

**MITIGATION OF DIFFERENT NETWORK ATTACKS AND OPTIMIZATION
IN SOFTWARE DEFINED NETWORK**

A Dissertation
Submitted to
the Temple University Graduate Board

In Partial Fulfillment
of the Requirements for the Degree of
DOCTOR OF PHILOSOPHY

by
Rajorshi Biswas
August 2021

Examining committee members:

Dr. Jie Wu, Dissertation Advisory Chair, Department of Computer and Information Sciences

Dr. Xiojiang Du, Department of Computer and Information Sciences

Dr. Yu Wang, Department of Computer and Information Sciences

Dr. Bo Ji, Department of Computer Science, College of Engineering, Virginia Tech

ABSTRACT

Cyber attacks are growing with the increase in internet usage. There are several types of network attacks including volumetric and non-volumetric attacks. In a volumetric attack, the target resource is taken down with a huge amount of traffic. Distributed denial-of-service (DDoS) and transit-link DDoS are examples of these types of attacks.

A DDoS attack is a cyber-attack in which the attacker sends out a huge number of requests to exhaust the capacity of a server so that it can no longer serve incoming requests and DoS occurs. The most devastating distributed DoS attack is performed by malicious programs called bots.

A transit-link DDoS attack is a special attack in which the attacker sends out a huge number of requests to exhaust the capacity of a link on the path the traffic comes to a server. As a result, user traffic cannot reach the server. As a result, denial-of-service and degradation of Quality-of-Service (QoS) occur. Because the attack traffic does not go to the victim, protecting the legitimate traffic alone is hard for the victim.

With the help of a special type of router called filter router (FR), the victim can protect itself and reduce useless congestion in the network. A server can send out filters to FRs for blocking attack traffic. The victim needs to select a subset of FRs wisely to minimize attack traffic and blockage of legitimate users (LUs). By using FR, the victim can also protect the legitimate traffic. An FR can receive a filter from servers and apply the filter to block a link incident to it. The victim needs to select some of these possible congested links and send a filter to the corresponding FR so that the legitimate traffic follows non-

congested paths. This way we can protect the victim/resources from the attackers that reside outside the datacenter network.

To protect the resources from the attackers that reside inside a datacenter we need to monitor all of the flows. Some of the free virtual machines (VMs) can be used as traffic monitors. The monitoring process induces some network overhead which should be the lowest for the best performance. Monitoring becomes more challenging when the number of monitors is not enough to monitor all the flows in a datacenter. In that case, the packets in flows are sub-sampled to fit the capacity of the monitors.

In a non-volumetric attack, the attackers try to steal or get illegal authorization of some resources in a network. This type of attack can be severe even with a small amount of traffic. Non-volumetric attacks can be stopped by applying a moving target defense approach at the nodes on the attack path. An attack path is a series of steps and the attacker needs to succeed in all of those steps to gain access to the resources.

The routing in an SDN switch is controlled by the rules installed in it or provided by the controller. The SDN switches have limited capacity for the rules and the performance dramatically drops when the number of stored rules is higher. To prevent the link congestion due to regular traffic and link flooding attacks (LFA), the rules need to be changed over time to reconfigure the flow path. It takes some time to adopt the changes by the SDN switches which causes an interruption in flow. We also aim at minimizing the number of rule changes while redirecting some of the traffic from the congested link.

We contribute to solving these problems in this paper. We formulate several problems for selecting filter routers given a constraint on the number of filters. The first problem considers the source-based filter and we provide several solutions include a dynamic programming solution. The second problem considers the destination-based filter and how to minimize the total amount of attack traffic and blocked LUs. We propose a dynamic programming solution for the second problem.

To defend the transit-link DDoS attack, we formulate optimization two problems for selecting the minimum number of possible congested links so that the legitimate traffic goes through a non-congested path. We consider the scenario where every user has at least one non-congested shortest path in the first problem. We extend the first problem to a scenario where there are some users whose shortest paths are all congested. We transform the original problem to the vertex separation problem to find the links to block.

We propose a mechanism to protect against DDoS attacks originated within a datacenter. Our system is composed of two parts: flow monitoring and traffic filtering. In flow monitoring, we formulate two problems: one for finding flow assignments to monitors and another for selecting the best locations of monitors. We provide an optimal and a greedy solutions for the first and second problems, respectively. We also propose a flow grouping and sampling rate distribution approach based on behavioral similarity among the VMs followed by hierarchical clustering of VMs. The sampling rate is uniform among all the flows in a group. We investigate the relationship between the sampling rate and the DDoS detection rate. Then, we formulate an optimization problem for finding an optimal sampling rate distribution and solve it using mix-integer linear programming.

To minimize the number of rule changes while redirecting some of the traffic from the congested link. We formulate two problems to minimize the number of rule changes to redirect traffic. The first problem is the basic and it considers a congested link and a flow to direct. We provide a Dijkstra-based and a rule merging-based solution to the problems. The second problem considers multiple flows and we propose flow grouping and rule merging based solutions.

To protect against the non-volumetric attack, we formulate an optimization problem to minimize the damage while securing the resources by deploying the minimum number of moving target defense methods. We provide a dynamic programming-based solution to this problem.

ACKNOWLEDGEMENTS

I would like to express my gratitude to my advisor Dr. Jie Wu for his tremendous support during my graduate study at Temple University. He has brought me into the world of research by his guidance and encouragements. I was always motivated by his hard work and excellence in research. He taught me how to find problems that connects with the real world and represent them in a formal way. He also guided me to find the solutions of the problems which might be eligible for publications. He is also very open minded to give me opportunity to explore different areas of research and pick based on my own interest. I also learned to implement different system level applications and run experiments under his supervision. He encourages to explore different research ideas and brainstorming with me to find a suitable one that might be a potential to explore deeply. Besides this, he always encourages me to do well in the courses that I have taken during my graduates study. Without his support is wont be possible to enroll and audit in many courses which helped me to develop myself for research. I feel the luckiest person to work with him in my graduate study.

I also feel lucky to work with Dr. Avinash Srinivasan in a network forensic project. Because of the project being a collaboration between academic institution (Temple University) and industry (TDI Technologies), i learned a lot about the industry trends and practical issues. I have learned a lot about network security required by US navy ships. I have experienced working with different computing devices used in navy ships which won't be possible without the help of Dr. Avinash Srinivasan and TDI Technologies.

I would like to thank Dr. Xiojiang Du for giving me opportunity to work with and learn. I specially thank him for his guidance on papers published in IEEE ICC and CPS journal. He provided me with many useful suggestions on exploring theoretical ideas during that research. Dr. Xiuqi Li has been another awesome collaborator and mentor. She guided and corrected me during a semester in my research. I feel happy to have a publication co-authored with her.

I was also benefited a lot from collaborations with many people that led to this thesis. I especially thank Ning Wand and Huanyang Zheng for insightful discussions in Chapter 2; Dr. Wei Chang and Dr. Pouya Ostovari for contributing in Chapter 5; Dr. Avinash Srinivasan for contributing in Chapter 6; and Yang Chen for contributing in Chapter 7. I would also like to thank all my colleagues and friends including Jiacheng Shang, Yubin Duan, Suhan Jiang, and Nadia Niknami on various projects. I would like to thank Yang Chen for her great support during my graduate study and research projects.

This dissertation could not have been completed without the support of the National Science Foundation (NSF), Army Research Office (ARO), and Office of Naval Research (ONR). Finally, I would like to thank my family for their patience and unconditional support.

This dissertation is dedicated to...
my loving and encouraging family members, friends, and colleagues who supported me a
lot during my graduate study at Temple University.

TABLE OF CONTENTS

ABSTRACT	iii
ACKNOWLEDGEMENTS	vi
LIST OF TABLES	xvi
LIST OF FIGURES	xvii
1 INTRODUCTION	1
1.1 Explored Research Areas	1
1.2 Introduction to Different Network Attacks	3
1.2.1 Distributed Denial-of-Service Attack	4
1.2.2 Transit-link DDoS Attack	5
1.2.3 Internal DDoS Attack	6
1.2.4 DDoS Attacks in Low Power Lossy Networks	7
1.2.5 Non-volumetric Attacks	8
1.2.6 Contributions: Defense Strategies and Network Optimizations . .	9
1.3 Chapter Overview	11
2 FILTER ASSIGNMENT POLICY TO MITIGATE DISTRIBUTED DENIAL-OF-SERVICE ATTACKS	13
2.1 Introduction	13
2.2 Related Works	16
2.3 System Model	17

2.4	Source-based Filter Assignment Policy	21
2.4.1	Problem 1: Find a filter assignment so that the contamination is minimal by ensuring that all the attack traffic is blocked before reaching the victim.	21
2.4.2	First Greedy Solution	22
2.4.3	Second Greedy Solution	23
2.5	Destination-based Filter Assignment Policy	26
2.5.1	Problem 2: Find a filter assignment so that the LU blockage and contamination are minimal.	26
2.5.2	A Dynamic Programming Solution	26
2.6	Simulations	31
2.6.1	Simulation Setting	31
2.6.2	Simulation Results	33
2.7	Run-time Improvement Using Heap	34
2.7.1	$O((N - B)^2 \log N)$ approach for Second Greedy Solution	34
2.8	Summary	36
3	OPTIMAL FILTER ASSIGNMENT POLICY AGAINST DISTRIBUTED DENIAL-OF-SERVICE ATTACK	37
3.1	Introduction	37
3.2	Related Works	40
3.3	System Model	42
3.4	Source-based Filter Assignment Policy	45
3.4.1	Problem 1: Find a filter assignment so that the contamination is the minimum by ensuring that all the attack traffic is blocked before reaching the victim.	45
3.4.2	An Optimal Solution	46
3.4.3	An Example	49
3.5	Destination-based Filter Assignment Policy	53

3.5.1	Problem 2: Find a filter assignment so that the LU blockage and contamination are the minimum.	54
3.5.2	An Optimal Solution	55
3.5.3	Example	57
3.6	Simulations	60
3.6.1	Simulation Setting	60
3.6.2	Simulation Results	62
3.7	Summary	64
4	PROTECTING RESOURCES AGAINST VOLUMETRIC AND NON-VOLUMETRIC NETWORK ATTACKS	66
4.1	Introduction	66
4.2	Related Works	69
4.3	System Model	71
4.3.1	Network Model	71
4.3.2	Attack Model	72
4.3.3	Cost, Budget and Damage Model	73
4.4	Volumetric Attack Problem Formulation	73
4.4.1	Solution	74
4.4.2	An Example	75
4.5	Non-volumetric Attack Problem Formulation	76
4.5.1	Solution	77
4.5.2	An Example	80
4.6	Simulation	83
4.6.1	Simulation Settings	83
4.6.2	Simulation Results of Volumetric Attack	85
4.6.3	Simulation Results of Non-volumetric Attack	88

4.7	Summary	91
5	OPTIMAL FILTER ASSIGNMENT POLICY AGAINST TRANSIT-LINK DISTRIBUTED DENIAL-OF-SERVICE ATTACK	92
5.1	Introduction	92
5.2	Related Work	95
5.3	System Model	97
5.3.1	Network Model	97
5.3.2	Attack Model	99
5.4	Problem Definition	101
5.5	Optimal Solution for Problem I	101
5.6	Extension of Problem I	106
5.7	An Optimal Solution for Problem II	107
5.8	Experimental Results	110
5.8.1	Experimental Settings	110
5.8.2	Simulation Results	112
5.9	Summary	115
6	OPTIMAL MONITOR PLACEMENT POLICY AGAINST DISTRIBUTED DENIAL-OF-SERVICE ATTACK IN DATACENTER	116
6.1	Introduction	116
6.2	Related Work	119
6.3	System Model	120
6.3.1	Datacenter Model	120
6.3.2	Attack Model	122
6.4	Flow Assignment	123
6.4.1	Problem 1: Find a flow assignment so that the network overhead is the minimum by ensuring coverage of all internal flows.	123
6.4.2	An Optimal Flow Assignment Scheme	124

6.5	Monitor Placement	127
6.5.1	Problem 2: Find locations for monitors and a flow assignment so that the network overhead is the minimum by ensuring coverage of all the internal flows.	127
6.5.2	Greedy Approximation: M/K -lowest cost approach	127
6.6	Experimental Results	129
6.6.1	Experimental Settings	129
6.6.2	Simulation Results	131
6.7	Summary	135
7	SAMPLING RATE DISTRIBUTION FOR FLOW MONITORING AND DDOS DETECTION IN DATACENTER	136
7.1	Introduction	136
7.2	Related Work	140
7.3	System Model	141
7.3.1	Datacenter Network Model	142
7.3.2	Attack Model	142
7.3.3	Monitoring System Overview	143
7.4	sampling rate Distribution	144
7.4.1	Flow Grouping	144
7.4.2	Relationship between Detection and Sampling Rate	148
7.5	Experiments	152
7.5.1	Experimental settings	152
7.5.2	Experiment results	154
7.6	Summary	161
8	MINIMIZING THE NUMBER OF RULES TO MITIGATE LINK FLOODING ATTACK IN SDN-BASED DATACENTERS	162
8.1	Introduction	162

8.2	Related Work	165
8.3	System Model	166
8.3.1	Network Model	167
8.3.2	Attack Model	168
8.4	Redirecting a flow	169
8.4.1	A Dijkstra-based Solution	170
8.4.2	An Example	170
8.5	Redirecting multiple flows	171
8.5.1	A Flow Grouping Solution	173
8.5.2	An Example of Common k Links Grouping	174
8.5.3	Greedy Rule Merging Solution	177
8.5.4	An example	178
8.6	Simulations and Experiments	179
8.6.1	Simulation Settings	179
8.6.2	Simulation result	181
8.7	Experiments in Datacenter	185
8.7.1	Experimental Settings	185
8.7.2	Experimental Results	186
8.8	Conclusion	187
9	EFFICIENT SWITCH MIGRATION FOR CONTROLLER LOAD BALANCING IN SOFTWARE DEFINED NETWORKING	188
9.1	Introduction	189
9.2	Related Works	191
9.3	Network Model	193
9.3.1	Cost Model	194
9.4	Initial Deployment	196

9.4.1	Solution Approaches	196
9.4.2	Greedy Controller Assignment	196
9.4.3	An Example to Greedy Controller Assignment	197
9.4.4	Hierarchical Clustering	199
9.4.5	An Example of Hierarchical Clustering Solution	200
9.5	Incremental Deployment	201
9.5.1	Solution Approaches	203
9.5.2	Greedy Incremental Deployment	203
9.5.3	An Example of Incremental Assignment	204
9.6	Simulation and Experiments	204
9.6.1	Experimental Settings	204
9.6.2	Simulation Settings	206
9.6.3	Simulation Results	207
9.7	Summary	210
10	CONCLUSION AND FUTURE WORKS	211
10.1	Summary of Contributions	211
10.2	Future Extensions	213
10.2.1	Defending Network Attacks	213
10.2.2	Optimization in Software Defined Networks	214
10.2.3	Improving Reliability in Service Function Chain	214
10.3	Concluding Remarks	215
	Related Publications	216
	Under Revision/Submitted	217
	BIBLIOGRAPHY	218

LIST OF TABLES

2.1	Topology Parameters	31
3.1	Topology Parameters	61
4.1	Topology Parameters	83
5.1	Table of notations	100
5.2	Topology Parameters	112
6.1	Topology Parameters	130
6.2	Random tree topology settings	130
7.1	Effect of SYN flood attack.	138
7.2	Groups of nodes for Flows I using Algorithm 19	152
7.3	Groups of nodes for Flows II using Algorithm 19	153
7.4	sampling rates for different K for Flows I.	159
7.5	sampling rates for different K for Flows II.	159
8.1	Flows and bandwidth	174
8.2	Topology Parameters	178
8.3	Shortest and alternative path rules	185

LIST OF FIGURES

1.1	Explored research areas.	3
1.2	DDoS attack by bots.	5
1.3	Transit-link DDoS attack by bots.	6
1.4	Internal DDoS attack (A_1 and A_2 : attacker, v : victim).	6
1.5	A DDoS attack in a LLN.	7
1.6	An example of non-volumetric attack.	8
2.1	DDoS attack by bots.	15
2.2	System model and constructed topology.	19
2.3	Topologies for problems 1 and 2.	22
2.4	Topology for Problem 2.	25
2.5	A , R , L , and C	29
2.6	Simulation results.	32
2.7	More simulation results.	34
2.8	H and g	35
3.1	DDoS attack by bots.	39
3.2	System model and constructed topology.	43
3.3	Topologies for problems 1 and 2.	46
3.4	Complete values of A_1 , A_2 , R_1 , and R_2	46
3.5	Recursion model.	48

3.6	<i>A</i> , <i>R</i> , and <i>L</i>	56
3.7	Randomly generated and real topologies.	60
3.8	Simulation results.	60
3.9	More simulation results.	61
4.1	An example of attack and defense mechanism.	68
4.2	An example for volumetric attack.	73
4.3	An example for non-volumetric attack.	78
4.4	Values of <i>A</i> and <i>D</i> for all nodes and <i>k</i>	78
4.5	Values of <i>C</i> and <i>L</i> for all nodes and <i>k</i>	78
4.6	Randomly generated topologies.	83
4.7	Simulation results of Problem 1.	84
4.8	More simulation results of Problem 1.	85
4.9	Simulation results of Problem 2.	88
5.1	Transit-link DDoS attack by bots.	93
5.2	System model.	97
5.3	Problem I example.	100
5.4	Problem II example.	104
5.5	Topologies I and II. (Green node=user, red node=bot, blue node=victim, red link=possible congested link)	110
5.6	Packet distribution of different scenarios in Topology II.	111
5.7	Simulation results.	113
5.8	More simulation results.	114
6.1	Internal DDoS attack (A_1 and A_2 : attacker, v : victim).	117
6.2	System model.	120
6.3	Problem I example.	124
6.4	Problem II example.	126

6.5	Randomly generated topologies.	129
6.6	Simulation results.	132
7.1	Attacker and sampling rate distribution.	137
7.2	Monitoring system.	142
7.3	An example of flow grouping using second level behavioral similarity.	144
7.4	Structure of the ANN.	151
7.5	Datacenter topology.	153
7.6	Effect of small scale SYN flood attack.	155
7.7	Grouping of nodes based on second level similarity.	155
7.8	Sampling rate vs. DDoS detection rate.	157
7.9	Uniform and grouped distribution.	158
7.10	Uniform and grouped distribution in Spark.	159
7.11	Comparison with k -means clustering.	161
8.1	(a) Flow redirection due to LFA and (b) transmission time vs. # rules.	164
8.2	An example for Problems I and II.	171
8.3	Merging rules.	176
8.4	CT of a randomly generated topology for different destinations (green node: source, blue node: destination, red link: new rules added, black link: existing rule).	180
8.5	Number of rules per redirected flow with different settings.	180
8.6	Comparison with shortest alternative path routing.	183
8.7	Effects on ping delay.	184
8.8	Datacenter topology (partial).	186
9.1	An Example of switch migration.	190
9.2	Example for initial switch migration.	198
9.3	Distance matrix for clustering.	199

9.4	Example for incremental switch migration.	201
9.5	Generated topologies.	205
9.6	Simulation results.	206

CHAPTER 1

INTRODUCTION

In this chapter, we present an overview of the research areas and different problems in some selective areas. We also present an overview of the solutions in a high level and outline the structure of this thesis briefly.

1.1 Explored Research Areas

During graduate study, we have explored different research areas of computer networks. We divide the explored research areas based on network types into two major parts: wireless and wired. In both areas, we have explored multiple security and optimization issues.

Firstly, in the wireless security areas we have explored research problems in source location privacy issue of wireless sensor networks. In this research, we propose the controlled routing protocol, a mixture of shorted path and random routing protocols that maintains a good balance between security and efficiency. Our proposed protocol is based on two principles: if all the messages do not follow the same path, then backtracking to the source node is not possible and when an adversary is very far away from the source and destination locations, then efficiency is more important than security. The research was published in [Related1]. We also propose a mitigation system of spectrum sensing falsifying attack in cognitive radio networks. We propose a reputation-based mechanism

for cooperative spectrum sensing which can minimize the effects of attackers on decision making. The research was published in [Related2, Related3]. We also propose a Distributed Denial-of-Service (DDoS) attack framework in low power lossy networks which is published in [Related4].

Secondly, in the wireless optimization areas we have explored research problems in 5G technologies and cognitive ad-hoc networks. We study the fair coexistence between LTE-U and Wi-Fi in the scenario where an LTE eNB exchanges information with Wi-Fi access points (AP). The communication is done in both wired and wireless mediums. The LTE eNB adjust its parameters according to the received information to achieve fairness. This research was published in [Related5]. When a user of cognitive ad-hoc network moves from one place to another, it needs to switch the channel to maintain the quality-of-service (QoS) required by different applications. In this research we study the mobility patterns of users, predict their next locations and probabilities to move there based on its history. We extract the mobility patterns from each user's location history and match the recent trajectory with the patterns to find future locations. We formulate a problem to select the current channel in order to minimize the total number of channel switches during a certain number of next moves of a user. This research was published in [Related6]. However, this researches are not directly related to our dissertation focus and we are not presenting them in this dissertation.

Thirdly, in the wired security areas we have explored different network attacks including, Distributed Denial-of-Service (DDoS) attacks, Transit-link DDoS attacks, internal DDoS attacks, and non-volumeric attacks. We proposed different mitigation systems which will be discussed through this dissertation.

Finally, we have conducted some researches on wired network optimization specially for software defined networks. This optimizations includes but not limited to minimization of network delays, load balancing of controllers, and minimizing the number of rules that will be presented in Chapters 8 and 9.

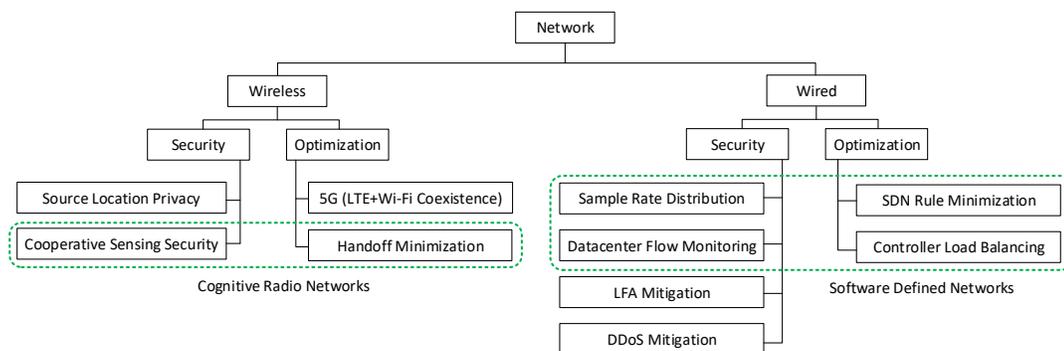


FIGURE 1.1: Explored research areas.

1.2 Introduction to Different Network Attacks

During the last few years the number of internet users are increasing rapidly. We observe a growth of about 300 millions of user each year on an average [1]. With the increase of internet users the amount of network attacks are also increasing rapidly. The objective and strategy of attacks are also evolving day by day. We discuss objectives and strategies of different attacks in this chapter. Based on the effectiveness of the volume of attack packets we divide the attacks into two categories:

1. **Volumetric attacks:** The amount of damage depends on the amount of attack volume. The packets are not harmful or contain malware but the amount creates congestion in the network and cause it to stop serving regular users.
2. **Non-volumetric attacks:** The amount of traffic is not related to the damage to the resources. The attacker intends to steal the content of the resource or gain special access permission to the resources.

Network attacks have evolved a lot during the last decade. The cost of conducting an attack has also decreased. For example, renting 1000 bots from any location costs from a few dollars to little more than 100 USD [2]. Attackers' strategies, capabilities, and goals have also evolved a lot during these years. For example, an attack against

Spamhaus in 2013 revealed that the attacker can change the target adaptively from the endpoint server according to the defense mechanism. Another attack against ProtonMail in 2015 demonstrated that the attackers changed their strategy in real-time according to the defense mechanism [3]. The moving target and attack strategy form an interactive game scenario between the defender and the attacker.

1.2.1 Distributed Denial-of-Service Attack

The distributed denial-of-service (DDoS) attack is one of the volumetric attacks. In a DDoS attack, a huge number of bots send a lot of service requests to the victim, resulting in network congestion and/or out of processing capability of the victim. As a result, the victim cannot serve its regular user and denial-of-service or degradation of quality-of-service (QoS) happens. The bots are usually malicious programs and controlled by a coordinator. The coordinator sends commands including the information of the victim to the bots and they act accordingly.

Nowadays, DDoS attacks are the cheapest and most powerful attacks among all others. Besides, DDoS attacks are increasing day by day in both number and size; CloudFlare [4] recently reported a 400 Gbps massive DoS attack that took place at their servers. There are several types of DDoS attacks including SYN Floods, Malformed Packets, UDP Floods, Amplification Attacks, and Distributed Attacks [5]. In a SYN Flood attack, the perpetrator sends many SYN messages for TCP connection setup. The server replies ACK and waits for the client's ACK but the attacker does not reply ACK and the connection remains half-open till timeout. The objective of a SYN flood is to simply fill up the limited slots that the target system has available for half-open connections. In some cases it's easy to detect a SYN Flood attack if a lot of SYN requests come from an address in short interval. Detection is harder, however, when the attacker spoofs IP address, SYNs come from multiple addresses, and arrival time varies. In a UDP Flood attack, the purpose would likely be to consume all available network bandwidth. Attackers send a large amount of

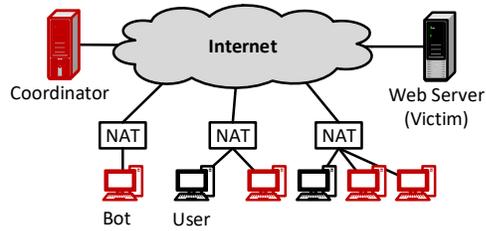


FIGURE 1.2: DDoS attack by bots.

spoofed requests with large useless payload. The application wastes CPU cycles trying to determine the meaning of the garbage. The objective of DDoS is to generate a lot of packets from different locations to exhaust incoming/outgoing bandwidth of the victim. A coordinator would send commands to workers who continue to send requests to the target. The workers are known as bots and the network of workers is known as botnet. As normal users also request through the NAT, it is difficult for the victim to differentiate between the bot requests and normal user requests. Fig. 1.2 shows the DDoS attack model by botnet.

1.2.2 Transit-link DDoS Attack

Transit-link DDoS or link flooding attacks are volumetric attacks. The transit-link DDoS attack using botnet is one of the most challenging DDoS attacks. The strategy and goal are slightly different from the traditional DDoS attack. In the traditional DDoS attack, the bots generate packets destined to the victim. In transit-link DDoS attack, the bots generate traffic that is not destined to the victim but the packets congest at least one of the links on the paths from the legitimate users to the victim. Because of the congested links, the legitimate traffic suffers from packet drop and delay. The transit-link DDoS can be so devastating that it can disconnect a server from the Internet.

Fig. 1.3 shows a scenario of a transit-link DDoS attack. The bots attached to NAT_1 generate huge traffic towards the decoy server server v'_2 . Decoy server servers are used to receive the attack packets from bots. Decoy server servers may be owned by the attacker and the attack packets congest one or more links on the path to decoy servers. The packets

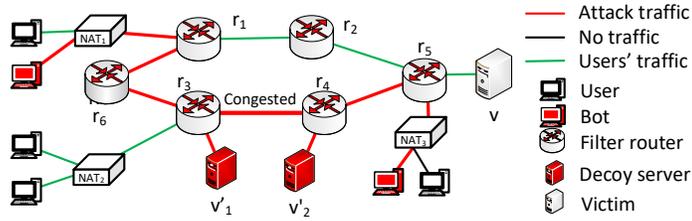


FIGURE 1.3: Transit-link DDoS attack by bots.

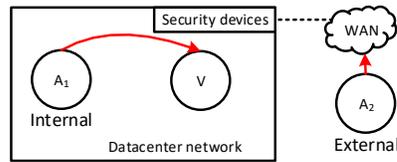


FIGURE 1.4: Internal DDoS attack (A_1 and A_2 : attacker, v : victim).

follow the path $r_1 \leftrightarrow r_6 \leftrightarrow r_3 \leftrightarrow r_4$. Similarly, the generated traffic from the bots attached to NAT_3 follows the path $r_5 \leftrightarrow r_4 \leftrightarrow r_3$ to reach v'_1 . The link $r_3 \leftrightarrow r_4$ becomes congested because of the traffic from bots from NAT_1 and NAT_2 . When the legitimate packets from the user attached to NAT_2 travel the shortest path $r_3 \leftrightarrow r_4 \leftrightarrow r_5$, they face delay, packet drop, and DoS.

1.2.3 Internal DDoS Attack

An internal DDoS attack is variant of DDoS attack, where the attackers reside inside the datacenter network. A traditional system cannot protect against internal DDoS attacks because the security modules remain in the core or aggregation layer, which monitors the incoming (or outgoing) traffic from (or to) a datacenter. Fig. 1.4 shows a scenario of an internal DoS attack. The victim V is protected by the security devices that remains at the edge routers. Therefore, the external attacker A_2 's traffic is blocked by the security modules. The internal attacker A_1 resides in the same datacenter as V . A_1 's traffic does not go through the datacenter security devices. As a result, the attack traffic reaches V

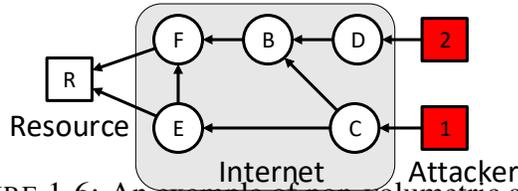


FIGURE 1.6: An example of non-volumetric attack.

at the root and DoS occurs. Fig. 1.5 shows an attack scenario in LLN. Nodes M , N , and K are attackers and they are controlled by a coordinator. The attackers are compromised devices and the coordinator can be one of them. The coordinator's goal is to maximize the attack strength. The easiest way to achieve this is to activate all the attackers. In reality, the strongest attack can be launched by activating a subset of the attackers. This is because each link has a limited bandwidth and attackers can attack a limited number of neighbors. Attacking a neighbor means sending packets with the wrong routing information at the highest rate. For example, if the coordinator selects N and K , and they both attack F and G , then the attack intensity will not be the highest. Assume an attacker can attack two neighbors at the same time and the bandwidth of each link is the same. This is because error messages from F for N and K 's packets travel through link $F \rightarrow C$ which has a limited bandwidth. Error messages from G travel back through link $G \rightarrow N$ which is used for the attack. The coordinator selects the attackers and the targeted neighbors wisely to maximize the strength of the attack.

1.2.5 Non-volumetric Attacks

In a non-volumetric attack, the amount of traffic is not proportional to the damage to the resources. The attacker intends to steal the content of the resource or gain special access permission to the resources. To achieve the final goal, the attacker needs to pass multiple steps. There can be one or multiple ways (series of steps) to reach the goal. For these types of attacks, the defender needs to cut all possible ways to reach the resources. Let us consider the figure in Fig. 1.6 and assume the nodes are the steps. The first step of the attackers is either passing through D or C . In reality, the D step can represent the guessing

of the password of a server. Step B can be gaining root privilege of that server. When an attacker passes one more step, it gets closer to the goal and causing damage to the network or datacenter. To prevent the attacker, the defender needs to apply a moving target defend (MTD) approach to stop the attacker at some particular steps. Simultaneously, the attacker needs to be stopped as early as possible to minimize the damage.

1.2.6 Contributions: Defense Strategies and Network Optimizations

To defend volumetric attacks including DDoS and transit-link DDoS attack we use a special type of router called filter router (FR). A FR is a special kind of router which is capable of two functionalities. Firstly, it can do packet marking, which is used to construct traffic topology at the victim. Secondly, it can receive filters from the victim and apply the filters to block the attack traffic according to the filter definitions. The filter are also capable of blocking links in a FR for specific traffic defined by source and/or destination address. These filters are used to block attack traffic and mitigate DDoS attacks. The details of the filter deployment policy is presented in Chapter 2. We have proposed filter assignment policy for different types of network topologies including tree and directed acyclic graphs.

To prevent an internal DDoS attack, we need to do two things: monitor all the internal traffic and block the attack traffic. To block traffic, the controller creates a rule in the hypervisor firewall which can be done easily. Therefore, we focus on monitoring all internal traffic. The security modules are expensive and they remain in the core or aggregation layer, which monitors the incoming (or outgoing) traffic from (or to) a datacenter. Therefore, the external attacker traffic is blocked by the DDoS protection server but the internal attacker's traffic does not go through the datacenter security devices. As a result, the attack traffic reaches the victim without any trouble. This is why we need to monitor all the traffic in a datacenter. Some of the free virtual machines in the datacenter (VMs) are used as traffic monitors. The SDN switches probabilistically copy packets of each flow to a monitor. We formulate a problem for assigning flows to monitors, considering that

the location of the monitors are predefined and provide an optimal solution by reducing the problem to a max-flow min-cost problem. Details are presented in Chapter 6. Now, if the capability of monitoring is lower than the amount of traffic in the datacenter, the flows need to be sub-sampled to meet the capability of the monitors. We propose an optimal sampling rate distribution approach for monitoring all the flows that maximizes the detection rate of attack traffic. Details are presented in Chapter 7.

Next, we focus on defending non-volumetric attacks. In this attack, the amount of damage does not depend on attack traffic. The attacker intends to steal the content of the resource or gain special access permission to the resources. To achieve the final goal, the attacker needs to pass multiple steps. The defender needs to cut all possible ways to reach the resources. To prevent the attacker, the defender needs to apply a moving target defend (MTD) approach to stop the attacker at some particular steps. In this research, we propose an MTD assignment policy for directed acyclic graph formed attack network to stop the attacker reaching its goal with the minimum number of MTD deployment. Chapter 4 presents details of the system.

Usage of software defined networking (SDN) in datacenters enables dynamic and convenient configuration management that makes it easy to reconfigure the network to mitigate the link flooding attacks (LFAs). The reconfiguration that redirects some of the traffic can be done in two ways: the shortest alternative path and the minimum changes in rule path. The SDN switches have a limited capacity for the rules and the performance dramatically drops when the number of stored rules is higher. Besides, it takes some time to adopt the changes by the SDN switches which causes interruption in flow. We aim at minimizing the number of rule changes while redirecting some of the traffic from the congested link. We consider multiple flows and propose flow grouping and rule merging based solutions for redirecting traffic on the congested link that minimized the number of rules changes.

As the size of the SDN datacenters are increasing, the number of controllers in a datacenter is also increasing. The performance of an SDN datacenter depends largely on the

delay of response from the controller. The load of a controller depends on the number of requests it receives from the switches it controls. Therefore, a well switch-controller assignment is very important for load balancing the controller and the performance of an SDN datacenter. We consider multiple controllers and formulate problems for initial and incremental load balancing. The initial assignment process is executed at the beginning of the network deployment and the incremental assignment process is executed periodically. The incremental process migrates some of the switches to another controller to improve the performance of the network. We propose greedy and clustering-based solutions for initial switch-controller assignment. We also propose a greedy solution for incremental assignment.

1.3 Chapter Overview

The dissertation is organized in the following sequence: DDoS attack defense, non-volumetric attack defense, transit-link attack defense, internal attack defense by flow monitoring, optimizing flow monitoring system, minimizing number of rules in SDN networks, load balancing SDN networks. In Chapter 2, we present the mitigation mechanism of DDoS attack using filters and filter routers and we propose some greedy and dynamic programming based non-optimal solutions for filter assignment problems in tree topologies. In Chapter 3, we study deeply the problem introduced in Chapter 2 and propose optimal solutions of them. Chapter 4 presents filter assignment policy for volumetric attack and MTD deployment strategies for non-volumetric attacks in directed acyclic graph based topologies. After that transit-link DDoS mitigation mechanism is proposed in Chapter 5. We propose optimal filter assignment policy to redirect legitimate traffic via a non-congested paths using minimum number of filters. Chapter 6 presents a flow monitoring system which is important to detect internal attacks. In Chapter 7, we propose a sampling rate distribution approach when the capability of the monitors is less than the amount of traffic in a datacenter. Chapters 8 and 9 focuses on performance optimization in SDN networks. Chapter 8

presents a mechanism to mitigate link congestion/link flooding attack by redirecting some flows by spending the minimum number of rules. In Chapter 9 we present our research on optimizing SDN network by load balancing the controllers in a multi-controller SDN network. Finally, Chapter 10 concludes our dissertation with some direction to future works.

CHAPTER 2

FILTER ASSIGNMENT POLICY TO MITIGATE DISTRIBUTED DENIAL-OF-SERVICE ATTACKS

In this chapter, we propose a Distributed Denial-of-Service (DDoS) attack by using filter router and filters. We present the proposed filter assignment policy that minimizes the amount of attack traffic in the network with a limited budget on the number of filters. The contents of this paper is published in [Related7].

2.1 Introduction

A denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource unavailable temporarily to its users. DoS attacks are considered a federal crime under the Computer Fraud and Abuse Act with penalties that include years of imprisonment [7]. The Computer Crime and Intellectual Property Section of the US Department of Justice handles cases of DoS attacks. Therefore, detecting DoS attacks and identifying attackers have been an important issue in Network Forensics. Moreover, DoS attacks are increasing day by day in both number and size; CloudFlare [4] recently reported a 400 Gbps massive DoS attack that took place at their servers.

There are several types of DoS attacks including SYN Floods, Malformed Packets, UDP Floods, Amplification Attacks, and Distributed Attacks [5]. The objective of DDoS

is to generate a lot of packets from different locations to exhaust incoming/outgoing bandwidth of the victim. A coordinator would send commands to workers who continue to send requests to the target. The workers are known as bots and the network of workers is known as botnet. As normal users also request through the NAT, it is difficult for the victim to differentiate between the bot requests and normal user requests. Fig. 2.1 shows the DDoS attack model by botnet. An effective method of preventing DDoS attack is to use filter routers (FRs) in network infrastructure. FRs are a special type of router which is capable of packet marking and receiving filter tasks. Packet marking task refers to attaching the FR's own IP address probabilistically to the packets it forwards while receiving filter task refers to receiving filters from a web server. A web server can block all or part of the traffic destined to it. The complete method of using FRs is a four-phase process. In the first phase, the FRs mark forwarded packets by appending its own IP address probabilistically. In the second phase, the victim constructs the traffic topology from the marking of the packets. In the third phase, the victim constructs filter, finds, and selects some FRs to assign the filters. In the last phase, the FRs evict unused filters from its storage. The packet marking is used to find the topology by the victim. The probability of marking is a tradeoff between topology construction time and router overhead. After topology construction, the victim generates filters and selects a subset of the FRs to assign them. There are two types of filters: source-based and destination-based. Using source-based filter a FR can allow/block traffic from specific sources which are destined for the victim. This type of filter is vulnerable to IP spoofing attacks. Though destination-based filter can prevent IP spoofing but it is more restrictive. Using a destination-based filter, a FR can block all traffic including LU traffic destined to the victim. It is challenging to find an optimal assignment of filter when the victim has a limit on selected number of FRs. A FR may get a number of filters from multiple victims. It has limitation of storage and computation power. Therefore, it evicts filters which are no longer used.

