:10.1109/BICTA.2009.5338074
- [27] B. Chandra Mohan; R. Baskaran (2012). A survey: Ant Colony Optimization based recent research and implementation on several engineering domain. , 39(4), 4618–4627. doi:10.1016/j.eswa.2011.09.076
- [28] ScottPlot (2022, October 19) <https://scottplot.net/>
- [29] NetworkX (2022, October 19) <https://networkx.org/>
- [30] R. W. Hamming, "Error detecting and error correcting codes," in The Bell System Technical Journal, vol. 29, no. 2, pp. 147-160, April 1950, doi: 10.1002/j.1538-7305.1950.tb00463.x.
- [31] Max Roser, Hannah Ritchie and Esteban Ortiz-Ospina (2015) - "Internet". (2022, October 27) 'https://ourworldindata.org/internet'
- [32] ICT Statistics (2022, October 27) <https://www.itu.int/ITU-D/ict/statistics/ict/>
- [33] Number of smartphone subscriptions worldwide from 2016 to 2021, with forecasts from 2022 to 2027 (2022, October 27) <https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/>



Network images

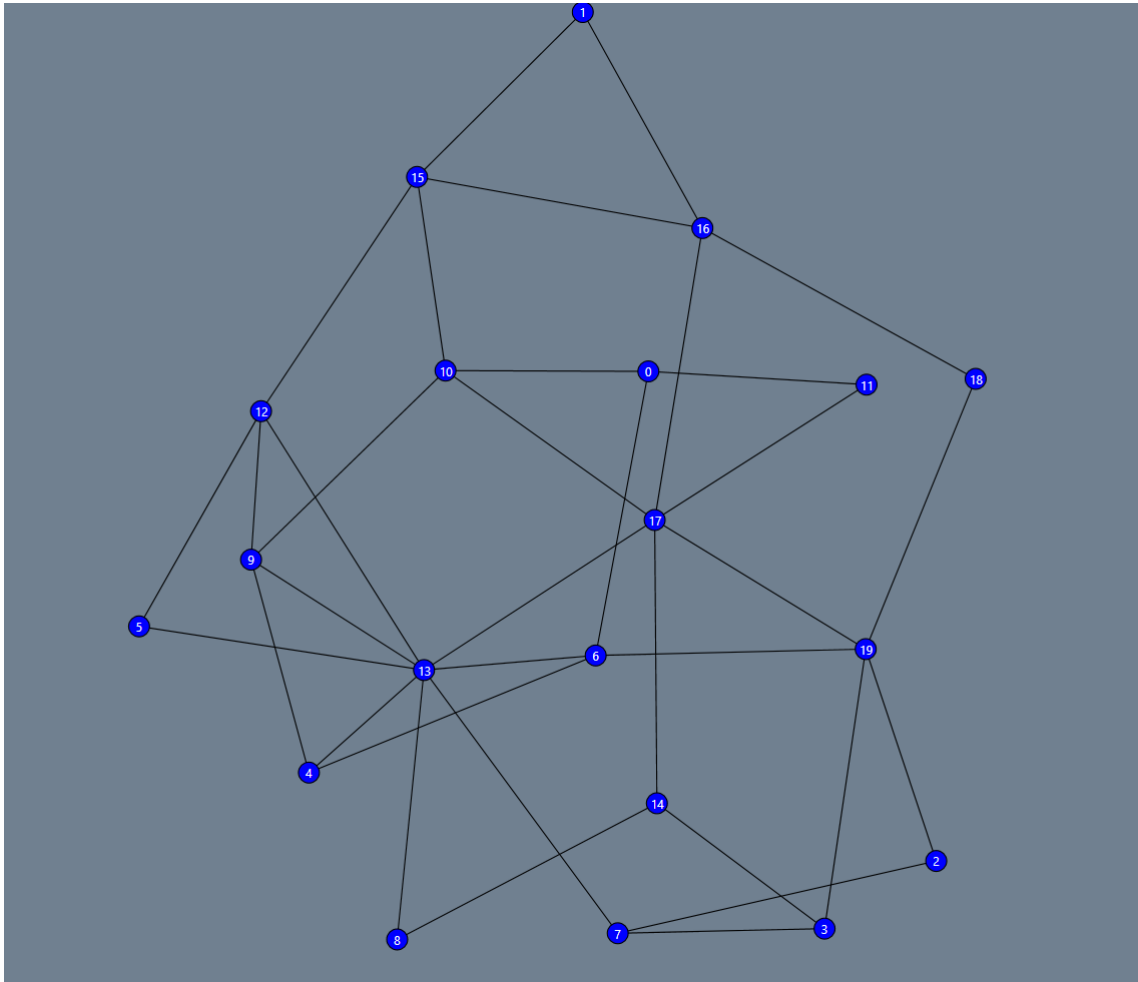
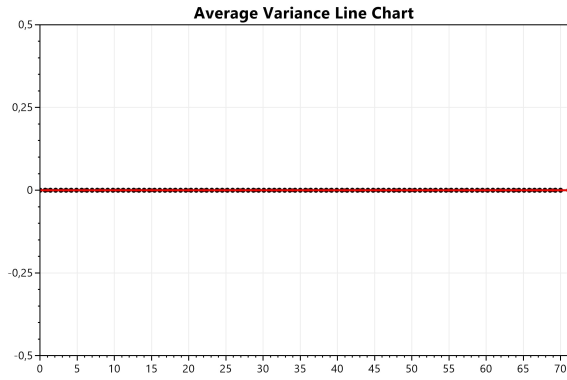
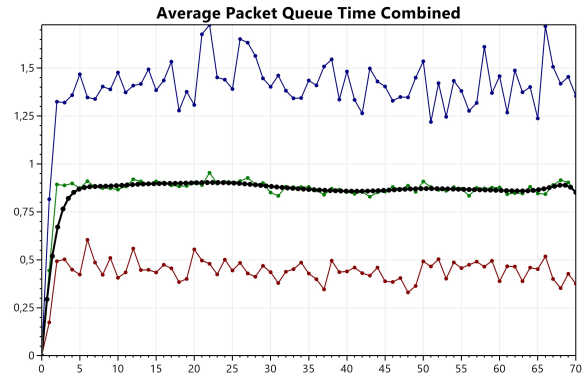


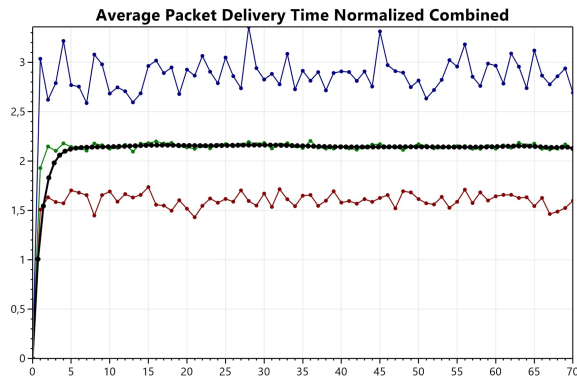
Figure A.1: Network with 20 nodes and probability of each pair of nodes having a link equal to 0.1; Longest path = 39, Average Path Length ~ 19.78 , $TTL = 59$



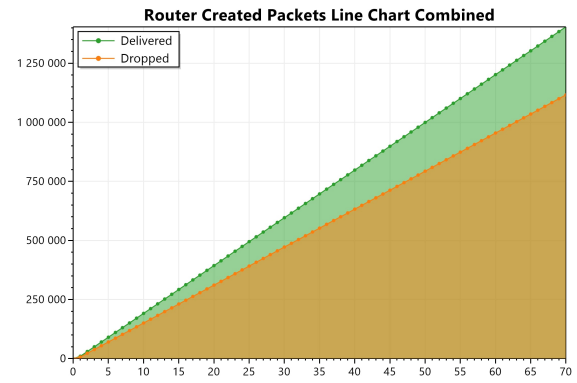
(a) Average variance



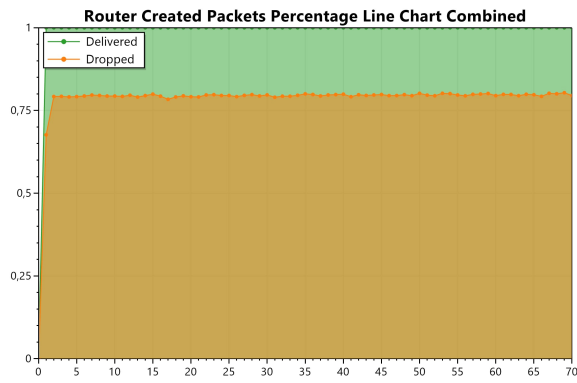
(b) Average packet queue time



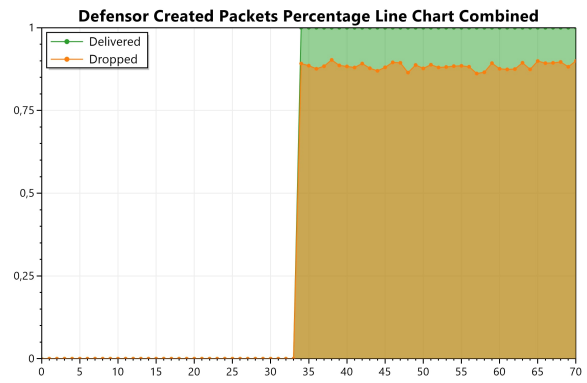
(c) Average packet delivery time normalized



(d) Router created packets

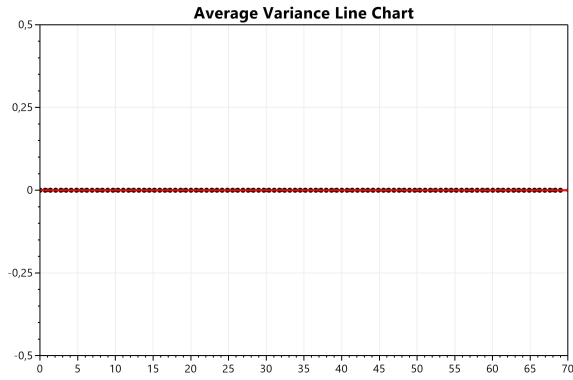


(e) Router created packets percentage

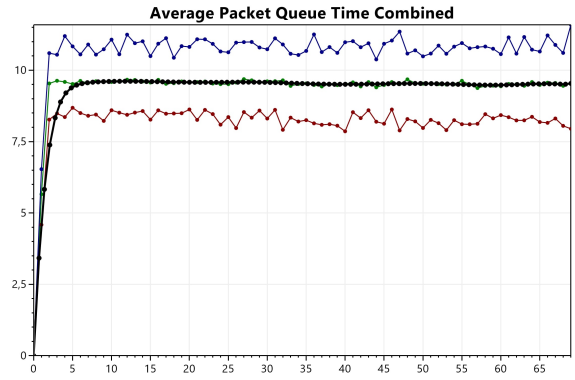


(f) Defender created packets percentage

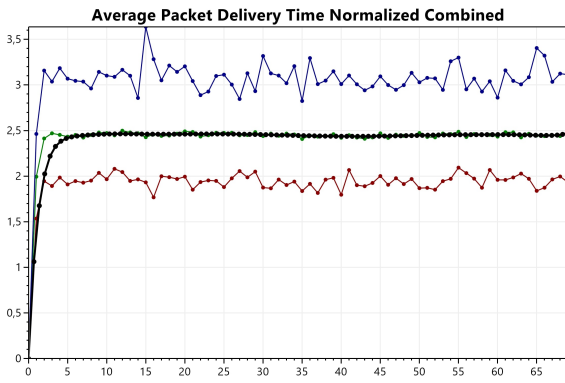
Figure A.2: Network A.1; random routing; no discovery; packet creation probability 0.1