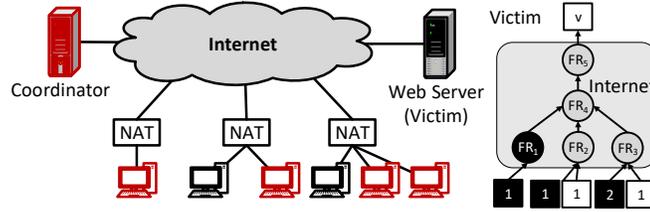


FIGURE 2.1: DDoS attack by bots.

In this paper, we focus on finding optimal filter assignment assuming that the victim has already constructed the traffic topology. We formulate two problems and propose solutions for them. In the first problem, a limited number of source-based filters are assigned to the FRs. For example, if the victim can assign 2 filters, it can select $\{FR_1, FR_2\}$, $\{FR_1, FR_5\}$, $\{FR_2, FR_4\}$ or another pair of FRs (see Fig. 2.1). If the victim selects the first pair of FRs, the attack traffic from FR_3 will reach the victim which is highly unexpected. If the second pair is selected then the attack traffic will travel through (FR_2, FR_4) , (FR_3, FR_4) , and (FR_4, FR_5) links. The amount of attack traffic in each link is not same. It is challenging to find a filter assignment for which the total amount of attack traffic is minimum. We propose greedy approximation solutions for this problem. In the second problem, a limited number of destination-based filters are assigned to the FRs. Destination-based filter blocks every packet at FR that is destined to the victim. If the victim selects the third pair, then all the legitimate users (LUs) will be blocked and the attack traffic will travel through (FR_2, FR_4) and (FR_3, FR_4) links. It is also challenging to find a filter assignment so that the total attack traffic and number of blocked LU are minimum. We propose a dynamic programming solution for this problem. Our main contributions are the following:

1. We formulate two problems for finding filter assignment with a budget (limited number of filters) and provide greedy and dynamic programming solutions.
2. We present simulation results to support our model.

The remainder of this paper is arranged as follows: Section 2.2 presents some related works and their limitations. In Section 2.3, we present the system model for preventing DDoS attack. Section 2.4 presents the formal definition of the first problem and our proposed greedy solutions. Section 2.5 presents the formal definition of the second problem and our proposed dynamic programming solution. In Section 2.6, we present some simulation results that strengthen our proposed solutions.

2.2 Related Works

There exist many statistical methods including correlation, entropy, covariance, divergence, cross-correlation, and information gain to detect anomalous DDoS requests [8]. A rank correlation-based method Rank Correlation-based Detection (RCD) is proposed in [9]. An information theoretical approach using Kolmogorov complexity is used for detection of DDoS attacks in [10]. A novel DDoS detection mechanism is proposed based on Artificial Neural Networks in [11]. There are other methods of detecting DDoS attack including [12, 13, 14, 15]. Authors in [16] introduced a model of randomized DDoS attacks with increasing emulation dictionary where the attackers use the attack definition from the dictionary that contains request patterns similar to those of LUs. They proposed an inference algorithm for identifying the botnets executing such DDoS attacks. Nowadays, static path identifiers are used for inter-domain routing objects, which makes it easy for attackers to launch the DDoS flooding attacks. In [17], the authors present a design dynamic path identification framework that uses path identifier negotiated between the neighboring domains as inter-domain routing objects.

In [18], the authors propose a method, RADAR, to detect and throttle DDoS attacks using adaptive correlation analysis on SDN switches. The system can defend against a wide range of flooding-based DDoS attacks including link flooding, SYN flooding, and UDP-based amplification attacks. In [19], the authors propose a new approach which reduces the resource utilization factor to a minimal value for quick absorption of the attack.

In [20], a DDoS protection mechanism, SkyShield, is proposed by taking advantages of the sketch techniques. To identify malicious hosts efficiently, they used the abnormal sketch obtained from the last detection cycle. The SkyShield could leverage other techniques including Bloom filters and the CAPTCHA. In [21], the authors propose a collaborative DDoS mitigation network system in which a domain helps another domain. A domain can direct excessive traffic to other trusted external domains for DDoS filtering. The filtered clean traffic is forwarded back to the targeted domain.

Most of the existing works are mainly concerned about the service availability of the server. In fact, the attack traffic may cause huge network congestion and DoS. Therefore, these techniques cannot protect the network from being contaminated by attack traffic. A victim and network component collaboration based system can help in this case. A four-phase DDoS protection system is proposed in [22]. The victim generates filters and sends them to the upstream FRs. The FRs send the filters to its upstream FRs and thus the filters propagate to the effective FRs. An adaptive version of PFS is proposed in [23]. The system sends directly filters to the high capable FRs first, then the filters propagate to the effective FRs. However, these two systems cannot select the FRs optimally when there is a limitation on selecting FRs.

2.3 System Model

Our system is composed of legacy routers (LRs), network address translators (NATs), filter routers (FRs), attackers, legitimate users (LUs), and a victim (v). Fig. 2.2 shows the complete system model. In reality, there are multiple victims in a network but for simplicity of explanation we are considering a single victim. We assume that end users are connected to a FR or a LR through NAT. The FRs are a special kind of router which are capable of two functionalities. Firstly, it can do packet marking which is used to construct traffic topology at the victim. Secondly, it can receive filter from the victim and apply the filter to block the attack traffic according to the filter definition. There can be two types

of filter: source-based and destination-based. The source-based filter specifies blocking of traffic based on source address. For example, a source-based filter can be understood: *if source address is X then discard the packet*. If we use a source-based filter at FR_3 (assume X and Y are the IP addresses of the NATs connected to FR_1 and FR_2 , respectively) then FR_3 will discard packets coming from NAT- X but forward packets from the NAT- Y .

The advantage of using source-based filter is that a FR can block the attack traffic by its source IP address and forward legitimate traffic. If the LU and attacker both remain behind the same NAT, then it is impossible to block only attack traffic. The limitation of the source-based filter is that it cannot protect if an attacker spoofs the IP address of a LU. If an attacker creates packet having Y as the source address, then the packet will not be blocked at FR_3 . To protect against DDoS attack with IP spoofing, we can use destination-based filter. A destination-based filter is *if the destination address is X then discard the packet*. For example, if we use a destination-based filter at FR_3 (assume that X is IP address of v), then all the packets including legitimate and spoofed attack packets will be blocked by FR_3 . The destination-based filter is more restrictive. When a FR uses it, it blocks all the attack and legitimate traffic destined for the victim. Therefore, spoofed attack traffic cannot penetrate.

The attackers are usually user devices which have compromised programs that can generate traffic destined for a target. The programs are controlled by a master. The master can send commands of attack to the program. This type of program is called bot and the network of bots is called botnet. Though the DDoS traffic is hard to differentiate from legitimate traffic, there exist several methods based on arrival time, packet size, and packet content for detecting attack packets [8]. In this paper, we are not focusing on the detection of attack packet and assume that the victim can find out the source address of attack traffic using these methods. The victim also knows the packet rate of each attacker. For example, if there are 100 attackers (or LUs) each with 1 Mbps attack traffic (or legitimate traffic)

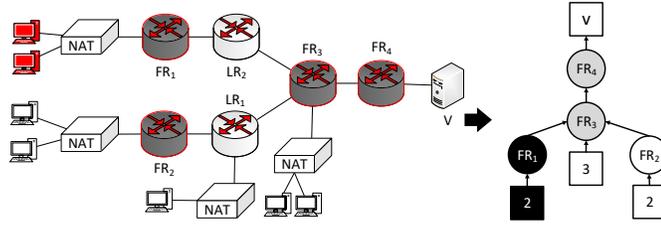


FIGURE 2.2: System model and constructed topology.

behind a NAT-X (having IP X), then the victim will identify the X as an attacker (or LU) with 100 Mbps attack traffic (or legitimate traffic).

The complete protection process consists of the four phases.

1. **Packet marking by FRs:** The process of appending FR's own IP address probabilistically in a special field of packet header.
2. **Construct traffic topology:** The process of a victim constructing the topology from the appended IP addresses of packets it receives.
3. **Construct filter and assign to FRs:** After analyzing the received packets over a period, the victim can identify the IP addresses of the attacker. Then it needs to find some FRs and send some filter to block the attack traffic.
4. **Evict unused filter from FR:** A FR has limited storage. When a filter is not used for a period of time, the FR evicts the filter from its storage.

In **phase 1**, the FRs probabilistically mark the packet it forwards by appending its own IP address. Assume the marking probability is 0.5. Then the victim v may get packets with $\{FR_1, FR_3\}$, $\{FR_3, FR_4\}$, or $\{FR_1, FR_4\}$. The victim may also get packets with $\{FR_2, FR_3\}$, $\{FR_3, FR_4\}$, or $\{FR_2, FR_4\}$. The $\{FR_1, FR_3\}$ marking indicates that the FR_1 remains before the FR_3 along the path from the user.

In **phase 2**, the victim constructs paths from all the sources after gathering enough information from marked packets. The victim can easily form a directly acyclic graph

(DAG) combining all the paths. For simplicity we will consider tree instead of a DAG. We consider that bots and LUs are behind NAT of internet service provider. We color the bots/attacker as black and LUs as white. The FRs which forward the end users' traffic first are called *entry nodes*. $\{FR_1, FR_2\}$ are the entry nodes in Fig. 2.2. The FRs are colored as black, white, or gray. A black (or white) FR means it only forwards messages from attacker/bot (or LU). A gray FR forwards packets from both LU and attacker.

In **phase 3**, some of the FRs in the traffic topology are selected to assign filters. The traffic topology is simplified by removing nodes with no fork. A node having at least two children is called a fork node. Non-fork nodes are not efficient for assigning filters. Instead, selecting child node reduces attack traffic in the network. Therefore, an optimal filter assignment policy should select a set of FRs (g) from the set of gray and black nodes (G) with minimal blockage of legitimate traffic and contamination by attack traffic, while ensuring that no attack traffic can reach the victim. We define the contamination as the total attack traffic in the network. For example, if the attack traffic is blocked at FR_3 then total contamination is 2 (assume all attackers' packet rate is 1). We denote the contamination in a network for the g filter assignment set by W_c .

$$W_c(g) = \sum_{a=1}^A \alpha_a \times d_a, \quad d_a = \min_{n \in \text{PRED}(n) \cap g} \text{dist}(a, n) \quad (2.1)$$

Here, $\text{PRED}(n)$ is the set of predecessor of n , α_a is the traffic load of attacker a , $\text{dist}(a, n)$ is the number of hops between a and n . A is the total number of attackers. Therefore, W_c is the total attack traffic load for selecting $|g|$ FRs out of $|G|$ FRs. If U is the set of LUs, then the number of blocked LUs for the filter assignment g is denoted by $U_b(g)$.

$$U_b(g) = \{u : u \in U \text{ and } \text{PRED}(u) \cap g \neq \emptyset\} \quad (2.2)$$

The best way to minimize blockage of LU and contamination is to block immediately after the attacker. In reality, there are a huge number of attackers and the victim needs to

select a huge number of FRs to block them which is not possible. So, a victim has budget B of selecting a number of FRs. Therefore, $|g|$ should be less than or equal to B .

In **phase 4**, unused filters are removed from the FRs. As the FRs have limited capacity and computation power, it is necessary to reduce the workload by removing the filters. Besides, the attacker can flood the FRs by sending useless filters. This type of filters are evicted soon because they are most likely not being used.

The filter sent by a user (or victim) is only applicable to the packets which are destined for that user (or victim). However, an attacker can spoof IP of the victim and send wrong filters to FRs. This spoofed filter can be detected using a simple handshake protocol. The spoofing attacker will not be able to handshake with the spoofed IP address. However, we are not focusing on finding an optimal filter assignment policy which is discussed in the next section.

2.4 Source-based Filter Assignment Policy

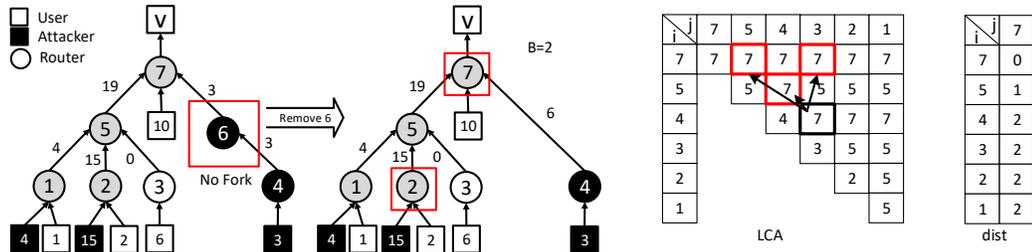
In this section, we formulate a problem of assigning source-based filters to the FRs so that the contamination is minimum.

2.4.1 Problem 1: *Find a filter assignment so that the contamination is minimal by ensuring that all the attack traffic is blocked before reaching the victim.*

In this problem, source-based filters are used. The contamination is defined by the total amount of attack traffic in the network. The problem can be expressed as the following:

$$\begin{aligned} & \text{minimize} && W_c(g) \\ & \text{subject to} && |g| \leq B, \forall g \subset G, v \notin G \end{aligned} \tag{2.3}$$

The victim v will be white ($v \notin G$) if all attacker traffic is blocked before reaching it. The complexity of optimal solution is unknown. Therefore, we propose and compare two greedy approximation solutions as follows.



(a) Topology for Problem 1. (b) LCA calculation.

FIGURE 2.3: Topologies for problems 1 and 2.

2.4.2 First Greedy Solution

A filter is first assigned to root to guarantee the blockage of all attack traffic. The rest of the filters are assigned using a greedy approach. We first calculate weight of each node. The weight of a node is its attack flow times the distance from the node to the root, or the closest node having filter on the path to the root. Then the highest weighed node is selected for filter assignment and weight is updated accordingly. After every new assignment we need to check for each node in assignment set whether attack flows from all children are blocked by other filters or not. If all attack flows are blocked, then we remove the filter from the node. The process continues until the number of filters is less than the budget.

Let us consider the tree in Fig. 2.3(a) that is constructed from information from marked packet by the victim. Assume we want to find assignment of 3 filters ($B = 3$). We first assign a filter to 7. The weights of nodes 1, 2, 5, and 8 are $4 \times 2 = 8$, $15 \times 2 = 30$, $(15 + 4) \times 1 = 19$, and $3 \times 2 = 6$, respectively. So, node 2 is taken for assignment. Now new weight of 2 is 0 and new weight of 5 will be $4 \times 1 = 4$. None of $\{7, 2\}$ nodes' incoming attack traffic is blocked. Therefore, none of the nodes are removed from assignment. The next highest weight is of node 4. Therefore, the filter assignment is $\{7, 2, 4\}$. The W_c of this assignment is 8 while the optimal W_c is 0 for $B = 3$. For randomly generated trees, filter assignment produced by this greedy approximation is almost twice as much as than the optimal filter assignment. Details are shown in Section 2.6.

Theorem 1. *The complexity of the First Greedy Blocking Strategy is $O(NB)$.*

Proof. If there are N nodes and B is the budget, then the algorithm would add a node to assignment set for $B - 1$ times. After each iteration, it takes $O(N)$ time to update weight. Therefore, the complexity of the strategy is $O(NB)$. In the worst case $B = N$ and the complexity is $O(N^2)$. \square

2.4.3 Second Greedy Solution

We start with the maximum number of filters needed to block attack traffic. In fact, we are assigning filters to all non-white entry nodes initially. Then gradually the number of filters and selected FRs will be reduced by one by merging a couple of filters. The merged filter will be sent to the least common ancestor (LCA) of the two FRs. Merging two filters means simply adding the conditions by “or” relation. There could be many options to merge two filters and merging will be associated with penalty. When we are merging two filters of two FRs and assigning the new filter to LCA of them, we are yielding attack traffic to the LCA FR. The amount of attack traffic we are yielding is the penalty associated with the merge. We select a couple of FRs with lowest merging penalty. Thus the number of selected FRs or filters is reduced by one. The reduction process continues until it meets the budget. The algorithm is shown in Alg. 1.

Let us consider the tree in Fig. 2.3(a). We use a double linked tree data structure. Each node contains a pointer to its parent, a number representing blocked attack traffic (BAT), an array of pointer to children with distance, and the color of the node. Initially the BAT of a “white” node is 0. The BAT of a “black” entry node is the total attack traffic. The BAT of non-entry nodes is 0. We keep an array \mathbb{N} of pointers to the nodes to quickly access a node by its label. $\mathbb{N}[i]$ is the node having level i . Firstly, we need to simplify the tree. There is only one node 6 without fork. We remove the node 6 and make 4 child of its parent 7. The new distance to 4 from 7 will increase by the distance of deleted link. The deletion of a node can be done in constant time. Finding out all non-forked nodes takes $O(N)$ time.

Algorithm 1 Second Greedy Blocking Strategy

Input: The number of filters B , topology tree T

Output: A set of nodes in T

```

1: Procedure: BLOCK-GREEDY-2( $B, T$ )
2:   Initialize  $LCA, dist, g$  and  $B_c \leftarrow |g|$ 
3:    $min \leftarrow \infty, i_{min} = 0, j_{min} = 0, A \leftarrow NULL$ 
4:   while  $B_c \neq B$  do
5:     for  $i = 1$  to  $B_c$  do
6:       for  $j = i$  to  $B_c$  do
7:          $A \leftarrow LCA[g[i], g[j]]$ 
8:         if  $min \leq P(i, j)$  then
9:            $min \leftarrow P(i, j), i_{min} \leftarrow i, j_{min} \leftarrow j$ 
10:         $g \leftarrow (g - \{i_{min}, j_{min}\}) \cup A$ 
11:        Increase BAT of  $A$  by BAT of  $i_{min}$  and  $j_{min}$ .
12:         $B_c \leftarrow B_c - 1$ 
13:    return  $g$ 
14: Procedure:  $P(i, j)$ 
15: return  $\mathbb{N}[i].BAT \times (dist[i, N] - dist[N, LCA[i, j]]) + \mathbb{N}[j].BAT \times (dist[j, N] - dist[N, LCA[i, j]])$ 

```

Therefore, the simplification can be done in $O(N)$ time. Next steps are taken according to the Alg. 1. B_c is the number of non-white entry nodes ($B_c = 3$). The assignment set $g = \{1, 2, 4\}$. This step takes $O(N)$ time to find all the entry nodes. Then the all-pair LCA is computed. LCA calculation can be represented as the following recurrence:

$$LCA[i, j] = \begin{cases} i, & \text{for } i = j \\ LCA[i, p(j)], & \text{for } LCA[i, p(j)] \neq null \\ LCA[p(i), j], & \text{for } LCA[p(i), j] \neq null \\ LCA[p(i), p(j)], & \text{otherwise} \end{cases} \quad (2.4)$$

Here $p(i)$ represents the parent of node i which can be found in constant time. The all pair LCA can be calculated in $O(N^2)$ time if we use dynamic programming. We have $LCA[i, j]$ is calculated in a top-down fashion. Therefore, $LCA[7, 7]$ is calculated first and $LCA[7, 7] = 7$. After that $LCA[7, 5]$ is calculated. As $P(5) = 7$ so that $LCA[7, 5] = 7$. When we calculate $LCA[4, 3]$, we need to look at $LCA[7, 3]$, $LCA[5, 4]$, or $LCA[7, 5]$.

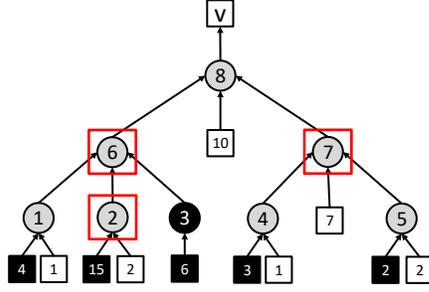


FIGURE 2.4: Topology for Problem 2.

The $LCA[7, 3] = 7$ so $LCA[4, 3] = 7$. Similarly, we calculate the rest of the pairs (see Fig. 2.3(b)).

Next we calculate distance (*dist*) of every node from the root. This calculation takes $O(N)$ as it needs to traverse the whole tree once again. The complete $dist[i, j]$ is shown in Fig. 2.3(b). When $B_c = 3$ the filter assignment is $\{1, 2, 4\}$. When $B_c = 2$, we have three options for merging filters: (1) merge filters of 1 and 2 and assign the merged filter to FR 5 ($P(1, 2) = 19$), (2) merge filters of 1 and 4 and assign the merged filter to FR 7 ($P(1, 4) = 14$), and (3) merge filters of 2 and 4 and assign the merged filter to FR 6 ($P(2, 4) = 36$). Therefore, option (2) is chosen and the assignment is $\{2, 7\}$ for $B_c = 2$. When $B_c = 1$, we have one choice which is to merge filters of 2 and 4 and assign the merged filter to 7 ($P(2, 7) = 30$). Therefore, for $B = 1$ the assignment is $\{7\}$.

Theorem 2. *The complexity of the Second Greedy Blocking Strategy is $O(N^2(N - B))$.*

Proof. The algorithm would iterate the step 5 loop for $N - B$ times. In each iteration, it takes $O(N^2)$ time to find out the pair of nodes with minimum penalty. The all pair LCA computation takes $O(N^2)$ time. The complexity of Alg. 1 is $O(N^2(N - B)) + O(N^2) = O(N^2(N - B))$. In the worst case $B = 1$ and the complexity is $O(N^3)$. The complexity can further be improved to $O((N - B)^2 \log N)$ using min-heap data structure (see Appendix A for details). \square

2.5 Destination-based Filter Assignment Policy

As we are using destination-based filters for protection against spoofed DDoS attack, we are blocking some LUs. In this section, we formulate another optimization problem of assigning destination-based filters to the FRs so that a weighted sum of the contamination and blocked LUs is minimum.

2.5.1 Problem 2: *Find a filter assignment so that the LU blockage and contamination are minimal.*

It is always better if the victim can select some FRs within its budget which minimizes both the number of blocked LUs and contamination. As discussed in Section 2.3 the source-based filter cannot ensure protection against IP spoofing DDoS attack. For example, if the attacker attached to node 1 uses IP address of the users attached to node 3 (see Fig. 2.3(a)). The filter used at node 1 or 5 would forward the packet. But if the FRs use destination-based filter then no spoofed attack packet can penetrate. Therefore, the victim would use the destination-based filters. The problem can be expressed as the following optimization problem:

$$\begin{aligned}
 & \text{minimize} && \omega W_c(g) + (1 - \omega)|U_b(g)| \\
 & \text{subject to} && |g| \leq B, \forall g \subset G, v \notin G
 \end{aligned} \tag{2.5}$$

Here $\omega = [0, 1]$ is considered a system parameter which determines priority of total contamination and LU blockage.

2.5.2 A Dynamic Programming Solution

Let us consider an N node tree with maximum node degree Δ . The nodes are labeled in bottom-up and left-right order. We define A as a $N \times B$ array which contains optimal cost for every node and budget. For example $A[i, j]$ is optimal cost of budget j on subtree rooted by node i .

We define L as a $1 \times N$ array which contains the number of LUs in subtree rooted by every node. $L[i]$ is the number of LUs in subtree rooted by node i . We also define R as

Algorithm 2 DP Blocking Strategy

Input: The number of filters B , topology tree T .

Output: A set of nodes in T .

```
1: Procedure: BLOCK-DP( $B, T$ )
2:    $N \leftarrow$  number of nodes in  $T$ 
3:   for  $i = 1$  to  $N$  do
4:     for  $j = 0$  to  $B$  do
5:       if  $i$  is an entry node then
6:         Initialize  $A[i, j]$ ,  $L[i]$ ,  $C[i]$ , and  $R[i, j, k]$ 
7:       else
8:          $L[i] \leftarrow \sum_{k=1}^{\Delta} L[c_k(i)]$ 
9:          $min \leftarrow \infty$ 
10:      if No attacker attached to  $i$  then
11:         $p \leftarrow$  cost according to Equation 2.6
12:      if  $min \geq p$  then
13:         $min \leftarrow \sum_{k=1}^{\Delta} A[c_k(i), x_k]$ 
14:         $A[i, j] \leftarrow min$ 
15:         $\forall_{1 \geq k \geq \Delta} R[i, j, k] \leftarrow x_k,$ 
16:         $R[i, j, \Delta + 1] \leftarrow 1$ 
17:      for  $\forall x_1, x_2, \dots, x_k : \sum_{k=0}^{\Delta} x_k = j$  do
18:         $p \leftarrow \sum_{k=1}^{\Delta} A[c_k(i), x_k]$ 
19:      if  $min \geq p$  then
20:         $A[i, j] \leftarrow p \leftarrow min$ 
21:         $\forall_{1 \geq k \geq \Delta} R[i, j, k] \leftarrow x_k$ 
22:         $R[i, j, \Delta + 1] \leftarrow 0$ 
23:      return FIND-ASSIGNMENT( $R, B$ )
```

an $N \times B \times (\Delta + 1)$ array which contains the number of filters assigned to node i and its subtrees for every node and budget. For example, $R[i, j, 1]$, $R[i, j, 2]$, and $R[i, j, \Delta + 1]$ are the number of filters to first subtree, second subtree, and node i of subtree rooted by i for budget j . The optimal cost of using j destination-based blocking/filter in subtree rooted by i is the minimum of the following quantity:

Case I: The minimum total weighted cost, if we assign 1 filter to the node i and the rest of the filters to some nodes of the subtree rooted by i . Therefore, the cost will be a weighted sum of the minimum contamination and LU in subtree rooted by i . The assignment of the $j - 1$ nodes can be done in a greedy way. We can apply the Alg. 1 to find an assignment

Algorithm 3 Find Assignment

```
1: Procedure: FIND-ASSIGNMENT(R, B)
2:    $x.i \leftarrow N, x.j \leftarrow B, g \leftarrow \emptyset$ , and  $Q \leftarrow \emptyset$ .
3:   ENQUEUE(Q, x).
4:   while  $Q \neq \emptyset$  do
5:      $x \leftarrow$  DEQUEUE(Q)
6:     if  $R[x.i, x.j, \Delta + 1] \neq 0$  then
7:        $g \leftarrow$  the  $R[x.i, x.j, \Delta + 1]$  nodes according to case 1.
8:     else
9:       for  $k = 1$  to  $\Delta$  do
10:         $x_c.i \leftarrow c_k(x.i), x_c.j \leftarrow R[x.i, x.j, k]$ 
11:        ENQUEUE(Q,  $x_c$ )
12:   return  $g$ 
```

in subtree rooted by i but the cost will no longer be optimal. First, we assume an attacker attached to the i . This assumption confines a filter to i . Then we find an assignment of budget j using Alg. 1. As i will be assigned a filter, the other $j - 1$ filters will be assigned to the subtree rooted by i .

Let $g'[i, j - 1]$ be the assignment which provides contamination ($C(g'[i, j - 1])$) to the subtree rooted by i for $j - 1$ filters. Then the cost for this option will be:

$$A[i, j] = \omega C(g'[i, j - 1]) + (i - \omega)L[i] \quad (2.6)$$

Case II: The minimum total weighted cost, if we divide the number of filters into x_1, x_2, \dots, x_k parts and assign them to the subtrees c_1, c_2, \dots, c_k , respectively. Therefore, the cost for this option will be:

$$A[i, j] = \sum_{k=1}^{\Delta} A[c_k, x_k] \quad (2.7)$$

Therefore, we take the minimum quantity from the above two options. If there are some attackers attached to node i , we do not consider the option 2. This is because, if we assign all the j filters to its subtree then the attack traffic from i will reach the victim v , which is not allowed by the constraint of the problem definition.

i \ j	0	1	2	3
1	∞	0.5	0.5	0.5
2	∞	1	1	2
3	∞	0	0	0
4	∞	1	1	1
5	∞	2	2	2
6	∞	14	6.5	1.5
7	∞	7.5	1.5	1.5
8	∞	36.5	21.5	14

i \ j	0	1	2	3
1	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
2	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
3	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
4	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
5	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
6	0,0,0,0	0,0,0,1	0,0,0,2	1,1,1,0
7	0,0,0,0	0,0,0,1	1,1,0,0	2,1,0,0
8	0,0,0,0	0,0,0,1	1,1,0,0	2,1,0,0

i	
1	1
2	2
3	0
4	1
5	2
6	3
7	10
8	23

FIGURE 2.5: A , R , L , and C .

Let us consider the traffic topology in Fig. 2.4 and $\omega = 0.5$. The leaf entry nodes are 1, 2, 3, 4, and 5. The calculations of A , R , and L are straightforward. For example $A[1, 0] = \infty$, $A[1, 1] = 0.5 \times 0 + 0.5 \times 1 = 0.5$, and $A[1, 2] = 0.5 \times 0 + 0.5 \times 1 = 0.5$. $R[i, 1] = [0, 0, 0, 1]$, $R[i, 2] = [0, 0, 0, 2]$, and $R[i, 3] = [0, 0, 0, 3]$. For node 6 and $j = 0$, we have one choice $x_1 = 0, x_2 = 0, x_3 = 0$ and the cost is ∞ . For $j = 1$, we have two cases. **Case 1:** 1 for node 6, and $x_1 = 0, x_2 = 0, x_3 = 0$. The total cost is $0.5 \times 25 + 0.5 \times 3 = 14$. **Case 2:** 0 for node 6, and (1) $x_1 = 1, x_2 = 0, x_3 = 0$, (2) $x_1 = 0, x_2 = 1, x_3 = 0$, or (3) $x_1 = 0, x_2 = 0, x_3 = 1$. For option (1): total cost is $\infty + \infty + \infty = \infty$. For option (2) and (3) total cost is also ∞ . Therefore, case 1 is minimum ($A[6, 1] = 14$) and $R[6, 1] = [0, 0, 0, 1]$.

For $j = 2$, there are also two cases. **Case 1:** 1 for node 6 and 1 is for its subtrees. We assume an attacker attached to 6. Then applying the Alg. 1 for $B = 2$, we find the assignment is $\{6, 2\}$. After assigning the filters the total contamination is $4 + 6 = 10$ The total cost in this option is $10(0.5) + 3(1 - 0.5) = 6.5$. **Case 1:** 0 for node 6, and 2 filters to subtrees of 6. This case is valid for node 6 because there is no attacker directly attached to it. There can be six options: (1) $x_1 = 2, x_2 = 0, x_3 = 0$, (2) $x_1 = 0, x_2 = 2, x_3 = 0$, (3) $x_1 = 0, x_2 = 0, x_3 = 2$, (4) $x_1 = 0, x_2 = 1, x_3 = 1$, (5) $x_1 = 1, x_2 = 0, x_3 = 1$, and (6) $x_1 = 1, x_2 = 1, x_3 = 0$. For option (1), the total cost is $A[1, 2] + A[2, 0] + A[3, 0] =$

$0.5 + \infty + \infty = \infty$. Similarly, the options (2) to (6) cost ∞ . Therefore, case 1 is minimum and $A[6, 2] = 6.5$ and $R[6, 2] = [0, 0, 0, 2]$. Similarly, we calculate the rest of the entries in A and R . The complete A , L and R are shown in Fig. 2.5.

According to the definition, $A[8, 3]$ contains the cost for budget $B = 3$ which is 14. From R we can find out which FRs are blocked. $R[8, 3, 1] = 2$ and $R[8, 3, 2] = 1$ means 2 and 1 filters are assigned to its 1st and 2nd subtrees, respectively. Then we need to look $R[6, 2, k]$ and $R[7, 1, k]$. $R[6, 2, 1] = 0$, $R[6, 2, 2] = 0$, $R[6, 2, 3] = 0$, and $R[6, 2, 4] = 2$ means no filter is assigned to its subtrees and two filters are assigned to itself. Therefore, we need to find the assignment according to Case I. According to Case I $\{6, 2\}$ is the assignment. Similarly, we can find that a filter is assigned to node 7. So, the filter assignment is $\{2, 6, 7\}$ for budget $B = 3$ and the cost is 14.

Theorem 3. *Complexity and space needed of the DP Blocking Strategy are $O(NB^{\Delta-1})$ and $O(NB\Delta)$.*

Proof. Let us consider the topology is a N node tree with maximum node degree Δ and the victim has budget of B . To find the partitions $x_1, x_2, \dots, x_\Delta$ we need $O(B^{\Delta-1})$ time if we use naive nested iteration approach. Therefore, the complexity of the Alg. 2 is $O(NB^{\Delta-1})$. The total space needed is $(N-1)B + (\Delta+1)(N-1)B + (N-1)$ which is an order of $O(NB\Delta)$. For a binary tree topology the complexity is $O(NB^2)$ and the space complexity is $O(NB)$. \square

Theorem 4. *The Alg. 2 provides optimal solution when $\omega = 0$.*

Proof. When $\omega = 0$ only blockage of LUs is taken into account. The Alg. 2 uses a dynamic programming bottom-up strategy to search the optimal assignment. For an one-node tree, if the node color is “black” or “gray” then there is no solution for $B = 0$. Because without any filter, the attack traffic will be forwarded to the downstream routers. For $B \geq 1$ there is only one choice of selecting FR which is that node. If that node

Table 2.1: Topology Parameters

	Topology I	Topology II	Topology III
Number of nodes	66	247	403
Internal user probability	0.1	0.25	0.1
Attacker ratio	0.4	0.4	0.4
Max Node Degree	4	4	20
Data Rate(pack/ms)	[0.1-0.4]	[0.1-0.4]	[0.03-0.13]

is selected, the optimal number of blocked LUs is the LUs attached to it. In each step, the Alg. 2 chooses the best allocation of filters to itself, left subtree, or right subtree. Therefore, the Alg. 2 provides optimal filter assignment to the FRs through exhaustive search. □

2.6 Simulations

2.6.1 Simulation Setting

We conducted the experiments with a custom build java simulator. The main reason for using custom build for simulator is its scalability. We do not need to analyze transmission time, bandwidth, or packet drop issues. We only need to count the number of legitimate (or attack) received (or blocked) packets. The network topologies we considered contain 100 – 500 routers. Using NS3 or other similar simulator for this kind of simulation would take several days. That is why we built our own java multi-threaded simulator to get the results quickly.

We conducted simulation for randomly generated tree topology and a subset from a real network topology. We used two randomly generated topology having node degree between $[0 - 4]$, internal node user probability between $1 - 0.25$, and maximum depth 6. The entry nodes' color and number of users or attackers were selected randomly from a uniform distribution. The Topology I and II were randomly generated trees with 66 and 247 nodes and max node degree 4. The Topology III was taken from a subset of the Stanford University Note Dame web graph [24]. The dataset contained 325,729 nodes and we took a subset (which is a tree) containing 403 nodes. Then we randomly assigned

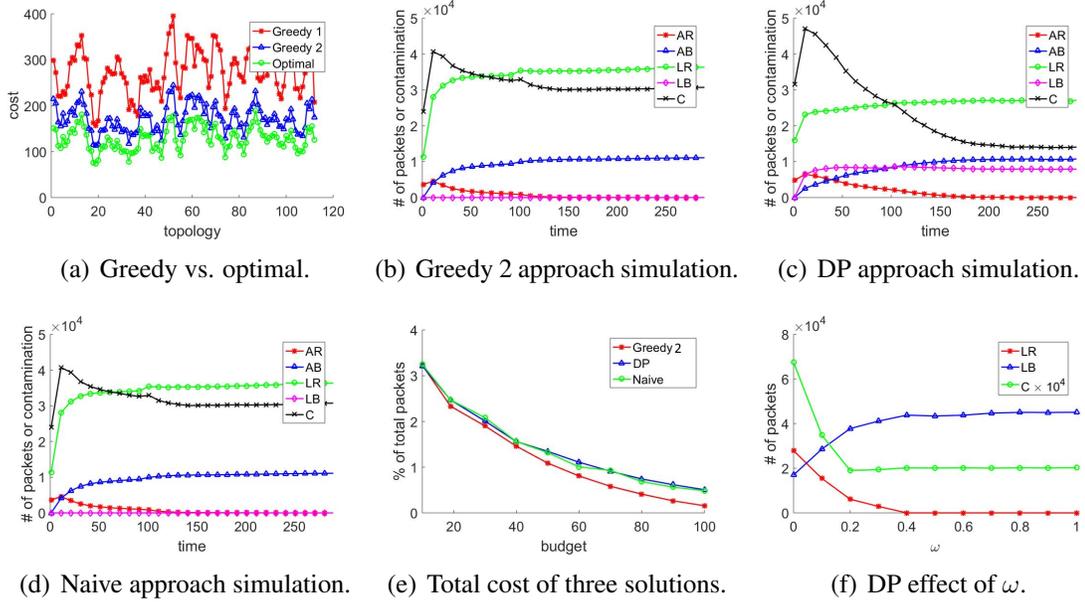


FIGURE 2.6: Simulation results.

users to the tree with internal user probability 0.1. The details are shown in Table 2.1 and Fig 2.7(b).

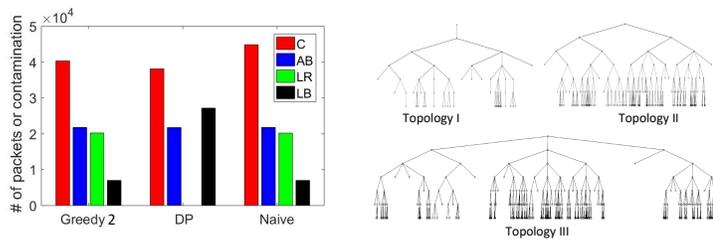
We compare the performances of our proposed second greedy blocking strategy (Greedy 2), DP blocking strategy (DP) and a naive approach. In the naive approach, source-based filters are used. At first, a filter is assigned to the node which is attached to v . Then the filter is split to its child nodes which have higher attack traffic flows. The split only occurs if the node does not have any attacker attached to it. Therefore, a split will increase the number of filters by its number of children. Every round the node with highest attack traffic is selected for split from the current assignment. The process continues until the number of assigned filters is less than budget. We show the contamination (C), number of blocked attack packets (AB), number of received attack packets (AR), number of blocked legitimate packets (LB), and number of received legitimate packets (LR) for the three topologies and three approaches.

2.6.2 Simulation Results

We compare the performances of the greedy solutions for different randomly generated topologies. The Fig. 2.6(a) shows the cost of first greedy (Greedy 1), second greedy (Greedy 2), and optimal cos. For $[10, 15]$ node trees and $[3, 5]$ filters the Greedy 2's average cost is about 9% higher than average optimal cost. For $[25, 35]$ node trees and $[5, 10]$ filters the Greedy 2's average cost is about 24% higher than optimal. Though the complexity of Greedy 2 is little higher than Greedy 1 it performs much better.

Figs. 2.6(b) , 2.6(c), and 2.6(d) show the C, AB, AR, LB, and LR for the Topology I. Here the contamination is the number of attack-packet forwarding events per attack packet. We can see that, at the beginning, the C in every approach is higher. The C reduces over time and becomes gradually more stable. This is because, at the beginning the victim knows a small subset of the complete topology. Over time, the victim gets more and more information from the marked packet and reconstructs the traffic topology. Finally, the victim succeeds in constructing the complete topology. That is why the AR is high at the beginning and decreases over time and finally converges to zero. The AB shows the opposite behavior for the same reason. We can also observe that the cost (contamination) of second greedy solution is less than the naive approach. The cost of DP is a weighted sum of C and BL.

Next we use the Topology II to compare the performances of three approaches for different budgets after convergence (considering the victim knows complete topology). In this topology we increased the internal user probability to see the amplified effect of different budget settings. Each run was observed for longer time (200-400 slots and each slot is about 1 second long) and the average measurements of slots are plotted. Fig. 2.6(e) shows the cost of the three approaches. Here, the cost is the number of forwarding events per attack traffic. We can see that the cost of Greedy 2 is then lowest. The costs of DP and Naive are similar.



(a) Simulation with real topology. (b) Traffic topologies.

FIGURE 2.7: More simulation results.

Fig. 2.6(f) shows the effect of ω in the DP. We observe the C, RL, and BL by ω . The C is multiplied by 10,000 to show the effect clearly with the other measurements. If we set $\omega = 0$, the total RL and C are highest (because we give no weight to contamination). On the other hand, the LB is lowest at $\omega = 0$. When $\omega \geq 0.4$ all the legitimate packets are blocked and the contamination remains stable. With the increase of ω , the LB increases and the LR and C decrease, which means that the more (or less) weight we put to contamination the more (or less) the legitimate traffic we block. Therefore, there is a trade-off between the number of legitimate traffic and attack traffic. The victim should set a proper ω to have a good balance between blocked legitimate packets and contamination.

Fig. 2.7(a) shows a comparison of C, AB, LR, and LB using a real topology. Here, we define C as the number of forwarding events of attack packets. We use the Topology III for this simulation. The simulation shows that the C of Greedy 2 is lower than Naive approach. In Greedy 2 and Naive approaches attack packets are forwarded 40,330 and 44,844 on average. The AB, LB, and LR are similar for both approaches. The DP shows the lowest contamination (38,126) but highest blocked legitimate traffic (for $\omega = 0.5$). The number of blocked LUs can be adjusted by choosing a lower ω .

2.7 Run-time Improvement Using Heap

2.7.1 $O((N - B)^2 \log N)$ approach for Second Greedy Solution

In the Alg. 4, Steps 1-6 are the same as in Alg. 1. Instead of searching for minimum penalty pair, we create a min-heap H from all pairs of elements in the current assignment

Algorithm 4 $O((N - B)^2 \log N)$ Greedy Blocking Strategy

```

1: Procedure: BLOCK-GREEDY-2( $B, T$ )
2:   Steps 1 – 6 in Alg. 1
3:    $S \leftarrow \{x : x.i, x.j \in g, i \neq j, \text{ AND } x.key = P(x.i, x.j)\}$ 
4:   Create minheap  $H$  from  $S$ .
5:   while  $B_c \neq B$  do
6:      $m \leftarrow \text{EXTRACTMIN}(H)$ 
7:     Remove  $x$  form  $H$ , where  $x.i$  or  $x.j$  is  $m.i$  or  $m.j$ 
8:      $g \leftarrow g - \{i, j\}$ 
9:     Insert all  $\{x : x.i = LCA[m.i, m.j], x.j \in g, x.key = P(x.i, x.j)\}$ 
10:     $g \leftarrow g \cap \{LCA[m.i, m.j]\}$ 
11:    Increase BAT of  $A$  by BAT of  $i_{min}$  and  $j_{min}$ .
12:     $B_c \leftarrow B_c - 1$ 
return  $g$ 
  
```

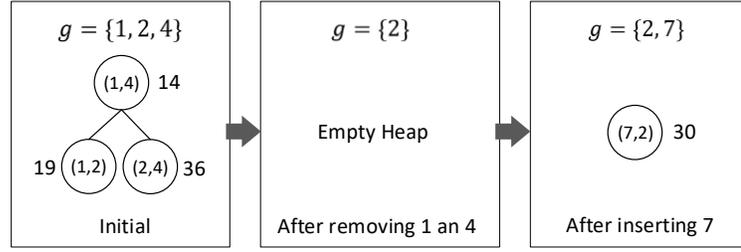


FIGURE 2.8: H and g .

g . Then, we do $\text{EXTRACTMIN}(H)$ from the heap and remove all associated elements with the removed nodes. Then, we insert all the associated elements with the new node. We create a min-heap of all the pairs of nodes in g (according to Fig. 2.3(a)). The heap H contains $\binom{N}{2}$ elements. The key of an element (i, j) is the penalty of merging the pair of nodes $P(i, j)$. For example, the key of node $(1, 2)$ is $P(1, 2) = \mathbb{N}[1].BAT \times (\text{dist}[1, 7] - \text{dist}[5, 7]) + \mathbb{N}[2].BAT \times (\text{dist}[2, 7] - \text{dist}[5, 7]) = 4 \times (2 - 1) + 15 \times (2 - 1) = 19$. Similarly, the key of node $(1, 4)$ and $(2, 4)$ is $P(1, 4) = 14$ and $P(2, 4) = 36$. Therefore, the heap H will have $\{(1, 4), (1, 2), (2, 4)\}$ elements. The heap construction takes $O(N^2)$ time for $\binom{N}{2}$ elements.

The next step is a while loop which iterates until B_c is not equal to the budget B of v . The $\text{EXTRACTMIN}(H)$ will return the $(1, 4)$ because its key is lowest. Generally, the

EXTRACTMIN(H) operation on heap takes $O(\log n)$ time. Here we have $\binom{N}{2}$ elements which take $O(\log N^2) = O(2 \log N) = O(\log N)$ time. After that we remove elements of heap which are associated with 1 or 4. Therefore, the heap H contains no elements because all the elements are associated with 1 or 4. The number of remove operations is $O(N)$ and each remove operation takes $O(\log N)$. Now we remove 1 and 4 from g . Therefore, $g = \{2\}$. Next we add each pair of elements associated with $LCA[4, 1] = 7$ and g . So, element $(7, 2)$ will be added to H . At this step we calculate the key of $(7, 2) = (15 \times 2 + 0) = 30$. The BAT of node 7 will be increased by $P(1, 4) = 14$. So, BAT of 7 is $(0 + 14) = 14$. Then $LCA[1, 4] = 7$ is added to g and B_c is decreased by 1. As the $B_c = 2$ is now equal to B the loop stops and our algorithm finishes by returning $g = \{2, 7\}$.

We can see that in the heap an element (a pair of nodes) is inserted or deleted once. The total number of removals or insertions is $\binom{N-B}{2}$. Each insert or delete operation takes $O(\log((N - B)^2)) = O \log(N)$. Therefore, the complexity of the algorithm is $O((N - B)^2 \log(N))$.

2.8 Summary

The DDoS attack is the most powerful attack to make a service unavailable to users. It is not possible to protect any server from DDoS attack without the help of the network equipment. As the most important component in a network, routers can be upgraded to filter routers easily. Besides, the filter router can work in a network with legacy routers. In the four-phase DDoS protection system, the filter routers block the attack traffic according to the victim's instruction. Though the blocking control of an internet service provider (ISP) is at victims hand who may not belong to the ISP but it will help the ISP to minimize traffic congestion. Therefore, both parties are benefited. In this work, we present three filter scheduling policies for two different settings. We compare the performances of proposed scheduling policies with the naive approach. Simulation results show that our proposed approaches work better than the naive approach. Both the source-based and destination-based filters have some advantages and limitations. In the next chapter we present optimal solutions of the problems defined in this chapter. We also use dynamic programming to solve the first problem.

CHAPTER 3

OPTIMAL FILTER ASSIGNMENT POLICY AGAINST DISTRIBUTED DENIAL-OF-SERVICE ATTACK

In Chapter 2, we have defined the problems of assigning filter with limited number of filters. We have proposed greedy solution for the first problems which is not optimal. As the solution of second problem uses the solution of the first problem, it becomes non-optimal. Now in this chapter we present an optimal solution to the first problem. As a result, both solution produce optimal results. The contents of this chapter is published in [Related8, Related9].

3.1 Introduction

A denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource unavailable temporarily to its users. DoS attacks are considered a federal crime under the Computer Fraud and Abuse Act with penalties that include years of imprisonment [7]. The Computer Crime and Intellectual Property Section of the US Department of Justice handles cases of DoS attacks. Therefore, detecting DoS attacks and identifying attackers have been an important issue in Network Forensics. Moreover, DoS attacks are increasing day by day in both number and size; CloudFlare [4] recently reported a 400 Gbps massive DoS attack that took place at their servers.

There are several types of DoS attacks including SYN Floods, Malformed Packets, UDP Floods, Amplification Attacks, and Distributed Attacks [5]. In a SYN Flood attack, the perpetrator sends many SYN messages for TCP connection setup. The server replies ACK and waits for the client's ACK but the attacker does not reply ACK and the connection remains half-open till timeout. The objective of a SYN flood is to simply fill up the limited slots that the target system has available for half-open connections. In some cases it's easy to detect a SYN Flood attack if a lot of SYN requests come from an address in short interval. Detection is harder, however, when the attacker spoofs IP address, SYNs come from multiple addresses, and arrival time varies. In a UDP Flood attack, the purpose would likely be to consume all available network bandwidth. Attackers send a large amount of spoofed requests with large useless payload. The application wastes CPU cycles trying to determine the meaning of the garbage.

The objective of the DDoS attack is to generate a lot of packets from different locations to exhaust incoming/outgoing bandwidth of the victim. A coordinator would send commands to workers who continue to send requests to the target. The workers are known as bots and the network of workers is known as botnet. As normal users also request through the NAT, it is difficult for the victim to differentiate between the bot requests and normal user requests. Fig. 3.1 shows the DDoS attack model by botnet. An effective method of preventing DDoS attack is to use filter routers (FRs) in network infrastructure. FRs are a special type of router which is capable of packet marking and receiving filter tasks. Packet marking task refers to attaching the FR's own IP address probabilistically to the packets it forwards while receiving filter task refers to receiving filters from a web server. A web server can block all or part of the traffic destined to it. The complete method of using FRs is a four-phase process. In the first phase, the FRs mark forwarded packets by appending its own IP address probabilistically. In the second phase, the victim constructs the traffic topology from the marking of the packets. In the third phase, the victim constructs filter, finds, and selects some FRs to assign the filters. In the last phase, the

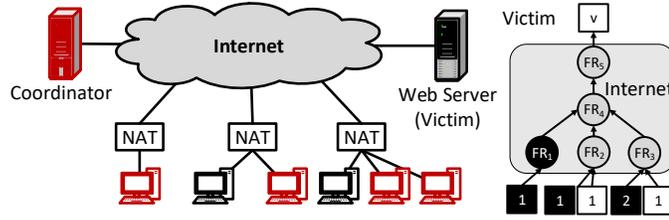


FIGURE 3.1: DDoS attack by bots.

FRs evict unused filters from its storage. The packet marking is used to find the topology by the victim. The probability of marking is a tradeoff between topology construction time and router overhead. After topology construction, the victim generates filters and selects a subset of the FRs to assign them. There are two types of filters: source-based and destination-based. Using source-based filter a FR can allow/block traffic from specific sources which are destined for the victim. This type of filter is vulnerable to IP spoofing attacks. Though destination-based filter can prevent IP spoofing but it is more restrictive. Using a destination-based filter, a FR can block all traffic including LU traffic destined to the victim. It is challenging to find an optimal assignment of filter when the victim has a limit on selected number of FRs. A FR may get a number of filters from multiple victims. It has a limitation of storage and computation power. Therefore, it evicts filters which are no longer used.

In this paper, we focus on finding the optimal filter assignment assuming that the victim has already constructed the traffic topology. We formulate two problems and propose solutions for them. In the first problem, a limited number of source-based filters are assigned to the FRs. For example, if the victim can assign 2 filters, it can select $\{FR_1, FR_2\}$, $\{FR_1, FR_5\}$, $\{FR_2, FR_4\}$ or another pair of FRs (see Fig. 3.1). If the victim selects the first pair of FRs, the attack traffic from FR_3 will reach the victim which is highly unexpected. If the second pair is selected then the attack traffic will travel through (FR_2, FR_4) , (FR_3, FR_4) , and (FR_4, FR_5) links. The amount of attack traffic in each link is not same. It is challenging to find a filter assignment for which the total amount of attack traffic is

minimum. We propose dynamic programming optimal solution for this problem. In the second problem, a limited number of destination-based filters are assigned to the FRs. Destination-based filter blocks every packet at FR that is destined to the victim. If the victim selects the third pair, then all the legitimate users (LUs) will be blocked and the attack traffic will travel through (FR_2, FR_4) and (FR_3, FR_4) links. It is also challenging to find a filter assignment so that the total attack traffic and number of blocked LU are minimum. We propose another dynamic programming solution for this problem. Our main contributions are the following:

1. We formulate two problems for finding filter assignment with a budget (limited number of filters) and provide optimal solutions using dynamic programming.
2. We present simulation results to support our model.

The remainder of this paper is arranged as follows: Section 3.2 presents some related works and their limitations. In Section 3.3, we present the system model for preventing DDoS attack. Section 3.4 presents the formal definition of the first problem and our proposed dynamic programming solutions. Section 3.5 presents the formal definition of the second problem and our proposed another dynamic programming solution. In Section 3.6, we present some simulation results that strengthen our proposed solutions.

3.2 Related Works

There exist many statistical methods including correlation, entropy, covariance, divergence, cross-correlation, and information gain to detect anomalous DDoS requests [8]. A rank correlation-based method Rank Correlation-based Detection (RCD) is proposed in [9]. An information theoretical approach using Kolmogorov complexity is used for detection of DDoS attacks in [10]. A novel DDoS detection mechanism is proposed based on Artificial Neural Networks in [11]. There are other methods of detecting DDoS attack including [12, 25, 26]. Authors in [16] introduced a model of randomized DDoS attacks

with increasing emulation dictionary where the attackers use the attack definition from the dictionary that contains request patterns similar to those of LUs. They proposed an inference algorithm for identifying the botnets executing such DDoS attacks. Nowadays, static path identifiers are used for inter-domain routing objects, which makes it easy for attackers to launch the DDoS flooding attacks. In [17], the authors present a design dynamic path identification framework that uses path identifier negotiated between the neighboring domains as inter-domain routing objects.

In [18], the authors propose a method, RADAR, to detect and throttle DDoS attacks using adaptive correlation analysis on SDN switches. The system can defend against a wide range of flooding-based DDoS attacks including link flooding, SYN flooding, and UDP-based amplification attacks. In [19], the authors propose a new approach which reduces the resource utilization factor to a minimal value for quick absorption of the attack. In [20], a DDoS protection mechanism, SkyShield, is proposed by taking advantages of the sketch techniques. To identify malicious hosts efficiently, they used the abnormal sketch obtained from the last detection cycle. The SkyShield could leverage other techniques including Bloom filters and the CAPTCHA. In [21], the authors propose a collaborative DDoS mitigation network system in which a domain helps another domain. A domain can direct excessive traffic to other trusted external domains for DDoS filtering. The filtered clean traffic is forwarded back to the targeted domain.

Most of the existing works are mainly concerned about the service availability of the server. In fact, the attack traffic may cause huge network congestion and DoS. Therefore, these techniques cannot protect the network from being contaminated by attack traffic. A victim and network component collaboration based system can help in this case. A four-phase DDoS protection system is proposed in [22]. The victim generates filters and sends them to the upstream FRs. The FRs send the filters to its upstream FRs and thus the filters propagate to the effective FRs. An adaptive version of PFS is proposed in [23]. The system sends directly filters to the high capable FRs first, then the filters propagate to the

effective FRs. However, these two systems cannot select the FRs optimally when there is a limitation on selecting FRs.

3.3 System Model

Our system is composed of legacy routers (LRs), network address translators (NATs), filter routers (FRs), attackers, legitimate users (LUs), and a victim (v). Fig. 3.2 shows the complete system model. In reality, there are multiple victims in a network but for simplicity of explanation we are considering a single victim. We assume that end users are connected to a FR or a LR through NAT. The FRs are a special kind of router which are capable of two functionalities. Firstly, it can do packet marking which is used to construct traffic topology at the victim. Secondly, it can receive filter from the victim and apply the filter to block the attack traffic according to the filter definition. There can be two types of filter: source-based and destination-based. The source-based filter specifies blocking of traffic based on source address. For example, a source-based filter can be understood: *if source address is X then discard the packet*. If we use a source-based filter at FR_3 (assume X and Y are the IP addresses of the NATs connected to FR_1 and FR_2 , respectively) then FR_3 will discard packets coming from NAT- X but forward packets from the NAT- Y .

The advantage of using source-based filter is that a FR can block the attack traffic by its source IP address and forward legitimate traffic. If the LU and attacker both remain behind the same NAT, then it is impossible to block only attack traffic. The limitation of the source-based filter is that it cannot protect if an attacker spoofs the IP address of a LU. If an attacker creates packet having Y as the source address, then the packet will not be blocked at FR_3 . To protect against DDoS attack with IP spoofing, we can use destination-based filter. A destination-based filter is *if the destination address is X then discard the packet*. For example, if we use a destination-based filter at FR_3 (assume that X is IP address of v), then all the packets including legitimate and spoofed attack packets will be blocked by FR_3 . The destination-based filter is more restrictive. When a FR uses

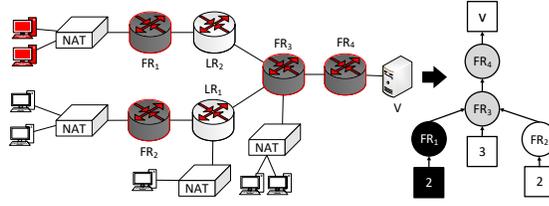


FIGURE 3.2: System model and constructed topology.

it, it blocks all the attack and legitimate traffic destined for the victim. Therefore, spoofed attack traffic cannot penetrate.

The attackers are usually user devices which have compromised programs that can generate traffic destined for a target. The programs are controlled by a master. The master can send commands of attack to the program. This type of program is called bot and the network of bots is called botnet. Though the DDoS traffic is hard to differentiate from legitimate traffic, there exist several methods based on arrival time, packet size, and packet content for detecting attack packets [8]. In this paper, we are not focusing on the detection of attack packet and assume that the victim can find out the source address of attack traffic using these methods. The victim also knows the packet rate of each attacker. For example, if there are 100 attackers (or LUs) each with 1 Mbps attack traffic (or legitimate traffic) behind a NAT- X (having IP X), then the victim will identify the X as an attacker (or LU) with 100 Mbps attack traffic (or legitimate traffic).

The complete protection process consists of the four phases.

In **phase 1**, the FRs probabilistically mark the packet it forwards by appending its own IP address. Assume the marking probability is 0.5. Then the victim v may get packets with $\{FR_1, FR_3\}$, $\{FR_3, FR_4\}$, or $\{FR_1, FR_4\}$. The victim may also get packets with $\{FR_2, FR_3\}$, $\{FR_3, FR_4\}$, or $\{FR_2, FR_4\}$. The $\{FR_1, FR_3\}$ marking indicates that the FR_1 remains before the FR_3 along the path from the user.

In **phase 2**, the victim constructs paths from all the sources after gathering enough information from marked packets. The victim can easily form a directly acyclic graph

(DAG) combining all the paths. For simplicity we will consider tree instead of a DAG. We consider that bots and LUs are behind NAT of internet service provider. We color the bots/attacker as black and LUs as white. The FRs which forward the end users' traffic first are called *entry nodes*. $\{FR_1, FR_2\}$ are the entry nodes in Fig. 3.2. The FRs are colored as black, white, or gray. A black (or white) FR means it only forwards messages from attacker/bot (or LU). A gray FR forwards packets from both LU and attacker.

In **phase 3**, some of the FRs in the traffic topology are selected to assign filters. The traffic topology is simplified by removing nodes with no fork. A node having at least two children is called a fork node. Non-fork nodes are not efficient for assigning filters. Instead, selecting child node reduces attack traffic in the network. Therefore, an optimal filter assignment policy should select a set of FRs (g) from the set of gray and black nodes (G) with minimal blockage of legitimate traffic and contamination by attack traffic, while ensuring that no attack traffic can reach the victim. We define the contamination as the total attack traffic in the network. For example, if the attack traffic is blocked at FR_3 then total contamination is 2 (assume all attackers' packet rate is 1). We denote the contamination in a network for the g filter assignment set by W_c .

$$W_c(g) = \sum_{a=1}^A \alpha_a \times d_a, \quad d_a = \min_{\forall n \in \text{PRED}(n) \cap g} \text{dist}(a, n) \quad (3.1)$$

Here, $\text{PRED}(n)$ is the set of predecessor of n , α_a is the traffic load of attacker a , $\text{dist}(a, n)$ is the number of hops between a and n . A is the total number of attackers. Therefore, W_c is the total attack traffic load for selecting $|g|$ FRs out of $|G|$ FRs. If U is the set of LUs, then the number of blocked LUs for the filter assignment g is denoted by $U_b(g)$.

$$U_b(g) = \{u : u \in U \text{ and } \text{PRED}(u) \cap g \neq \emptyset\} \quad (3.2)$$

The best way to minimize blockage of LU and contamination is to block immediately after the attacker. In reality, there are a huge number of attackers and the victim needs to select a huge number of FRs to block them which is not possible. So, a victim has budget B of selecting a number of FRs. Therefore, $|g|$ should be less than or equal to B .

In **phase 4**, unused filters are removed from the FRs. As the FRs have limited capacity and computation power, it is necessary to reduce the workload by removing the filters. Besides, the attacker can flood the FRs by sending useless filters. This type of filters are evicted soon because they are most likely not being used.

The filter sent by a user (or victim) is only applicable to the packets which are destined for that user (or victim). However, an attacker can spoof IP of the victim and send wrong filters to FRs. This spoofed filter can be detected using a simple handshake protocol. The spoofing attacker will not be able to handshake with the spoofed IP address. However, we are not focusing on finding an optimal filter assignment policy which is discussed in the next section.

3.4 Source-based Filter Assignment Policy

In this section, we formulate a problem of assigning the source-based filters to the FRs so that the contamination is the minimum.

*3.4.1 **Problem 1:** Find a filter assignment so that the contamination is the minimum by ensuring that all the attack traffic is blocked before reaching the victim.*

In this problem, source-based filters are used. The contamination is defined by the total amount of attack traffic in the network. The problem can be expressed as the following:

$$\begin{aligned}
 & \text{minimize} && W_c(g) \\
 & \text{subject to} && |g| \leq B, \forall g \subset G, v \notin G
 \end{aligned} \tag{3.3}$$

The victim v will be white ($v \notin G$) if all attacker traffic is blocked before reaching it. We propose an optimal solution using dynamic programming.

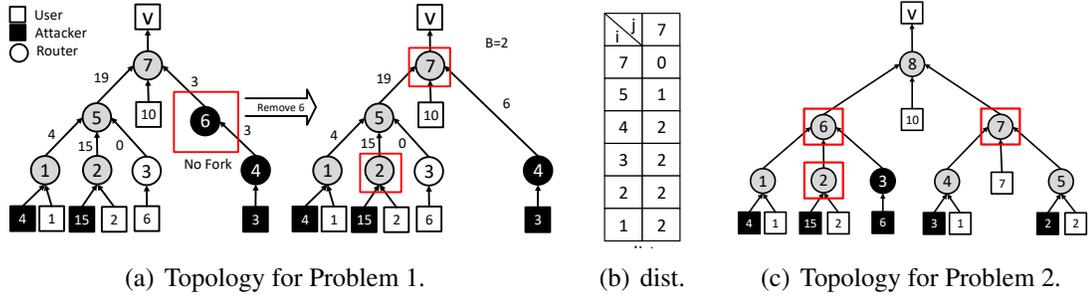


FIGURE 3.3: Topologies for problems 1 and 2.

i \ j	0	1	2	3
1	4,4	0,0	0,0	0,0
2	15,15	0,0	0,0	0,0
4	6,3	0,0	0,0	0,0
5	38,19	8,4	0,0	0,0
7	44,22	14,7	6,3	0,0

A_1

i \ j	0	1	2	3
1	0,0,0	0,0,1	0,0,2	0,0,3
2	0,0,0	0,0,1	0,0,2	0,0,3
4	0,0,0	0,0,1	0,0,2	0,0,3
5	0,0,0	0,1,0	1,1,0	1,2,0
7	0,0,0	1,0,0	2,0,0	2,1,0

R_1

i \ j	0	1	2	3
1	∞	0	0	0
2	∞	0	0	0
4	∞	0	0	0
5	∞	19	0	0
7	∞	44	14	0

A_2

i \ j	0	1	2	3
1	0,0,0	0,0,1	0,0,2	0,0,3
2	0,0,0	0,0,1	0,0,2	0,0,3
4	0,0,0	0,0,1	0,0,2	0,0,3
5	0,0,0	0,1,0	1,1,0	1,2,0
7	0,0,0	0,0,1	1,0,1	2,1,0

R_2

(a) Tables A_1 and R_1 .

(b) Tables A_2 and R_2 .

FIGURE 3.4: Complete values of A_1 , A_2 , R_1 , and R_2

3.4.2 An Optimal Solution

To solve the problem, we define following two problems.

1. $P_1(i, j)$: Find and return optimal contamination in the tree rooted by i including the link to its parent for j number of filters regardless of blocking all attack traffic. The optimal contaminations, unblocked attack traffic loads, and filter assignments are stored in $A_2[i, j, 0]$, $A_2[i, j, 1]$, and $R_2[i, j]$ to reuse in dynamic programming. For this problem, when we refer to subtree rooted by i we mean the subtree rooted by i including the link from i to its parent.
2. $P_2(i, j)$: Find and return optimal contamination in the tree rooted by i for j number of filters by ensuring blockage of all attack traffic. This is the problem defined in Problem 1. The optimal contamination is stored in $A_1[i, j]$ and the filter assignment is stored in $R_1[i, j]$ to reuse in dynamic programming.

There are two options to assign filters. $P_1(i, j)$ is the minimum of the following two options:

Option I: The minimum total contamination, if we assign 1 filter to node i , divide the rest of the $j - 1$ filters into $b_1, b_2, \dots, b_\Delta$ parts, and assign the parts to the subtrees $c_1, c_2, \dots, c_\Delta$, respectively. Therefore, the cost will be the sum of the minimum contaminations in subtrees rooted by i and the contamination for unblocked attack traffic in subtrees rooted by i . In this case, the filter assigned to i blocks all the attack traffic. Therefore, the unblocked attack traffic load is 0 ($A_2[i, j, 1] = 0$) for this option.

$$P_1(i, j) = \min_{\forall b_k \sum_{k=1}^{\Delta} b_k = j-1} \sum_{k=1}^{\Delta} P_1(c_k(i), b_k) \quad (3.4)$$

Option II: The minimum total contamination, if we divide the number of filters into $b_1, b_2, \dots, b_\Delta$ parts and assign them to the subtrees $c_1(i), c_2(i), \dots, c_\Delta(i)$, respectively. Therefore, the contamination for this option will be:

$$\begin{aligned} P_1(i, j) = & \min_{\forall b_k \sum_{k=1}^{\Delta} b_k = j} \sum_{k=1}^{\Delta} \{P_1(c_k(i), b_k) \\ & + A[c_k(i), j_k, 1] \times dist[p(i), i]\} \\ & + AL[i] \times dist[p(i), i] \end{aligned} \quad (3.5)$$

If there are some attackers attached to node i , we add the attack load times the distance to the total contamination. We take the minimum quantity of contamination from these two options.

The optimal contamination $P_2(i, j)$ of using j source-based blocking/filter in subtree rooted by i is the minimum of the following quantity:

Option I: The minimum total contamination, if we assign 1 filter to node i and the rest of the filters to some nodes of the subtree rooted by i . Therefore, the contamination will be the sum of the minimum contaminations in subtrees rooted by i and the contamination for

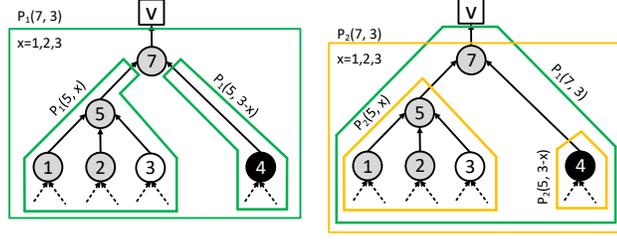


FIGURE 3.5: Recursion model.

unblocked attack traffic in subtrees rooted by i , which is $P_1(i, j)$.

$$P_2(i, j) = \min_{\forall b_k, \sum_{k=1}^{\Delta} b_k = j-1} \sum_{k=1}^{\Delta} P_1(c_k(j), b_k) \quad (3.6)$$

Option II: The minimum total contamination, if we divide the number of filters into $b_1, b_2, \dots, b_{\Delta}$ parts and assign them to the subtrees $c_1(i), c_2(i), \dots, c_{\Delta}(i)$, respectively. Therefore, the contamination for this option will be:

$$P_2(i, j) = \min_{\forall b_k, \sum_{k=1}^{\Delta} b_k = j} \sum_{k=1}^{\Delta} P_2(c_k(j), b_k) \quad (3.7)$$

Therefore, we take the minimum quantity from the above two options. If there are some attackers attached to node i , we do not consider the Option II. This is because, if we assign all the j filters to its subtree then the attack traffic from i will reach the victim v , which is not allowed by the constraint of the problem definition. Fig. 3.5 shows the high level recursion model of the problems.

Let us consider an N node tree with maximum node degree Δ . The nodes are labeled in bottom-up and left-right order. We define A_1 as a $N \times B \times 2$ array which contains optimal contamination and unblocked attack load for every node and budget. For example, $A_1[i, j, 0]$ and $A_1[i, j, 1]$ are optimal contamination and unblocked attack load of budget j in subtree rooted by node i .

We also define A_2 as a $N \times B$ array which contains the contamination for every node and budget. For example, $A_2[i, j]$ is the amount of unblocked attack load for the minimum contamination of budget j on subtree rooted by node i .

We define AL as a $1 \times N$ array which contains the traffic loads of attackers attached to every node. $AL[i]$ is the attack traffic load by the attacker attached to node i .

We also define R_1 as an $N \times B \times (\Delta + 1)$ array which contains the number of filters assigned to node i and its subtrees for every node and budget. For example, $R_1[i, j, 1]$, $R_1[i, j, 2]$, and $R_1[i, j, \Delta + 1]$ are the number of filters to first subtree, second subtree, and node i of subtree rooted by i for budget j . R_1 is the assignment according to P_1 . We define R_2 similar to R_1 but it contains the assignment according to P_2 .

We use a double linked tree data structure. Each node contains a pointer to its parent, an array of pointer to children with distance, and the color of the node. The complete algorithm is shown in Alg. 5.

3.4.3 An Example

Let us consider the tree in Fig. 3.3(a). Firstly, we need to simplify the tree. There is only one node (node 6) without fork. We remove node 6 and make 4 child of its parent 7. The new distance to 4 from 7 increases by the distance of deleted link. The deletion of a node can be done in constant time. Finding out all non-forked nodes takes $O(N)$ time. Therefore, the simplification can be done in $O(N)$ time. Next we calculate distance ($dist$) of every node from the root. This calculation takes $O(N)$ as it needs to traverse the whole tree once again. The complete $dist[i, j]$ is shown in Fig. 3.3(b).

Next, we compute $A_1[i, j]$ and $R_1[i, j]$ for $i = 1, \dots, 7$ and $j = 0, 1, \dots, 3$. $A_1[1, 0, 0]$ is 4 because of no filter is assigned to node 1. The attack traffic in subtree rooted by 1 (including the link from 1 to 5) travels the link $1 \rightarrow 5$. The unblocked attack traffic load $A_1[1, 0, 1]$ is 4. $R_1[1, 0] = [0, 0, 0]$, which means no filter is assigned to left subtree,

right subtree, or itself. We can calculate the $A_1[2, 0]$ and $A_1[4, 0]$ trivially. Similarly, $R_1[2, 0] = [0, 0, 0]$ and $R_1[4, 0] = [0, 0, 0]$.

For all $j \geq 1$ and $i \in \{1, 2, 4\}$ $A_1[i, j] = [0, 0]$. This is because, node 1, 2, and 4 are leaf nodes. A filter can block all the incoming attack traffics entering into network at a leaf node. All the j filters are assigned to node i itself as there is no children ($R_1[i, j] = [0, 0, j]$).

For $i = 5$ and $j = 0$, we have one option. **Option II:** 0 for nodes 5, 1 and 2 ($b_1 = 0, b_2 = 0$). The total contamination in this option is $4 + 15 = 19$. The total unblocked attack load is $4 + 15 = 15$. Therefore, the minimum contamination $A_1[5, 1, 0] = 19$ and unblocked traffic at the minimum contamination is $A_1[5, 1, 1] = 19$. The assignment $R_1[5, 0] = [0, 0, 0]$.

For $i = 5$ and $j = 1$, we have two options. **Option I:** 1 for node 5, and 0 for node 1 and 2. ($b_1 = 0, b_2 = 0$). The total contamination in this option is $4 + 15 = 19$. The total unblocked attack load is 0. Therefore, $A_1[5, 1] = [19, 0]$.

Option II: In this option, we have two choices. Choice (1): 0 for node 5, 1 for node 1, and 0 for node 2 ($b_1 = 1, b_2 = 0$). The total unblocked attack load is $0 + 15 = 15$. The total contamination in this choice is $0 + 15 + 15 = 30$.

Choice (2): 0 for node 5, 0 for node 1, and 1 for node 2 ($b_1 = 0, b_2 = 1$). The total unblocked attack load is $4 + 0 = 4$. The total contamination in this choice is $4 + 0 + 4 = 8$. For option 2, the minimum contamination is 8 and the total unblocked attack load is 4. Therefore, the minimum contamination between option 1 and 2 is $A_1[5, 1, 0] = 8$ and unblocked traffic at minimum contamination is $A_1[5, 1, 1] = 4$. The minimum contamination is found with the assignment $R_1[5, 1] = [0, 1, 0]$. Similarly, we calculate the rest of the values in A_1 and R_1 . Fig. 3.4(a).

Next, we compute $A_2[i, j]$ and $R_2[i, j]$ for $i = 1, \dots, 7$ and $j = 0, 1, \dots, 3$. $A_2[1, 0]$ is ∞ because of no filter is assigned to node 1 the attack traffic in subtree rooted by 1. Therefore it cannot block all of the attack traffics. $R_2[1, 0] = [0, 0, 0]$, which means no

filter is assigned to left subtree, right subtree, or itself. We can calculate easily that the value of $A_2[2, 0]$ and $A_2[4, 0]$ is ∞ . Similarly, $R_2[2, 0] = [0, 0, 0]$ and $R_2[4, 0] = [0, 0, 0]$.

For all $j \geq 1$ and $i \in \{1, 2, 4\}$ $A_2[i, j] = 0$. This is because, nodes 1, 2, and 4 are leaf nodes. A filter can block all the incoming attack traffic entering into network at a leaf node. All the j filters are assigned to node i itself as there is no children ($R_2[i, j] = [0, 0, j]$).

For $i = 5$ and $j = 1$, we have two options. **Option I:** 1 for node 5, and 0 for node 1 and 2. ($b_1 = 0, b_2 = 0$). The total contamination in this option is already calculated in A_1 . We need to lookup $A_1[1, 0]$ and $A_1[2, 0]$. The total contamination is $15 + 4 = 19$.

Option II: In this option, we have two choices. Choice (1): 0 for node 5, 1 for node 1, and 0 for node 2 ($b_1 = 1, b_2 = 0$). The total contamination in this choice is $0 + \infty = \infty$.

Choice (2): 0 for node 5, 0 for node 1, and 1 for node 2 ($b_1 = 0, b_2 = 1$). The total contamination in this choice is $\infty + 0 = \infty$. For option 2, the minimum contamination is ∞ . Therefore, the minimum contamination between option 1 and 2 is $A_2[5, 1] = 19$. The minimum contamination is found with the assignment $R_2[5, 1] = [0, 0, 1]$.

Similarly, we calculate the rest of the values in A_1 and R_1 . Fig. 3.4(b) shows the complete value of $A_1, A_2, R_1,$ and R_2 . $A_2[7, 3]$ contains the optimal contamination for 3 filters in the topology in Fig. 3.3(a).

Assignment Set Formulation We generate the filter assignment set using R_1 and R_2 . According to the definition $R_2[7, 3]$ contains the assignment at node 7. $R_2[7, 3] = [2, 1, 0]$ means left subtree is assigned 2 filters and right subtree is assigned 1 filter. Next, we need to lookup $R_2[5, 2]$ and $R_2[4, 1]$. $R_2[5, 2]$ is $[1, 1, 0]$, which means left and right subtrees both are assigned 1 filter. So, we need to lookup $R_2[1, 1]$ and $R_2[2, 1]$. $R_2[1, 1]$ and $R_2[2, 1]$ is $[0, 0, 1]$ which means a filter is assigned to both node 1 and 2. Now our assignment set is $\{1, 2\}$. Next, we lookup $R_2[4, 1]$. $R_2[4, 1]$ is $[0, 0, 1]$, which means a filter is assigned to node 4. Therefore, the final assignment set is $\{1, 2, 4\}$.

Algorithm 5 DP Blocking Strategy for Problem 1

Input: The number of filters B , topology tree T .

Output: A set of nodes in T .

```
1: Procedure: BLOCK-DP1( $B, T$ )
2:    $N \leftarrow$  number of nodes in  $T$ 
3:   for every entry node  $i$  do
4:     Initialize  $AL[i]$ .
5:     for  $j = 0$  to  $B$  do
6:       Initialize  $A_1[i, j]$ ,  $A_2[i, j]$ ,  $R_1[i, j]$ , and  $R_2[i, j]$ 
7:   CALC-P1( $N, B$ )
8:   CALC-P2( $N, B$ )
9:   return ASSIGNMENT( $R, N, B$ )
```

In the above example, we do not need to lookup in R_1 . Next, we see an example where we need to lookup both R_1 and R_2 . According to the definition $R_2[7, 2]$ contains the assignment at node 7 for a budget of 2 filters. $R_2[7, 2] = [1, 0, 1]$ means left subtree is assigned 1 filter and right subtree is assigned 0 filter. Node 7 itself is assigned 1 filter. Now, our assignment set is $\{7\}$. As the node itself is assigned a filter the afterwards assignments come from R_1 . Next we need to lookup $R_1[5, 1]$. $R_1[5, 1]$ is $[0, 1, 0]$, which means right subtree is assigned 1 filter. So, we need to lookup $R_1[2, 1]$. $R_1[2, 1]$ is $[0, 0, 1]$, which means a filter is assigned to both node 2. Therefore, the final assignment set is $\{7, 2\}$. The algorithm is shown in Alg. 5.