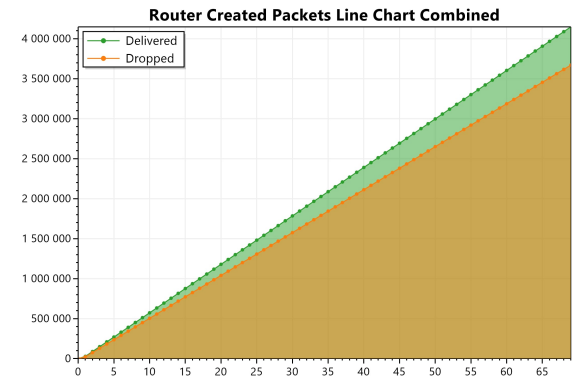
(a) Average variance



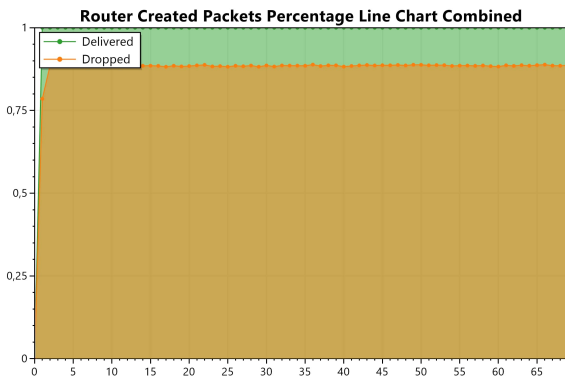
(b) Average packet queue time



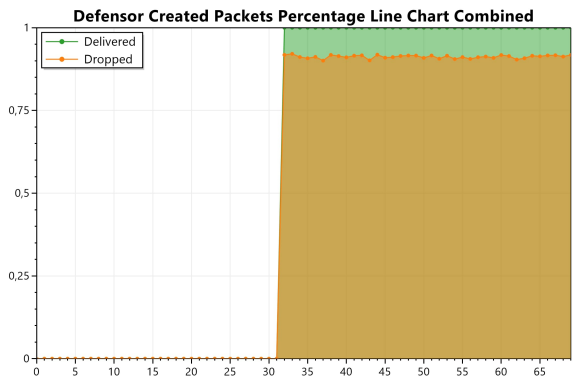
(c) Average packet delivery time normalized



(d) Router created packets

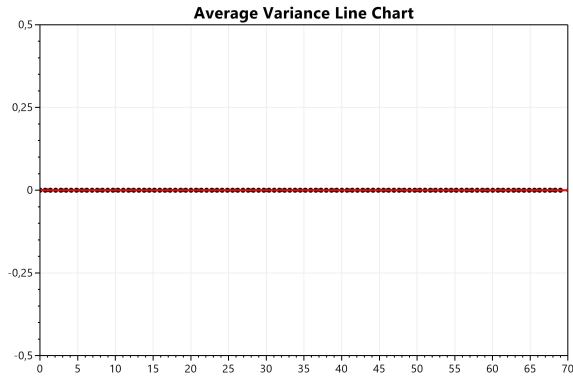


(e) Router created packets percentage

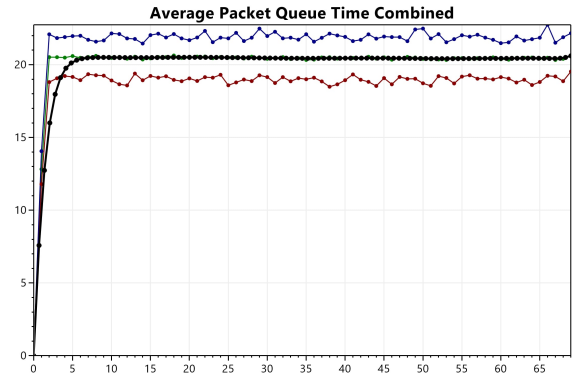


(f) Defender created packets percentage

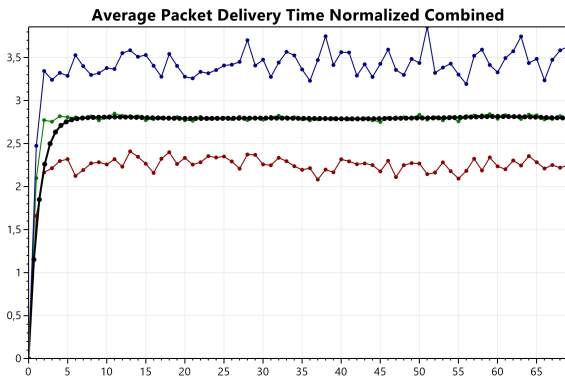
Figure A.3: Network A.1; random routing; no discovery; packet creation probability 0.3



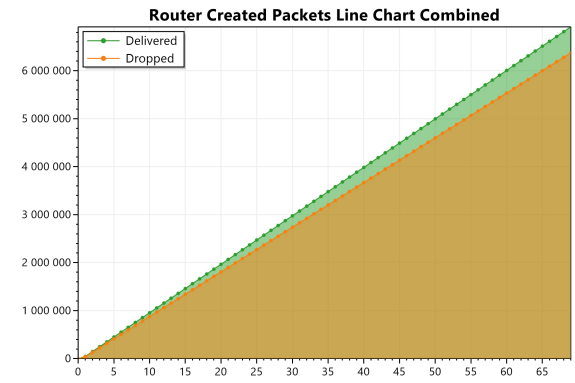
(a) Average variance



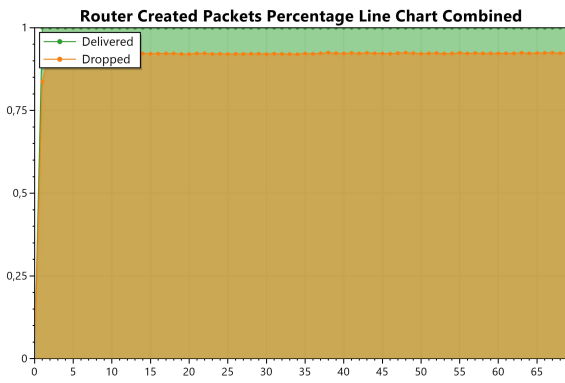
(b) Average packet queue time



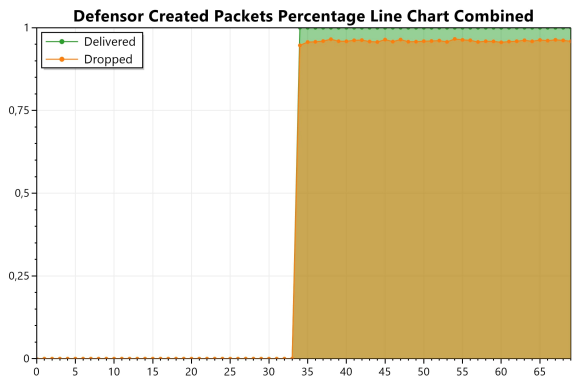
(c) Average packet delivery time normalized



(d) Router created packets

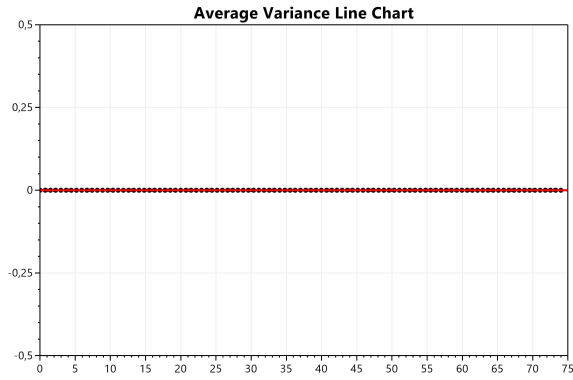


(e) Router created packets percentage

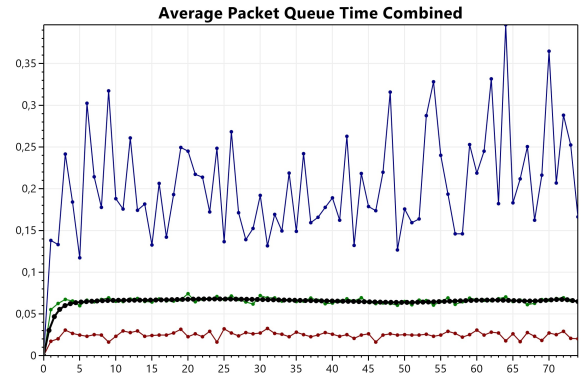


(f) Defender created packets percentage

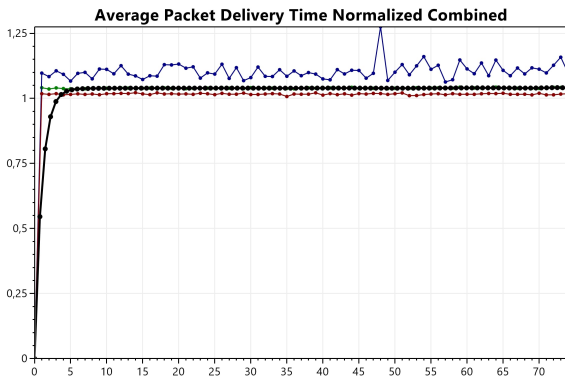
Figure A.4: Network A.1; random routing; no discovery; packet creation probability 0.5