Theorem 5. *Complexity and space needed of the Alg. 5 are $O(NB^\Delta)$ and $O(NB\Delta)$.*

Proof. Let us consider the topology is an N node tree with maximum node degree Δ and the victim has budget of B . To find the partitions $b_1, b_2, \dots, b_\Delta$ we need $O(B^{(\Delta-1)})$ time if we use naive nested iteration approach. Therefore, the complexity of the Alg. 5 is $O(NB^{(\Delta)})$. The total space needed for A_1, A_2, R_1, R_2 , and AL are $2NB, NB, (\Delta + 1)NB, (\Delta + 1)NB$, and N , respectively. The total space needed is an order of $O(NB\Delta)$. \square

Theorem 6. *The Alg. 5 provides optimal solution.*

Algorithm 6 Calculate A_1 and R_1

```
1: Procedure: CALC-P1( $N, B$ )
2:   for  $i = 1$  to  $N$  do
3:     for  $j = 0$  to  $B$  do
4:        $min \leftarrow \infty, map \leftarrow \emptyset$ 
5:       for  $\forall b_1, b_2, \dots, b_\Delta : \sum_{k=1}^\Delta b_k = j$  do
6:          $p \leftarrow \sum_{k=1}^\Delta A_1[c_k(i), b_k]$ 
7:         PUT( $map, [b_1, b_2, \dots, b_\Delta, 0], p$ )
8:       for  $\forall b_1, b_2, \dots, b_\Delta : \sum_{k=1}^\Delta b_k = j - 1$  do
9:          $p \leftarrow \sum_{k=1}^\Delta \{A_1[c_k(i), b_k] + A_1[i] \text{dist}[p(i), i]\}$ 
10:         $+ AL[i] \times \text{dist}[p(i), i]$ 
11:        PUT( $map, [b_1, b_2, \dots, b_\Delta, 1], p$ )
12:        $A_1[i, j, 0] \leftarrow \text{MIN}(map)$ 
13:        $R_1[i, j] \leftarrow \text{ARGMIN}(map)$ 
14:        $A_1[i, j, 1] \leftarrow \sum_{k=1}^\Delta A_1[c_k(i), R_1[i, j, k], 1] + AL[i]$ 
```

Proof. The Alg. 5 uses a dynamic programming bottom-up strategy to search the optimal assignment. For an one-node tree, if the node color is not “white”, then there is no solution for $B = 0$. Because without any filter, the attack traffic will be forwarded to the downstream routers and finally reach v . For $B \geq 1$ there is only one choice of selecting FR which is that node. If that node is selected, the optimal contamination is 0. In each step, the Alg. 9 chooses the allocation of filters to itself, left subtree, or right subtree which produce the minimum contamination. Therefore, the Alg. 5 provides optimal filter assignment to the FRs through exhaustive search. \square

3.5 Destination-based Filter Assignment Policy

As we are using destination-based filters for protection against spoofed DDoS attack, we are blocking some LUs. In this section, we formulate another optimization problem of assigning destination-based filters to the FRs so that a weighted sum of the contamination and blocked LUs is the minimum.

Algorithm 7 Calculate A_2 and R_2

```
1: Procedure: CALC-P2( $N, B$ )
2:   for  $i = 1$  to  $N$  do
3:     for  $j = 0$  to  $B$  do
4:        $min \leftarrow \infty, map \leftarrow \emptyset$ 
5:       for  $\forall b_1, b_2, \dots, b_\Delta : \sum_{k=1}^{\Delta} b_k = j - 1$  do
6:          $p \leftarrow \sum_{k=1}^{\Delta} A_1[c_k(i), b_k]$ 
7:         PUT( $map, [b_1, b_2, \dots, b_\Delta, 1], p$ )
8:       if  $AL[i] \neq 0$  then
9:         for  $\forall b_1, b_2, \dots, b_\Delta : \sum_{k=1}^{\Delta} b_k = j$  do
10:           $p \leftarrow \sum_{k=1}^{\Delta} A_2[c_k(i), b_k]$ 
11:          PUT( $map, [b_1, b_2, \dots, b_\Delta, 0], p$ )
12:           $A_2[i, j] \leftarrow \text{MIN}(map)$ 
13:           $R_2[i, j] \leftarrow \text{ARGMIN}(map)$ 
```

3.5.1 **Problem 2:** Find a filter assignment so that the LU blockage and contamination are the minimum.

It is always better if the victim can select some FRs within its budget which minimizes both the number of blocked LUs and contamination. As discussed in Section 3.3, the source-based filter cannot ensure protection against IP spoofing DDoS attack. For example, if the attacker attached to node 1 uses IP address of the users attached to node 3 (see Fig. 3.3(a)). The filter used at node 1 or 5 would forward the packet. But if the FRs use destination-based filter then no spoofed attack packet can penetrate. Therefore, the victim would use the destination-based filters. The problem can be expressed as the following optimization problem:

$$\begin{aligned} & \text{minimize} && \omega W_c(g) + (1 - \omega)|U_b(g)| \\ & \text{subject to} && |g| \leq B, \forall g \subset G, v \notin G \end{aligned} \tag{3.8}$$

Here $\omega = [0, 1]$ is considered a system parameter which determines priority of total contamination and LU blockage.

Algorithm 8 Find Assignment

```
1: Procedure: ASSIGNMENT( $R_1, N, B$ )
2:   if  $B = 0$  then
3:     return  $\emptyset$ 
4:    $x.i \leftarrow N, x.j \leftarrow B, g \leftarrow \emptyset$ , and  $Q \leftarrow \emptyset$ .
5:   ENQUEUE( $Q, x$ ).
6:   while  $Q \neq \emptyset$  do
7:      $x \leftarrow$  DEQUEUE( $Q$ )
8:     if  $R_1[x.i, x.j, \Delta + 1] \neq 0$  then
9:        $g' \leftarrow \bigcup_{i=1}^{\Delta} \text{ASSIGNMENT}(R_2, c_k(x.i), R_1[x.i, x.j, i])$ .
10:       $g = g \cup g' \cup \{x.i\}$ 
11:     else
12:       for  $k = 1$  to  $\Delta$  do
13:          $b_c.i \leftarrow c_k(x.i), b_c.j \leftarrow R[x.i, x.j, k]$ 
14:         ENQUEUE( $Q, b_c$ )
15:   return  $g$ 
```

3.5.2 An Optimal Solution

To solve the problem, we define following problem.

1. $P_3(i, j)$: Find and return the optimal cost in the subtree rooted by i by ensuring blockage of all attack traffics before reaching v . The number of LUs, optimal cost, and filter assignments are stored in L, A and R to reuse in dynamic programming.

The optimal cost of using j destination-based blocking/filter in subtree rooted by i is the minimum of the following quantity:

Option I: The minimum total weighted cost, if we assign 1 filter to node i and the rest of the filters to some nodes of the subtree rooted by i . Therefore, the cost will be a weighted sum of the minimum contamination and LU in subtree rooted by i . When we assign a filter to i , then number of blocked LUs remains constant regardless of other filter assignment. The problem becomes similar to problem 1. We can apply the Alg. 5 to find an optimal assignment in subtree rooted by i . First, we assume an attacker attached to the i . This assumption confines a filter to i . Then we find an assignment of budget j using

i \ j	0	1	2	3
1	∞	0.5	0.5	0.5
2	∞	1	1	2
3	∞	0	0	0
4	∞	1	1	1
5	∞	2	2	2
6	∞	14	6.5	1.5
7	∞	7.5	1.5	1.5
8	∞	36.5	21.5	14

i \ j	0	1	2	3
1	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
2	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
3	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
4	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
5	0,0,0,0	0,0,0,1	0,0,0,2	0,0,0,3
6	0,0,0,0	0,0,0,1	0,0,0,2	1,1,1,0
7	0,0,0,0	0,0,0,1	1,1,0,0	2,1,0,0
8	0,0,0,0	0,0,0,1	1,1,0,0	2,1,0,0

i	
1	1
2	2
3	0
4	1
5	2
6	3
7	10
8	23

FIGURE 3.6: A , R , and L .

Alg. 5. As i will be assigned a filter, the other $j - 1$ filters will be assigned to the subtree rooted by i .

Then the cost for this option will be:

$$P_3(i, j) = \omega P_2(i, j) + (1 - \omega)L[i] \quad (3.9)$$

Option II: The minimum total weighted cost, if we divide the number of filters into $b_1, b_2, \dots, b_\Delta$ parts and assign them to the subtrees $c_1(i), c_2(i), \dots, c_\Delta(i)$, respectively. Therefore, the cost for this option will be:

$$P_3(i, j) = \sum_{k=1}^{\Delta} P_3(c_k(i), b_k) \quad (3.10)$$

Therefore, we take the minimum quantity between the above two options. If there are some attackers attached to node i , we do not consider the option II. This is because, if we assign all the j filters to its subtree then the attack traffic from i will reach the victim v , which is not allowed by the constraint of the problem definition.

Let us consider an N node tree with maximum node degree Δ . The nodes are labeled in bottom-up and left-right order. We define A as a $N \times B$ array which contains the optimal cost for every node and budget. For example $A[i, j]$ is optimal cost of budget j on subtree rooted by node i .

We define L as a $1 \times N$ array which contains the number of LUs in subtree rooted by every node. $L[i]$ is the number of LUs in subtree rooted by node i . We also define R as an $N \times B \times (\Delta + 1)$ array which contains the number of filters assigned to node i and its subtrees for every node and budget. For example, $R[i, j, 1]$, $R[i, j, 2]$, and $R[i, j, \Delta + 1]$ are the number of filters to first subtree, second subtree, and node i of subtree rooted by i for budget j . The complete algorithm is shown in Alg. 9.

3.5.3 Example

Let us consider the traffic topology in Fig. 3.3(c) and $\omega = 0.5$. The leaf entry nodes are 1, 2, 3, 4, and 5. The calculations of A , R , and L are straightforward. For example $A[1, 0] = \infty$, $A[1, 1] = 0.5 \times 0 + 0.5 \times 1 = 0.5$, and $A[1, 2] = 0.5 \times 0 + 0.5 \times 1 = 0.5$. $R[i, 1] = [0, 0, 0, 1]$, $R[i, 2] = [0, 0, 0, 2]$, and $R[i, 3] = [0, 0, 0, 3]$. For node 6 and $j = 0$, we have one choice $b_1 = 0, b_2 = 0, b_3 = 0$ and the cost is ∞ . For $j = 1$, we have two options.

Option I: 1 for node 6, and $b_1 = 0, b_2 = 0, b_3 = 0$. The total cost is $0.5 \times 25 + 0.5 \times 3 = 14$.

Option II: 0 for node 6, and (1) $b_1 = 1, b_2 = 0, b_3 = 0$, (2) $b_1 = 0, b_2 = 1, b_3 = 0$, or (3) $b_1 = 0, b_2 = 0, b_3 = 1$. For option (1): total cost is $\infty + \infty + \infty = \infty$. For option (2) and (3) total cost is also ∞ . Therefore, Option I is the minimum ($A[6, 1] = 14$) and $R[6, 1] = [0, 0, 0, 1]$.

For $j = 2$, there are also two options.

Option I: 1 for node 6 and 1 is for its subtrees. We assume an attacker attached to 6. Then applying the Alg. 5 for $B = 2$, we find the assignment is $\{6, 2\}$. After assigning the filters the total contamination is $4 + 6 = 10$. The total cost in this option is $10(0.5) + 3(1 - 0.5) = 6.5$.

Option II: 0 for node 6, and 2 filters to subtrees of 6. This option is valid for node 6 because there is no attacker directly attached to it. There can be six choices:

Algorithm 9 DP Blocking Strategy

Input: The number of filters B , topology tree T .

Output: A set of nodes in T .

```
1: Procedure: BLOCK-DP2( $B, T$ )
2:    $N \leftarrow$  number of nodes in  $T$ 
3:   for every entry node  $i$  do
4:     Initialize  $AL[i]$  and  $L[i]$ .
5:     for  $j = 0$  to  $B$  do
6:       Initialize  $A_1[i, j], A_2[i, j], A[i, j], R_1[i, j], R_2[i, j]$ , and  $R[i, j]$ 
7:     COMP-P1( $N, B$ )
8:     COMP-P2( $N, B$ )
9:     COMP-P3( $N, B$ )
10:  return ASSIGNMENT( $R, N, B$ )
```

(1) $b_1 = 2, b_2 = 0, b_3 = 0$, (2) $b_1 = 0, b_2 = 2, b_3 = 0$, (3) $b_1 = 0, b_2 = 0, b_3 = 2$, (4) $b_1 = 0, b_2 = 1, b_3 = 1$, (5) $b_1 = 1, b_2 = 0, b_3 = 1$, and (6) $b_1 = 1, b_2 = 1, b_3 = 0$. For choice (1), the total cost is $A[1, 2] + A[2, 0] + A[3, 0] = 0.5 + \infty + \infty = \infty$. Similarly, the choices (2) to (6) cost ∞ . Therefore, option I is the minimum and $A[6, 2] = 6.5$ and $R[6, 2] = [0, 0, 0, 2]$. Similarly, we calculate the rest of the entries in A and R . The complete A , L and R are shown in Fig. 3.6. According to the definition, $A[8, 3]$ contains the cost for budget $B = 3$ which is 14.

Assignment Set Formulation From R we can find out which FRs need to be blocked. $R[8, 3, 1] = 2$ and $R[8, 3, 2] = 1$ means 2 and 1 filters are assigned to its first and second subtrees, respectively. Then we need to look $R[6, 2]$ and $R[7, 1]$. $R[6, 2, 1] = 0$, $R[6, 2, 2] = 0$, $R[6, 2, 3] = 0$, and $R[6, 2, 4] = 2$ means no filter is assigned to its subtrees and two filters are assigned to itself. Therefore, we need to find the assignment according to Option I. According to Option I $\{6, 2\}$ is the assignment. Similarly, we can find that a filter is assigned to node 7. So, the filter assignment is $\{2, 6, 7\}$ for budget $B = 3$.

Theorem 7. *Complexity and space needed of the DP Blocking Strategy are $O(NB^{\Delta-1})$ and $O(NB\Delta)$.*

Algorithm 10 Compute A and R

```
1: Procedure: COMP-P3( $N, B$ )
2:   for  $i = 1$  to  $N$  do
3:     for  $j = 0$  to  $B$  do
4:        $L[i] \leftarrow \sum_{k=1}^{\Delta} L[c_k(i)]$ 
5:        $min \leftarrow \infty$ 
6:       if  $AL[i] > 0$  then
7:          $p \leftarrow \omega A_2[i, j] + (1 - \omega)L[i]$ 
8:         PUT( $list, A_2[i, j], p$ )
9:       else
10:        for  $\forall b_1, b_2, \dots, b_k : \sum_{k=0}^{\Delta} b_k = j$  do
11:           $p \leftarrow \sum_{k=1}^{\Delta} A[c_k(i), b_k]$ 
12:          PUT( $list, [b_1, b_2, \dots, b_{\Delta}, 0], p$ )
13:        $A[i, j] \leftarrow \text{MIN}(map)$ 
14:        $R[i, j] \leftarrow \text{ARGMIN}(map)$ 
```

Proof. Let us consider the topology is a N node tree with maximum node degree Δ and the victim has budget of B . To find the partitions $b_1, b_2, \dots, b_{\Delta}$ we need $O(B^{(\Delta-1)})$ time if we use naive nested iteration approach. Therefore, the complexity of the Alg. 9 is $O(NB^{(\Delta-1)})$. The total space needed is $(N-1)B + (\Delta+1)(N-1)B + (N-1)$ which is an order of $O(NB\Delta)$. For a binary tree topology the complexity is $O(NB^2)$ and the space complexity is $O(NB)$. \square

Theorem 8. *The Alg. 9 provides optimal solution.*

Proof. The Alg. 9 uses a dynamic programming bottom-up strategy to search the optimal assignment. For an one-node tree, if the node color is “black” or “gray” then there is no solution for $B = 0$. Because without any filter, the attack traffic will be forwarded to the downstream routers. For $B \geq 1$ there is only one choice of selecting FR which is that node. If that node is selected, the optimal number of blocked LUs is the LUs attached to it. In each step, the Alg. 9 chooses the best allocation of filters to itself, left subtree, or right subtree. Therefore, the Alg. 9 provides optimal filter assignment to the FRs through exhaustive search. \square

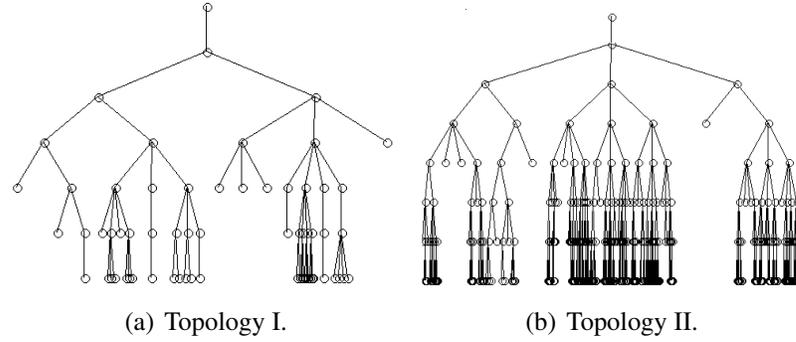
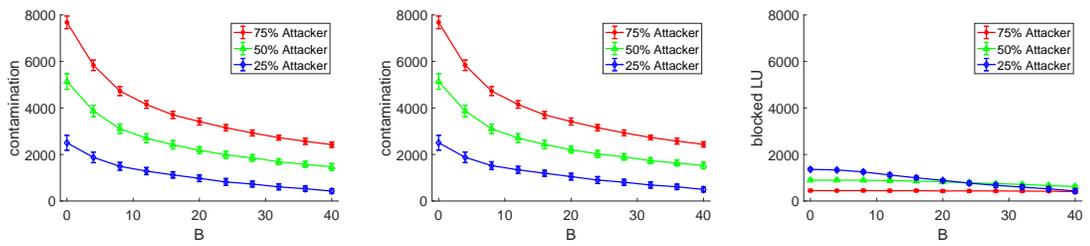
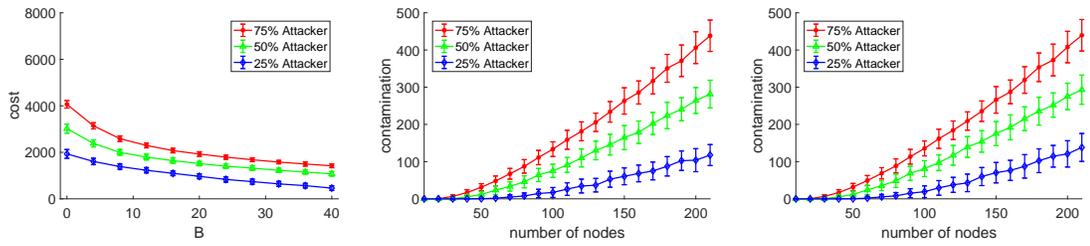


FIGURE 3.7: Randomly generated and real topologies.



(a) Contamination (source-based). (b) Contamination (dest-based). (c) Blocked LU traffic (dest-based).



(d) Cost (dest-based). (e) Contamination (source-based). (f) Contamination (dest-based).

FIGURE 3.8: Simulation results.

3.6 Simulations

In this section, we present our experimental settings and simulation results.

3.6.1 Simulation Setting

We conduct the experiments with a custom build Java simulator. The main reason for using a custom build for simulator is its scalability. We do not need to analyze transmission time, bandwidth, or packet drop issues. We only need to count the number of legitimate

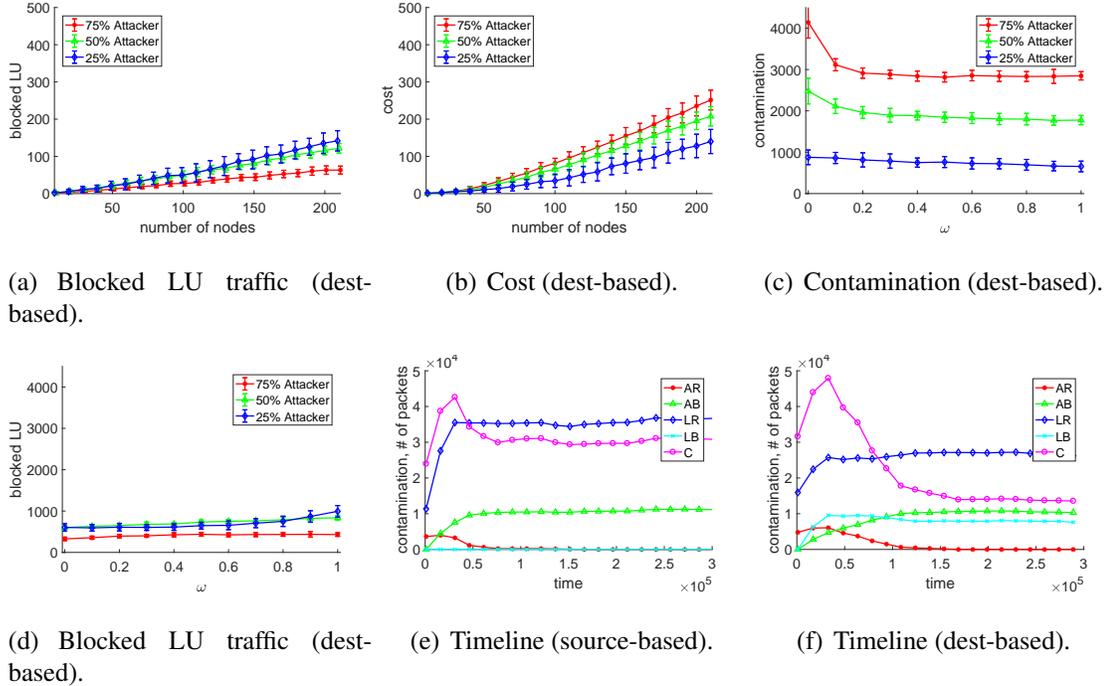


FIGURE 3.9: More simulation results.

Table 3.1: Topology Parameters

	Topology I	Topology II
Number of nodes	66	403
Internal user probability	0.1	0.1
Attacker ratio	0.4	0.4
Max Node Degree	4	20
Data Rate(pack/ms)	[0.1-0.4]	[0.1-0.4]

(or attack) received (or blocked) packets. The network topologies we considered contain about 100 – 500 routers. Using NS3 or other similar simulator for this kind of simulation would take several days. That is why we built our own Java multi-threaded simulator to get the results quickly.

We conduct simulations for randomly generated tree topologies and a subset from a real network topology. To generate a random tree, we first generate the desired number of nodes. Then, we randomly pick a root among the nodes. After that a random node from the generated nodes is picked up and added as a child to a random node in the tree. The

process continues until all of the generated nodes are added to the tree. We use a randomly generated topology having node degree between $[0 - 4]$, internal node user probability between $[0.1 - 0.25]$, and maximum depth of 6. The entry nodes' color and number of users or attackers are selected randomly from a uniform distribution. The Topology I is a randomly generated tree of 66 nodes and max node degree of 4. The Topology II is taken from a subset of the Stanford University AS-733 dataset [24]. The dataset contains 6,474 nodes and we took a subset (which is a tree) containing 403 nodes. Then we randomly assigned users to the tree with internal user probability 0.1. The details are shown in Table 3.1 and Fig. 3.7.

We measure the performances of our proposed solution in terms of contamination (C), blocked LU traffic, number of blocked attack packets (AB), number of received attack packets (AR), number of blocked legitimate packets (LB), and number of received legitimate packets (LR) for the two topologies.

3.6.2 Simulation Results

For the following experiments we use the Topology II to observe the performances of both approaches for different budgets. We change the attacker ratio and repeat the experiments. We plot the average and standard deviation of 100 random attacker and LU distributions.

Fig. 3.8(a) shows the contamination by the number of source-based filters. We vary the number of filters from 1 to 40. The contamination of 75% attacker is the highest, 25% attacker is the lowest, and 50% attacker is in between for all budgets. The contaminations of all attacker distributions decrease by the number of filters. The more the number of filters is, the closer the filters are deployed to the attackers. As a result, higher number of filters produce a lower contamination. For 75% attacker the contamination with 1 and 40 filters are 7,681 and 2,424. Therefore, the contamination is 68% reduced. For 25% attacker the contamination with 1 and 40 filters are 2,503 and 426. Therefore, the contamination is 82% reduced. Figs. 3.8(b), 3.8(c), and 3.8(d) show the contamination, blocked

LU traffic, and cost for destination-based filters. We keep the ω as 0.5. We observe that the contamination of source-based filter and destination-based filters are similar. This is because, we give equal priority to the contamination and blocked LU traffic. The amount of blocked LU traffic is also decreasing by the number of filters. As a result, the cost is decreasing by the number of filters. For 75% attacker the blocked LU traffic with 1 and 40 filters are 446 and 402. Therefore, the blocked LU traffic is 10% reduced. For 25% attacker the blocked LU traffic with 1 and 40 filters are 1,360 and 427. Therefore, the blocked LU traffic is 69% reduced.

Fig. 3.8(e) shows the contamination by the number of nodes. We vary the number of nodes from 10 to 210. We keep the number of filters as 20. Similar to the above experiment, contamination of 75% attacker is the highest, 25% attacker is the lowest, and 50% attacker is in between for all number of nodes. The contaminations of all attacker distributions increase by the number of nodes. The more the number of nodes is, the more the number of attackers and height of the tree. As a result, higher number of nodes produce higher contamination. For 75% attacker the contamination with 10 and 210 noded trees are 0 and 438. For 25% attacker the contamination with 10 and 210 noded trees are 0 and 117. In 10 noded trees, we observe contamination is 0, this is because 20 filter are more than enough to block every attacker at the closest router. Therefore, no attack traffic enters into the network and contamination is zero. Figs. 3.8(f), 3.9(a), and 3.9(b) show the contamination, blocked LU traffic, and cost for destination-based filters. We keep the ω as 0.5. We observe that the contamination of source-based and destination-based filters are also similar. The amount of blocked LU traffic is also increasing by the number of nodes. As a result, the cost is increasing by the number of nodes. For 75% attacker the blocked LU traffic in 10 and 210 noded trees are 1 and 63. For 25% attacker the blocked LU traffic in 10 and 210 noded trees are 2 and 147. The contamination, blocked LU traffic, and cost increase almost linearly by the number of nodes.

Figs. 3.9(c) and 3.9(d) show the contamination, blocked LU traffic, and cost for destination-based filters. We vary the value of ω from 0 to 1. We keep the number of filters as 30. The contamination and amount of blocked LU traffic decrease and increase by ω , respectively. For 75% attacker when $\omega = 0$, the contamination and blocked LU traffic are 4,141 and 327. When $\omega = 1$, the contamination and blocked LU traffic are 2,848 and 431. When ω is lower the blocked LU traffics are prioritized over contamination. Therefore, when $\omega = 0$ the contamination is higher than of when $\omega = 1$. Similarly, when $\omega = 0$ the blocked LU traffic is lower than of when $\omega = 1$.

Figs. 3.9(e) and 3.9(f) show the C, AB, AR, LB, and LR using source-based and destination-based filters, respectively. The Topology I is used in this experiment because it is smaller than the Topology II. For this reason we can observe the effect of topology construction better in Topology I than in Topology II. Here the contamination is the total number of attack-packet forwarding events. We can see that, at the beginning (ignoring the warm-up period from time 0 to 0.025), the C in every approach is higher. The C reduces over time and becomes gradually more stable. This is because, at the beginning the victim knows a small subset of the topology. Over time, the victim gets more and more information from the marked packet and constructs the traffic topology. Finally, the victims knowledge about the topology becomes stable. That is why the AR is high at the beginning, decreases over time, and finally converges to zero. The AB shows the opposite behavior for the same reason. We also observe that the number of LB is zero in source-based filter. The AB is zero initially when no filter is deployed. Gradually, the AB increases and becomes stable after some time.

3.7 Summary

The DDoS attack is the most powerful attack to make a service unavailable to users. It is not possible to protect any server from DDoS attack without the help of the network equipment. As the most important component in a network, routers can be upgraded to

filter routers easily. Besides, the filter router can work in a network with legacy routers. In the four-phase DDoS protection system, the filter routers block the attack traffic according to the victim's instruction. Though the blocking control of an internet service provider (ISP) is at victims hand who may not belong to the ISP but it will help the ISP to minimize traffic congestion. Therefore, both parties are benefited. In this work, we present two filter assignment policies for two different settings. We observe the performances of proposed policis in synthetic and real topologies. Both the source-based and destination-based filters have some advantages and limitations. Upto this chapter, we have optimal solutions for the problems where we consider the traffic typologies are trees. This solution is not applicable for generalised graph topology structures. Practically, the traffic topologies are directed acyclic graphs in most cases as the ISPs exploits load balancing techniques. In Chapter 4 we propose optimal solution of similar problems defined in this chapter.

CHAPTER 4

PROTECTING RESOURCES AGAINST VOLUMETRIC AND NON-VOLUMETRIC NETWORK ATTACKS

In this chapter, we propose an architecture to defend against both volumetric and non-volumetric types of attacks. We formulate a problem to minimize the damage caused by the volumetric attack by using a limited number of blockage at some routers. In our previous chapters we considered tree based topologies and in this chapter we consider directed acyclic graph based topologies. The contents of this chapter is submitted to [Submitted1].

4.1 Introduction

The growth of internet usage and services produces a lot of opportunities for network attackers. Based on the effectiveness of the volume of attack packets we divide the attacks into two categories: volumetric and non-volumetric attacks. In a volumetric attack, the amount of damage depends on the amount of attack volume. The packets are not harmful or contain malware but the amount creates congestion in the network and cause it to stop serving regular users.

The distributed denial-of-service (DDoS) and link flooding attacks (LFA) are this kind of attack. In a DDoS attack, a huge number of bots send a lot of service requests to the victim, resulting in network congestion and/or out of processing capability of the victim.

As a result, the victim cannot serve its regular user and denial-of-service or degradation of quality-of-service (QoS) happens. The bots are usually malicious programs and controlled by a master. The master sends commands including the information of the victim to the bots and they act accordingly. Nowadays, DDoS attacks are the cheapest and most powerful attacks among all others. In an LFA attack, instead of directly targeting a server, one or multiple links are targeted which are an important part of the path to the victim. The master chooses multiple pairs of bots and decoy servers to generate traffic. The bot and decoy server pairs are chosen in such a way that the flow from the bot to the decoy server travels at least one target link. Decoy servers are owned by the attackers and they are used to receive the attack traffic from the bots. When the capacity of the target link is overwhelmed by the attack traffic, the victim gets disconnected from the network and denial-of-service happens.

To defend against this attack, we use filters that consist of some rules to block certain traffic based on its source and destination address. A filter can be applied to the routers by the defender to reduce the unnecessary traffic reaching the resources. There is limited storage for filters and the owner ISP may charge money for assigning filters. Therefore, we consider a limited budget for the number of filters. A good filter assignment can block more traffic and waste least resources. For example, in Fig.4.1, if we are allowed to place only one filter, we may place it on router B . A filter on B will allow a maximum of 1 attack traffic to the resource R . If we place it on E then 3 attack traffic can reach R at most. Therefore, a good filter assignment is needed to minimize the amount of attack traffic reaching the resources to minimize damage.

In a non-volumetric attack, the amount of traffic is not proportional to the damage to the resources. The attacker intends to steal the content of the resource or gain special access permission to the resources. To achieve the final goal, the attacker needs to pass multiple steps. There can be one or multiple ways (series of steps) to reach the goal. For these types of attacks, the defender needs to cut all possible ways to reach the resources.

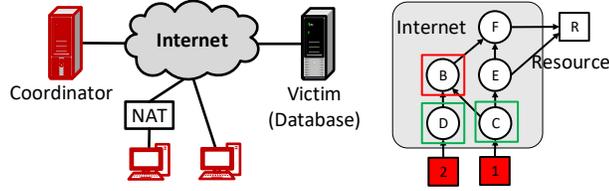


FIGURE 4.1: An example of attack and defense mechanism.

Let us consider the figure in Fig. 4.1 and assume the nodes are the steps. The first step of the attackers is either passing through D or C . In reality, the D step can represent the guessing of the password of a server. Step B can be gaining root privilege of that server. When an attacker passes one more step, it gets closer to the goal and causing damage to the network or datacenter. To prevent the attacker, the defender needs to apply a moving target defend (MTD) approach to stop the attacker at some particular steps. Simultaneously, the attacker needs to be stopped as early as possible to minimize the damage. In Fig. 4.1, we need at least two MTDs to deploy to stop the attack. The best locations for the MTDs are at step D and C because they yield no damage to the network/datacenter. If we assign the MTDs at B and E it would stop the attacker from reaching the goal but we would lose D and C . Therefore, a good MTD assignment is necessary to minimize the damage while ensuring the protection of the resources.

In this paper, we study the defending mechanism of volumetric and non-volumetric attacks and model the problem in an optimization framework. We formulate two problems for these two types of attacks with different objectives and constraints. We consider the amount of damage to be proportional to the amount of traffic in volumetric attacks. The amount of damage in a non-volumetric attack is proportional to the number of steps. For simplicity, we are considering the amount of damage for each step to be the same. Therefore, the main contributions are the following:

1. We study an optimization problem for minimizing damage caused by the volumetric attack and provide an approximation solution with a performance guarantee.

2. We formulate another problem for minimizing damage for non-volumetric attacks by ensuring the protection of the resources. A dynamic programming solution is provided for this problem.
3. An extensive simulation is conducted to evaluate the solutions with synthetic and real data.

The remainder of this paper is arranged as follows: Section 4.2 presents some related works. In Section 4.3, we present the system, attacker, and cost models. Section 4.4 and Section 4.5 present the formal definitions of the problems and our proposed solutions, respectively. Section 4.6 presents some simulation results. Finally, Section 4.7 concludes the paper.

4.2 Related Works

For volumetric attacks there exist many statistical methods, including correlation, covariance, entropy, cross-correlation, and information gain to detect anomalous attack requests [8]. A rank correlation-based method and information theoretical approach are proposed in [9] and [10], respectively. An Artificial Neural Network based approach is proposed in [11]. In proposed in [27] authors propose a dynamic neural network that learns to activate the neuron based on input data using unsupervised learning. Other type of DDoS attack defense mechanism includes statistical methods to classify packets and block them [9, 10, 12, 13, 14].

There exists many research on another type of volumetric attack called link flooding attack. In [28], the authors propose SPIFFY that logically increases link capacity when it detects congestion. The attacker may increase the data rate of each bot after the capacity increases and gets identified. A router functionality based mechanism is proposed in [29], in which each router detects and preferentially drop packets that likely belong to an attacker. The upstream routers gets notified of the drop event so that they do not waste the resources by forwarding attack traffic. In ColDef [30] mechanism, the domains which are

uncontaminated by attackers help to route the legitimate traffic. It also enables routers to detect low-rate attacks flows. In [31], authors proposed a link flooding attack mitigation system by using BGP rules in BGP routers. If a link congestion is detected, then the BGP router advertises its neighbors to avoid the congested link. To do so, it creates a false path by appending its own address and advertises the false path. The BGP routers from other autonomous systems find that the path has a loop and avoid it.

There also exists several works for mitigating non-volumetric attacks. In [32], authors present a path discovery method of cyber-attacks. The method uses DFS search to effectively generate attack graphs. Authors first generate the graph from capability and location data. Then they reduce the graph by removing resources that are out of reach of the attacker. Then a DFS is run to identify all the paths from the locations of attacker can access. In [33] authors propose a model only with the path of the network nodes involved in the attack to be analyzed in detail. A network attack path detection model based on attack graph is also proposed. First, they formulate an attack graph and use it to describe the transfer relationship between nodes. They map the process of the attack from one host to the next host and discover the path to identify the attack intention. An MTD based defense mechanism is proposed in [34] for non-patchable vulnerabilities. They propose to change the attack surface of the IoT network to increase the attack effort. They develop two proactive defense mechanisms that reconfigure the SDN-based IoT network topology. In [35], authors propose a strategy to use a diverse set of security mechanisms, such that the impact from a vulnerability in any security mechanism is minimized. They introduce a game-theoretic graph coloring technique to get the optimal allocation of security mechanisms that minimizes the impact of security vulnerabilities to the power grid.

We discussed three types of existing systems: (1) statistical approaches that analyze packets or traffic properties to detect and block volumetric attack traffic (2) usage of machine learning to detect both volumetric and non-volumetric attack traffic and block, and (3) game theory based system that incorporate attack paths to defense non-volumetric at-

tacks. None of these system consider FR or MTD as defense mechanism and utilize them perfectly. Earlier works on volumetric attacks based on FR use tree based topologies which is rate nowadays. Therefore it is important to develop a system that can work on any kind o topology.

4.3 System Model

4.3.1 *Network Model*

Our network is composed of filter routers (FR), attackers, a defender, resources, and legacy routers (LR). The resources can be databases, files, and credential servers which are the most valuable resources that need high protection. The servers can be connected to any location of the network. A filter router is a special type of router which is capable of accepting filters and applying them to block some traffic [36]. A filter is a packet blocking rule based on source and destination IP address. The filter sent by the ISP is only applicable to the packets which are destined for the resources owned by that ISP. It is possible that an attacker spoofs the IP addresses of the ISP and sends the wrong filters to FRs so that legitimate traffic is blocked. This spoofed filter request can be detected using a simple handshaking protocol. The spoofing attacker is not capable of exchanging handshake messages with the spoofed IP address. The defender is responsible for deploying filters and the moving target defense (MTD) module to protect the resources owned by that ISP. An MTD module changes the configuration of some node or server periodically so that the attacker cannot succeed. The configuration changes while the attacker is in the progress of action so that it needs to start over. Therefore, when the MTD module is applied the attacker cannot succeed in that step of the attack. Each Filter or MTD module deployment incurs some cost to the ISP. Therefore, the ISP wants to deploy a limited number of filters and MTD modules while getting maximum protection.

4.3.2 *Attack Model*

We consider two types of attacks in our attack model: volumetric and non-volumetric attacks. In a volumetric attack, the attack traffic is not harmful but the amount of traffic harms the service. For example, DDoS attacks are volumetric because the packets fired by attackers are not harmful but the amount of traffic exhaust the capacity of the servers.

The attackers are usually user devices with malicious programs that can generate traffic as commanded by the master. The master can send different types of attack commands to the programs. This type of malicious program is called a bot, and a network of bots is called a botnet. The bots are capable of several types of attacks, such as SYN flooding, malformed packet attacks, UDP Flooding, amplification attacks, and password guessing attacks [37]. These attacks are classified into two classes: volumetric and non-volumetric attacks. In a non-volumetric attack, the amount of traffic is not important but the packets are harmful. The attackers are either program that may reside on compromised machines that can generate attack traffic destined for a target. The programs are usually controlled by a human attacker. Sometimes the human attacker can devise an attack based on vulnerabilities of a network. For example, a password guessing attack may produce a scanty amount of traffic but have the outcome of exposing the password database to the attackers. There are several types of non-volumetric attacks such as malware installation, administrator access acquiring and accessing private networks and data. These attacks are conducted in series for more optimistic goal. For example, an attacker trying to get data from a server that is connected to a private network. One simple way to do this is to gain access to the machine which is connected to both networks. This may include gaining root privilege of the intermediate machine which increases one more step to the goal. Then using that machine an attacker can gain access to the target machine.