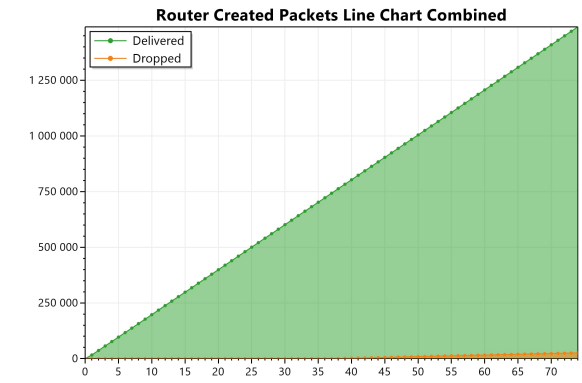
(a) Average variance



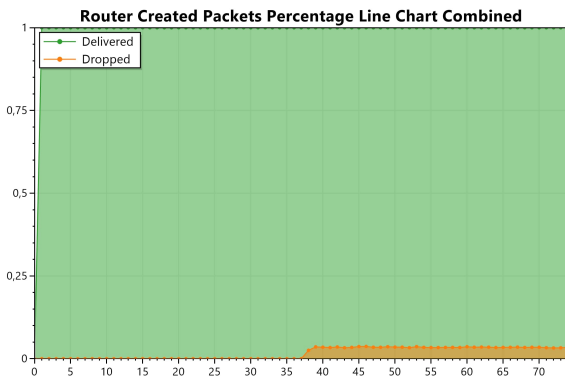
(b) Average packet queue time



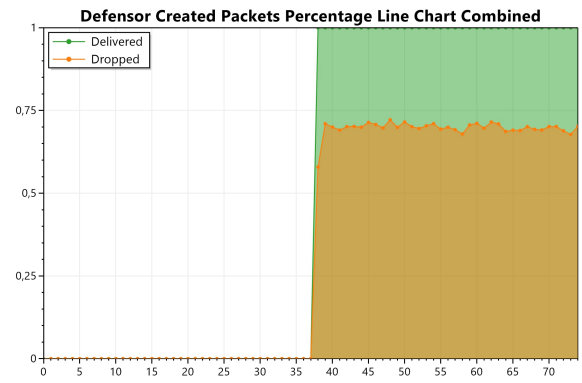
(c) Average packet delivery time normalized



(d) Router created packets

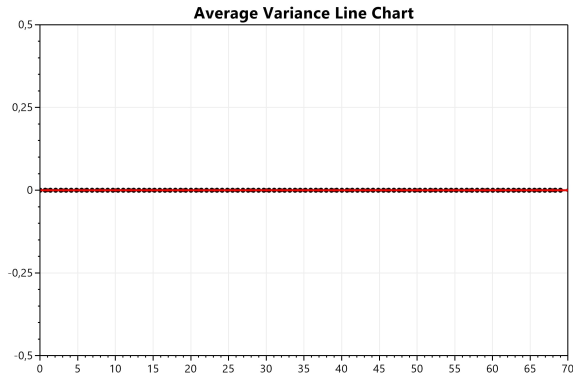


(e) Router created packets percentage

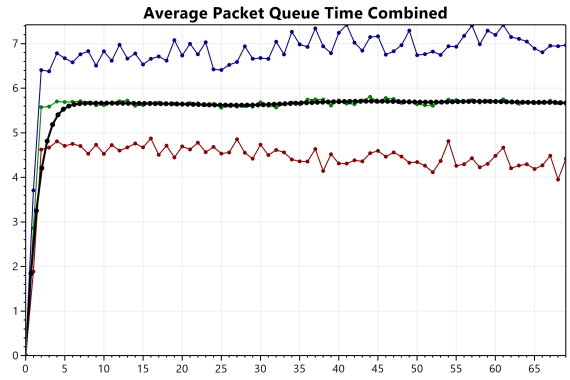


(f) Defender created packets percentage

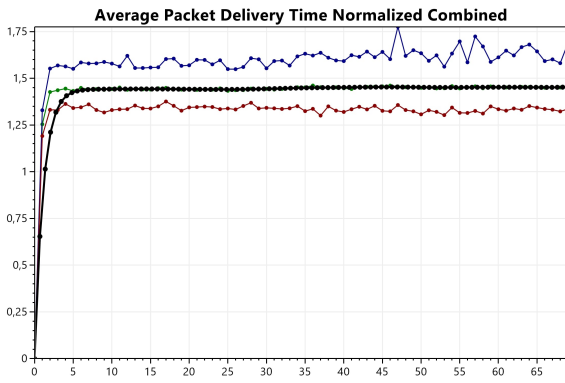
Figure A.5: Network A.1; random routing; best path only; packet creation probability 0.1



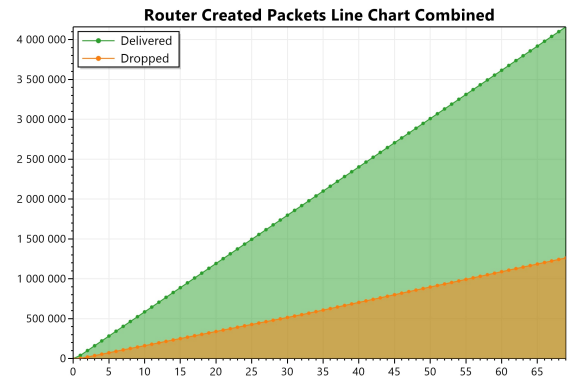
(a) Average variance



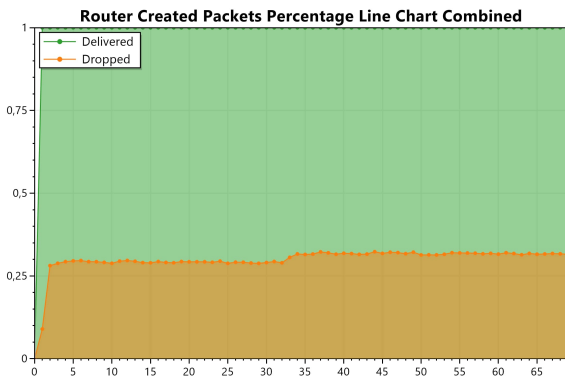
(b) Average packet queue time



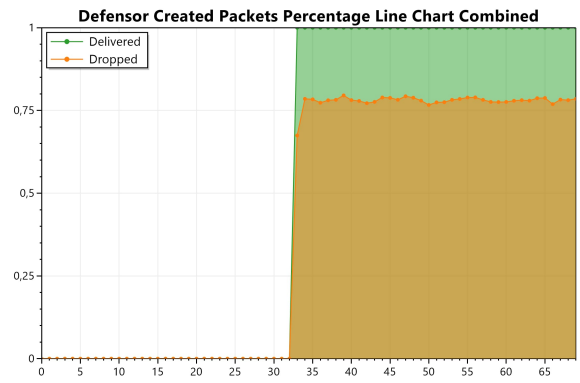
(c) Average packet delivery time normalized



(d) Router created packets

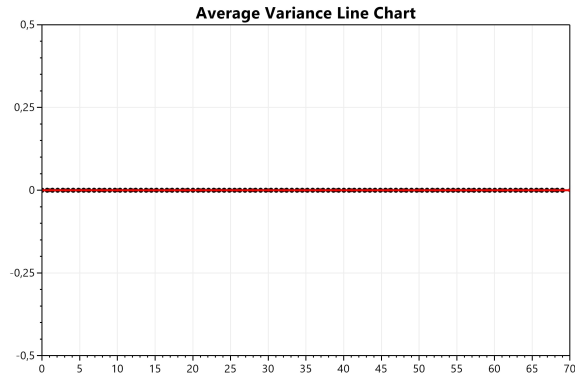


(e) Router created packets percentage

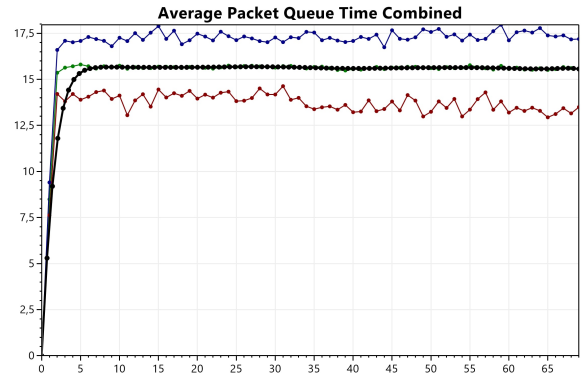


(f) Defender created packets percentage

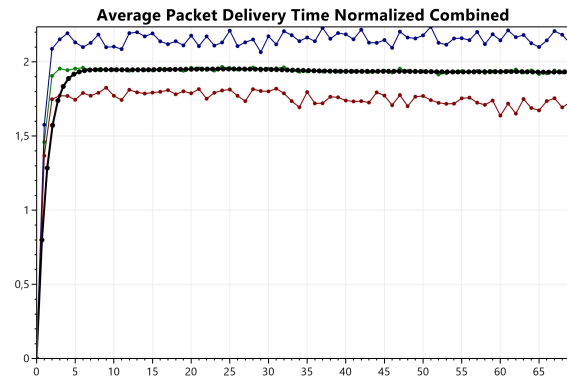
Figure A.6: Network A.1; random routing; best path only; packet creation probability 0.3



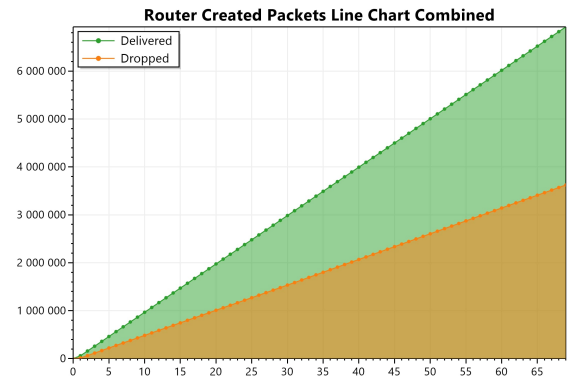
(a) Average variance



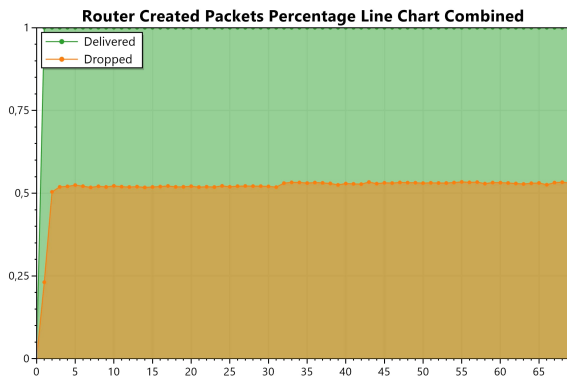
(b) Average packet queue time



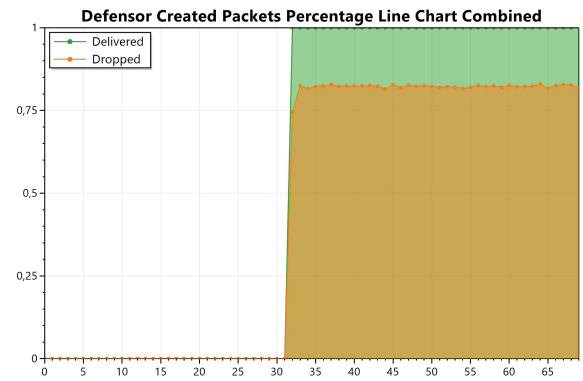
(c) Average packet delivery time normalized



(d) Router created packets

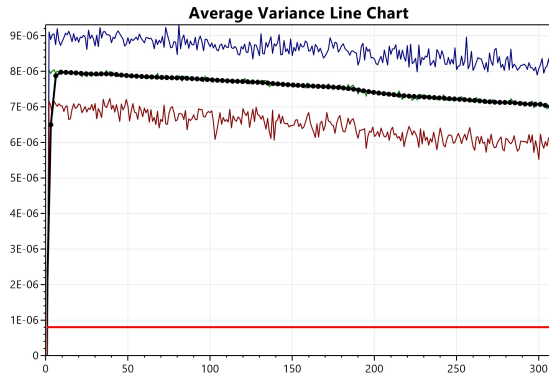


(e) Router created packets percentage

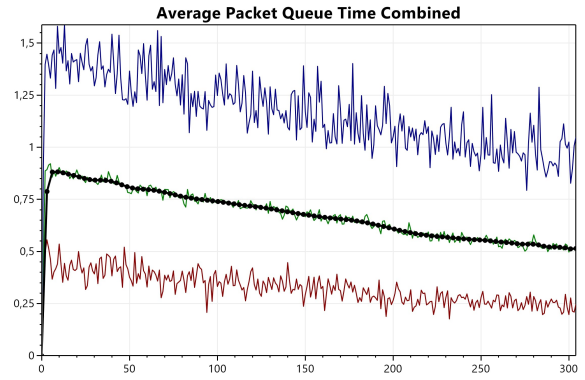


(f) Defender created packets percentage

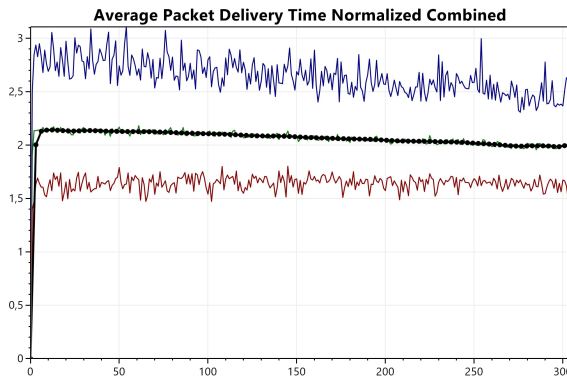
Figure A.7: Network A.1; random routing; best path only; packet creation probability 0.5



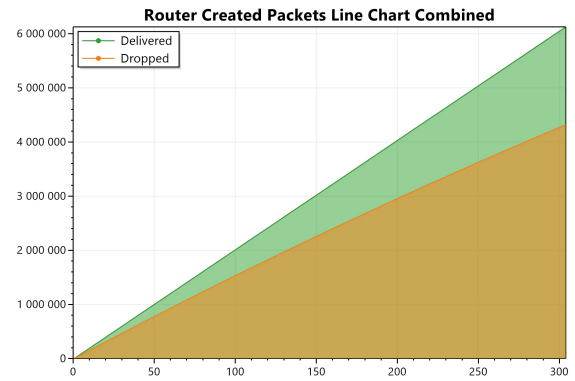
(a) Average variance



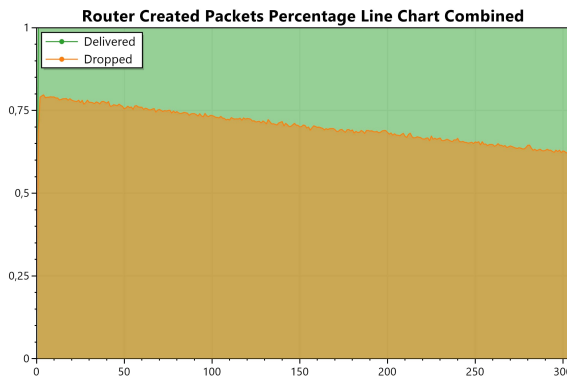
(b) Average packet queue time



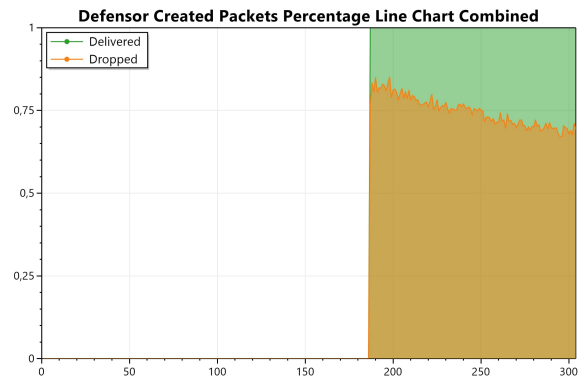
(c) Average packet delivery time normalized



(d) Router created packets

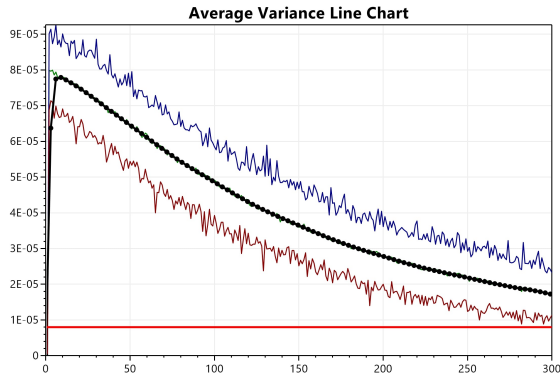


(e) Router created packets percentage

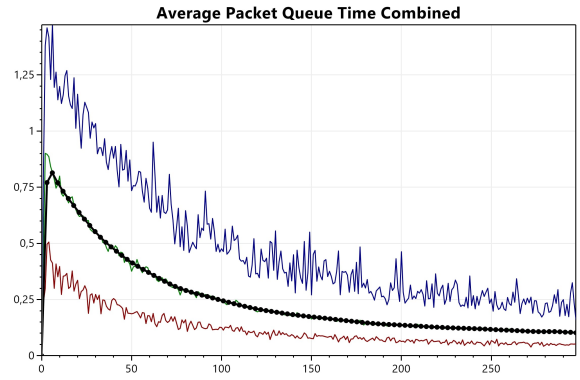


(f) Defender created packets percentage

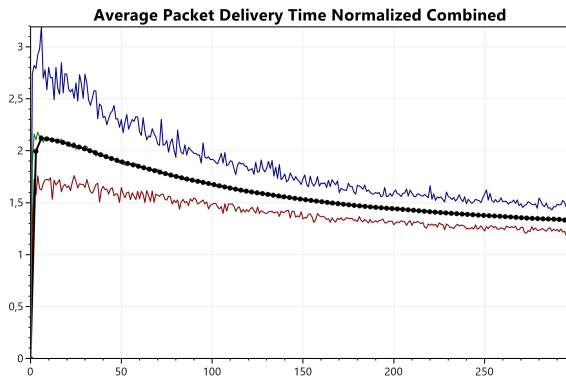
Figure A.8: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; low learning rate



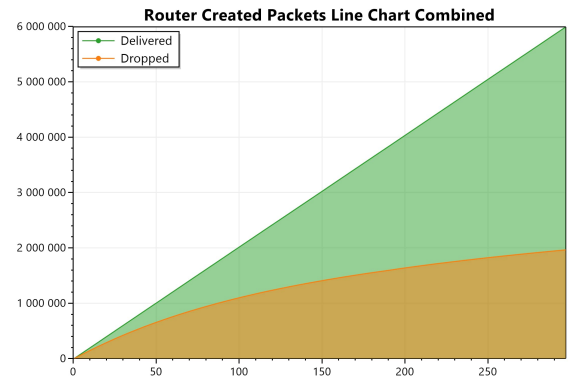
(a) Average variance



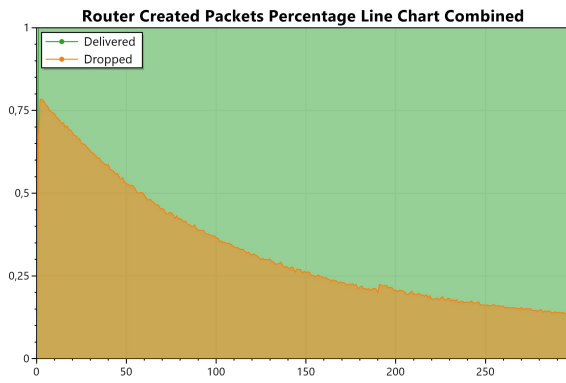
(b) Average packet queue time



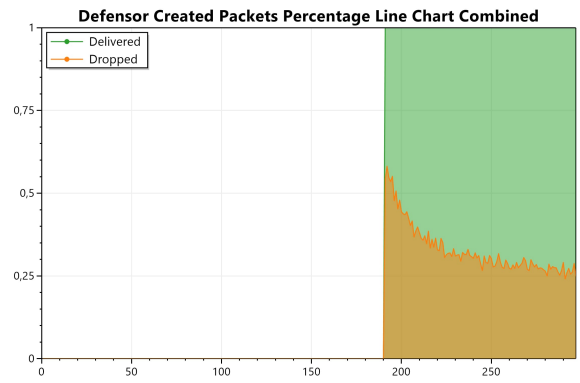
(c) Average packet delivery time normalized



(d) Router created packets

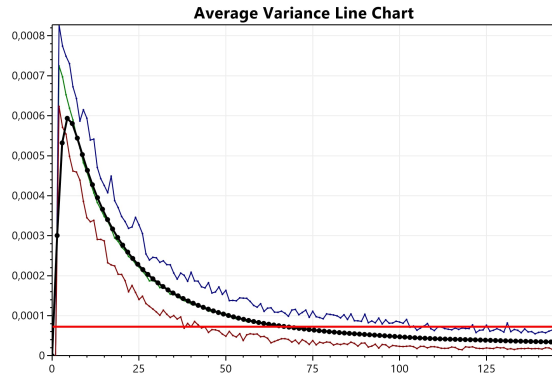


(e) Router created packets percentage

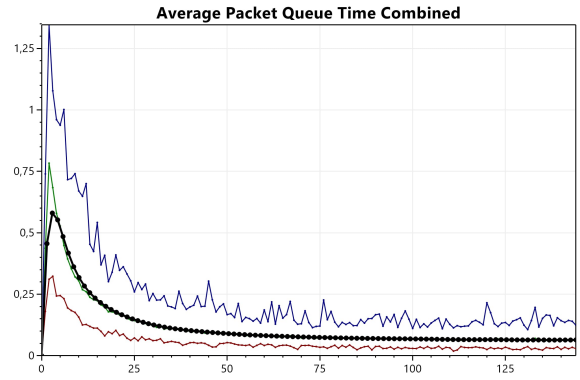


(f) Defender created packets percentage

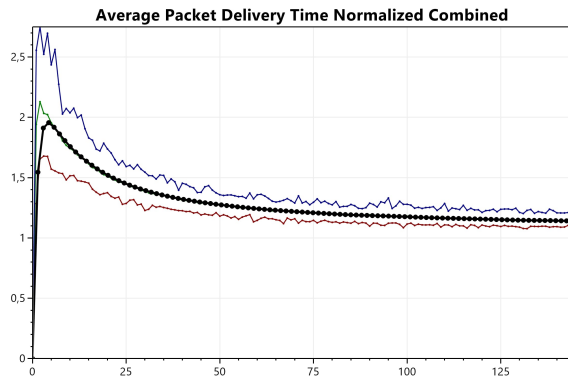
Figure A.9: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; medium learning rate