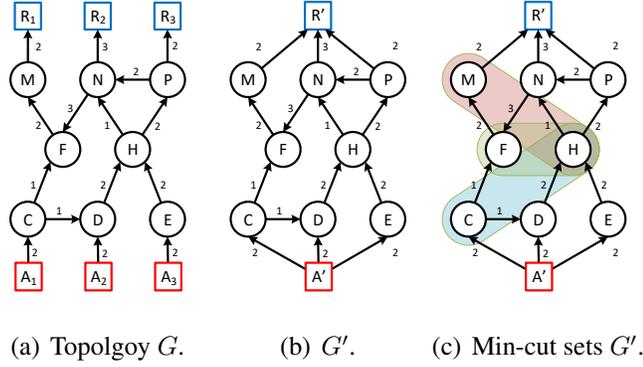


FIGURE 4.2: An example for volumetric attack.

4.3.3 Cost, Budget and Damage Model

The cost of defense and damages incurred by the attack are different for these two types of attack. In a volumetric attack, the cost is incurred by the filter assignment. The hosting internet service provider of FRs may charge money for applying filters. In reality, the cost for filters may vary for different ISPs but for simplicity, we assume a uniform cost of filters. We assume the network provider has a limited budget for the number of filters. We denote the budget of the service provider as K . Therefore, the service provider wants to minimize the damage incurred by the attack to the minimum.

As the traffic is benign (not intent to steal), the service provider can allow some of it pass through to the resource servers. The amount of damage is proportional to the amount of traffic received by the resources. So, the cost can be defined as $C = \sum_{r \in R} B(r)$ Here, the set of resources is R . $B(n)$ denotes the amount of attack traffic incoming to node n .

4.4 Volumetric Attack Problem Formulation

In this section, we formulate the problem of filter assignment so that traffic reaching the resources is minimal.

Problem I: Find K number of nodes to apply filters so that the traffic reaching the resources is minimum.

Let the topology be $G = (V, E)$ where V is the set of nodes and E is the set of links. Let the set of attacker and resources are A and R ($R \subset V$ and $A \subset V$). $B(n)$ denotes the amount of attack traffic of attack success through node n . Therefore, the problem can be expressed as the following:

$$\begin{aligned} & \text{minimize} && \sum_{r \in R} B(r) \\ & \text{subject to} && \sum_{n \in V} M(n) \leq K, \end{aligned} \tag{4.1}$$

Here, $M(n)$ is 1 if a MTD is applied on n , otherwise 0.

4.4.1 Solution

This problem is NP-Hard and we provide a greedy solution based on the min-cut problem. We first create $G' = (V', E')$ from the original topology G . In G' , we combine all of the attackers and create a super attacker A' . We also combine all the resources and create a super resource R' . Therefore, $V' = V \cup A' \cup R' - (A \cup R)$. Now, the problem can be viewed as a flow problem where we want to minimize the maximum flow by removing some nodes.

Next, we need to find all possible minimum cuts in the flow network. To find all possible min-cuts we adopt the Kanevsky [38] method. If the size of the cut set is equal or less than K , then any of the cut set is the best solution. If the size of the cut set is higher than K , then we use a greedy procedure to find the best K nodes. For each cut set, we calculate the maximum blockage for K nodes.

To find the maximum blockage for a cut set $S_c = n_1, n_2, \dots$ for K nodes, we calculate the contribution to max flows of each nodes in S_c . We calculate the maximum incoming and outgoing flows $\text{MAX-FLOW}(A, n_i)$ and $\text{MAX-FLOW}(n_i, R)$ to and from node n_i . The contribution of node n_i in the max-flow is the minimum of the incoming and outgoing flows. We choose the node with the maximum contribution to max-flow first. Then, the capacities of each link carrying the flows are reduced. Similarly, we calculate the contribu-

tions of the rest of the nodes and pick the one with the highest contribution. This process continues until we pick up K nodes. The complete algorithm is shown in Alg. 11.

4.4.2 An Example

Let us consider the example in Fig. 4.2. Fig. 4.2(a) shows the original topology G of the network. $R_1, R_2,$ and R_3 are the resources and they need to be protected as much as possible. $A_1, A_2,$ and A_3 are the attackers and they launch a DoS attack. The links are directed since we only consider the incoming traffics of the resources. The number beside each link shows the capacity of the links. Other nodes in the topology are considered as FRs. We transform the original topology G to G' by combining the resources and the attacker. Fig. 4.2(b) shows the G' . In G' , $A_1, A_2,$ and A_3 are combined to A' and the links are added with the corresponding capacities. Similarly, the resources $R_1, R_2,$ and R_3 are combined to R' . Next, we find all of the minimum cut sets in G' using the Kanevsky [38] method. We do not show the details of this process to save space. There are three minimum cut sets $\{C, H\}, \{C, AD\},$ and $\{H, M\}$. The cut sets are shown in Fig. 4.2(c).

If the maximum allowable number of filter (K) is 2, then any of the cut sets is the optimal filter assignment. In this case, no attack traffic will reach the resources. Let us assume that $K = 1$. Now we need to find the maximum blockage for each cut sets for one blockage. For $\{C, H\}$, if we remove H from G' , then the maximum flow going through C is 1. Therefore, the contribution of node C is 1. Similarly, if we remove C from G' , the maximum flow going through H is 3. Therefore, the contribution of node H is 3. We select the best 1 node ($K = 1$), which is H . Therefore, the maximum blockage for $\{C, H\}$ is 3 for $K = 1$. Similarly, the maximum blockage of $\{C, D\}$ and $\{H, M\}$ cut sets is 3. Therefore, we pick $\{C, H\}$, and the filter needs to be installed at node H . The amount of traffic reaching the resources after installing the filter is 1.

Theorem 9. *The complexity of Alg. 11 is $O(|S_c||V|(|V| + |E|f))$.*

Proof. To calculate the complexity of Alg. 11, we need to calculate complexity of $\text{MAXBLOCKAGE}(S_c, K)$. According to [38], Step 9 takes $O(\kappa(|V| + |E|))$, where κ is the connectivity of graph G . Step 12 and 13 take $O(|E|f)$. Here, f is the maximum attack traffic flow in the network. The loop at Step 10, takes $|V||E|f$. Here, $|V|$ is the maximum number of nodes in a cut sets. Therefore, $\text{MAXBLOCKAGE}(S_c, K)$ takes $O(\kappa(|V| + |E|) + |V||E|f)$. In the worst case, the connectivity can be $|V|$. The complexity of $\text{MAXBLOCKAGE}(S_c, K)$ is $O(|V|^2 + |V||E| + |V||E|f)$ which is $O(|V|(|V| + |E|f))$. Therefore, the Alg. 11 takes $O(|S_c||V|(|V| + |E|f))$. \square

Theorem 10. *The approximation ratio of Alg. 11 is $(1 - 1/e)$*

Proof. To find the approximation ratio of Alg. 11, we need to find the approximation ratio of procedure $\text{MAXBLOCKAGE}(S_c, K)$. The procedure picks the best node having maximum max flow by deleting the other cut nodes. When we remove the other cut nodes, all the traffic passes through that node. Then we reduce the capacity of the link passing traffic. This process is similar to the solution of the greedy maximum coverage problem. If we consider each node in the cut set as a set and each link on each distinct path with a unit amount of flow as an element of the set represented by the node, then the problem is to find K sets with maximum elements. The approximation ratio of the maximum coverage problem is $(1 - 1/e)$. Therefore, the approximation ratio of the $\text{MAXBLOCKAGE}(S_c, K)$ is $(1 - 1/e)$. The rest of the parts of Alg. 11 search in all possible options. Therefore, the approximation ratio of Alg. 11 is $(1 - 1/e)$. \square

4.5 Non-volumetric Attack Problem Formulation

In this section, we formulate the problem of assigning the MTD methods to the machines so that no attack traffic can reach the resource servers.

Problem II: *Find K number of nodes to apply MTD so that the system is secured and the damage is the minimum.*

Algorithm 11 Greedy Blocking Strategy

Input: The number of filters K and topology graph G .

Output: A set of nodes in G .

```
1: Procedure: BLOCK( $K, G$ )
2:    $N \leftarrow$  number of nodes in  $G$ 
3:    $S \leftarrow$  All minimum cut sets.
4:   for  $S_c \in S$  do
5:      $M[S_c] \leftarrow$  MAXBLOCKAGE( $S_c, K$ ).
6:   return ARGMAX( $M$ )
7: Procedure: MAXBLOCKAGE( $S_c, K$ )
8:    $N \leftarrow$  number of nodes in  $G$ 
9:    $S \leftarrow$  All minimum cut sets.
10:  for  $n \in S_c$  do
11:     $G' \leftarrow$  Remove  $S_c - n$  nodes from  $G$ .
12:     $f_i \leftarrow$  MAX-FLOW( $A, n, G'$ ).
13:     $f_o \leftarrow$  MAX-FLOW( $n, R, G'$ ).
14:     $C[n] \leftarrow$  MIN( $f_i, f_o$ )
15:   $MaxC \leftarrow$  ARGMAX( $C$ )
16:   $MaxBlockage \leftarrow$   $MaxBlockage +$  MAX( $C$ )
17:  Reduce capacity of link carrying flows through  $MaxC$ 
18:  return  $MaxBlockage$ .
```

Let, the topology is $G = (V, E)$ where V is the set of nodes and E is the set of links. Let the set of attacker and resources are A and R . The set of nodes that can be chased by an attacker is D . Therefore, the problem can be expressed as the following optimization problem:

$$\begin{aligned} & \text{minimize} && |D| \\ & \text{subject to} && \sum_{n \in V} M(n) \leq K, \\ & && \sum_{r \in R} B(r) = 0 \end{aligned} \tag{4.2}$$

Here, $M(n)$ is 1 if a MTD is applied on n , otherwise 0.

4.5.1 Solution

We solve the problem using dynamic programming. To solve the problem, we define the following problem:

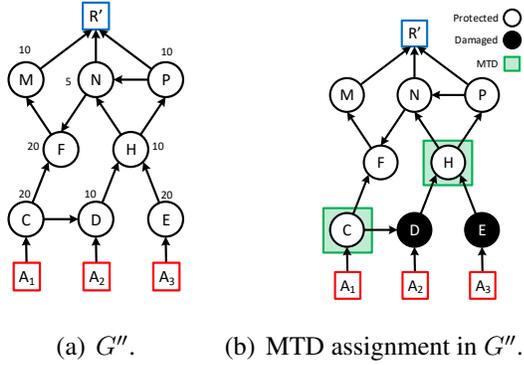


FIGURE 4.3: An example for non-volumetric attack.

D[R']			D[M]			D[F]			D[N]			D[P]			D[H]			D[D]			D[E]			D[C]					
k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1
0	-	105	0	-	105	0	-	95	0	-	75	0	-	70	0	-	60	0	-	30	0	-	20	0	-	20	0	-	20
1	-	80	1	75	80	1	75	70	1	50	55	1	50	50	1	50	30	1	20	10	1	0	-	1	0	-			
2	30		2	30	50	2	30	40	2	20	35	2	20	30	2	20	20	2	0	20	2	0	-	2	0	-			

A[R']			A[M]			A[F]			A[N]			A[P]			A[H]			A[D]			A[E]			A[C]					
k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1
0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset
1	-	H	1	F	H	1	F	H	1	H	C	1	H	C	1	H	C	1	H	C	1	D	C	1	E	-	1	C	-
2	CH		2	CH	DE	2	CH	DE	2	DE	CE	2	DE	CE	2	D,E	CE	2	C,D	C	2	E	-	2	C	-			

FIGURE 4.4: Values of A and D for all nodes and k .

C[R']			C[M]			C[F]			C[N]			C[P]			C[H]			C[D]			C[E]			C[C]					
k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1
0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset	0	-	\emptyset
1	-	HPN	1	F	PN	1	F	PN	1	HPN	C	1	H	C	1	H	C	1	D	C	1	E	-	1	C	-			
2	CHPNF M		2	HPNF	CE	2	HPNF	DEHP N	2	DEHP N	CE	2	DEHP	CD	2	DEH	CE	2	C,D	C	2	E	-	2	C	-			

L[R']			L[M]			L[F]			L[N]			L[P]			L[H]			L[D]			L[E]			L[C]					
k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1	k	T	1
0	-	\emptyset	0	-	CDEH PNFM	0	-	CDEH PNFM	0	-	CDEHP N	0	-	CDEHP	0	-	CDEH	0	-	C,D	0	-	E	0	-	C			
1	CDEHP NFM	CDEF M	1	CDEHP N	CDEF M	1	CDEHP N	CDEF	1	CDE	DEHP N	1	CDE	DEHP	1	CDE	DEH	1	C	D	1	\emptyset	-	1	\emptyset	-			
2	DE		2	DE	CFM	2	DE	CF	2	C	DHPN	2	C	DHP	2	C	DH	2	\emptyset	D	2	\emptyset	-	2	\emptyset	-			

FIGURE 4.5: Values of C and L for all nodes and k .

$P(n, k, t)$: Find and return the minimum damage in the subgraph (DAG) rooted by n for k number of MTD by yielding ($k = 1$) or blocking all ($k = 0$) attack traffic. The optimal damage, MTD assignments, covered nodes, and damaged nodes are stored in $D[n, k, t]$, $A[n, k, t]$, $C[n, k, t]$, and $L[n, k, t]$ to reuse in dynamic programming.

There are two options to assign MTD. $P(n, k, t)$ is the minimum of the following two options:

Option I: The minimum total damage, if we assign 1 MTD to node n , divide the rest of the $k-1$ MTDs into $k_1, k_2, \dots, k_\Delta$ parts, assign the parts to the subgraphs $c_1(n), c_2(n), \dots, c_\Delta(n)$, and either block or partially block attack traffic, respectively. Therefore, the number of damaged nodes will be the union of the minimum damages for blocked/unblocked attack traffics in $c_1(n), c_2(n), \dots, c_\Delta(n)$. In this case, the MTD assigned to n blocks all of the attack traffic. Therefore, this option is applicable to $k = 0$ only. As there are overlap among the subgraphs, and if a node is covered by MTD assignment by a subgraph and not covered by other subgraphs, then the node will be covered in the current DAG.

$$L[n, k, 0] = \min_{\forall k_\delta, t} |\bigcup_{\delta=1}^{\Delta} P(c_\delta(n), k_\delta, t) - C[c_\delta(n), k, t]|$$

$$P(n, k, 0) = \sum_{n' \in L[n, k, t0]} V(n')$$
(4.3)

Here, $\forall_\delta \sum_{\delta=1}^{\Delta} k_\delta = K - 1$.

Option II: The minimum total damage, if we divide the number of MTDs into $k_1, k_2, \dots, k_\Delta$ parts, assign them to the subtrees $c_1(n), c_2(n), \dots, c_\Delta(n)$, and either block or partially block attack traffic, respectively. Therefore, the damage for this option will be:

$$L[n, k, t] = \min_{\forall k_\delta, t} |\bigcup_{\delta=1}^{\Delta} P(c_\delta(n), k_\delta, t) - C[c_\delta(n), k, t]|$$

$$P(n, k, t) = \sum_{n' \in L[n, k, t]} V(n') + tV(n)$$
(4.4)

Here, $\sum_{\delta=1}^{\Delta} k_\delta = K$. We take the minimum quantity from the above two options. Let us consider an N node DAG with the maximum node degree Δ . We define D as an $N \times K \times 2$ array that contains optimal damage for every node, budget, and blocked or unblocked attack traffic. For example, $D[n, k, 0]$ contains optimal damage in the sub DAG rooted by n of budget k by blocking all attack traffic.

Algorithm 12 DP MTD assignment strategy for Problem 2

Input: Budget on MTD K , and topology graph G'' .

Output: A set of nodes in G'' .

```
1: Procedure: ASSIGN-MTD-DP( $K, G$ )
2:    $N \leftarrow$  number of nodes in  $G''$ 
3:    $S \leftarrow$  topological order of nodes in  $G''$ 
4:   for every entry node  $n$  do
5:     for  $k = 0$  to  $K$  do
6:       Initialize  $D[n, k]$ ,  $T[n, k]$ , and  $A[n, k]$ 
7:     for every  $n \in S$  do
8:       for  $k = 0$  to  $K$  do
9:          $OP_1 \leftarrow D[n, k]$  using equation 4.3
10:         $OP_2 \leftarrow D[n, k]$  using equation 4.4
11:         $D[n, k] \leftarrow \text{MIN}(OP_1, OP_2)$ 
12:         $A[n, k] \leftarrow \text{ARGMIN}(A_{OP_1} \cup A_{OP_2})$ 
13:   return  $A[R', K]$ 
```

We define A as an $N \times K \times 2$ array which contains the MTD assignments in subgraph rooted by every node. We also define C and L as $N \times K \times 2$ arrays that contain the protected and damaged nodes in subgraph rooted by every node, respectively.

The complete algorithm is shown in Alg. 12.

4.5.2 An Example

Let us consider the attack path graph in Fig. 4.3(a). We compute the values of $D[n, k, t]$, and $A[n, k, t]$ for every node, $k = 0, 1$, and 2 and $t = 0$ and 1 . We are not showing details calculation of $C[n, k, t]$ and $L[n, k, t]$ because of the limited space. The values of $V(n)$ are given in the DAG in Fig. 4.3(a). We first find a topological order for the calculation. One of the topological orders of the DAG in Fig. 4.3(a) is $\{C, E, D, H, P, N, F, M, R'\}$.

The nodes C and E do not have any children. Therefore, the calculations of D and A are straightforward. For example, $D[C, 0, 0] = -$. Here “-” indicates an invalid option. This is because without any MTD, it is not possible to block all attacks C . If we assign a MTD to node C , then we are saving one node, thus $D[C, 1, 0] = 0$. Similarly, $D[C, 2, 0]$ is 0. If we want to yield attack traffic without any MTDs, then the node C gets damaged.

Therefore, $D[C, 0, 1] = 20$. If we assign any MTD to C , then we cannot yield any attack to its ancestors. Therefore, $D[C, 1, 1]$ and $D[C, 2, 1]$ is $-$.

For $k = 2$, we have two options for assigning the MTDs.

Option I: 1 MTD for node H and one MTDs for its subgraphs. We can assign 1 MTD to the subgraphs in two ways: $(k_1 = 1, k_2 = 0)$ or $(k_1 = 0, k_2 = 1)$.

For the first way, $(k_1 = 1, k_2 = 0)$, we assign the subgraph rooted by D and E to one MTD and zero MTDs. We can also consider the choices for $t = 0$ and $t = 1$. Therefore, we have eight choices for two ways (each way can be assigned in four ways).

Choice(1): $(k_1 = 1, k_2 = 0, t_1 = 0, t_2 = 0)$ The total lost nodes for this choice is $(L[D, 1, 0] \cup L[E, 0, 0]) \setminus (C[D, 1, 0] \cup L[E, 0, 0]) = (\{C\} \cup -) \setminus (\{D\} \cup -)$. Therefore, this choice is invalid.

Choice(2): $(k_1 = 1, k_2 = 0, t_1 = 0, t_2 = 1)$ The total lost nodes for this choice is $(L[D, 1, 0] \cup L[E, 0, 1]) \setminus (C[D, 1, 0] \cup L[E, 0, 1]) = (\{C\} \cup \{E\}) \setminus (\{D\} \cup \emptyset) = \{C, E\}$. Therefore, the damage for this choice is 40 and because of $t_2 = 1$, this choice will yield attack traffic.

Choice(3): $(k_1 = 1, k_2 = 0, t_1 = 1, t_2 = 0)$ The total lost nodes for this choice is $(L[D, 1, 1] \cup L[E, 0, 0]) \setminus (C[D, 1, 1] \cup L[E, 0, 0]) = (\{D\} \cup -) \setminus (\{C\} \cup -)$. Therefore, this choice is also invalid.

Choice(4): $(k_1 = 1, k_2 = 0, t_1 = 1, t_2 = 1)$ The total lost nodes for this choice is $(L[D, 1, 1] \cup L[E, 0, 1]) \setminus (C[D, 1, 1] \cup L[E, 0, 1]) = (\{D\} \cup \{E\}) \setminus (\{C\} \cup \emptyset) = \{D, E\}$. Therefore, the damage for this choice is 30 and because of $t_1 = 1$ and $t_2 = 1$, this choice will yield attack traffic.

Similarly, we can calculate the damages other choices: **Choice(5):** $(k_1 = 0, k_2 = 1, t_1 = 0, t_2 = 0)$, **Choice(6):** $(k_1 = 0, k_2 = 1, t_1 = 0, t_2 = 1)$, **Choice(7):** $(k_1 = 0, k_2 = 1, t_1 = 1, t_2 = 0)$, and **Choice(8):** $(k_1 = 0, k_2 = 1, t_1 = 1, t_2 = 1)$.

Option II: We assign 0 MTDs to node H and rest of the MTDs to its subgraphs. We have three ways to assign MTDs to its subgraphs: $(k_1 = 2, k_2 = 0)$, $(k_1 = 1, k_2 = 1)$, or

($k_1 = 0, k_2 = 2$). For each way we need to consider four choices. Because of the limited space we are only showing the choice that produce minimum damage. The choice ($k_1 = 1, k_2 = 1, t_1 = 0, t_2 = 0$) will produce the minimum damage by yielding no attack traffic. The total lost nodes for this choice is $(L[D, 1, 0] \cup L[E, 1, 0]) \setminus (C[D, 1, 0] \cup L[E, 1, 0]) = (\{C\} \cup \emptyset) \setminus (\{D\} \cup \{E\}) = \{C\}$. Therefore, the damage for this choice is 20 and because of $t_1 = 0$ and $t_2 = 0$, this choice will yield no attack traffic. Therefore, $A[H, 2, 0] = 20$ and $A[H, 2, 0] = (A[D, 1, 0] \cup A[E, 1, 0]) = \{D, E\}$.

The choice ($k_1 = 1, k_2 = 1, t_1 = 1, t_2 = 0$) will produce the minimum damage by yielding attack traffic. The total lost nodes for this choice is $(L[D, 1, 1] \cup L[E, 1, 0]) \setminus (C[D, 1, 1] \cup L[E, 1, 0]) = (\{D\} \cup \emptyset) \setminus (\{C\} \cup \{E\}) = \{D\}$. Therefore, the damage for this choice is $10 + 10 = 20$ as the attack traffic damages node H ($V(H) = 10$). Therefore, $A[H, 2, 1] = 20$ and $A[H, 2, 1] = (A[D, 1, 1] \cup A[E, 1, 0]) = \{C, E\}$. Similarly, we calculate the rest of the values in the tables. The complete tables is shown in Fig. 4.4. According to the table, if we want to assign two MTDs, then the best location for applying the MTDs are on nodes C and H .

Theorem 11. *The complexity of Alg. 12 is $O(|V|^2 K^2 \Delta)$.*

Proof. In Alg. Step 4 to 6 take $O(K|V|)$ to initialize the values of D , T , and A . Steps 9 and 10 take $O(\Delta|V|K)$ in the worst case because they needs to compute the union of sets and the maximum size of the can be $|V|$. Therefore, Steps 7 to 12 dominates the complexity of the algorithm. Steps 7 to 12 take $O(|V|^2 K^2 \Delta)$. Therefore, the complexity of the algorithm is $O(|V|^2 K^2 \Delta + K|V|)$ which is $O(|V|^2 K^2 \Delta)$. \square

Theorem 12. *Alg. 12 is produces an optimal MTD assignment.*

Proof. Alg. 12 uses a dynamic programming bottom-up strategy to search the optimal assignment. For a one-node DAG, if the node color is attached to the attacker, then there is no solution for $K = 0$ and $T = 1$. This is because, without any MTD, the attack

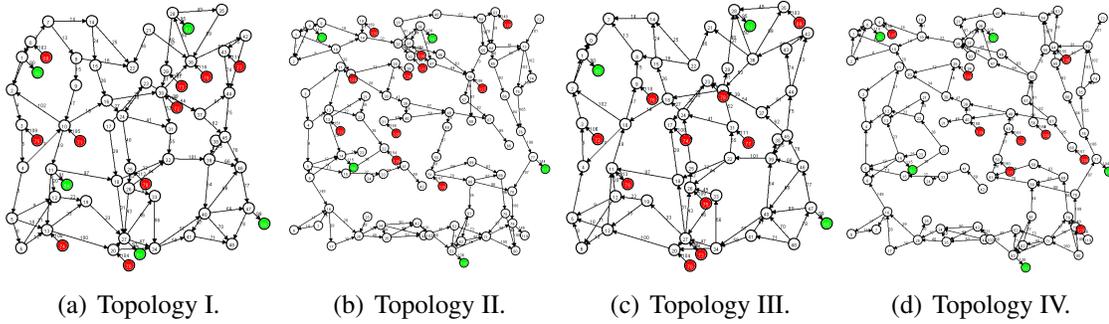


FIGURE 4.6: Randomly generated topologies.

Table 4.1: Topology Parameters

	Topology I & III	Topology II & IV
Number of nodes	68	129
Number of edges	103	149
Number of attackers	15	32
Number of resources	5	5

will be succeeded in the next step. For $K \geq 1$, there is only one choice for selecting MTDs, which is that node. If that node is selected, the attack is stopped and the number of damaged nodes is 0. In each step, Alg. 12 chooses the best allocation of MTDs to itself or the sub-DAGs. Therefore, Alg. 12 provides an optimal MTD assignment to the nodes through an exhaustive search. \square

4.6 Simulation

4.6.1 Simulation Settings

We built a java simulator to conduct all of the simulations. We want to count the amount of damage and attack traffic reaching the victim for the different settings. We do not need to analyze the real transmission time, link bandwidth, congestion, or packet drop scenarios. Besides the topologies, in this simulation we consider contain hundreds of nodes, links, resources, and attackers. The NS3 or other similar simulators would take a long time to produce results compared to our java simulator.

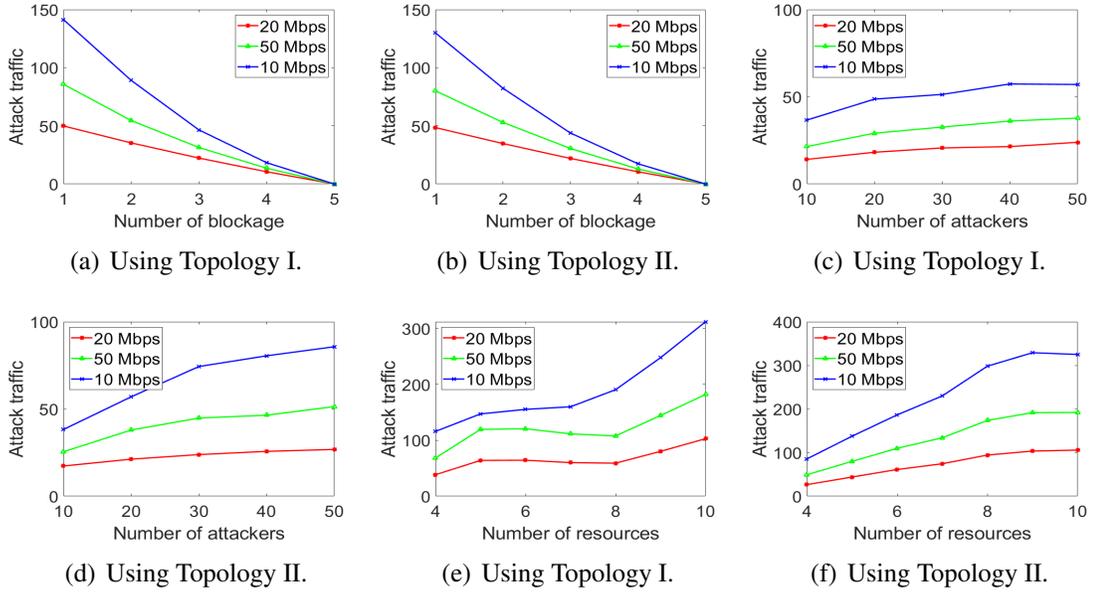


FIGURE 4.7: Simulation results of Problem 1.

We use randomly generated topologies for the simulations. We first divide an area of 500×500 square units into 50×50 blocks. A certain number of nodes are placed at random locations in each block. We limit the minimum distance between a couple of nodes. Then, the edges are generated based on the distance between the nodes. When a node is within a certain distance of another node, we add an edge between them. Then a few edges are added by picking up a pair of nodes randomly. Finally, we connect a certain number of resources and attackers to some of the randomly selected nodes. We set the remaining capacities of each edge randomly from a range. The minimum remaining capacity of a link is set to 10 Mbps and different maximum capacity is set for different simulations. In a real-world network, the link maximum capacity varies (100Mbps/1Gbps/10Gbps), and other flows use up some of the link capacities. We do not simulate the other flows so that we set the remaining link capacities randomly. If the link capacity is higher, then the amount of traffic arriving at resources and damages are higher. Topologies I and II are randomly generated with 68 and 129 nodes. Topology I smaller and contains 103 edges and 15 attackers. Topology II relatively large and contains 149 edges and 32 attackers.

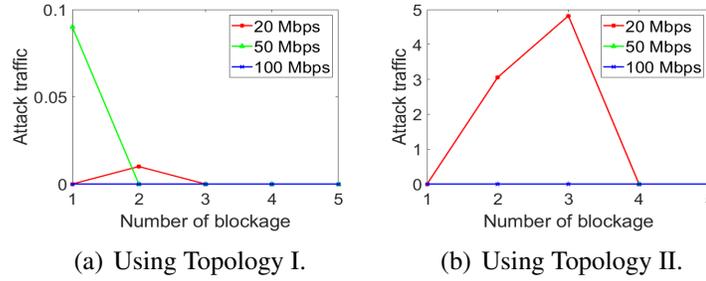


FIGURE 4.8: More simulation results of Problem 1.

Simulations related to problem 1 is conducted using Topologies I and II. Topologies III and IV are generated from topologies I and II, respectively. The directions of the links are changed to ensure no cycle in the topologies. Simulations related to problem 2 are conducted using Topologies III and IV. The details are shown in Table 4.1 and Fig. 4.6.

We measure the performances of our proposed solutions in terms of received attack traffic (RAT) and the amount of damage for different numbers of filters, MTDs, attackers, and resources. We compare the result of the first problem with the optimal solution. The optimal solution is obtained using the brute-force method. As the solution to the second problem is optimal we do not need to compare it with other works. All of the results presented in plots are average of 1000 runs.

4.6.2 Simulation Results of Volumetric Attack

We first conduct a simulation to measure the amount of attack traffic for different numbers of filters. Fig. 4.7(a) shows the amount of attack traffic received by the resources in Topology I. We vary the number of filters from 1 to 5 and keep the number of attackers and resources as in the original topology. For 20 Mbps maximum remaining link capacity, if the number of filters is 1, the amount of RAT is 49.90 Mbps. If the number of filters is 4, the amount of RAT is 10.54 Mbps. For the increase of 3 filters, the attack traffic reduces about 78%. For a higher maximum remaining link capacity, we observe a higher amount of attack traffic received by the resources. For 100 Mbps maximum remaining link capacity, if the number of filters is 1, the amount of RAT is 141.33 Mbps. If the number

of filters is 4, the amount of RAT is 18.345 Mbps. For the increase of 3 filters, the attack traffic reduces about 87%. When there are 5 filters, all of the attack traffic is blocked. The amount of attack traffic decreased almost linearly with the increase in the number of filters.

Figs. 4.7(b) shows the amount of attack traffic received by the resources in Topology II. We keep the same settings as before for this simulation. For 20 Mbps maximum remaining link capacity, if the number of filters is 1, the amount of RAT is 49.90 Mbps. If the number of filters is 4, the amount of RAT is 10.54 Mbps. For the increase of 3 filters, the attack traffic reduces about 78%. For higher maximum remaining link capacity we observe a higher amount of attack traffic received by the resources. For 100 Mbps maximum remaining link capacity, if the number of filters is 1, the amount of RAT is 141.33 Mbps. If the number of filters is 4, the amount of RAT is 18.345 Mbps. For the increase of 3 filters, the attack traffic reduces about 87%. When there are 5 filters, all of the attack traffic is blocked. The amount of attack traffic decreased almost linearly with the increase in the number of filters.

Next, we vary the number of attackers to measure the amount of attack traffic. Fig. 4.7(c) shows the amount of attack traffic received by the resources in Topology I for different numbers of attackers. We vary the number of attackers from 10 to 50 and keep the number of filters at 3. For 20 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of RAT is 14.11 Mbps. If the number of attackers is 50, the amount of RAT is 23.9 Mbps. For the increase of 40 attackers, the attack traffic increases about 69%. For higher maximum remaining link capacity, we also observe a higher amount of attack traffic received by the resources. For 100 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of RAT is 36.65 Mbps. If the number of attackers is 50, the amount of RAT is 57.06 Mbps. For the increase of 40 attackers, the attack traffic increases about 35%. The amount of attack traffic increases almost linearly with the increase in the number of attackers.

Fig. 4.7(d) shows the amount of attack traffic received by the resources in Topology II for a different number of attackers. We keep the same settings as the previous simulation. For 20 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of RAT is 17.32 Mbps. If the number of attackers is 50, the amount of RAT is 26.82 Mbps. For the increase of 40 attackers, the attack traffic increases about 54%. For higher maximum remaining link capacity, we also observe a higher amount of attack traffic received by the resources. For 100 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of RAT is 38.27 Mbps. If the number of attackers is 50, the amount of RAT is 85.68 Mbps. For the increase of 40 attackers, the attack traffic increases about 12.3%.

After that, we vary the number of resources and measure the amount of attack traffic. Fig. 4.7(e) shows the amount of attack traffic received by the resources in Topology I for different numbers of resources. We vary the number of resources from 4 to 10 and keep the number of filters at 3. For 20 Mbps maximum remaining link capacity, if the number of resources is 4, the amount of RAT is 38.44 Mbps. If the number of resources is 10, the amount of RAT is 102.85 Mbps. For the increase of 6 resources, the attack traffic increases about 16%. For 100 Mbps maximum remaining link capacity, if the number of resources is 4, the amount of RAT is 115.86 Mbps. If the number of resources is 10, the amount of RAT is 311.79 Mbps. For the increase of 6 resources, the attack traffic increases about 17%. The amount of attack traffic increases with the increase in the number of resources.

Fig. 4.7(f) shows the amount of attack traffic received by the resources in Topology II for different numbers of resources. For 20 Mbps maximum remaining link capacity, if the number of resources is 4, the amount of RAT is 26.55 Mbps. If the number of resources is 10, the amount of RAT is 105.74 Mbps. For the increase of 6 resources, the attack traffic increases about 30%. For 100 Mbps maximum remaining link capacity, if the number of resources is 10, the amount of RAT is 85.48 Mbps. If the number of resources is 50, the amount of RAT is 324.99 Mbps. For the increase of 6 resources, the attack

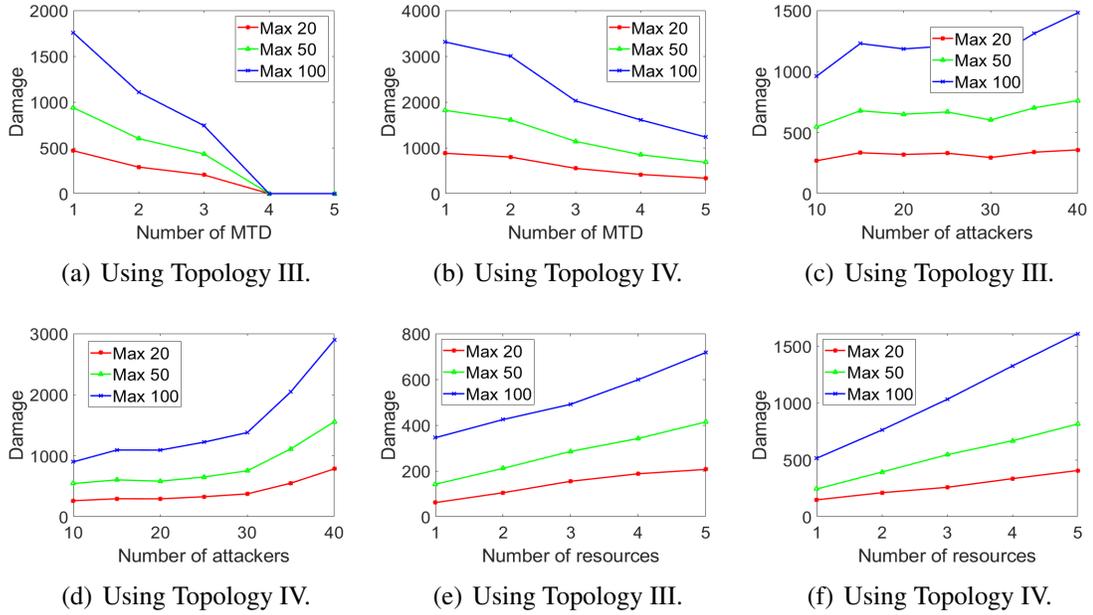


FIGURE 4.9: Simulation results of Problem 2.

traffic increases about 28%. The amount of attack traffic increases with the increase in the number of resources.

Finally, Figs. 4.8(a) and 4.8(b) show the difference in attack traffic between the optimal and our approaches. We can observe that most of the cases our proposed approach produce optimal result. Few cases shows little higher than optimal for 20 and 50 Mbps max remaining link capacity in topology I. In topology II, only 20 Mbps remaining link capacity shows higher than optimal. Compared to the total attack traffic the amount of difference is negligible.

4.6.3 Simulation Results of Non-volumetric Attack

We conduct a simulation to measure the damage corresponding to different numbers of MTDs. Figs. 4.9(a) shows the damage caused by the attackers in Topology III. We vary the number of MTDs from 1 to 5 and keep the number of attackers and resources the same as in the original topology. For 20 Mbps maximum remaining link capacity, if the number of MTDs is 1, the amount of damage is 467.66. If the number of MTDs is 4, the amount

of damage becomes 0. For higher maximum remaining link capacity we observe higher damage caused by the attackers. For 100 Mbps maximum remaining link capacity, if the number of MTDs is 1, the amount of damage is 1755.3. If the number of MTDs is 4, the amount of damage also becomes 0. When the number of MTDs is more than 4, all of the attack traffic is blocked and no damage caused.

Figs. 4.9(b) shows the damage caused by the attackers in Topology IV. We keep the same settings as before for this simulation. For 20 Mbps maximum remaining link capacity, if the number of MTDs is 1, the amount of damage is 879.96. If the number of MTDs is 5, the amount of damage is 336.33. For the increase of 4 MTDs, the damage reduces about 61%. For higher maximum remaining link capacity we observe higher damage caused by the attackers. For 100 Mbps maximum remaining link capacity, if the number of MTDs is 1, the amount of damage is 3309.9. If the number of MTDs is 5, the amount of damage is 1235.26. For the increase of 4 MTDs, the damage reduces about 63%.