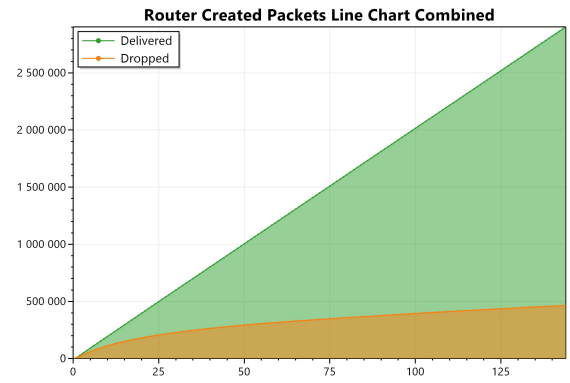
(a) Average variance



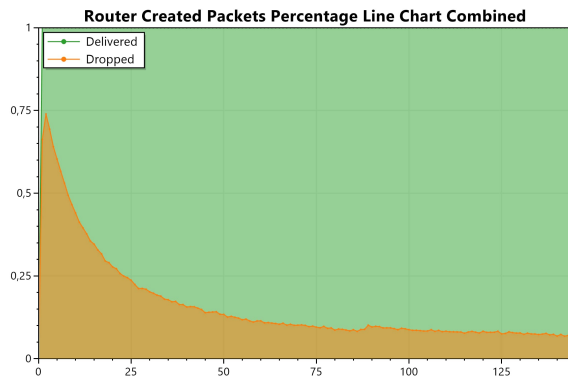
(b) Average packet queue time



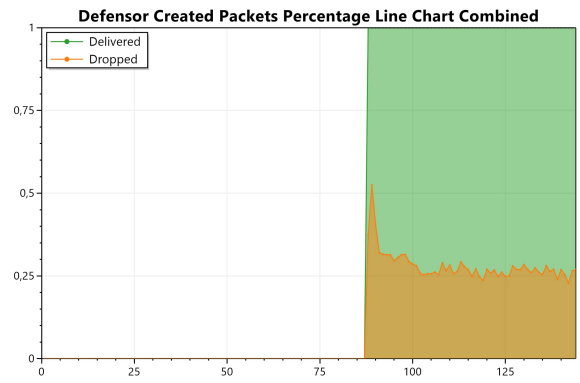
(c) Average packet delivery time normalized



(d) Router created packets

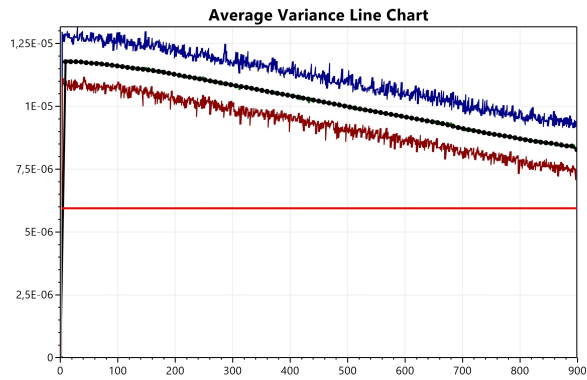


(e) Router created packets percentage

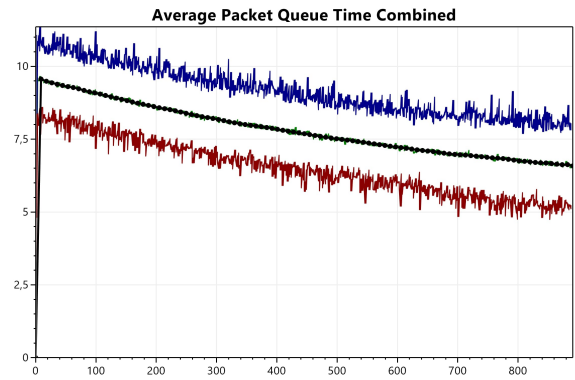


(f) Defender created packets percentage

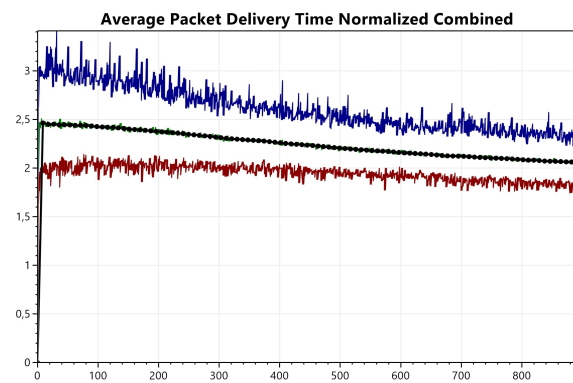
Figure A.10: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.1; high learning rate



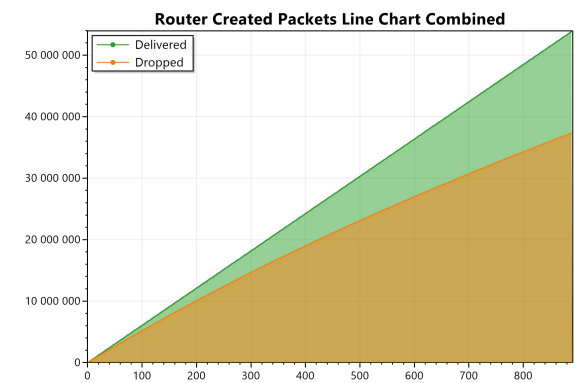
(a) Average variance



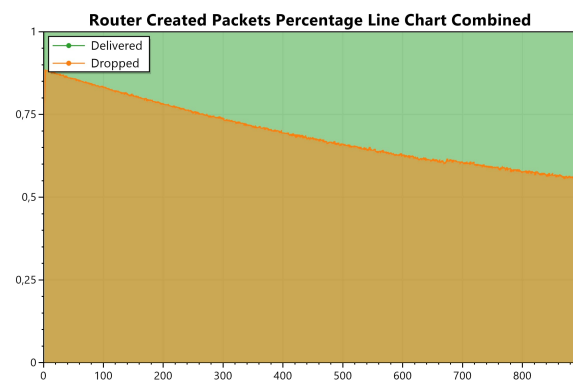
(b) Average packet queue time



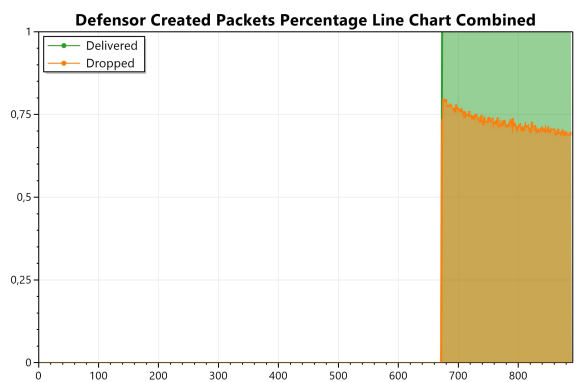
(c) Average packet delivery time normalized



(d) Router created packets

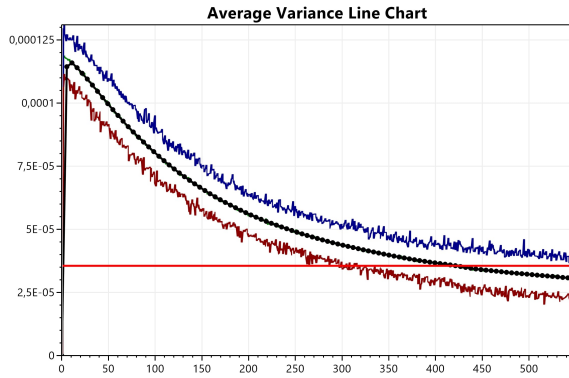


(e) Router created packets percentage

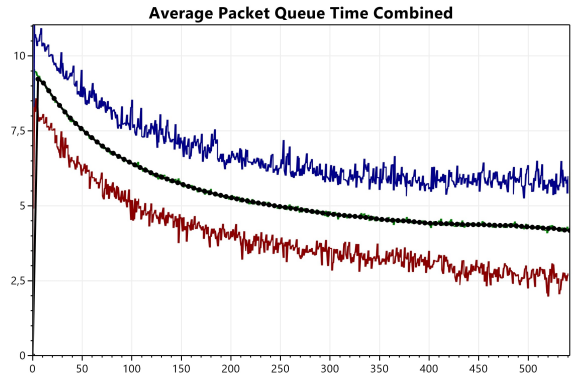


(f) Defender created packets percentage

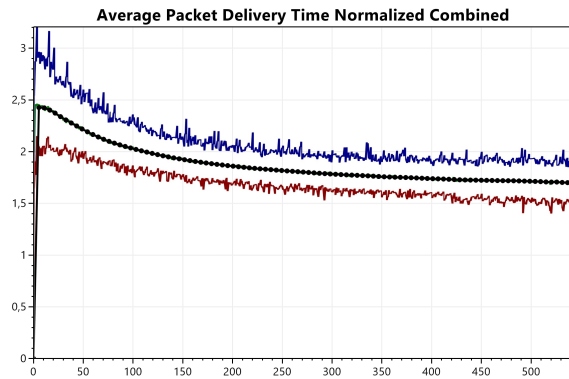
Figure A.11: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; low learning rate



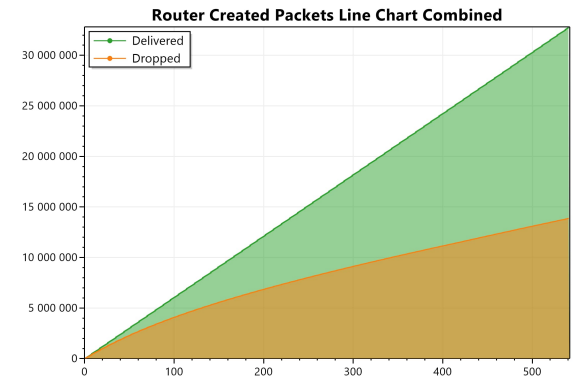
(a) Average variance



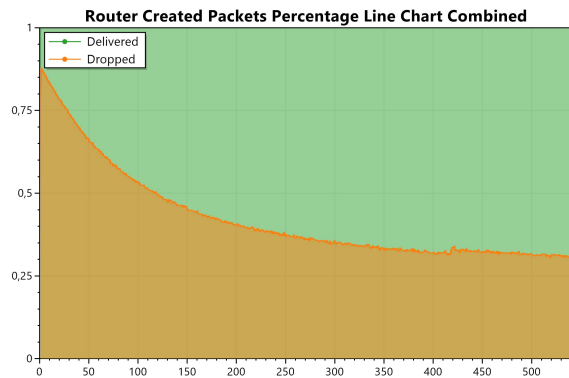
(b) Average packet queue time



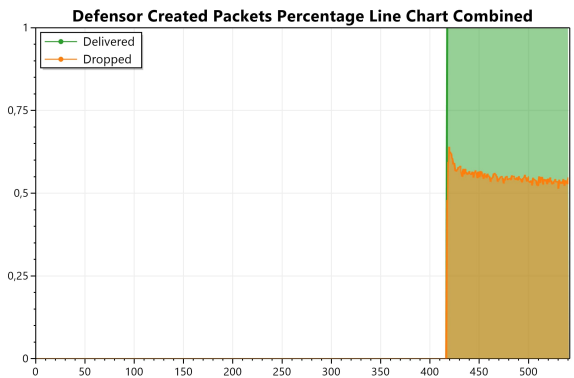
(c) Average packet delivery time normalized



(d) Router created packets

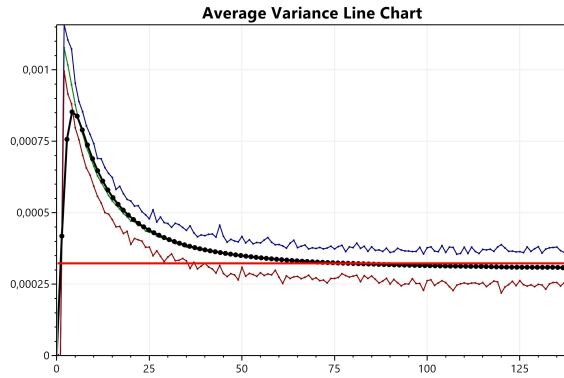


(e) Router created packets percentage

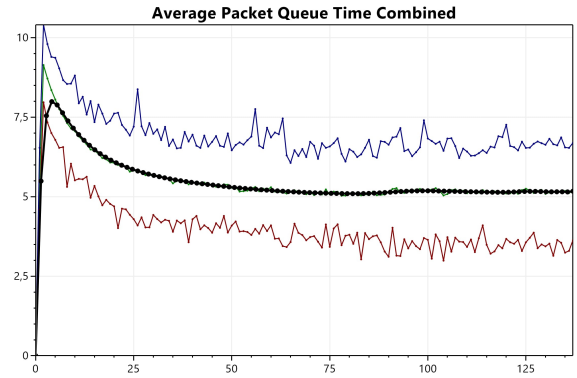


(f) Defender created packets percentage

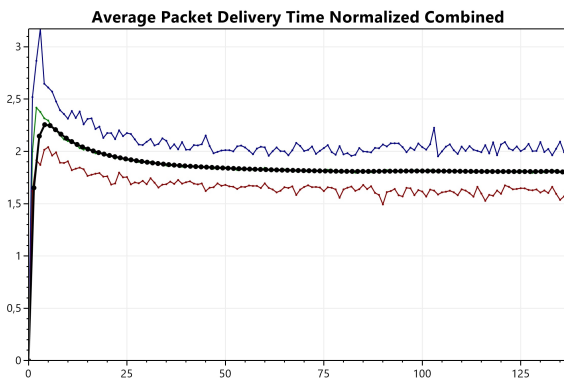
Figure A.12: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; medium learning rate



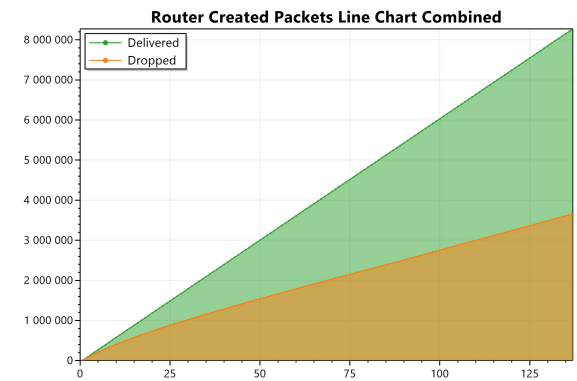
(a) Average variance



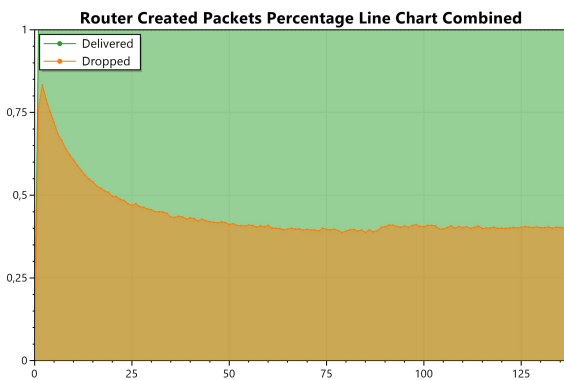
(b) Average packet queue time



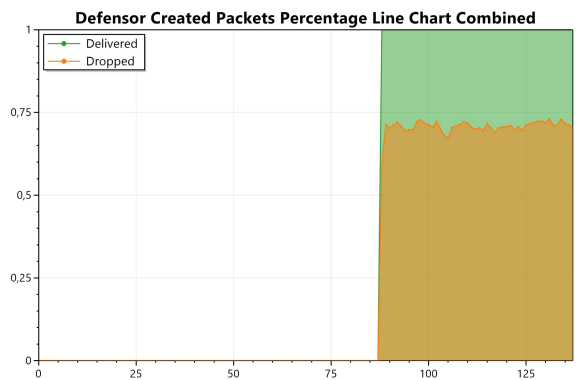
(c) Average packet delivery time normalized



(d) Router created packets

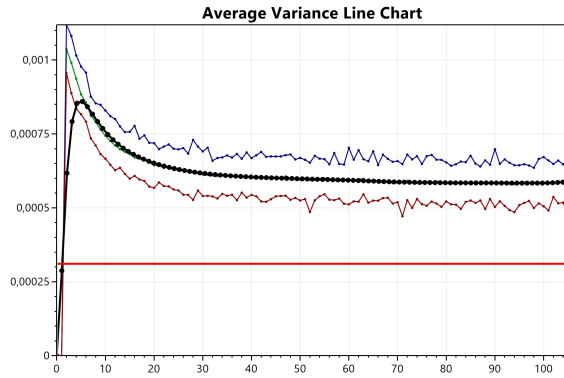


(e) Router created packets percentage

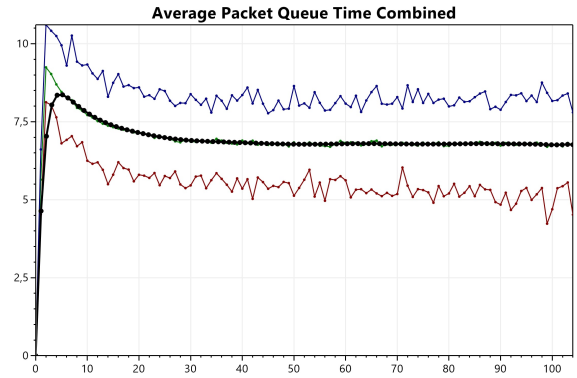


(f) Defender created packets percentage

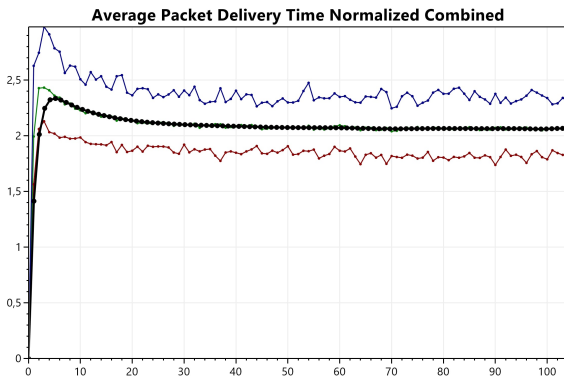
Figure A.13: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; high learning rate



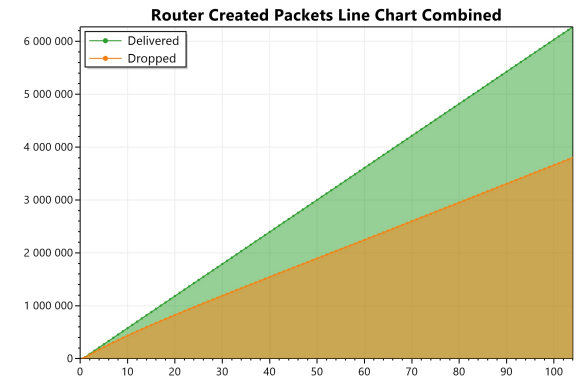
(a) Average variance



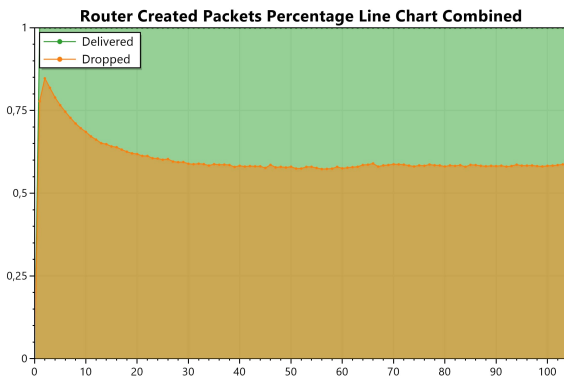
(b) Average packet queue time



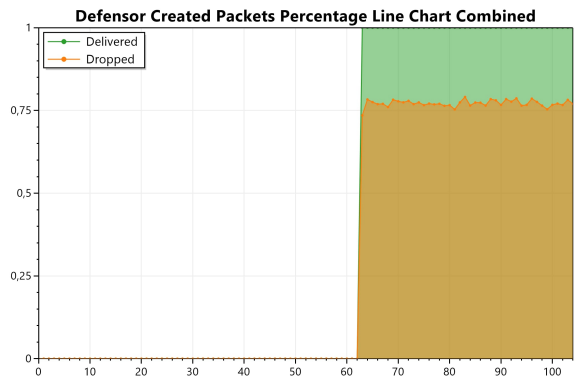
(c) Average packet delivery time normalized



(d) Router created packets

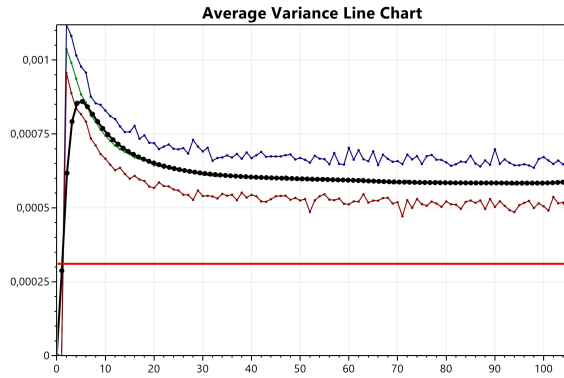


(e) Router created packets percentage

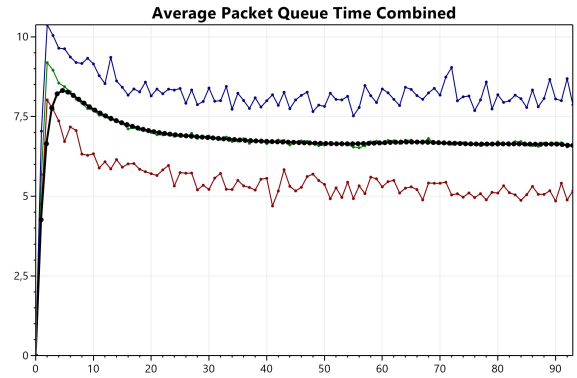


(f) Defender created packets percentage

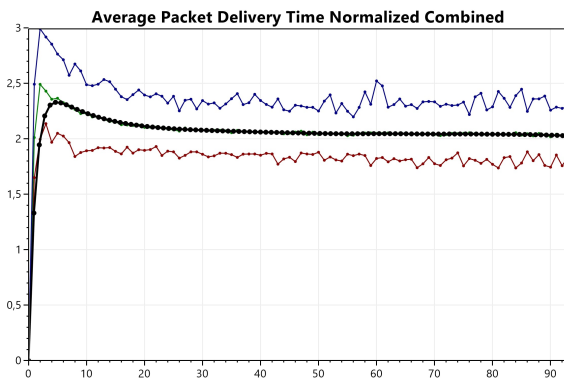
Figure A.14: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; low reward rate and high penalty rate