Next, we vary the number of attackers to measure the damage. Fig. 4.9(c) shows the damage caused by the attackers in Topology III for different numbers of attackers. We vary the number of attackers from 10 to 40 and keep the number of MTDs as 3. For 20 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of damage is 269.3. If the number of attackers is 40, the amount of damage is 357.83. For the increase of 30 attackers, the attack traffic increases about 32%. For higher maximum remaining link capacity, we also observe higher damage caused by the attackers. For 100 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of damage is 962.2. If the number of attackers is 40, the amount of damage is 1479.46. For the increase of 30 attackers, the attack traffic increases about 53%. The damage increases about linearly with the increase in the number of attackers.

Fig. 4.9(d) shows the damage caused by the attackers in Topology IV for different numbers of attackers. We keep the same settings as the previous simulation. For 20 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of dam-

age is 262.16. If the number of attackers is 40, the amount of damage is 784.06. For the increase of 30 attackers, the damage increases about 200%. For higher maximum remaining link capacity, we also observe higher damage caused by the attackers. For 100 Mbps maximum remaining link capacity, if the number of attackers is 10, the amount of damage is 899.13. If the number of attackers is 40, the damage is 2893.26. For the increase of 30 attackers, the damage increases about 220%. Because of being a larger topology, topology IV gets higher damage than topology III for higher number of attackers.

After that, we vary the number of resources and measure the damage in the network. Fig. 4.9(e) shows the damage caused by the attackers in Topology III for different numbers of resources. We vary the number of resources from 1 to 5 and keep the number of MTDs as 3. For 20 Mbps maximum remaining link capacity, if the number of resources is 1, the amount of damage is 61.76. If the number of resources is 5, the amount of damage is 206.6. For the increase of 4 resources, the damage increases about 237%. For 100 Mbps maximum remaining link capacity, if the number of resources is 1, the amount of damage is 345.4. If the number of resources is 5, the amount of damage is 717.0. For the increase of 4 resources, the attack traffic increases about 107%. The damage increases with the increase in the number of resources.

Fig. 4.9(f) shows the damage caused by the attackers in Topology IV for different numbers of resources. For 20 Mbps maximum remaining link capacity, if the number of resources is 1, the amount of damage is 148.56. If the number of resources is 5, the amount of damage is 406.53. For the increase of 4 resources, the attack traffic increases about 175%. For 100 Mbps maximum remaining link capacity, if the number of resources is 1, the amount of damage is 515.06. If the number of resources is 5, the amount of damage is 1611.33. For the increase of 4 resources, the attack traffic increases about 212%. The damage increases with the increase in the number of resources.

Therefore, from the above simulation results we can conclude that the proposed filter assignment approach for volumetric attack performs nearly optimal. The MTD assignment

approach for non-volumetric attack depends on the parameters which need to be adjusted based on network properties.

4.7 Summary

Due to increased threats on the internet, it is important to protect the network-connected resources such as web, database, and file servers. We have considered two types of attacks with different objectives for attackers and formulate the defense mechanism as optimization problems. We considered that the amount of attack traffic is proportional to the damage from the volumetric attacks. We provide an approximation filter assignment solution to block the maximum amount of attack traffic with a limited number of filters. The amount of attack traffic is not related to the damage caused by the non-volumetric attack. We considered that the steps passed by the attacker are equal to the damage. We propose a dynamic programming-based optimal solution to assign a moving target defense approach to some nodes. We conducted extensive simulations to observe behaviors of the defense mechanisms for different settings. We also observe that the approximation solution is very close to the optimal solution. Upto this chapter we have proposed multiple solutions for defending regular DDoS and non-volumetric attacks. In the next chapter we are going to extend our work for transit-link DDoS attacks.

CHAPTER 5

OPTIMAL FILTER ASSIGNMENT POLICY AGAINST TRANSIT-LINK DISTRIBUTED DENIAL-OF-SERVICE ATTACK

In this chapter, we propose a transit-link DDoS attack mitigation technique by using filter. We formulate two optimization problems for selecting the minimum number of possible congested links so that the legitimate traffic goes through a non-congested path. We consider the scenario where every user has at least one non-congested shortest path in the first problem. We extend the first problem to a scenario where there are some users whose shortest paths are all congested. The research in this chapter has been partially published in [Related10] and under revision in [Submitted2].

5.1 Introduction

The growing amount of distributed denial-of-service (DDoS) attacks is a threat to critical services on the Internet. The primary objective of a DDoS attack is to generate a lot of packets from different workers to exhaust incoming/outgoing bandwidth or resources used by the victim. A coordinator would send commands to workers who continue to send requests to the target. The workers are known as bots and the network of workers is known as botnet. Bots are usually computers that are infected with malicious program. Because

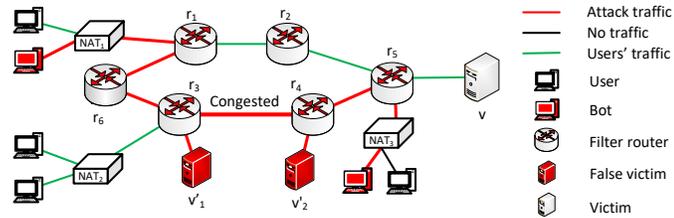


FIGURE 5.1: Transit-link DDoS attack by bots.

of the huge amount of traffic or service requests a machine or network resource becomes temporarily unavailable to its users.

DDoS attacks have evolved a lot during the last decade. The cost of conducting an attack has also decreased. For example, renting 1000 bots from any location costs from a few dollars to little more than 100 USD [2]. Attackers' strategies, capabilities, and goals have also evolved a lot during these years. For example, an attack against Spamhaus in 2013 revealed that the attacker can change the target adaptively from the endpoint server according to the defense mechanism. Another attack against ProtonMail in 2015 demonstrated that the attackers changed their strategy in real-time according to the defense mechanism [3]. The moving target and attack strategy form an interactive game scenario between the defender and the attacker.

The transit-link DDoS attack using botnet is one of the most challenging DDoS attacks. The strategy and goal are slightly different from the traditional DDoS attack. In the traditional DDoS attack, the bots generate packets destined to the victim. In transit-link DDoS attack, the bots generate traffic that is not destined to the victim but the packets congest at least one of the links on the paths from the legitimate users to the victim. Because of the congested links, the legitimate traffic suffers from packet drop and delay. The transit-link DDoS can be so devastating that it can disconnect a server from the Internet.

Fig. 5.1 shows a scenario of a transit-link DDoS attack. The bots attached to NAT_1 generate huge traffic towards the false victim server v'_2 . False victim servers are used to receive the attack packets from bots. False victim servers may be owned by the attacker

and the attack packets congest one or more links on the path to false victims. The packets follow the path $r_1 \leftrightarrow r_6 \leftrightarrow r_3 \leftrightarrow r_4$. Similarly, the generated traffic from the bots attached to NAT_3 follows the path $r_5 \leftrightarrow r_4 \leftrightarrow r_3$ to reach v'_1 . The link $r_3 \leftrightarrow r_4$ becomes congested because of the traffic from bots from NAT_1 and NAT_2 . When the legitimate packets from the user attached to NAT_2 travel the shortest path $r_3 \leftrightarrow r_4 \leftrightarrow r_5$, they face delay, packet drop, and DoS. On the other hand, legitimate traffic from users attached to NAT_1 and NAT_3 do not encounter any congested links and the victim observes good traffic rate from them. Because of low data rates from users attached to NAT_3 , the victim assumes that either any or both of the $r_3 \leftrightarrow r_4$ and $r_4 \leftrightarrow r_5$ links are congested. The victim can disable a link for the packets destined to him by sending a filter to the corresponding filter router (FR). The FRs are special kinds of routers. In addition to the routing task, they do two operations: packet marking and traffic filtering. The packet marking operation probabilistically appends FR's own IP address to the packets it forwards. The marked packets are used to identify paths traveled by a users' packets. The victim can obtain traffic topology by combining the paths from all sources. Traffic filtering is associated with two operations: receiving filters from victims and applying the filters to block or modify the routes that packets follow to the victim. The concept of FR is not new and well studied in [39, 40, 36] for preventing DDoS attacks, but it has never been studied for transit-link DDoS attacks.

Assigning a filter to a FR has a cost. The costs include monetary and communications overhead. For example, if the victim and FR belong to different ISP, then ISP of FR can earn by accepting filters from victim. On the other hand, the victim improves its QoS by changing user's route by deploying filters. This way the victim and owner ISP of FR both get benefited. Therefore, for the victim, it costs a lot to disable all the possible congested links. In this paper, we focus on directing all legitimate traffic through non-congested paths with a minimum number of filters. The main contributions of our paper are:

1. We formulate an optimization problem for directing legitimate traffic through the alternate shortest non-congested paths with the minimum number of filters/blockage.
2. We propose an optimal solution in polynomial time by narrowing down the problem to the vertex separation problem.
3. We formulate another optimization problem for the scenario where there are no alternate non-congested shortest path from the users. We solve the problem narrowing down the problem to the first problem.
4. We provide extensive simulation with synthetic and real topology and compare it with existing approaches to validate our model.

The remainder of this paper is arranged as follows: Section 5.2 presents some related work and their limitations. In Section 5.3, we present the system model for preventing transit-link DDoS attacks. In Section 5.4 and 5.5, we formulate a problem in a special scenario where there is a non-congested shortest path from every user, and we provide a solution. In Section 5.6 and 5.7, we extend the problem to to a more general scenario and present an optimal solution. In Section 5.8, we present the simulation results. Section ?? describes some limitations and future works.

5.2 Related Work

There exist many works defending transit-link DDoS attacks. Many statistical methods, including correlation, entropy, covariance, divergence, cross-correlation, and information gain have been used for network traffic analysis to detect anomalous traffic [8]. There are many multi-path routing mechanisms that help to mitigate congestion, including [41, 42, 43, 44, 45]. These schemes do not give priority to legitimate traffic and the destination machine does not have any control over it.

There are several methods that employ routers to identify attack traffic and block them. The SPIFFY [28] logically increases link capacity when it detects congestion. After the

capacity increases, the attacker may increase the data rate of each bot to keep the cost constant and be identified. The ColDef [30] enables routers to distinguish low-rate attack flows from legitimate flows. In this mechanism, the domains which are uncontaminated by botnet help route the legitimate traffic. In [29], the authors propose PUSHBACK, a router functionality based mechanism which enables each router to detect and preferentially drop packets that likely belong to an attacker. The upstream routers are also notified of the drop event so that they can use the resources for legitimate traffic. In [46], the authors propose a router subsystem called FLoc (Flow Localization) that provides differential bandwidth guarantee to legitimate traffic at congested links. The FLoc distinguishes between the traffic from a bot-contaminated and uncontaminated domains. Legitimate traffic from uncontaminated domains have more homogeneous congestion characteristics - including mean packet drop rates and mean delay - than flows of contaminated domains. Based on the traffic characteristics, FLoc determines how much bandwidth to allocate to the traffic from a domain.

In [31], authors proposed a scheme which mitigates transit-link DDoS by using some BGP rules in BGP routers. When a link congestion is detected, the BGP router advertises its neighbors in such a way so that the congested link is avoided. When a BGP router detects congestion, it creates a false path by appending its own address and advertises the false path. Other BGP routers from different autonomous systems find that the path has a loop and avoid it.

We discussed two types of existing defense systems: (1) attack packet/flow detection at a router followed by packet drop, rate control, and redirection and (2) identification of congested links at a router or controller and avoiding the link. The first type increases router computation overhead and the false positive creates great loss to the victims. Besides, bots are smart enough to create traffic similar to users and are unidentifiable using statistical techniques. For the second type, the identification of the congested link for a victim far from the congested link is tough without additional access. The victim can dis-

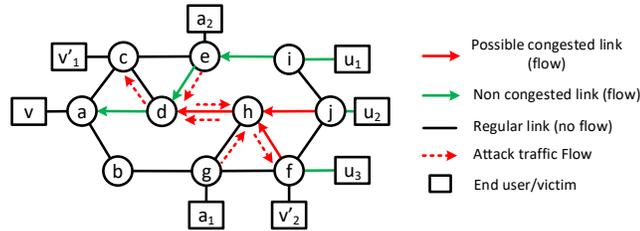


FIGURE 5.2: System model.

tinguish between the attacker and legitimate traffic. Therefore, a defense system in which routing is controlled by the victim is necessary.

The most related work preventing DDoS attacks is the probabilistic filter scheduling (PFS) system [39]. The victim generates filters and sends them to the upstream FRs. The FRs send the filters to its upstream FRs and thus the filters propagate to the effective FRs. An adaptive version of PFS is proposed in [40]. The system directly sends filters to the high capable FRs first, then the filters propagate to the effective FRs. However, these two systems cannot optimally select the FRs when there is a limitation on selecting FRs. It is not applicable for transit-link DDoS protection.

5.3 System Model

In this section, we define the network and the attack model.

5.3.1 Network Model

Our network is composed of network address translators (NATs), filter routers (FRs), bots, users, and a victim. We assume that all the users are connected to FRs through NAT. We also assume that there is at least one non-congested path from every user to the victim.

The FRs are a special kind of router capable of two functions. Firstly, they can mark packets that is used to construct traffic topology for the victim. Secondly, they can receive filters from the victim and apply the filters to block a link. A filter is a special instruction to a FR to block an incident link. When the victim v sends a filter to a FR, any packet destined to the v will be affected by the filter. For example, in Fig. 5.2, assume a filter

“block link $h \leftrightarrow d$ ” is sent to a FR h by v . The FR h may forward an incoming packet through link $h \leftrightarrow d$ if the destination is not v . Therefore, the attack traffic from a_1 heading to the false victim v'_1 is not blocked. If the packet is destined to v , then the FR will never forward the packet through the link $h \leftrightarrow d$. FR h will not forward user traffic from u_2 through $h \leftrightarrow d$.

Sometimes attacker can impersonate v and send filter to FRs but it is identifiable using a simple handshaking protocol. For example, an attacker may send a wrong filter by sending a filter request message as the victim. When a filter request comes to a victim, it will ask the sender whether it has sent the filter or not. The attacker will not get the verification message and will not be succeed in the filter spoofing attack.

We assume that the victim knows the network topology. The topology and traffic topology (the way the traffic comes to the victim) can be obtained from packet marking. From the packet arrival rate, the victim can identify the users that are suffering from congestion. It is hard for the victim to know which links are congested using the data rate of users. For example, in Fig. 5.2, the attackers congest the link $d \leftrightarrow h$. As a result, traffic from u_2 and u_3 suffer from congestion. The victim v observes that data rates from u_2 and u_3 are below the regular data rate. Therefore, one or multiple links on the path $a \leftrightarrow d \leftrightarrow h \leftrightarrow j$ and $a \leftrightarrow d \leftrightarrow h \leftrightarrow f$ may be congested. The victim also observes that the data rate from u_1 is good. Therefore, none of the links on $a \leftrightarrow d \leftrightarrow e \leftrightarrow i$ are congested. With this observation the victim concludes that $d \leftrightarrow h$, $h \leftrightarrow j$, or $h \leftrightarrow f$ links have possibility to be congested.

After finding the possible congested links, the victim wants his incoming traffic to avoid the possible congested links. A trivial solution is to block all possible congested links by sending filters to the FRs that are associated with them. For example, if v wants to disable link $d \leftrightarrow h$, it would send a filter “block $d \leftrightarrow h$ ” to FR h . Sending a filter to FR has a cost. There are too many FR routers in the network, and the victim wants to reduce the filtering cost. Moreover, there is no need to block all of the possible congested links. For example, if v blocks $h \leftrightarrow j$ and $f \leftrightarrow h$, then all legitimate traffic will follow

non-congested paths. Therefore, blocking $d \leftrightarrow h$ is unnecessary. The victim wants the legitimate traffic to follow the non-congested path by blocking a minimum number of links by spending the minimum amount of resources. The victim can run a centralized algorithm to find the minimum number of required blocking.

The attack traffic continues to follow the same path because the filters do not affect any packet that is not destined to the victim. Therefore, the system does not reduce network congestion by blocking attack traffic. It only forces the legitimate traffic to follow a non-congested path. Whenever we mention blocking a link, we refer to blocking only the victim's legitimate traffic through that link.

5.3.2 Attack Model

The attacker knows the topology of the network but does not know the traffic topology. The attacker knows the location of the victim and users. By analyzing the location of the victim and users, the attacker selects one or multiple links as a target. Target links are selected based on two principles. Firstly, the target links should be used by a reasonable number of users. Secondly, the target links should be reachable by the flow from bots to false victims. Then, based on the location of the target link, the attacker selects false victims and bots to generate traffic. The selection of a pair $\langle bot, false\ victim \rangle$ is done in such a way that the traffic from the bot to the false victim passes through one or multiple targeted links. The traffic generated by a small number of bots cannot congest a link, but when a large number of attack flows pass through a link it becomes congested. As a result, the critical traffics that are passing through the congested links suffer from packet drop, delay, and low data rate.

In Fig. 5.2, after analyzing the topology, location of the victim, and the users, the attacker finds that link $a \leftrightarrow d$ carries the most legitimate traffic but cannot be reached by the false victim and bots. The second most important link is $d \leftrightarrow h$ which carries two legitimate flows and is reachable by bots and false victim. Therefore, the attacker starts

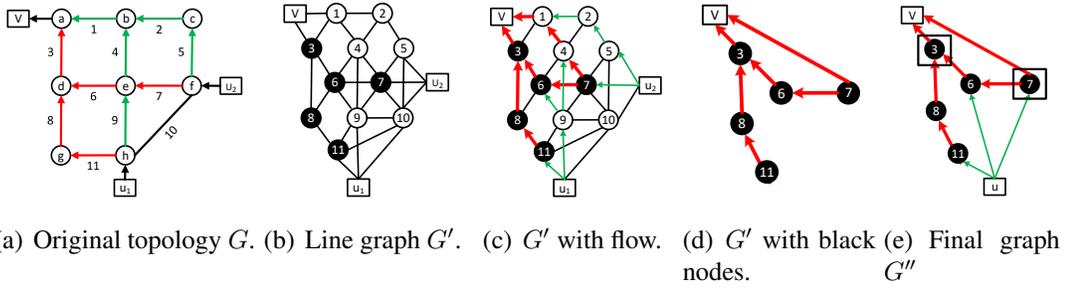


FIGURE 5.3: Problem I example.

Table 5.1: Table of notations

N	Number of nodes
v	Victim
B	Number of blocked edges
P_u^s	Set of the shortest paths from u to v
P_u	Set of paths from u to v
$P_u^s(B)$	Set of shortest paths from u to v after blocking B links
$G = (V, E)$	Topology graph
$G' = (V', E')$	Line graph
$G'' = (V'', E'')$	Congested flow graph
U	Number of users
e_{ij}	Edge between node i and j
L_c	Set of congested links
$C_e(e)$	Color of edge e
$C_n(n)$	Color of Node n
$C_f(f)$	Color of Flow f
$Neighbor(n)$	Neighbor set of node n

$\langle a_1, v'_1 \rangle$ and $\langle a_2, v'_2 \rangle$ flows by sending commands to a_1 and a_2 . $\langle a_1, v'_1 \rangle$ and $\langle a_2, v'_2 \rangle$ pass through paths $g \leftrightarrow h \leftrightarrow d \leftrightarrow c$ and $e \leftrightarrow d \leftrightarrow h \leftrightarrow f$. The traffic created by a bot cannot congest a link so that $c \leftrightarrow d$, $e \leftrightarrow d$, $g \leftrightarrow h$, and $f \leftrightarrow h$ are not congested. Link $d \leftrightarrow h$ becomes congested because both flows pass through it. As a result, legitimate traffic from u_2 and u_3 suffer from low data rates.

5.4 Problem Definition

In this section, we formulate the problem for finding a minimum number of links to block and provide an optimal solution. The notations used in this paper are summarized in Table 5.1. To solve the problem, we break the problem into two steps. In the first step, we assume that there exist multiple users and at least one shortest non-congested path from every user (defined as Problem I). In the second step, we extend the problem to solve the situation where there exists some entry node from which all of the shortest paths are possibly congested (defined as Problem II).

Problem I: *Find the minimum number of filter assignments so that all legitimate traffics follow non-congested shortest paths.*

In this problem, we assume that, from every user, there is at least one non-congested shortest path. Let, P_u^s be the set of the shortest paths from a user u to victim v . L_c is the set of possible congested links. Some of the paths in P_u^s contain possible congested links and are defined as possible congested paths. After deploying B number of filters to FRs in G , the set of the shortest paths from u is $P_u^s(B)$.

In this case, the problem can be expressed as the following optimization problem:

$$\begin{aligned}
 & \text{minimize} && B \\
 & \text{subject to} && \forall_{1 \leq u \leq U, p \in P_u^s(B)} \quad p \cap L_c = \emptyset, \\
 & && P_u^s(B) \subseteq P_u^s
 \end{aligned} \tag{5.1}$$

5.5 Optimal Solution for Problem I

Let us consider the topology in Fig. 5.3(a). The victim is connected to the FR a . A couple of users are connected to the FR h and FR f . The victim v observes that when traffic from u_1 follows the paths $h \leftrightarrow g \leftrightarrow d \leftrightarrow a$ or $h \leftrightarrow e \leftrightarrow d \leftrightarrow a$, the data rate falls below a threshold. Similarly, if the traffic from u_2 follows $f \leftrightarrow e \leftrightarrow d \leftrightarrow a$ the data rate falls below the threshold. On the other hand, the rate is good if the traffic from u_1 and u_2 follow $h \leftrightarrow e \leftrightarrow b \leftrightarrow a$ and

$f \leftrightarrow c \leftrightarrow b \leftrightarrow a$, respectively. From this observation, v finds out that links 3, 6, 7, 8, and 11 are possible congested links. Because of limited knowledge or access, the v cannot detect the actual congested links. A victim in an autonomous system might not know the status of a link in other autonomous systems. For example, this situation could happen if only links 3, 6, and 8, or 7 and 11 are congested. The v does not want to compromise data rates from its users and wants all the legitimate traffics coming to it to follow non-congested paths. We color the congested links red and non-congested links green. The black links do not carry any legitimate traffic and are assumed to be non-congested.

We divide the process of determining the minimum number of filter assignments into four steps:

- Step 1: Transformation: In this step, we transform the topology graph G to line graph G' .
- Step 2: Edge coloring: In this step, we create flows from each user which follow the shortest paths and we color the edges green or red.
- Step 3: Elimination: In this step, create graph G'' by eliminating white nodes.
- Step 4: Min separation: We apply the graph separation algorithm to find a min cut in G'' .

In Step 1, we transform the original graph $G = (V, E)$ to line graph $G' = (V', E')$. V is the set of FRs and E is the set of links in the topology. The edges in G become the nodes in G' . Two nodes are neighbors in G' if the corresponding links in G are adjacent. We color the corresponding node of a possible congested link as black and other nodes as white. In Fig. 5.3(a), link 6 is connected to d and e so that all the links which are connected to d $\{3, 8\}$ and e $\{4, 7, 9\}$ will be neighbor of 6 in G' . Therefore, node 6 is connected to 3, 8, 4, 7, and 9 in G' . In G , v is connected to a and a is connected with two other links $\{1, 3\}$, therefore v will be connected to node 3 and 1 in G' . Similarly, we add users to G' .

The color of nodes 3, 6, 7, 8, and 11 is black in G' because the links 3, 6, 7, 8, and 11 are possible congested G . Fig. 5.3(b) shows the line graph G' .

In Step 2, we create traffic flow from every user in G' . The flow travels through all the next hops remaining on the shortest paths to the victim. The flow continues and the traveled links are colored green. When a flow encounters a black node, the flow becomes red and continues coloring the traveled links red. If a link is already colored red, the color never changes, but if it is marked as green, then a red flow changes the link color to red. The flow terminates at the victim. The process is shown in Alg. 14. In Fig. 5.3(c) the green flows travels nodes 11 and 9 which remain on the shortest paths from u_1 to v . The edges $u_1 \leftrightarrow 11$ and $u_1 \leftrightarrow 9$ become green. Since, 11 is a black node, the flow becomes red and the edge $11 \leftrightarrow 8$ becomes red. The process continues until all the flows reaches v .

In Step 3, we create another graph $G'' = (V'', E'')$ with the victim node, red links, and nodes which are endpoints of the red links. Then, we eliminate the white nodes from G'' . When we eliminate a white node we connect its the incoming neighbors (who are sending flow to the white node) with the outgoing neighbors (who are receiving flows from the white node). Alg. 15 shows this step in details. For example, when we eliminate node 1 we connect v and 4 by $4 \leftrightarrow v$. Similarly, we connect v and 7 by $7 \leftrightarrow v$ when we remove node 4. We create a virtual user u and edges from u to any black node that is connected to at least one green edge. Any black node that appears first on the shortest path from any user to the victim has a green incident edge. According to Fig. 5.3(c), node 11, 6, and 7 will be connected to the virtual source. Now G'' contains only source, victim, and black nodes. Graph G'' represents all the congested shortest path flow from every user to the victim. Fig. 5.3(e) shows G'' .

In Step 4, a minimum number of nodes are selected to separate the victim and virtual user in G'' using Acid and Campos's minimum d-separating set [47]. When we are separating the victim and the virtual source, we are actually cutting all the shortest possible congested paths with a minimum number of link blockage. Applying Acid and Campos's

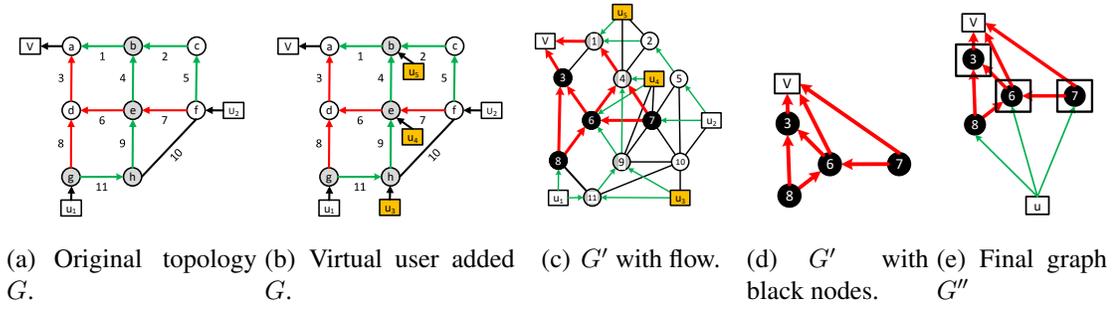


FIGURE 5.4: Problem II example.

method we find the minimum separation set $\{3, 7\}$. Therefore, if we block the links 3 and 7 in G by sending filters to FR d and f , no legitimate traffic will follow any congested path. The complete procedure is shown in Alg. 13.

Theorem 13. *Alg. 13 finds optimal number of links to block.*

Proof. Alg. 13 transforms the original topology to a flow graph where all shortest congested flows are considered. According to the assumption there is at least a non-congested shortest path. The flows travel one of the non-congested shortest paths after disabling all the possible congested paths. To prove the optimality of the Alg. 13, we need to prove that G'' contains all the possible congested flows. Assume there is another red path from a black node n to v which does not use any link in the E'' . Then, there might be another node n' which was not selected for forwarding flow during flow creation process in Step 5 of Alg. 14. Which means n' offers a path longer than the shortest path. FR n will never select a next hop offering longer path when it already has a neighbor offering shortest path. Therefore, there is no other congested flows from a user which is not in G'' . Cutting all the flows by removing some nodes means blocking the congested shortest paths in G . As the Acid and Campos's method provides the minimum size vertex cut set, the algorithm will return the optimal solution. \square

Theorem 14. *The complexity of Alg. 13 is $O(m^3d)$.*

Algorithm 13 Find a minimum number of blockage

Input: Topology G , possible congested links L_c .

Output: A set of links to block from L_c .

- 1: **Procedure:** FIND-ASSIGNMENT(G)
 - 2: $G' \leftarrow$ the line graph of G .
 - 3: CREATE-FLOW(G')
 - 4: $G'' \leftarrow$ ELIMINATE(G')
 - 5: $g \leftarrow$ minimum vertex separation set in G''
 - 6: **return** g
-

Algorithm 14 Create flow in the line graph

Input: Topology G' .

- 1: **Procedure:** CREATE-FLOW(G')
 - 2: $u \leftarrow$ all users in G'
 - 3: **while** $u \neq \emptyset$ **do**
 - 4: **for** all $n \in u$ **do**
 - 5: $nh \leftarrow \{n' : n' \text{ offers shortest paths to } n\}$
 - 6: **if** $C_n(n) = \text{black}$ or $C_f(n) = \text{red}$ **then**
 - 7: $\forall_{n' \in nh} C_e(e_{nn'}) \leftarrow \text{red}$
 - 8: $\forall_{n' \in nh} C_f(n') \leftarrow \text{red}$
 - 9: **else if** $C_e(e_{nn'}) \neq \text{red}$ **then**
 - 10: $\forall_{n' \in nh} C_e(e_{nn'}) \leftarrow \text{green}$
 - 11: $\forall_{n' \in nh} C_f(n') \leftarrow \text{green}$
 - 12: $u' \leftarrow u' \cup nh$
 - 13: $u \leftarrow u'$
-

Proof. Let us assume that G has n nodes, m edges, and d maximum node degree. Step 1 stage takes $O(md)$ time. If d is the maximum node degree in G , then G' has $2d - 1$ max node degree which is $O(d)$. Step 2 takes $O(md)$ to create flow for a user. In the worst case the number of users can be m and Step 2 takes $O(m^2d)$ time. To create G'' from G' , $O(md)$ time is needed in the worst case. Therefore, the conversion from G to G'' takes $O(m^2d)$. Step 4 takes $O(m^3d)$ to find a minimum vertex separator. Therefore, the algorithm takes $O(m^3d)$. □

Algorithm 15 Elimination of white nodes from the flow graph

Input: Topology G' .

Output: The final graph G'' .

```
1: Procedure: ELIMINATE( $G'$ )
2:    $G'' \leftarrow G'$ 
3:   for all  $n \in V'$  do
4:     if no incident edge to  $n$  is red then
5:       Remove  $n$  from  $V''$ 
6:        $\forall_{j \in Neighbor(n)}$  remove  $e_{nj}$  from  $E''$ 
7:     else if  $C_n(n) = white$  then
8:       Remove  $n$  from  $V''$ .
9:       Add edges from incoming to outgoing neighbors of  $n$ .
10:  Create a virtual node  $u$ 
11:  for all nodes  $n$  in  $V''$  do
12:    if # of green incident edge to  $n \geq 1$  then
13:      Add edge  $e_{un}$  to  $E''$ 
```

5.6 Extension of Problem I

In this section, we formulate a similar problem for a more general scenario and provide an optimal solution. In problem I, we consider that there is at least a shortest non-congested path from every user. In some cases, all of the shortest paths and some paths longer than a shortest path can become congested. Alg. 13 fails to solve the problem in this situation. Therefore, we formulate problem II, in which we assume that there exists a non-congested path from each user to victim.

***Problem II:** Find a minimum number of filter assignments so that all legitimate traffic follows non-congested paths.*

In this problem, we assume that from every user there is a non-congested path. Let P_u be the set of paths from a user u to the victim v . After deploying B filters to FRs, we get $P_u^s(B)$ where $P_u^s(B) \not\subseteq P_u$ and none of the paths in $P_u^s(B)$ use any possible congested links. Therefore, the problem can be expressed as the following optimization problem:

Algorithm 16 Find a minimum number of blockage

Input: Topology G , possible congested links L_c .

Output: A set of links to block from L_c .

```
1: Procedure: FIND-ASSIGNMENT-2( $G$ )
2:   ADD-IMAGINARY-U( $G$ )
3:    $G' \leftarrow$  the line graph of  $G$ .
4:   CREATE-FLOW-2( $G'$ )
5:    $G'' \leftarrow$  ELIMINATE( $G'$ )
6:    $g \leftarrow$  minimum vertex separation set in  $G''$ 
7:   return  $g$ 
8: Procedure: ADD-IMAGINARY-U( $G$ )
9:   Remove all links in  $L_c$  from  $E$ 
10:  for  $n \in V$  do
11:    if all shortest path of  $n$  are congested then
12:       $Q \leftarrow \{n\}$  ▷ initialize a queue with  $n$ .
13:      while  $Q \neq \emptyset$  do
14:         $n' \leftarrow$  DEQUEUE( $Q$ ).
15:         $C_n(n') \leftarrow gray$ .
16:         $n'' \leftarrow$  neighbors of  $n'$  offering shortest path.
17:        ENQUEUE( $Q, n''$ ).
18:  Add all links in  $L_c$  to  $E$ 
```

$$\begin{aligned} & \text{minimize } B \\ & \text{subject to } \forall_{1 \leq u \leq U, p \in P_u^s(B)} p \cap L_c = \emptyset \end{aligned} \tag{5.2}$$

5.7 An Optimal Solution for Problem II

Let us consider the topology in Fig. 5.4(a). From the FR g , there is only one shortest path which contains possible congested links. Therefore, the traffic from u_1 should go through a shortest detour to avoid the congested links. We divide the procedure to find the minimum number of links to block into five steps:

- Step 1: Virtual user creation: In this step, we find the nodes which are on detour and add virtual users.
- Step 2: Transformation: same as Step 1 in problem I

- Step 3: Modified edge coloring: This step is a simple modification of step 2 in problem II.
- Step 4: Elimination: same as Step 3 in problem I.
- Step 5: Min separation: same as Step 4 in problem I.

In Step 1, we compute the shortest path distance and next hops from each user whose shortest paths are all congested, ignoring the possible congested links. We color the nodes that are on the shortest paths gray. In Fig. 5.4(a), ignoring the possible congested links (3, 6, 7, and 8), we get the detour from g which is $g \leftrightarrow h \leftrightarrow e \leftrightarrow b \leftrightarrow a$. We color g , h , e , and b as gray and ignore the the node a which is connected to the victim. This is because, when a packet reaches a it will always reach the victim without any trouble. Then, we add a virtual user to each gray node. u_3 , u_4 , and u_5 are virtual users. It is clear that when a packet arrives at a gray node, there might be other possible congested paths to the victim which have equal or less length compared to the detour. The gray FR might choose a shorter path than the detour. If there is no shorter path than the detour, then there is a possibility to choose one of the congested paths by a FR. Therefore, we want to block all the congested shortest paths from every gray node by using a modified version of the previously described method.

In Step 2, we transform G to G' using Step 1 of problem I. In Step 3, green flows are created from every user (both real and imaginary). Unlike Step 2 of problem I, here, the flow travels through the paths that are less than or equal to the length of the detour. For example, in Fig. 5.4(b), from the gray node e the detour $e \leftrightarrow b \leftrightarrow a$ is 3 hops. The neighbors that offer the shortest path less than or equal to 2 are selected, and flow propagates to those nodes. The nodes d and b offer 2 length paths. The flow travels to d and b . When the flow reaches b , the traveled hop is 1 and we look for neighbors of b that offer the shortest path less than or equal to 1. The flow continues this way. This way the flow travels all possible ways which could destruct the user traffic passing through the detour. Due to

space limitation, the procedure $\text{CREATE-FLOW-2}(G')$ is not shown here. The green flows from each virtual user ensures a green incident edge to each node they travel first. This ensures presence of green flow in each detour node. Therefore, it is important to add virtual users at each detour node. Otherwise, red flows from other users may overwrite the color of the green incident edge to a black/gray node.

Step 4 and Step 5 are same as Step 3 and Step 4 in problem I. According to Fig. 5.4(e), the minimum vertex separation in G'' is $\{3, 6, 7\}$. Blocking the edges 3, 6, and 7 by sending filters to the FRs d , e , and f ensures the shortest non-congested paths from each legitimate to the victim. The complete procedure is shown in Alg. 16.

Theorem 15. *The complexity of Alg. 16 is $O(m^3 + n)$.*

Proof. Alg. 16 has an additional computation for adding imaginary users. The procedure takes $O(nd)$ to find out the detour nodes for all users. Then, adding virtual users takes $O(n)$ time. The modified flow generation procedure takes a similar amount of time. Therefore, Alg. 16 takes $O(m^3d + nd)$. If we consider mesh topology or other networks where the maximum node degree is small, then Alg. 16 takes $O(m^3 + n)$. \square

Theorem 16. *Alg. 16 finds the optimal number of links to block.*

Proof. Alg. 16 transforms the original topology to a flow graph where all congested flows are considered. Disconnecting the source from the victim will disable all possible congested paths which are not longer than the detour. To prove the optimality of Alg. 16, we need to prove that G'' represents all possible congested flow not longer than the detour.

Assume that there is another red path from a black node n to v which does not use any links in E'' . Then, there might be another node n' which was not selected for forwarding flow during the flow creation process in Step 5 of Alg. 14. This means n' offers the shortest path longer than the detour. The FR n will never select a next hop offering a longer path when it already has a neighbor offering a shorter path. Therefore, there are no other

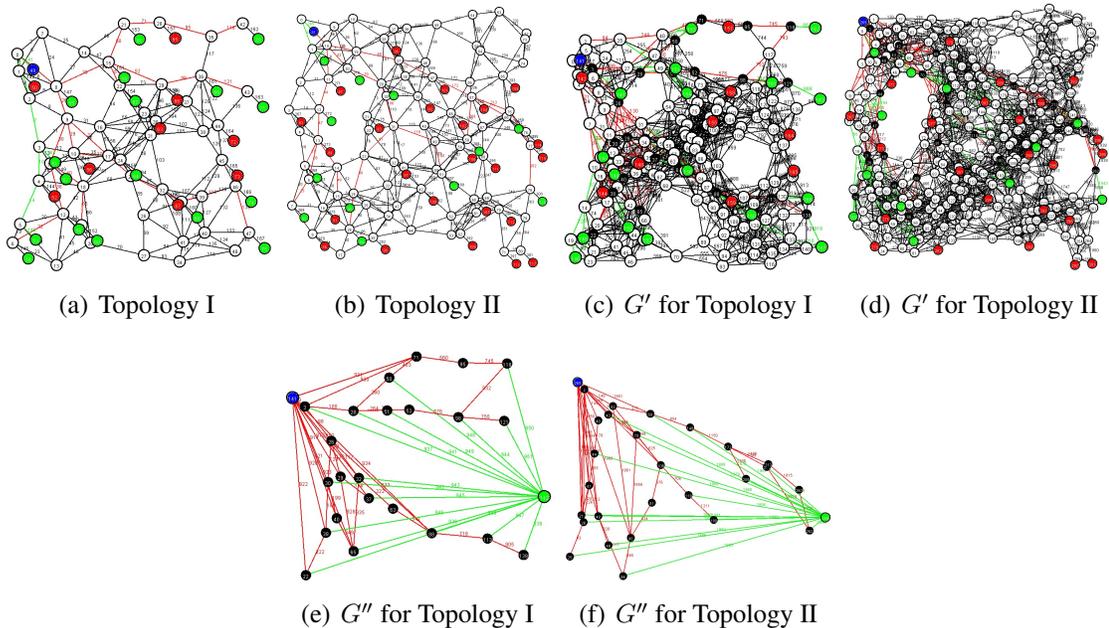


FIGURE 5.5: Topologies I and II. (Green node=user, red node=bot, blue node=victim, red link=possible congested link)

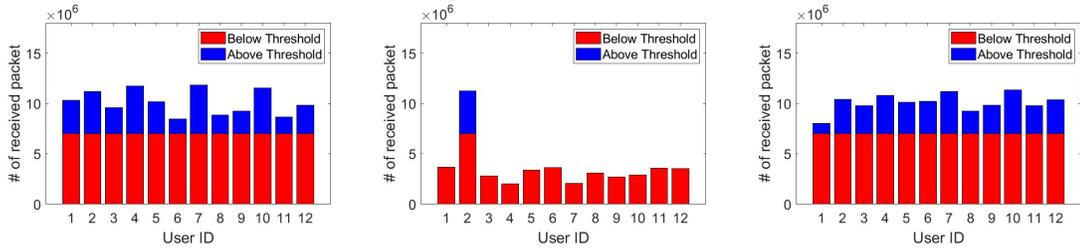
congested flows from a user that are not in G' . Cutting all the flows by removing some nodes means blocking the congested paths in G . Acid and Campos's method provides the minimum size vertex cut set. Therefore, the algorithm will return the optimal solution. \square

5.8 Experimental Results

In this section, we present our experimental setting and simulation results comparing some existing works.

5.8.1 Experimental Settings

We conducted all the experiments with a custom built Java simulator. The main reason for using a custom built simulator is its scalability. We do not need to analyze transmission time or natural packet drop issues. We only need to count the number of legitimate received packets, number of possible congested links, number of links to block, and number of hops to the victim. The network topologies we considered contain 68 – 2000 routers. Using



(a) Before attack number of re- (b) During attack number of re- (c) During defense number of re-
 ceived packets. ceived packets. ceived packets.

FIGURE 5.6: Packet distribution of different scenarios in Topology II.

NS3 or other similar simulators for this kind of simulation would take several days. That is why we built our own Java multi-threaded simulator to get the results quickly.

We conducted simulations for randomly generated graph topologies and a subset of real network topology. We used two randomly generated topologies with node degrees between 1 and 15 and internal node user probabilities between 0.2 and 0.5. The number of users, and bots were selected randomly from a uniform distribution. The target links are selected based on the number of users using a link.

If a topology has an internal user probability of 0.5, it means that 50% of the nodes are attached to attackers or to users. The Topology III was taken from the Autonomous systems AS-733 dataset [24]. The details are shown in Table 5.2. Topologies I and II are depicted in Figs. 5.5(a) and 5.5(b). The green, red, blue, and white nodes represent the users, bots, victim, and FRs, respectively. The red links represent the possible congested links. The line graph G' with flows for Topologies I and II are shown in Figs. 5.5(c) and 5.5(d). The black nodes represent the congested links. As there is no effect of bots in coloring the edges, we ignore the red nodes when creating flows. The final flow graphs G'' are shown in Figs. 5.5(e) and 5.5(f). The black rectangles around a black node represent that the corresponding links of that node are selected for blocking.

Table 5.2: Topology Parameters

	Topology I	Topology II	Topology III
Number of nodes	68	117	2000
Number of edges	179	302	10800
Internal user probability	0.2	0.5	0.5
Max node degree	10	15	1459

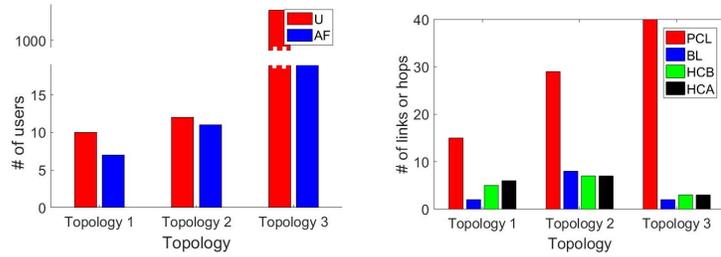
5.8.2 Simulation Results

As the bots are distributed uniformly over the network, the attacker can congest any of the links in the network. We assume that the attacker is capable of congesting only one link. It selects the link which carries traffic from the highest number of users. Simulations shown in Figs. 5.6(a)-5.6(c) are conducted with this setting.

Fig. 5.6(a) shows the regular number of received packets by the victim from the users in Topology II. There are 12 users and the number of received packets from all of them is above the threshold (150,000,000) for a particular period. When the attack is started, the numbers of received packets from some of the users are reduced. From Fig. 5.6(b), we can see that the number of received packets of 11 users fall below the threshold. We also see in Fig. 5.7(a) that 11 users are affected by the attack. When the filters are deployed, we can see that the number of received packets by all of the users is above the threshold (see Fig. 5.6(c)).

Fig. 5.7(a) shows the number of affected users (AF) and total users (U). In Topology I, we see that after congestion, 7 of the 10 users become affected. For Topology II, 11 out of 12 users are affected. The ratio of the affected users in Topology III is less than the other topologies. This is because Topology III has a higher node degree and many shortest paths. Therefore, the traffic from different users goes through many distinct paths.

Fig. 5.7(b) shows possible congested links (PCLs), the number of links needed to block to avoid all possible congested links (BL), and average number of hops to reach the victim before filter deployment (HCB) and after filter deployment (HCA). We can observe that the BL is much smaller than the PCL. Specially in Topology III, the number of PCLs



(a) Total users and affected users (b) Different measurements for three topologies.

FIGURE 5.7: Simulation results.

is 40, but they can all be avoided by blocking only 2 links. After deploying the filters, the average number of hops to the victim increases in Topology I, but remains the same for Topologies II and III.

Figs. 5.8(a) and 5.8(b) show the number of affected users, possible congested links, blocked links, and hop counts after filter deployment. We observe that all the measurements increase with a higher number of attacked links. The HCA does not increase in Topology III, but it increases by 1 or 2 in Topology I. This is because Topology III has more redundant connections than Topology I.

Fig. 5.8(c) shows the comparison between our proposed FR-based model and BGP advertisement-based model. It is hard to compare between these two models because the BGP advertisement-based model assumes that the deployer autonomous system knows exactly which links are congested. We assume that the victim (deployer) does not know exactly which links are congested, which is more practical. We assume that the victim in FR-based model knows the congested links to give both models equal information. The BGP advertisement-based model will block all of the congested links (it avoids the congested links by false path advertisement) but our FR-based model blocks much fewer links than the BGP advertisement-based model. Fig. 5.8(c) shows the number of blocked links by the number of attacked links for both approaches. The attacked links are chosen randomly and an average of several rounds of simulation are shown in the figures.

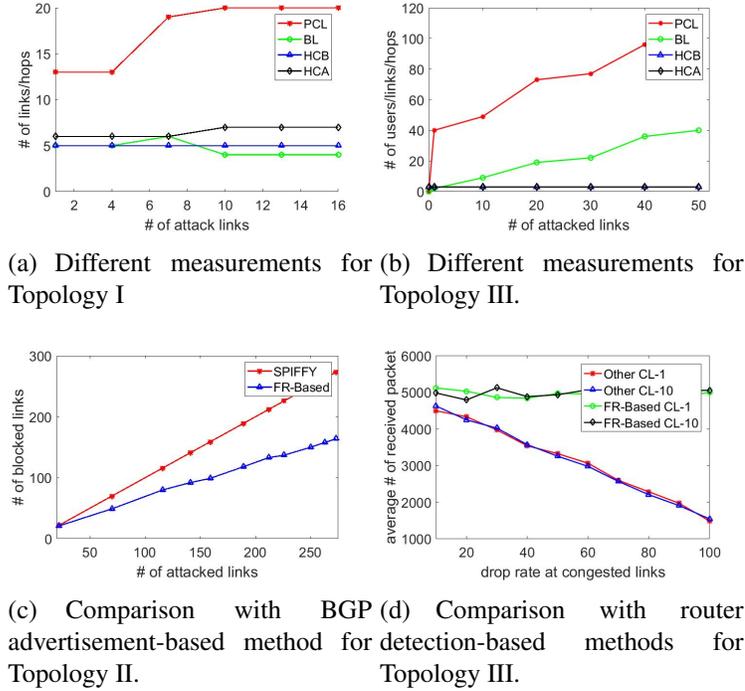


FIGURE 5.8: More simulation results.

We compare the FR-based model with some other approaches including FLoc [46], SPIFFY [28], ColDef [30], which use attack traffic detection at the router. We show the detrimental effects when the routers cannot detect the attack traffic. Fig. 5.8(d) shows the average number of received packets by the ratio of packet drop at congested links. The average number of received packets is reduced by the drop rate at congested links. The number of received packets for 10 congested links is slightly less than that of one congested link. The numbers of received packets are similar regardless of the number of congested links or drop rate for FR-based model. This is because when filters are deployed, none of the user traffic goes through any congested links.

From the above experimental results we can conclude that the FR-based model can efficiently protect the users' traffic. We need to block few number of links to avoid a large number of possible congested links. The model works with limited knowledge about the networks status and can protect its users with the minimum cost.

5.9 Summary

The transit-link DDoS attack is the most powerful attack that makes a service unavailable to legitimate users. It is not possible to protect any server from DDoS attack without the help of the network equipments. Current DDoS defense system cannot protect the transit-link DDoS attack because the attack packets do not go to the victim. Therefore, victim remains unaware of the location of the attacker. Routers which are the most important component in network can be upgraded to FRs easily. Besides, the FR can work in a network with legacy routers. An ISP can gain benefit economically by deploying FRs. Therefore, both the deployer and victim get benefited from this model. In this work, we present an optimal filter assignment policy. We compare performances of proposed assignment policy with some existing approaches. Simulation results show that our proposed approaches work much better than the other existing approaches. Up-to this chapter we considered only the external attackers. Therefore, it is important to defend the internal attacker and the next chapter is going to be on defending internal attackers.

CHAPTER 6

OPTIMAL MONITOR PLACEMENT POLICY AGAINST DISTRIBUTED DENIAL-OF-SERVICE ATTACK IN DATACENTER

In the previous chapters, we have proposed multiple solutions to defend against external attackers. In this chapter, we are going to explore the areas to defend against the internal attacker in datacenter by monitoring internal flows. The network considered in this is software defined networks. The research in this chapter has been published in [Related11].

6.1 Introduction

A denial-of-service attack (DoS attack) is a cyber-attack in which the attacker seeks to make a machine or network resource temporarily unavailable to its users. CloudFlare is one of the biggest companies that provides DDoS protection services. CloudFlare users change the DNS of their domain and point to the CloudFlare DNS server. The CloudFlare DNS server returns a CloudFlare IP address instead of the user's IP address in DNS lookups. The server located at that IP address analyzes and filters the attack traffic. However, if the attackers remain in the same datacenter and know the public or private IP address of the victim, then the traffic sent to that IP address does not go through the

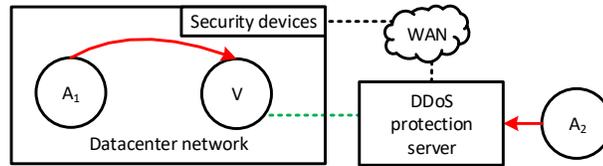


FIGURE 6.1: Internal DDoS attack (A_1 and A_2 : attacker, v : victim).

CloudFlare server. Therefore, the internal DDoS traffic is not filtered by the CloudFlare server.

There are several methods to find the IP address of the server behind the CloudFlare protection server. Firstly, some websites give historical data about a domain, including IP address changes, sub-domains, and cloud providers. It is sometimes possible to get the previous IP addresses of the target domains. It is possible that the IP address before the CloudFlare IP address is the actual IP address of the user. Secondly, if the target server runs any mail-server, then any email directly sent from the mail-server contains the IP address of the target server in its header. Generally, an email addressed to a wrong username (email) to a mail-server gets a reply that the username does not exist on the mail-server. Finally, a brute force approach can find the internal IP of the server. For example there are 16,777,216 class A private IP addresses (10.0.0.0 – 10.255.255.255). If one second is spent checking whether an IP address is the targeted server or not, a program with 194 threads can finish the checking in a day. This approach might not work if the target disables internal incoming flows. There are some applications, including hadoop and spark, that require internal communication. Therefore, it is not always hard to get the internal IP address of the target.

The security modules are expensive and they remain in the core or aggregation layer, which monitors the incoming (or outgoing) traffic from (or to) a datacenter. Therefore, an existing system cannot protect against internal DDoS attacks. Fig. 6.1 shows a scenario of an internal DoS attack. The victim V uses a commercial DDoS protection service.

Therefore, the external attacker A_2 's traffic is blocked by the DDoS protection server. The internal attacker A_1 resides in the same datacenter as V . A_1 's traffic does not go through the datacenter security devices. As a result, the attack traffic reaches V without any trouble. Other filter based DDoS attack defense mechanisms such as [36, 48, 49] cannot protect internal DDoS attacks. In order to prevent an internal DDoS attack, we need to do two things: monitor all the internal traffic and block the attack traffic. To block traffic, the controller creates a rule in the hypervisor firewall. Therefore, we focus on monitoring all internal traffic.

In this paper, we assume that the datacenter switches are all software defined networking (SDN) switches. Some of the free virtual machines (VMs) are used as traffic monitors. The monitors are special network function virtualization (NFV) units which can detect DDoS behavior of flows. The SDN switches probabilistically copy packets of each flow to a monitor. A monitor runs machine learning techniques to detect any DDoS behavior in the flows it receives. If a monitor classifies that a flow belongs to a DDoS attack, it creates a firewall rule at the hypervisor of source VM to stop the flow. The main contributions are as follows:

- We formulate a problem for assigning flows to monitors, considering that the location of the monitors are predefined. We provide an optimal solution by reducing the problem to a max-flow min-cost problem.
- We formulate another problem to optimally select some of the VMs among the free VMs for monitor placement, considering a budget on the number of VMs. This is an NP-hard problem and we provide a greedy solution.

The remainder of this paper is arranged as follows: Section 6.2 presents some related works. In Section 6.3, we present the system model. Section 6.4 and Section 6.5 present the formal definition of the first and second problem and our proposed optimal solutions,

respectively. In Section 6.6, we present some simulation results. Finally, Section 6.7 concludes the paper.

6.2 Related Work

There exist many statistical methods including correlation, entropy, covariance, cross-correlation, and information gain to detect anomalous DDoS requests [8]. A rank correlation-based method, Rank Correlation-based Detection (RCD), is proposed in [9]. An information theoretical approach using Kolmogorov complexity is used for the detection of DDoS attacks in [10]. A novel DDoS detection mechanism is proposed based on Artificial Neural Networks in [11]. There are other methods of detecting DDoS attacks, including [12, 13]. These methods can be used in monitors to detect DDoS attacks.

In [50], authors propose a three layer DDoS defence system, called Bohatei. It exploits the functionality of NFV/SDN architecture to mitigate distributed DoS attacks. There are some OpenFlow-based intrusion detection systems which utilize NFV/SDN functionalists [51, 52].

Authors in [53] propose a scheme to monitor the internal and external flows in a datacenter. The SDN switches mirror every flow to monitors. They use a greedy approach to find the locations for the monitors. At first, they find the districts that need at least one monitor. Then, they find assignments inside each district. Their approach sometimes fails to find an assignment due to the limited number of VMs. They also do not consider a budget on the number of VMs and fractionally copy packets of network flows. In [54], authors propose a two-stage DDoS mitigation framework. In the first step, it screens the traffic and determines what kind of processing is important for the flow, including network-layer security and application-layer security. In the second step, the advanced specialized detection system is used to detect DDoS behavior.

Large cloud providers such as Amazon EC2 and Microsoft Azure claim to provide protection against several traditional network security attacks, including DDoS. The DDoS

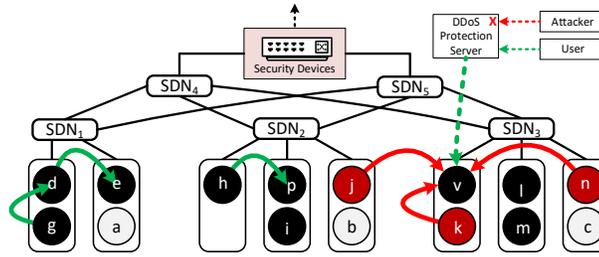


FIGURE 6.2: System model.

mitigation techniques, including SYN cookies and connection limiting, are employed within EC2 [55]. Microsoft Azure is also capable of detecting internal DDoS attacks and remove the attacking VMs or accounts [56]. Though Azure and EC2 have mechanisms to protect against traditional DDoS attacks, they recommend the tenants to implement their own protection mechanism. Besides, many other datacenters are vulnerable to internal DDoS attacks. Authors in [57] discuss the detrimental effects by internal attacks including low rate DDoS attack.

The aforementioned works that exploit the benefit, of SDN and NFV either do not consider the network overhead and limited budget on number of VMs, or they partially process the traffic flows. Therefore, it is necessary for a new system to consider a limited budget on the number of VMs and aim to partially or fully monitor all the network flows.

6.3 System Model

6.3.1 Datacenter Model

Our system is composed of physical machines (PM), top of rack (TOR) switches, SDN switches, users, attackers, and a victim. The system model is depicted in Fig. 6.2. The victim (v) uses a DDoS protection service from a commercial DDoS protection service provider, which is located outside the datacenter. The top level of the datacenter is equipped with flow monitoring devices which monitor incoming and outgoing traffic to/from the datacenter. Any external attacker or user (that remains outside the datacenter) has to go through the DDoS protection server and the datacenter security module. There-

fore, the external attackers are easily blocked by either the DDoS protection server or the datacenter security module. The traffic from an internal attacker/user does not go through the security module or the commercial DDoS protection server.

The SDN controller controls all the flow monitoring process. It deploys monitors to some free VMs. Then, it assigns each flow to a monitor and the monitor analyzes the packets in the flow to detect whether it belongs to an attacker or not. There are many methods, including [12, 13], to classify whether packets in a flow belong to an attacker or not, discussed in Section 6.2. If the flow belongs to an attacker then the monitor notifies the controller. The controller creates a firewall rule in the originating VM's hypervisor to block the flow. Assigning a flow to a monitor happens in two steps. In the first step, the SDN switch on the flow path closest to the monitor is selected. In the second step, an SDN or hypervisor rule is created to copy the packets and forward them to the monitor. Let us assume that the flow f_{jv} (flow from VM j to v) is assigned to the monitor installed in VM c . The packets of flow f_{jv} can be copied and forwarded to c from any of the SDN switches in $\{SDN_2, SDN_5, SDN_3\}$. SDN_3 is closest to c . Therefore, the controller selects SDN_3 , copies, and forwards the packets to c . The reason for selecting the closest SDN switch is that it increases the minimum network overhead for copying a flow. We also consider the limited capability of monitors. We assume that each monitor can handle a limited number of flows. Though in reality, capability of each VM can be different. For simplicity, we assume that all VMs are homogeneous with the same capabilities. Besides, we can easily extend the solution for homogeneous VMs to the solution for heterogeneous VMs.

When the number of required monitors for all the flows is less than the budget, the controller creates rules to probabilistically copy a certain portion of each flow from the SDN switch. For example, assume there are 2 available VMs for monitors and each monitor can handle a flow. If the number of flows is 4, then each of the monitors needs to handle 2 flows. Therefore, the SDN switches copy 50% of the packets of each flow and forward them to the monitors. The copy of the original packets creates extra network overhead. We

measure the increased network overhead by multiplying the total data rate by the number of hops the flow copy travels. Let $f_{ab} \in F$ be a flow between VM a and b . r_{ab} is the data rate of flow f_{ab} . V is the set of free VMs and $A : F \rightarrow V$ is the monitor assignment. The network overhead for assignment A is $C(A)$.

$$C(A) = \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} dist(p, A(f_{ab})) \quad (6.1)$$

Here, $a \rightarrow b$ is an ordered set of SDN switches on the path from a to b , including the physical machines of a and b . $dist(p, A(f_{ab}))$ is the number of hops from SDN switch p to the monitor $A(f_{ab})$. For example, if the data rate of the flow f_{jv} is 1 (Mbps) and it is assigned to monitor c , then the increased network overhead for monitoring the flow is $1 \times 1 = 1$. If the flow f_{jv} is assigned to monitor b , then there is no network overhead. This is because, the hypervisor can copy the flow to the monitor from the source VM of flow f_{jv} . So, the copied packets do not travel any links. The increased network overhead can be a reason to degrade QoS of the datacenter. Therefore, our main goal is to minimize the increased network overhead. To minimize the network overhead, we need to select the best subset of the free VMs for the monitor placement and find the best flow assignment. The flows in a datacenter is not static. Therefore, the controller needs to re-calculate flow assignments periodically.

6.3.2 Attack Model

We assume all the attackers are internal attackers and controlled by a master. The master may not reside inside the datacenter. It is capable of finding the actual IP (public or private) address of the victim. When the master sends commands telling the attackers the victim's IP address, they start sending attack packets. The victim becomes overwhelmed with the attack packets and DoS occurs. For example, in Fig. 6.2, VMs j , k , and n are the attackers.

6.4 Flow Assignment

In this section, we formulate the problem to find an optimal flow assignment to the monitors. We consider that the cloud provider allocates some VMs for monitor deployment. The number of VMs, location, and type of VMs are decided based on affordability and the business strategy of the cloud provider. For example, the cloud provider may not want to allocate VMs in a PM, which is highly demanded by customers.

6.4.1 Problem 1: Find a flow assignment so that the network overhead is the minimum by ensuring coverage of all internal flows.

Let $f_{ab} \in F$ be the flow between VM_a and VM_b ($VM_a \in V$ and $VM_b \in V$). The data rate of the flow f_{ab} is r_{ab} . There are M ($|F| = M$) number of flows and N ($|V| = N$) number of free VMs. The cloud provider can afford VMs in V' ($|V'| = K$) from some predefined locations. Therefore, each VM needs to monitor M/K flows. We need to find a flow assignment $A : F \rightarrow V'$ for which the increased network overhead is minimum. The problem can be expressed as the following:

$$\begin{aligned} & \text{minimize} && \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} \text{dist}(p, A(f_{ab})) \\ & \text{subject to} && \forall v \in V' |\{f : f \in F \text{ and } A(f) = v\}| \end{aligned} \tag{6.2}$$

Here, p is the node on the path $a \rightarrow b$ and $\text{dist}(p, A(f))$ is the number of hops from the node p to the VM that monitors the flow f_{ab} . The location of K VMs are predetermined. Therefore, we only need to find an optimal assignment between the flows and free VMs. Usually, datacenters use regular switches with SDN switches which are not programmable and cannot copy packets. If a flow does not go through any SDN switches, then the hypervisor of source or destination is the only way of copying the flow. For simplicity, we consider all the switches in our model to be SDN switches.

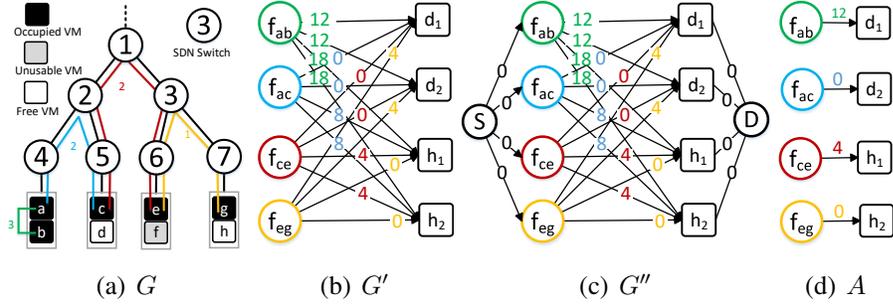


FIGURE 6.3: Problem I example.

6.4.2 An Optimal Flow Assignment Scheme

The assignment process consists of two steps. In the first step, we construct a bipartite graph $G' = (V_1, V_2, E)$, where V_1 is the set of flows ($|V_1| = M$) and V_2 is the set of parts of VMs. Each VM is split into VM_1, VM_2, \dots, VM_B parts so that each part can handle one flow ($|V_2| = B \times K$). From each flow, edges are added to all parts. The weight of the edge is the increased network overhead. So, G' is a complete bipartite graph ($K_{M, BK}$).

In the second step, we find a minimum cost perfect matching. We can easily reduce the problem to a minimum cost flow problem. A flow graph $G'' = (V'', E'')$ is created from G' ($V'' = V_1 \cup V_2$ and $E'' = E$). A source and a destination are added to V'' . Then, edges from the source to all of V_1 and the destination to all of V_2 are added to E'' . The capacity of each edge/link is 1 and the cost of flow is the weight of each edge. Newly added edges will have zero cost. The max-flow min-cost paths in G'' produce a minimum cost perfect matching in G' . We can find max-flow min-cost paths using any of the algorithms in [58]. The complete algorithm is shown in Alg. 17. Procedures CREATE-BIPARTITE and CREATE-FLOW-GRAPH are not shown in details to save space.

Let us consider the simple topology in Fig. 6.3(a). There are seven SDN switches, each connected to a PM. Each PM can host two VMs. There are four flows f_{ab} , f_{ac} , f_{ce} , and f_{eg} . The data rates of these flows are $r_{ab} = 3$, $r_{ac} = 2$, $r_{ce} = 2$, and $r_{eg} = 1$. There

Algorithm 17 Flow Assignment

Input: The topology G , set of flows F , and set of VMs V' .

Output: A flow assignment.

- 1: **Procedure:** FLOW-ASSIGNMENT(G, F, V')
 - 2: $G' \leftarrow$ CREATE-BIPARTITE (G, F, V')
 - 3: $G'' \leftarrow$ CREATE-FLOW-GRAPH (G')
 - 4: $A \leftarrow$ find assignment using max-flow min-cost in G'' .
 - 5: **return** A .
-

are three free VMs $\{d, f, h\}$. The cloud provider can only afford d and h . We want to find a mapping from the four flows to two VMs. Therefore, each VM will handle two flows.

According to the first step, we create a bipartite graph $G' = (V_1, V_2, E)$ where $V_1 = \{f_{ab}, f_{ac}, f_{ce}, f_{eg}\}$. We split d into d_1, d_2 and h into h_1, h_2 . So, $V_2 = \{d_1, d_2, h_1, h_2\}$. Then we add edges between all pairs of nodes from V_1 and V_2 . The weight of each edge is the increased network overhead for monitoring it. For example, the weight of the edge between f_{ab} and h_1 is 12. If we want to monitor f_{ab} in d , then the hypervisor needs to copy the flow and the flow travels $4 \rightarrow 2 \rightarrow 5 \rightarrow d$, which is four hops. Therefore, the increased network overhead is $4 \times 3 = 12$. Similarly, the weight of the edge between f_{ab} and h_2 is also 12. If we want to monitor f_{ac} in d , then the hypervisor of d and c can copy the flow to d and there is no additional network overhead. Therefore, the weight of the edge between f_{ac} and d_1 (or d_2) is 0.

According to the second step, we create the flow graph $G'' = (V'', E'')$. We add S and D to G'' ($V'' = V_1 \cup V_2 \cup \{S, D\}$). After adding all edges in E , edges from S to each node in V_1 and each node in V_2 to D are created and added to E'' . Weights of newly created edges are zero. We assign the capacity of every edge to one. Fig. 6.3(c) shows the flow graph. Then we use one of the maximum flow minimum cost algorithms [58] to find a perfect matching between V_1 and V_2 . Fig. 6.3(d) shows the perfect matching (A). Therefore, f_{ab} and f_{ac} will be monitored in VM d , and f_{ce} and f_{eg} will be monitored in VM h . The total network overhead for monitoring these four flows is $12+4=16$.

Theorem 17. *The complexity of Alg. 17 is $O(M^3 + S^3)$.*

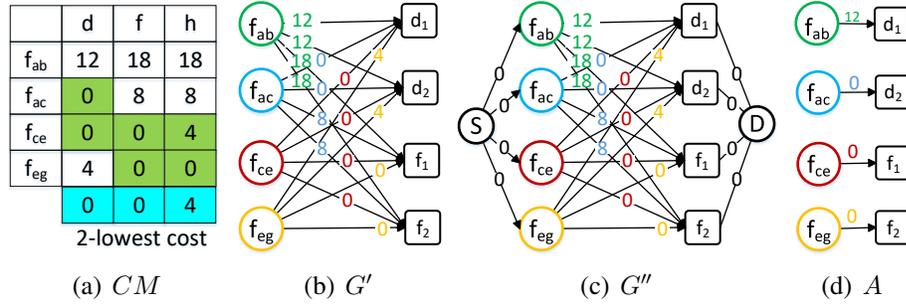


FIGURE 6.4: Problem II example.

Algorithm 18 Flow Assignment 2

Input: The topology G , set of flows F , and set of VMs V' .

Output: A flow assignment

- 1: **Procedure:** FLOW-ASSIGNMENT-2(V, K)
 - 2: Calculate M/K -lowest cost for each VM in V .
 - 3: Select K lowest M/K -lowest cost VMs V' from V .
 - 4: $A \leftarrow$ FLOW-ASSIGNMENT(G, F, V')
 - 5: **return** A .
-

Proof. In step 2 of Alg. 17, creating a bipartite graph takes $O(M^2)$ if we pre-compute all pairs' shortest path distances. Computation of all pairs' shortest path distances takes $O(S^3)$, where S is the number of SDN switches. Then, in step 3, creating the flow graph takes $O(M)$. In step 4, finding the assignments using the cheapest augmenting path algorithm takes $O(M^3 + M^2 \log(M))$ time, which is $O(M^3)$. Therefore, the complexity of the Alg. 17 is $O(M^3 + S^3)$. □

Theorem 18. *The Alg. 17 returns optimal flow assignment.*

Proof. The capacity of all edges in E'' is 1. Therefore, for any assignment, the maximum flow is M . To ensure M amount of flows between S and D , each node in V_1 must have an incoming flow of 1. Similarly, each node in V_2 must have an outgoing flow of 1. Therefore, no node in V_1 is assigned to multiple nodes in V_2 . Secondly, the minimum cost maximum flow algorithms [58] return the minimum cost path in G'' . Therefore, Alg. 17 returns the optimal flow assignment. □

6.5 Monitor Placement

In this section, we formulate a problem to find optimal locations for the monitors among some free VMs. We consider a limited budget on the number of VMs for allocating monitors. The large number of running VMs in the datacenter increases the energy consumption and cost. The cloud provider may want to reserve some VMs for a sudden increase in the customers' demand. Therefore, a cloud provider wants to deploy monitors on the limited number of VMs. At the same time, the cloud provider wants to get maximum security from the monitors and less network overhead.

6.5.1 Problem 2: *Find locations for monitors and a flow assignment so that the network overhead is the minimum by ensuring coverage of all the internal flows.*

In this problem, we assume that the locations of K VMs for installing monitors are not predetermined. We need to find a flow assignment $A : F \rightarrow V'$ and find V' ($V' \subset V$) for which the network overhead is minimum. The problem can be expressed as the following:

$$\begin{aligned}
 & \text{minimize} && \sum_{f_{ab} \in F} r_{ab} \times \min_{p \in a \rightarrow b} \text{dist}(p, A(f_{ab})) \\
 & \text{subject to} && |V'| = K, \\
 & && \forall v \in V' |\{f : f \in F \text{ and } A(f) = v\}|
 \end{aligned} \tag{6.3}$$

So, we need to find an optimal assignment of the flows to the monitors for which the network overhead is the minimum. The problem 2 is NP-hard. Its NP-hardness can be proved by reducing it to the vertex cover problem [53].

6.5.2 Greedy Approximation: *M/K -lowest cost approach*

M/K is the number of flows to be monitored in a VM. M/K -lowest cost of a VM is the lowest total network overhead for assigning M/K flows to the VM. Inclusion of a VM to the assignment set increases at least M/K -lowest cost. Therefore, we select K VMs with the lowest M/K -lowest cost. We denote the M/K -lowest cost of VM a by C'_a . The complete algorithm is shown in Alg. 18. Next flows are assigned to the selected VMs using Alg. 17.

Let us consider the example in Fig. 6.3(a). Here, $M = 4$ and $K = 2$. We need to find the 2-lowest cost for each VM. The costs for monitoring f_{ab} , f_{ac} , f_{ce} , and f_{eg} in VM d are 12, 0, 0, and 4, respectively. The 2-lowest cost of VM d is 0. Similarly, the 2-lowest costs of h and f are 4 and 0, respectively. The VMs with lowest 2-lowest costs are f and d . Therefore, $\{f, d\}$ is the set of selected K VMs. The cost matrix (CM), G' , G'' , and A are shown in Fig. 6.4.

Next, we create a bipartite graph $G' = (V_1, V_2, E)$. The $V_1 = \{f_{ab}, f_{ac}, f_{ce}, f_{eg}\}$ and $V_2 = \{d_1, d_2, f_1, f_2\}$. Then, we add edges between all pairs of nodes from V_1 and V_2 , and assign weights to all edges. Fig. 6.4(b) shows the bipartite graph. Then, we create the flow graph $G'' = (V'', E)$ by adding S and D to G' . Fig. 6.4(c) shows the flow graph. Then, we use one of the maximum flow minimum cost algorithms to find a perfect matching between V_1 and V_2 . Fig. 6.4(d) shows the perfect matching. Therefore, f_{ab} and f_{ac} will be monitored in VM d , and f_{ce} and f_{eg} will be monitored in VM f . Fig. 6.4(d) shows the final flow assignment. The total network overhead for monitoring these four flows is $12+0=12$. The final flow assignment is shown in Fig. 6.4(d).

Theorem 19. *The complexity of Alg. 18 is $O(M^3 + S^3)$.*

Proof. In step 2, calculating the M/K -lowest cost takes $O(MN)$. In step 3, finding K lowest M/K -cost takes $O(N)$. Step 4 which is finding an assignment using Alg. 17 takes $O(M^3 + S^3)$. So, the complexity of Alg. 17 is $O(M^3 + S^3)$. \square

Theorem 20. *The M/K -lowest cost approach increases at most $2R \log S$, where R is total bandwidth of the flows.*

Proof. Let the optimal network overhead be denoted by C^* . The cost approximated by the M/K -lowest cost approach is C . The set V' ($|V'| = K$) has the lowest M/K -lowest cost among the V VMs. V^* is the optimal set of VMs. In the worst case, $V - V'$ and $V - V^*$

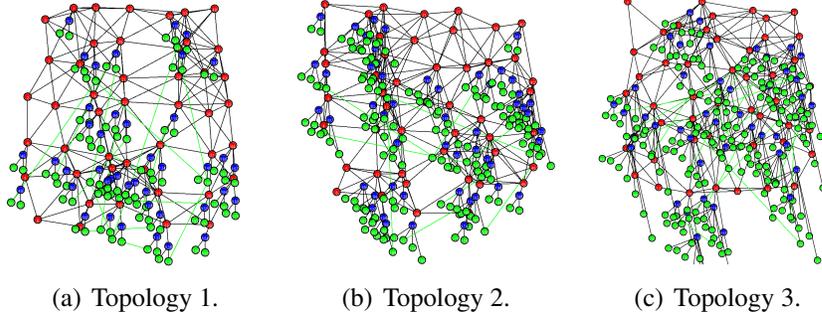


FIGURE 6.5: Randomly generated topologies.

are disjoint. Therefore, we can write

$$C^* = C - 2 \log(S) \sum_{a \in V-V'} R_a + \sum_{b \in V-V^*} C'_b \quad (6.4)$$

Here, R_a is the total incoming data rate to VM a and C'_b is M/K -lowest cost of VM b . In the worst case, $\sum_{b \in V-V^*} C'_b$ is zero. Therefore, we can write the following:

$$C^* = C - 2 \log(S) \sum_{a \in V-V'} R_a, \quad (6.5)$$

$$C^* \geq C - 2 \log(S) \sum_{a \in V} R_a, \quad C \leq C^* + 2 \log(S)R$$

So, C will be at most $2R \log(S)$ more than the optimal cost. In tree topology, $2 \log(S)$ is the diameter. Therefore, for any kind of topology the increased network overhead will be at most $R \times diameter$ more than the optimal. \square

6.6 Experimental Results

In this section, we describe our experimental settings and simulation results.

6.6.1 Experimental Settings

We conduct the experiments with a custom build java simulator. The main reason for using a custom build for the simulator is its scalability. We only need to count the increased network overhead for monitoring all flows. The network topologies we consider contain 200 – 300 SDN switches, PMs, and VMs. Using NS3 or other similar simulators for this

Table 6.1: Topology Parameters

Number of	Topology 1	Topology 2	Topology 3
Nodes	172	249	277
VMs	84	150	184
PMs	42	52	44
SDN switches	46	47	49
Links	304	392	427
PMs in an SDN switch	{1}	{1, 2}	{1, 2}
VMs in a PM	{2}	{2, 3, 4}	{1, 2, 3, 4, 5, 6, 7}
Datarate	[1,6]	[1,6]	[1,6]

Table 6.2: Random tree topology settings

Number of	Settings 1	Settings 2	Settings 3
Max degree	3	5	8
PMs in an SDN switch	3	3	3
VMs in a PM	5	5	5

kind of simulation would take several days. That is why we built our own java multi-threaded simulator to get the results quickly.

We generate three random topologies for some experiments (in Fig. 6.6). We consider a 500×500 rectangular region. The region is divided into 50×50 blocks. In each block, an SDN switch is placed by choosing a random location uniformly. Links between two SDN switches are added if their distance is less than 100 units. SDN switches in Topologies 1, 2, and 3 are generated with this setting. Then, PMs are attached to each SDN switch with probability of 0.5. The number of PMs attached to an SDN switch is 1 in Topology 1. The number of PMs attached to an SDN switch is taken randomly from $\{1, 2\}$ in Topologies 2 and 3. Then VMs are added to each PM. The number of VMs is taken randomly from $\{2\}$, $\{2, 3, 4\}$, and $\{1, 2, 3, 4, 5, 6, 7\}$ in Topologies 1, 2, and 3. Node colors red, blue, and green represent SDN switches, PMs, and VMs, respectively. Detailed information is listed in Table 6.1.