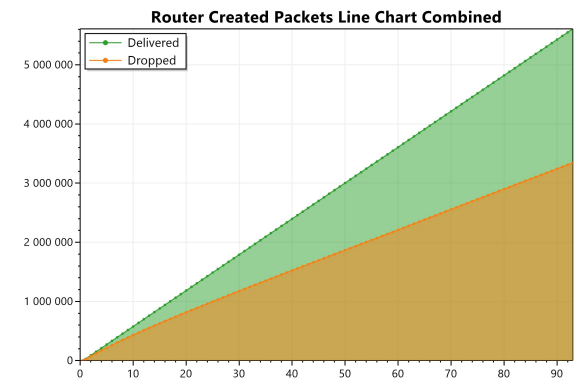
(a) Average variance



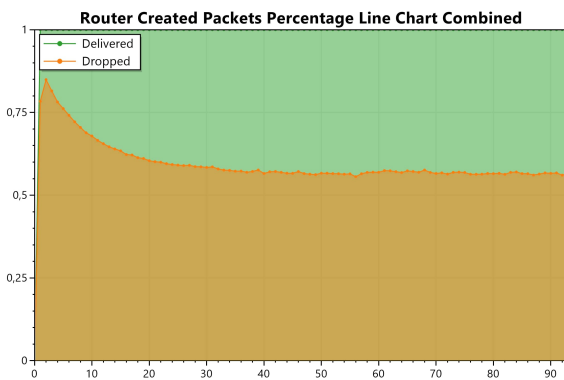
(b) Average packet queue time



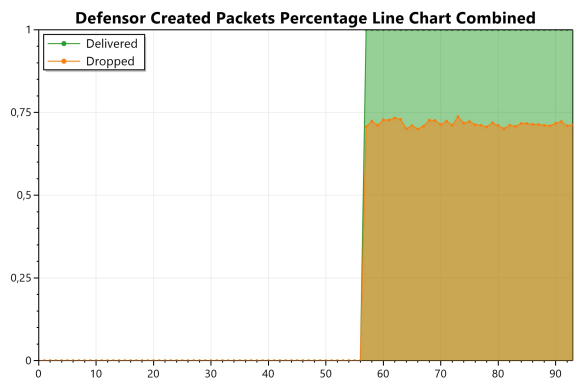
(c) Average packet delivery time normalized



(d) Router created packets

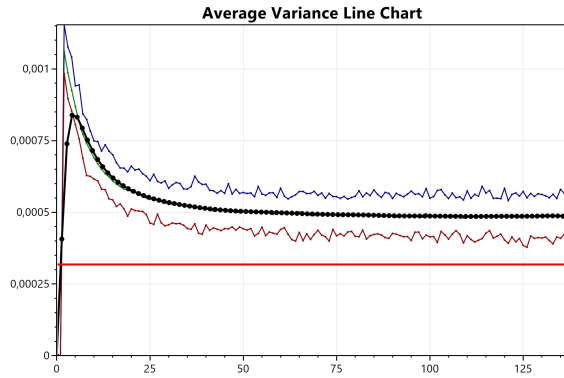


(e) Router created packets percentage

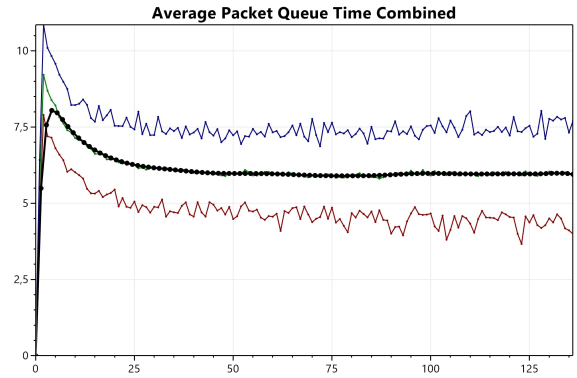


(f) Defender created packets percentage

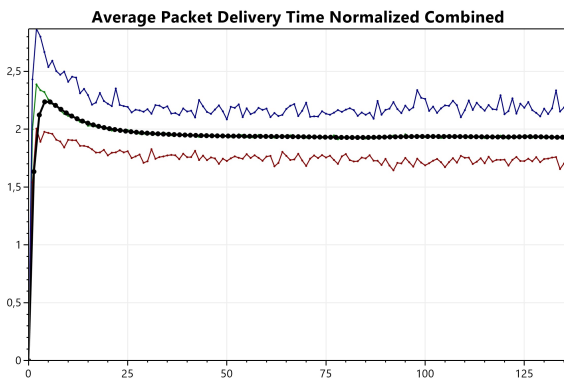
Figure A.15: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; medium reward rate and high penalty rate



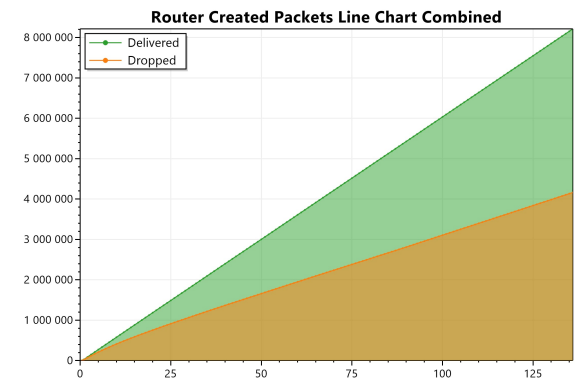
(a) Average variance



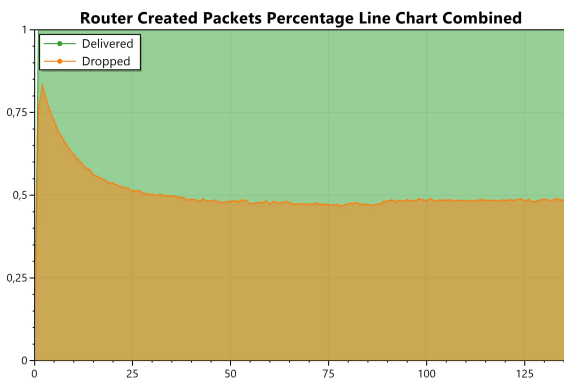
(b) Average packet queue time



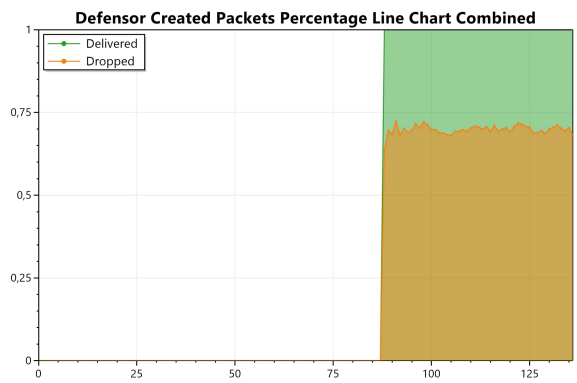
(c) Average packet delivery time normalized



(d) Router created packets

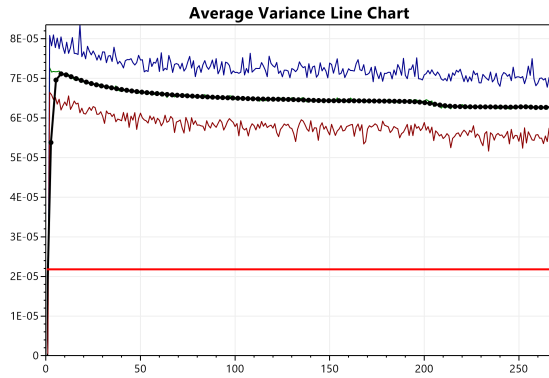


(e) Router created packets percentage

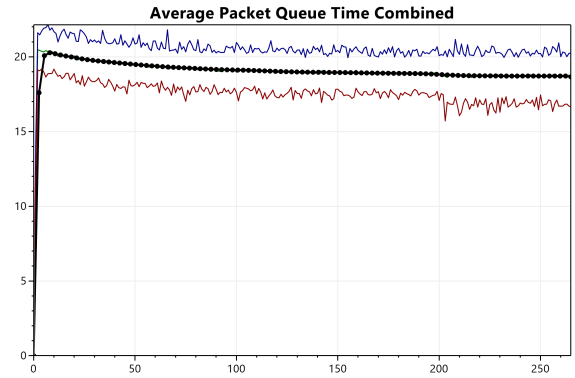


(f) Defender created packets percentage

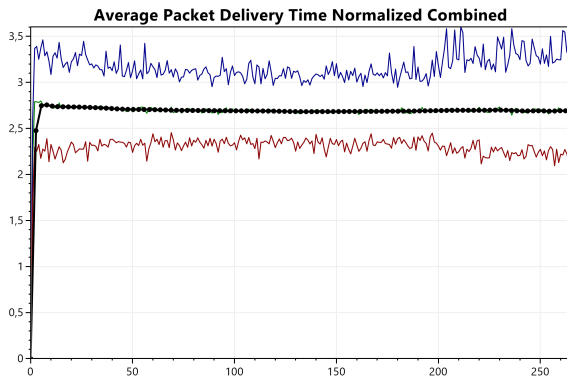
Figure A.16: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.3; high reward rate and high penalty rate



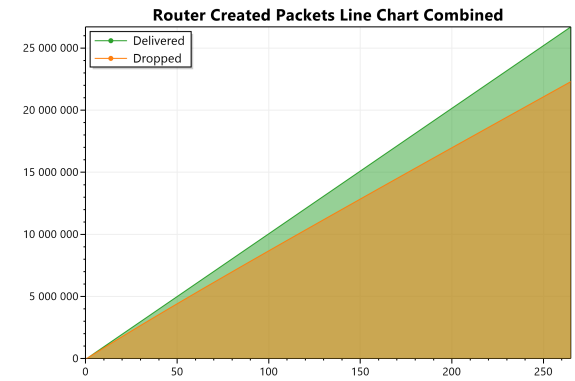
(a) Average variance



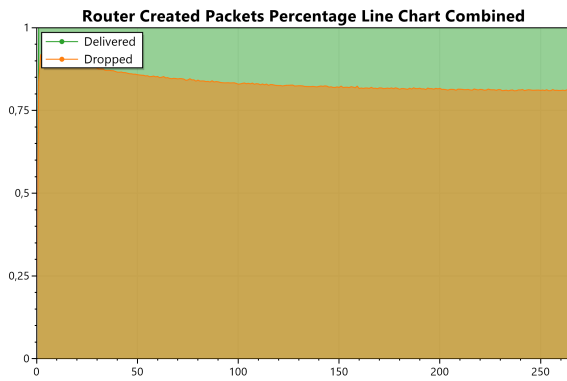
(b) Average packet queue time



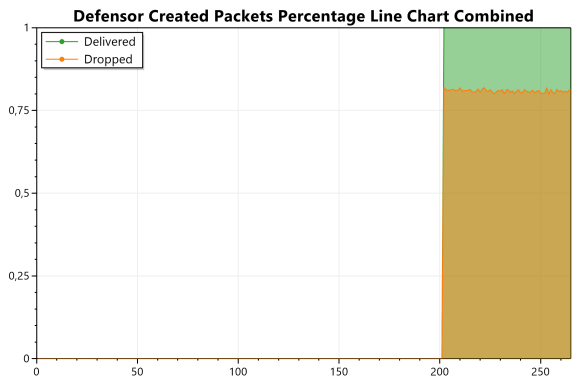
(c) Average packet delivery time normalized



(d) Router created packets

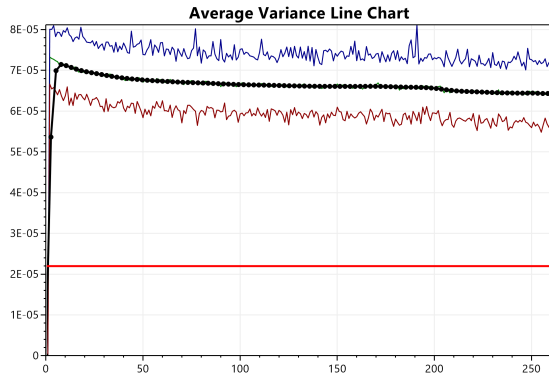


(e) Router created packets percentage

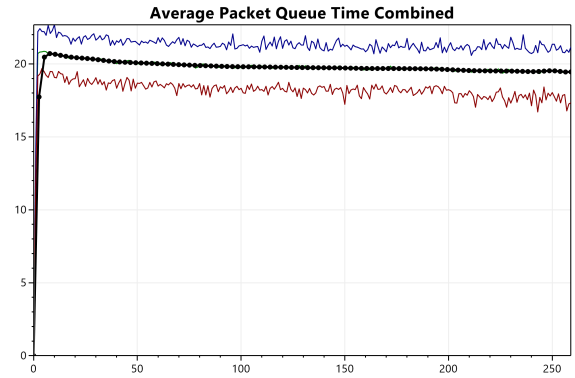


(f) Defender created packets percentage

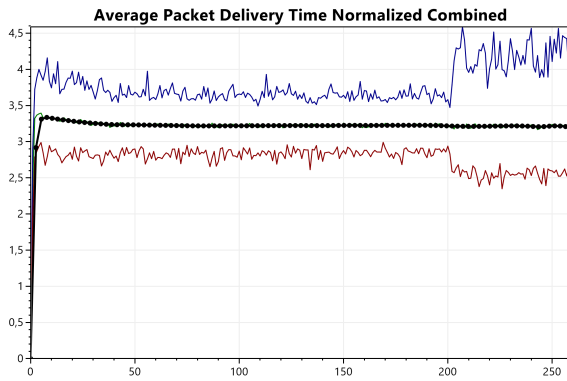
Figure A.17: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; random packet picking



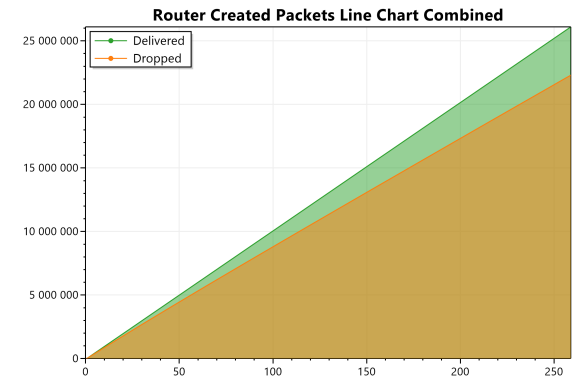
(a) Average variance



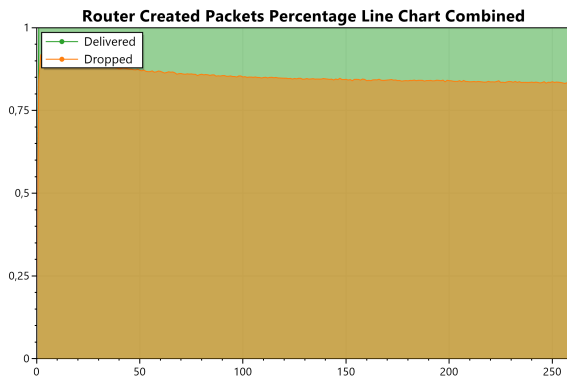
(b) Average packet queue time



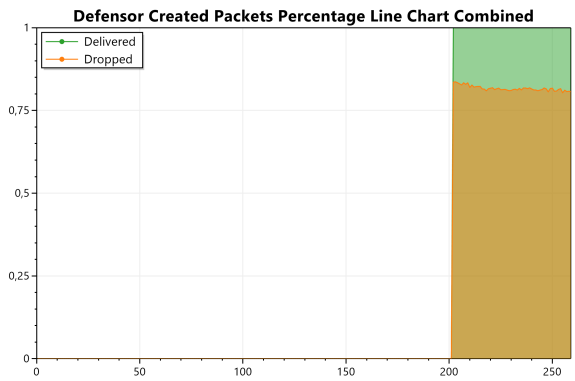
(c) Average packet delivery time normalized



(d) Router created packets

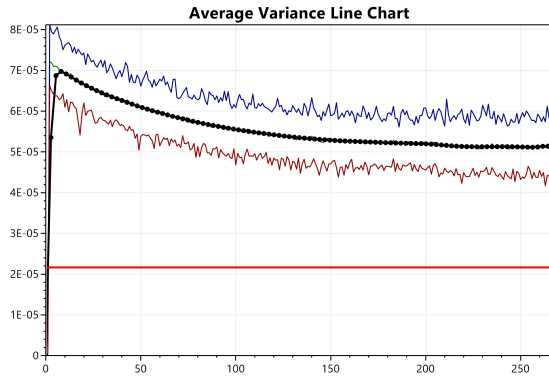


(e) Router created packets percentage

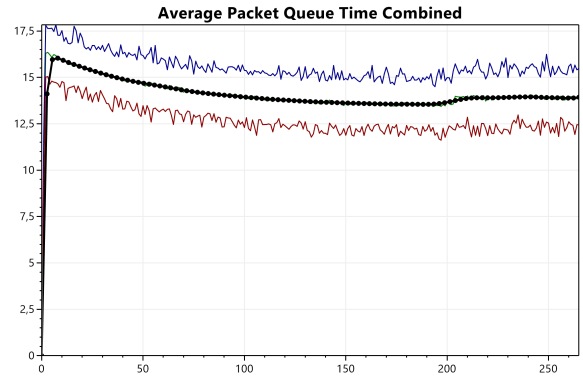


(f) Defender created packets percentage

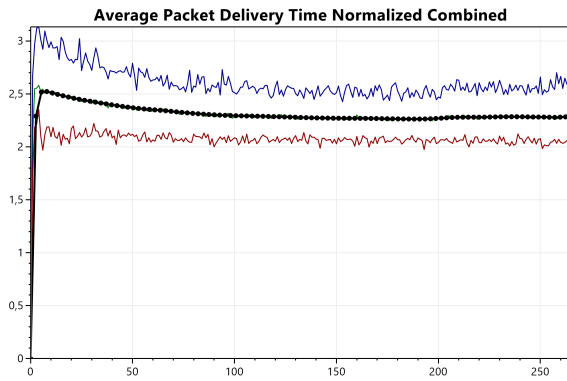
Figure A.18: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; FIFO packet picking



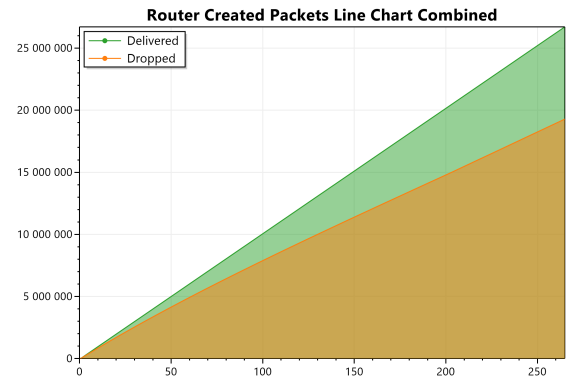
(a) Average variance



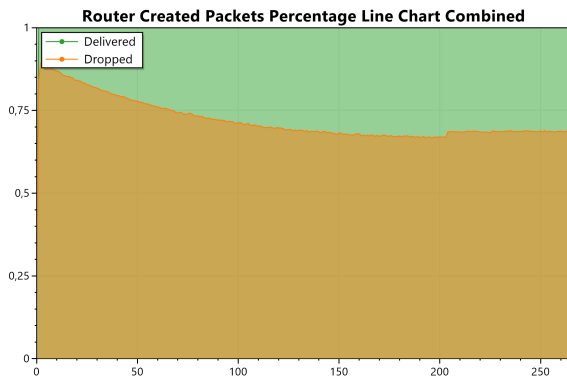
(b) Average packet queue time



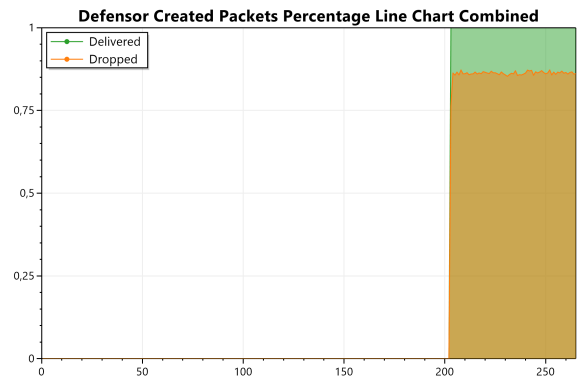
(c) Average packet delivery time normalized



(d) Router created packets



(e) Router created packets percentage



(f) Defender created packets percentage

Figure A.19: Network A.1; linear reward penalty routing; no discovery; packet creation probability 0.5; random remove unreachable packet picking