We conduct some experiments with tree datacenter topologies. In reality, many datacenters use the tree structure. We generate random trees with three different settings. We first generate the desired number of nodes and randomly pick a root among the nodes. Then, a random node from the generated nodes is picked up and added as a child to a

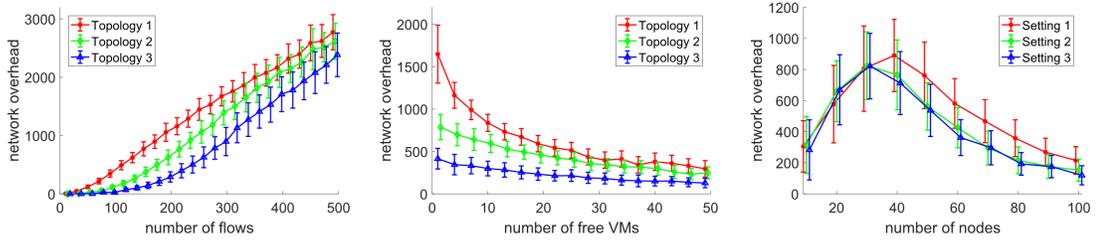
random node in the tree. Then, at each leaf node, PMs and VMs are added similarly. We define three different settings for the random tree generation. All settings allocate 3 PMs in each leaf node and 5 VMs in each PM. The maximum node degrees in Setting 1, 2, and 3 are 3, 5, and 8, respectively.

To generate flows, we consider two parameters: the number of free VMs, and the number of flows. Firstly, the desired number of VMs is excluded from the VM list. The refined VMs are used as the source and destination of a flow. Then, all possible pairs of VMs are generated. From the pairs of VMs, the desired number of pairs are chosen randomly and a flow is created between the VMs in each pair. The data rate of each flow is chosen randomly from the range $[1, 6]$.

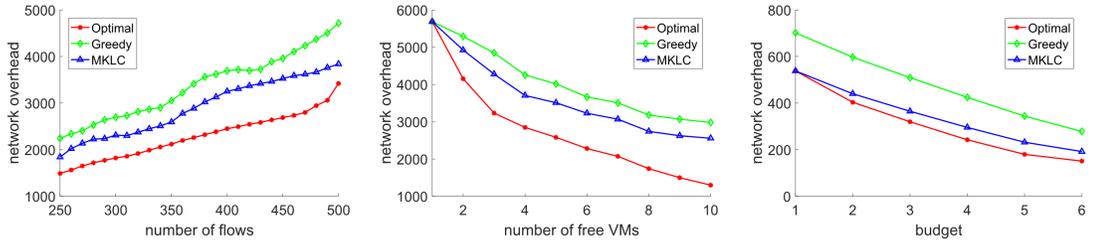
We record the increased network overhead for 100 samples and plot the average values. Since the solution for problem 1 is optimal, we compare the network overhead for different settings by varying different parameters. For the second problem, we compare the optimal, M/K-lowest cost, and greedy approaches' network overhead. To find the optimal solution, we needed to check all combination of free VMs. Therefore, we pick a small number for free VMs and budget, then observe the effect of changing multiple parameters. The greedy approach is mainly introduced in [53] for intra-district flow assignments in a datacenter. We consider a slightly modified greedy approach compared with the original. For each flow, we pick the best VM and add it to the selected VMs list. We keep adding VMs to the selected VMs list unless the budget is met. When the number of selected VMs is equal to the budget, the new flow selects the best VM among the selected VMs. When a VM's capacity becomes full, we do not consider the VM for the next flow. Thus, we assign VMs for each flow and the selected VMs are used for monitor placement.

6.6.2 *Simulation Results*

We compare the increased network overhead of three topologies. In Topology 1, we observe a small number of PMs. Topologies 2 and 3 have a higher number of PMs than



(a) Effect of the number of flows. (b) Effect of the number of free VMs. (c) Effect of the number of nodes.



(d) Effect of the number of flows. (e) Effect of the number of free VMs. (f) Effect of the number of nodes.

FIGURE 6.6: Simulation results.

Topology 1. The number of VMs in Topology 3 is the highest, Topology 1 is lowest, and Topology 2 is in between. Therefore, it is expected that Topology 1 has a higher network overhead. Though Topology 2 has a higher number of PMs, it does not help to reduce overhead. This is because VMs in the same PM do not increase network overhead between them, but VMs under the same SDN switch increase network overhead.

Fig. 6.6(a) shows the average network overhead and the standard deviation for different number of flows. We vary the number of flows from 1 to 500. We keep the minimum number of free VMs as 20 for all topologies. When the numbers of flows are small, the number of free VMs is higher than 20. In Topology 1, if the number of flows is less than 32 ($((84 - 20)/2)$), there might be more than 20 free VMs. If the number of flows is greater than 32, it does not guarantee that the number of VMs is 20. The higher the number of flows, the higher the probability of having exactly 20 free VMs. When the number of flows is 500, the increased network overheads for Topologies 1, 2, and 3 are 2,772, 2,617, and

2, 385. The standard deviation of the network overhead for Topologies 1, 2, and 3 are 302, 310, and 372, respectively.

Fig. 6.6(b) shows the average network overhead and standard deviation for different numbers of free VMs. We vary the number of free VMs from 1 to 50. We keep the number of flows as 500 for all topologies. When the number of free VMs is smaller, the network overhead of different topologies is higher. This is because in Topology 1, there are a small number of VMs and less flexibility of placing the monitors. As a result, the network overhead is higher. The network overhead decreases with the number of free VMs. When the number of free VMs is higher the difference of the network overhead is smaller. The standard deviation of network overhead is also decreased by the number of free VMs. When the number of free VMs is 1, the network overhead for Topologies 1, 2, and 3 are 1646, 785, and 414, respectively. When the number of free VMs is 50, the network overhead for Topologies 1, 2, and 3 are 295, 248, and 128, respectively.

Fig. 6.6(c) shows the average overhead and the standard deviation for different numbers of SDN switches. We vary the number of SDN switches from 10 to 100. We use the random tree generation algorithm, as discussed in experimental settings. We record the network overhead for 100 randomly generated trees with 500 flows. The settings 1 trees are supposed to have a greater height than settings 2 or 3 trees. So, the network overhead is supposed to be higher in topologies with smaller node degrees. The maximum number of flows is set to 500 for all settings. We keep the number of free VMs equal to 50% of the number of SDN switches. We observe that there is no significant difference in network overhead when the number of SDN switches ranges from 10 to 30. This is because the maximum allowable flows almost saturate the network. When the number of SDN switches is greater than 30, we see that settings 1 topologies have a higher network overhead than settings 2 and 3. A datacenter with moderate flows and a higher node degree helps to reduce network overhead. We also observe that the overhead is increasing by the number of SDN switches till 30. This is because the maximum possible flows do not

cross 500 in the topologies. When the number of SDN switches increases, the number of VMs and the maximum possible flows also increase. When the number of SDN switches is more than 30, the number of flows in the network remains constant (500). The higher the number of SDN switches, the higher the number of VMs, free VMs, and flexibility. As a result, the network overhead decreases with the number of SDN switches.

We compare the performances of the optimal (brute force), greedy, and M/K -lowest cost (MKLC) approaches by varying different parameters. Fig. 6.6(d) shows the average network overhead of optimal, greedy, and MKLC for different numbers of flows. We varied the number of flows from 250 to 500. We keep the number of SDN switches to be 100. We choose the range because they neither saturate the network, nor are very small for the network. We use the settings 3 and the number of flows as 500 for all topologies. We keep the budget constant (9 VMs) to see the performance of three approaches and their effect on the number of flows. The overhead for greedy is higher than MKLC. On average, the network overhead is 3,366, 2,913, and 2,279 for greedy, MKLC, and optimal approaches, respectively. The MKLC increases the network overhead by 27% while the greedy increases overhead by 48% more than the optimal network overhead.

Fig. 6.6(e) shows the average network overhead of the three approaches for different numbers of free VMs flows. We vary the number of free VMs from 1 to 10. We keep the same settings as the previous experiment for the topology generation. We set the budget to be 50% of the number of free VMs. When there is a free VM, all approaches will assign all flows to that VM. Therefore, all approaches have the same network overhead. When the number of free VMs is 2, the greedy approach is more likely to not choose the best VMs for placing monitors than the MKLC approach. A higher number of free VMs gives more options and a higher location flexibility. As a result, the network overhead decreases by the number of VMs. We observe that the MKLC always performs better than the greedy approach. On average, the network overhead is 4,049, 3,607, and 2,507 for greedy, MKLC, and optimal approaches, respectively.

Fig. 6.6(f) shows the average network overhead of the three approaches for different budgets. We vary the budget from 1 to 6. We keep the number of SDN switches as 20. We use the setting 3 and number of flows as 100 for all topologies. It is clearly observed that the overhead decreases by the budget. The network overhead in MKLC approach is closer to the optimal than the greedy approach. On average, the network overheads are 470, 336, and 291, respectively. Therefore, we can conclude from the above experiments that, the MKLC approach produces less overhead than the greedy approach.

6.7 Summary

The internal DDoS attack is less common than the regular DDoS attack, and is also harder to detect and protect against. SDN switches can be utilized to monitor internal network flows. Besides, the regular SDN switches can be used to copy network packets to monitor by developing a custom action plugin. The OpenFlow framework supports custom plugins, which can be used for probabilistic packet forwarding. In this work, we present two problems for assigning flows to monitors and selecting the best locations for the monitors. We compare the performances of the proposed placement policy with an existing greedy approach. Simulation results show that our proposed approach works better than the greedy approach. In the next chapter, we consider the scenario where the available monitors are not capable to monitor all the flows in a network.

CHAPTER 7

SAMPLING RATE DISTRIBUTION FOR FLOW MONITORING AND DDOS DETECTION IN DATACENTER

In Chapter 6, we have assumed that the available monitors can process all the flows in a datacenter. In this chapter, we consider that the capacities of the monitors are smaller than that is needed to monitor all the flows. Therefore, we subsample the flows to fit in the capacity of the monitors. We investigate the relationship between the sampling rate and the DDoS detection rate. Then, we formulate an optimization problem for finding an optimal sampling rate distribution and solve it using mix-integer linear programming. The research in this chapter has been published in [Related11, Related12].

7.1 Introduction

A denial-of-service (DoS) attack is a cyber attack that aims to make a machine or service temporarily unavailable to its users. Attackers send a huge number of requests to the victim machine. There are several types of DoS attacks, including SYN flood and UDP flood [5]. The goal of these attacks is to fill-out the limited slots for the half-open connection so that the server cannot accept a new TCP connection. The purpose of the UDP flood attack is to consume all of the available network bandwidth of the victim. In this attack, the payloads

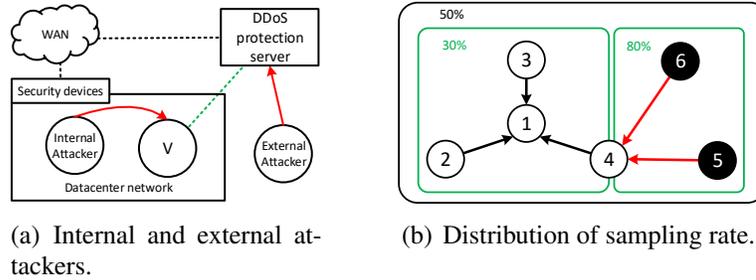


FIGURE 7.1: Attacker and sampling rate distribution.

of the attack packets are kept large. In a distributed denial-of-service (DDoS) attack, many attackers - controlled by a master - simultaneously send requests to the victim.

Based on the locations of attackers, we can divide DDoS attacks in a datacenter into two types: internal and external, as shown in Fig. 7.1(a). In external DDoS attacks, the attackers reside outside the datacenter. A commercial DDoS protection service cannot defend against the internal DDoS attacks. The filter-based DDoS attack defense mechanisms such as [36] also cannot protect against internal DDoS attacks. The clients of a commercial DDoS protection point their domain to the commercial DDoS protection server. The protection server creates a private connection with the client's server and passes all the requests and responses between the client's server and users. When the attackers know the internal or the public IP address and send attack packets to that IP address, the packets do not travel through the commercial DDoS protection server. The security modules of a datacenter usually remain in the aggregation or core layer of the network. Therefore, internal flows usually do not go through the security modules. As a result, internal flows are not monitored.

One way to protect the server against internal DDoS attacks is to block all internal flows, but there are some applications, including Hadoop and Spark, that need internal communication. We conducted an experiment to show how an internal DDoS attack affects

This research was supported in part by NSF grants CNS 1824440, CNS 1828363, CNS 1757533, CNS 1629746, CNS 1651947, and CNS 1564128.

Table 7.1: Effect of SYN flood attack.

Measurements	Normal	Attacked
Killed map tasks	1.18	3.66
MapReduce success rate	100%	50%
Total time for map (ms)	1122082	2116012
Total time for reduce (ms)	360243	438762

Hadoop. Table 7.1 shows the effect of a SYN flood attack on the Hadoop master from fourteen bots. The bots produce a SYN attack using hping3 tools. The bots fire SYN packets with an interval of 1 ms. Each packet is 66 bytes long. With this low attack rate, the mapper time increases to almost double that of without an attack. With the attack, the MapReduce job failed 50% of the time. Details will be presented in Section 7.5.

To protect against internal DDoS attacks, we need to monitor all the internal flows. To monitor all the internal flows, unused virtual machines (VMs) are used as monitors. The monitors run some machine learning approaches to detect attack behavior in the flows. Obviously, there is a limited number of VMs and capacity of a VM is limited. We assume a VM can monitor a limited number of flows. If the number of flows is larger than the total capacity, then each flow is partially copied to the monitors. The partial copy is done probabilistically. Flow copying is done by the software defined networking switches. When a flow is sub-sampled, the DDoS attack detecting rate is also reduced. Sub-sampling a flow is probabilistically taking a portion of the packets in that flow. The rate of detection also depends on the type of attack.

Let us assume that we have a DDoS detection mechanism that works based on the arrival time intervals of packets. If a flow has low arrival time intervals, then it is classified as a DDoS flow. In the datacenter, we have five flows, as shown in Fig. 7.1(b). We assume all the flows have the same data rate. Flows $2 \rightarrow 1$, $3 \rightarrow 1$, and $4 \rightarrow 1$ are normal flows that have a high interval of packets. Other flows, $6 \rightarrow 4$ and $5 \rightarrow 4$, are DDoS flows that have a small interval of packets. The monitoring system can handle 50% of the total packets. If we sample 50% from each flow and send them to the monitor, the packet arrival

interval will increase for the DDoS flows. Therefore, flows $6 \rightarrow 4$ and $5 \rightarrow 4$ might be classified as normal flows. Let us divide the flows into two groups such as flows $2 \rightarrow 1$, $3 \rightarrow 1$, and $4 \rightarrow 1$ in group 1 and flows $6 \rightarrow 4$ and $5 \rightarrow 4$ in group 2. If we sample 30% from group 1 and 80% from group 2, then the overall sampling rate remains 50%. A higher sampling rate for group 2 increases the probability of DDoS flow detection. On the other hand, 30% samples from group 1 slightly reduce the DDoS detection rate. So, the overall rate is higher than the previous approach.

It is challenging to find a perfect grouping of flows. The relationship between the detection rate and the sampling rate is important. By using the relationship between the sampling rate and the detection rate of each group, we can get an optimal sampling rate distribution with the help of a traditional optimization solver in the given groups of flows. There is also a trade-off between the number of groups and computation complexity of finding an optimal sampling rate distribution. Therefore, a grouping of flows based on their characteristics and a uniform sampling rate for each group might be a good solution. In this paper, we focus on finding an optimal sampling rate distribution among groups of flows. To the best of our knowledge, we propose a group-based sampling rate distribution for the first time. Our main contributions in this paper are the following:

1. We propose a flow grouping approach based on the behavioral similarity of VMs.
2. We study the relationship between the sampling rate and DDoS detection rate using an artificial neural network second order regression.
3. Using the relationship between the sampling rate and DDoS detection we find the sampling rate distribution among groups that produces the maximum detection rate.
4. We conduct experiments in a datacenter with 15 SDN switches and 36 servers with Hadoop and Spark.

The remainder of this paper is arranged as follows. In Section 7.2, we discuss some related works and their limitations. In Section 7.3, we present the network model and an overview of the monitoring system. Section 7.4 contains our proposed sampling rate distribution approach and a formal definition of the problem. In Section 7.5, we present some experimental results that strengthen our proposed solutions. Finally, Section 7.6 concludes our paper.

7.2 Related Work

There exist many works on DDoS detection methods and flow monitoring. Firstly, there are many statistical and machine learning methods, including correlation, entropy, divergence, cross-correlation, co-variance, and information gain that detect anomalous DDoS requests [8]. In [11], a DDoS detection mechanism is proposed based on Artificial Neural Networks. There are several other methods to detect DDoS attacks, including [12, 13, 59].

There are also many works on flow monitoring and packet sampling. In [60], an adaptive non-linear sampling method for passive measurement is proposed. The system dynamically sets the sampling rate for a flow depending on the number of packets. The sampling rate diminishes with the increase of packet count in a flow. Their approach samples a small flow with a large sampling rate and samples a large flow with a small sampling rate, both of which provide good accuracy. Several counter-based sampling techniques are proposed in [61]. The systematic count-based technique selects packets through a deterministic and invariable function based on the packet position. The systematic time-based technique selects packets based on arrival time with a deterministic interval. The random count-based technique selects the starting points of the sampling intervals randomly. Adaptive linear prediction [62] increases (or decreases) sampling rate in order to identify the new traffic pattern when the network activity increases (or decreases).

In [63], a multiadaptive sampling technique is proposed. In addition to adjusting the sampling rate, it adjusts the sample size. This approach avoids overload of the measure-

ment points. When there is less activity, the multiadaptive technique decreases sampling rate and increases sample size to get more information about the network. In [64], the authors propose a distributed and collaborative monitoring system called DCM. DCM enables flow monitoring tasks and balances the measurement load at the switch. DCM can monitor different groups of flows using different actions. The system uses bloom filters to represent rules which use a small amount of memory in switch, but monitoring is very limited and cannot produce good accuracy. In [65], the authors use a deep neural network model to build an intrusion detection system. They use six basic features for SDN network without any sampling. In [66], the authors propose a flow monitoring algorithm to record some features of the DDoS attack traffic on the data plane.

We discussed three types of existing systems: (1) statistical approaches that analyze packets to detect and block attack traffic (2) dynamic sampling of packets based on traffic volume and other parameters, and (3) machine learning based systems for detecting attack packets without sampling the packets. Since, there are some differences among the attackers, victims, and other nodes in the way they communicate with each other or the master, we can use that for grouping their communications/flows. None of the existing works discussed above utilize grouping the flows based on characteristics of their source and destination. In addition, the detection rate of DDoS behavior depends on the sampling rate. The effect of the sampling rate on the detection rate also depends on the characteristics of the flows. Therefore, a group-based sampling rate distribution is necessary to increase the detection rate.

7.3 System Model

In this section, we describe the datacenter structure, attack model, and monitoring system. The overall system works in three steps. In the first step, the controller divides the flows into several groups. In the second step, packets in all flows in each group are analyzed to find the relationship between sampling rate and accuracy. Then, an optimization problem

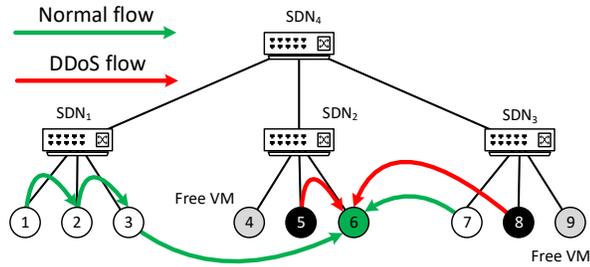


FIGURE 7.2: Monitoring system.

is solved to find an optimal sampling rate in each group. In the third step, the monitoring system optimally assigns each flow to a monitor.

7.3.1 Datacenter Network Model

Our datacenter is composed of virtual machines (VM), physical machines (PM), top of rack (TOR) switches, SDN switches, users, attackers, and a victim. The victim uses DDoS protection service from a commercial DDoS protection service provider. The datacenter also has some security modules that are installed at the top layer of the network. Therefore, all the external DDoS attack traffics are caught by DDoS protection server or security modules.

7.3.2 Attack Model

We assume that the attackers reside in the same datacenter as the victim. Attackers are controlled by a master who may reside outside of the datacenter. The attacker is capable of launching SYN flood and UDP flood. The attackers cannot spoof IP addresses and launch an amplification attack. This is because, in most of the datacenters, the VMs are not allowed to spoof others' IP addresses and spoofed packets are blocked at the hypervisor level. Amplification attacks are not possible nowadays because, new versions of the operating systems do not reply to an ICMP/ping packet which is destined to a broadcast address. The attackers target the master of Hadoop or Spark system and launch a mix of SYN flood and UDP flood. The goal of the attacker is to increase the computation time or hamper the operation of the Hadoop or Spark.

7.3.3 Monitoring System Overview

Fig. 7.2 depicts the overall monitoring system. VM 6 is the victim and master. VMs 3 and 7 are the workers (e.g. data nodes). Attackers 5 and 8 send DDoS packets to the master 6. VMs 4 and 9 are unused VMs and they will be used as monitors. Let the capabilities of all VMs be the same and all flows have equal data rates. A VM can monitor only one flow. There are 6 flows in the datacenter. To monitor all the flows we need at least 6 monitors. We have only 2 monitors, so the system sub-samples the flows and copies 33% of the total packets probabilistically to the monitors. 33% of the packets can be copied in many different ways from the flows. However, the copied packets in each flow increase the network overhead. The network overhead for copying a flow is the data rate of the copied flow multiplied by the number of hops it travels. For example, if the flow $1 \rightarrow 2$ is monitored in VM 4, then network overhead is $r_{12} \times 0.33 \times 3$, where r_{12} is the data rate of the flow $1 \rightarrow 2$. $r_{12} \times 0.33$ is the data rate of the copied flow, and 3 is the number of hops that the copied packets travel. Flow $1 \rightarrow 2$ can be copied from SDN_1 and copied flow travels path $SDN_1 \rightarrow SDN_4 \rightarrow SDN_2 \rightarrow 4$. Therefore, the total increased network overhead for copying all flows is:

$$C = \sum_{f_{ij} \in F} r_{ij} s_{ij} \times \min_{p \in i \rightarrow j} dist(p, A(f_{ij})). \quad (7.1)$$

Here, F is the set of all flows, s_{ij} is the sampling rate of flow f_{ij} , and A is an assignment function that maps a flow to a monitor VM. The goal of the monitoring system is to find an assignment that produces the minimum network overhead.

The assignment process of the monitoring system consists of two steps. In the first step, from the flow graph $G = (V, E)$ it constructs a bipartite graph $G' = (V', E')$, where $V' = V_1 \cup V_2$, V_1 is the set of flows, and V_2 is the set of monitor VMs. From each flow, edges are added to all flow VMs. The weight of an edge is the increased network overhead for assigning that VM to the monitor.

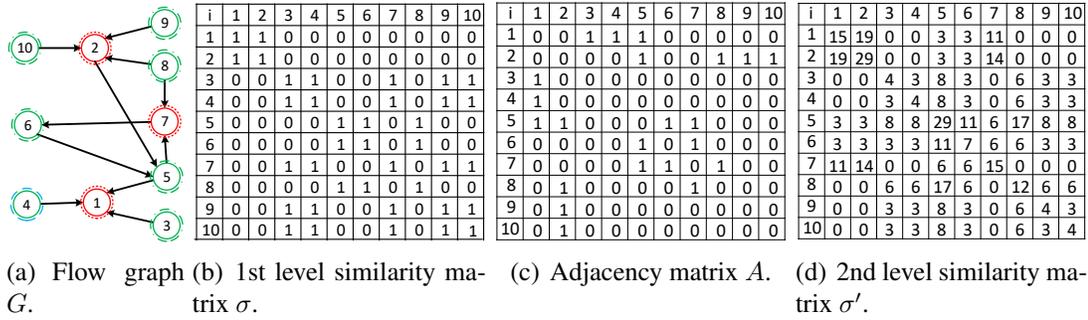


FIGURE 7.3: An example of flow grouping using second level behavioral similarity.

In the second step, it transforms the G' to a flow graph $G'' = (V'', E'')$. V'' contains all the vertices of V_1 and V_2 , including a source S and destination D . Then, edges from S to all nodes in V_1 are added with capacities equal to the data rates of the corresponding flows. Edges are added from all nodes in V_2 to D having a capacity of the data rate the monitor can handle. The costs of new edges are zeros. The cost of other edges are set to the increased network overheads. Then it finds a max-flow min-cost paths and the edges with flow indicate the flow assignment to the monitors [67].

7.4 sampling rate Distribution

In this section, we present the sampling rate distribution approach among the flows. We first divide the VMs into several groups. Then, we group the flows based on the VM groups. Finally, we assign sampling rates to each group.

7.4.1 Flow Grouping

We first group the nodes/VMs in the datacenter and then group the flows by the groups of nodes. We use second level behavioral similarity to classify the nodes. The uniqueness of second level behavioral similarity is that the behavior of neighbors determines the behavior of a node along with its own behavior. Therefore, two nodes that have similar type of nodes in their neighborhood are considered similar. When a group of bots attack, they send a lot of traffic to the victim. The victims also gets a lot of traffic from users. Therefore, if there are multiple victims in a network, they should be similar according to second level

behavioral similarity because they will have a lot of traffic from separate groups of bots and users. On the other hand, the master will receive traffic from the bots but not from the users. Therefore, all of the masters in a network might be similar according to the second level behavioral similarity. Thus, traffics related to similar nodes are treated similarly which is not possible in any other type of grouping approaches including machine learning and other attribute based approaches [68].

We classify the nodes into three types: sender, receiver, and forwarder. A sender (or receiver) node's incoming data rate is much lower (or higher) than the outgoing data rates. A forwarder node has similar incoming and outgoing data rate. We classify the nodes into these three categories based on general behaviors of attackers and victims. Usually, attackers send a lot more traffic than they receive. A victim usually is overwhelmed by traffic and thus its incoming traffic is much higher than the other nodes. Forwarder nodes are usually neutral nodes which are neither attackers or victims. The types are considered as first level similarity and similarity among the neighbors is considered second level similarity. We use the second level similarity to group nodes because nodes that have similar types of neighbors are subject to suffer similarly by their neighbors. The type of a node n , $T(n)$ is defined as the following:

$$T(n) = \begin{cases} \text{receiver,} & \text{if } r_i(n)/r_o(n) > \theta, \\ \text{sender,} & \text{if } r_i(n)/r_o(n) < \frac{1}{\theta}, \\ \text{forwarder,} & \text{otherwise.} \end{cases} \quad (7.2)$$

Here, $r_i(n)$ and $r_o(n)$ are the incoming and outgoing data rates of node n . θ is a system-defined threshold. For example, if $\theta = 2$, then a node will be classified as a sender (or a receiver) if its outgoing data rate is higher than double (or less than half) of its incoming data rate. If the incoming data rate is less than double and higher than half of the outgoing data rate, then it is a forwarder node. The value of θ can be set based on statistics of the nodes in the network. It should not be close to 1 because a slight change in flow rates

might change the type of the nodes. On the other hand, it should not be a large number. In that case, most of the nodes will be classified as forwarder nodes.

Next, we define the first level similarity matrix σ . σ is a matrix containing similarity between every pair of nodes based on the type of nodes. $\sigma[m, n]$ contains the similarity between node m and n . If the types of nodes n and m are the same, then $\sigma[m, n]$ is equal to 1. If the types of nodes n and m are different, then $\sigma[m, n]$ is equal to 0. σ can be expressed as the following:

$$\sigma[m, n] = \begin{cases} 1, & \text{if } T(m) = T(n), \\ 0, & \text{otherwise.} \end{cases} \quad (7.3)$$

Then, we calculate the second level similarity matrix σ' . We use regular equivalence to calculate σ' . The regular equivalence is widely used in social network analysis. In regular equivalence, two nodes are similar if they have many neighbors who are themselves similar. $\sigma[m, n]$ can be expressed as the following [69]:

$$\sigma'[m, n] = \alpha \sum_{kl} A[m, k]A[n, k]\sigma[k, l] + \delta[m, n]. \quad (7.4)$$

Here, A is the adjacency matrix of the flow graph. α is an eigenvalue of σ . $\delta[m, n]$ is 1 (or 0) if m and n are equal (or different). Equation 7.4 can be expressed as the following matrix multiplication form:

$$\sigma' = \alpha A\sigma A + I. \quad (7.5)$$

Fig. 7.3(a) shows an example of a flow graph with 10 nodes. A directed edge indicates a flow direction. Inspecting visually, we can say that nodes 1 and 2 look similar because they have incoming/outgoing flows from a similar type of node. They could be masters of a distributed application or victims of a DDoS attack. So, we should treat the incoming flows of these nodes the same way.

In this example, we assume the data rates are equal for all flows and θ is 2. For node 1, the numbers of incoming flows ($r_i(1)$) and outgoing ($r_o(1)$) flows are 3 and 0,

respectively. $r_i(1)/r_o(1) = 3/0 = \infty$, which is greater than 2. Therefore, node 1 ($T(i)$) is a receiver. Similarly, node 2 is also a receiver. According to the first level similarity, nodes 1 and 2 are similar. The first level similarity cannot produce what we expect. If we look at node 10, it has 1 incoming flow and 0 outgoing flows. Therefore, node 10 is also similar to nodes 1 and 2, which is unexpected. Nodes $\{1, 2, 7\}$, $\{3, 4, 8, 9, 10\}$, and $\{5, 6\}$ are receivers, senders, and forwarders, respectively. Fig. 7.3(b) shows the similarity matrix σ . Therefore, the similarity among nodes 1, 2, and 7 is 1. Next, we calculate second level similarity among nodes 1, 2, and 7. The maximum eigenvalue of the Q is $\alpha = 2.75$. Using Equation 7.4, we calculate $\sigma'[1, 2]$. Sender neighbors of nodes 1 and 2 are $\{3, 4\}$ and $\{8, 9, 10\}$. Forwarder neighbors of nodes 1 and 2 are $\{5\}$ and $\{5\}$. Therefore, $\sigma'[1, 2] = 2.75 \times \{(2 \times 3) + (1 \times 1) + 0\} = 19.25$.

Similarly, we calculate that $\sigma'[1, 7] = 11$ and $\sigma'[2, 7] = 13.75$. So, we can see that nodes 1 and 2 are more similar than nodes 1 and 7 or 2 and 7. The similarity score between 1 and 3 is 0. This is because they are different in type and they do not have any similar neighbors. Using the matrix multiplication form, we calculate σ' (shown in Fig. 7.3(c), rounded to the nearest integer).

Then, we group the nodes using the hierarchical clustering algorithm. We formulate the distance matrix from the second level similarity matrix. The similarity values between a pair of nodes are subtracted from the maximum of the similarity values (except the similarity value of a node with itself) to get dissimilarities. The dissimilarities are used as distances for hierarchical clustering. The distance matrix is represented as follows:

$$D[m, n] = \begin{cases} \max_{i \neq j} \{\sigma'[i, j]\} - \sigma'[m, n], & \text{if } m \neq n, \\ 0, & \text{otherwise.} \end{cases} \quad (7.6)$$

Next, we use D to cluster the nodes into K groups. There is trade off between the number of groups and the execution time. When the number of groups is higher, the system takes a long time to find the optimal sampling rate. Therefore, the system needs

to adjust the value of K based on several parameters including the topology size, machine configuration, and performance of detection algorithm. Firstly, we use the hierarchical clustering algorithm. We set the maximum class to be the number of nodes. This clustering will group the most similar nodes in a cluster which will be the largest cluster. We pick that cluster and remove all the nodes in that cluster. Then, we repeat the process $K - 1$ to find $K - 1$ clusters. The rest of the nodes are grouped and treated as another cluster. The complete approach is shown in Algorithm 19. The procedure IN-FLOWS(c) returns the incoming flows of the nodes in set c . Details of the procedure are not shown due to limited space.

If we use $K = 2$, then we get groups $\{1, 2, 7\}$ and $\{3, 4, 5, 6, 8, 9, 10\}$. Therefore, the groups of flows are $\{3 \rightarrow 1, 4 \rightarrow 1, 5 \rightarrow 1, 8 \rightarrow 2, 9 \rightarrow 2, 10 \rightarrow 2, 5 \rightarrow 7, 8 \rightarrow 7\}$ and $\{2 \rightarrow 5, 7 \rightarrow 6, 6 \rightarrow 5\}$. If we use $K = 3$, then we get groups $\{1, 2, 7\}$, $\{4\}$, and $\{3, 5, 6, 8, 9, 10\}$. Therefore, the groups of flows are still the same because there are no incoming flows to node 4.

After grouping flows, we need to find the relationship between the DDoS detection rate and the sampling rate using an artificial neural network (ANN).

7.4.2 Relationship between Detection and Sampling Rate

To find the relationship between the sampling rate and the detection rate, we need to vary the sampling rate and record the detection rate for each flow group. We use a pre-trained model of ANN. The input of the ANN consists of the features of a certain number of packets. The packets are grouped based on the order they are received. The ANN uses two types of features: features of an individual packet and features of packets in a group. Features of an individual packet include the following:

- Protocol: The protocol of the packet. A protocol of a packet can be TCP, UDP, HTTP, or ICMP.
- Packet Size: The total size of the packet in bytes.

Algorithm 19 Flow grouping

Input: The adjacency matrix A , set of flows F , and number of clusters K .

Output: A group of flows.

```
1: Procedure: FLOW-GROUPING( $A, F, k$ )
2:   Calculate types  $T[n]$  from flows for each node  $n$ .
3:   Calculate  $\sigma$  from the types  $T$ .
4:    $\alpha \leftarrow$  the maximum Eigen value of  $\sigma$ .
5:    $\sigma' = \alpha A \times \sigma \times A + I$ .
6:   Calculate  $D$  from  $\sigma'$  using Equation 7.6.
7:   return GROUP( $D, k$ )
8: Procedure: GROUP( $D, k$ )
9:    $g \leftarrow \emptyset$ 
10:  for  $i = 1$  to  $K - 1$  do
11:     $C \leftarrow$  clusters using hierarchical clustering
12:     $c \leftarrow$  largest cluster in  $C$ 
13:    Remove all nodes in  $c$  from  $D$ .
14:     $g \leftarrow g \cup \text{IN-FLOWS}(c)$ 
15:     $r \leftarrow$  set of nodes that are not in any cluster
16:     $g \leftarrow g \cup \text{IN-FLOWS}(r)$ 
17:  return  $g$ .
```

- Entropy: The entropy is calculated over all bytes in a packet using the Tsallis method.

Features of a group of packets are the following:

- Variance in packet size: The variance of the packet size (in milliseconds) difference between two consecutive packets.
- Entropy: The entropy is calculated over all bytes in all packets in a group using the Tsallis method.
- Variance in arrival time: The variance of the time (in milliseconds) difference between two consecutive packets.

Though there is a protocol number for each protocol, we use an indexing method to provide a numeric value for each protocol. In our approach, DNS, TCP, UDP, ICMP, HTTP, DHCP, and SSH protocols are considered. All other protocols are considered to be

the same. To calculate the entropy of a packet we use the Tsallis entropy formula [70]. Let there be N bytes in a received packet p . Then, the entropy of the packet p is:

$$E(p) = \frac{1}{q-1} \left\{ 1 - \sum_{n=0}^{N-1} P(p[n])^q \right\}. \quad (7.7)$$

Here, P is the probability density function of the bytes of packet p . $p[n]$ denotes the n th byte of p . q is a real parameter which is called the entropic-index.

We split the packets in each flow by a 60-40 ratio. We keep the first 60% of the packets in each flow as the training packets and the rest as the test packets. Then from each flow, we divide the packets into non-overlapping groups based on arrival time. Each group contains w consecutive packets. Then the four features of all packets are organized in a specific order. Then, the three group features are appended. For example, if we have $w = 5$, then the total number of inputs of the ANN is $5 \times 4 + 3 = 23$. The class of each input is either regular or DDoS. The packets in the flows from the attackers to the victim are considered as DDoS packets. Other packets are considered as regular packets.

From the test packets, we pick a portion of the packets from a group. From each group, packets are picked with different probabilities. For example, if we have two ($w = 2$) groups of flows and from each group, we pick a packet with 10 different probabilities, then there will be $10 \times 2 = 20$ different test packet groups. From each test group, we formulate a test dataset. Using the retrained model, we calculate the DDoS detection rate of each test dataset.

The details of the ANN structure are shown in Fig. 7.4. We have $4w + 3$ inputs in the input layer. We have two hidden layers with w and 2 neurons. We use the most commonly used activation function, ReLU, as the activation function in our ANN. This structure works well to differentiate between DDoS and regular flows. This two-hidden layered ANN works much better than a one-hidden layered ANN. The relationship depends greatly on the machine learning model. If the model is very good and can detect the attack with

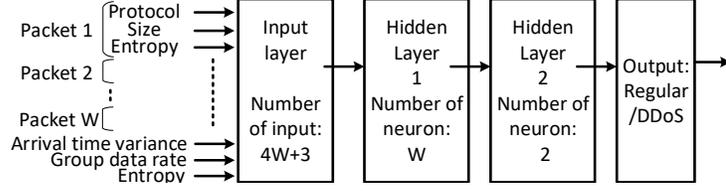


FIGURE 7.4: Structure of the ANN.

even 1% of the sampling rate, then all of the relationships will be straight lines parallel to the X-axis. In that case, any sampling rate is good. In practical it is impossible to find a perfect model because of the dynamic attack behavior. If the model is changed, then it is required to retrain using the train dataset. In this paper, we are not focusing on finding the ANN structure that performs best. We assume the machine learning model is not perfect. We are focusing on finding out how the sampling rate affects the DDoS detection rate.

After getting the DDoS detection rates of each test dataset, we apply second-order polynomial regression to find the relationship between the sampling rate and the DDoS detection rate for each group. Let there be K number of groups. We denote the relationship between the sampling rate and DDoS detection rate for a group k as $f_k(s)$. Here, $k = 0, 1, 2, \dots, K - 1$ is the group number and s is the sampling rate. The total data rate of all flows in group k is r_k . The problem can be expressed as the following:

$$\begin{aligned}
 \text{maximize: } & \sum_{k=0}^{K-1} f_k(s_k) \times r_k \\
 \text{subject to: } & \sum_{k=0}^{K-1} s_k \times \hat{r}_k \leq S, \quad \forall_k s_k > S_{min}
 \end{aligned} \tag{7.8}$$

Here, \hat{r}_k is the normalized data rate of group k . S is the maximum overall sampling rate that the system can process. S_{min} is the minimum sampling rate that produces good accuracy. We consider the sampling rates as integer-valued and the problem as a mixed-integer program. The problem can be solved using any optimization problem solver.

Table 7.2: Groups of nodes for Flows I using Algorithm 19

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
K=2	12, 33, 34	rest of the nodes				
K=4	12, 33, 34	16, 22, 26	2, 4, 20	rest of the nodes		
K=6	12, 33, 34	16, 22, 26	2, 4, 20	6, 8, 10, 14, 18, 24, 28, 30, 32	0, 1, 3	rest of the nodes
# of packets	793,340	476,068, 2,139,044 (for K=2)	314,351	1,008,455, 1,348,625 (for K=4)	60,142	280,028

7.5 Experiments

7.5.1 Experimental settings

We first present the datacenter structure. Fig. 7.5 shows the datacenter structure. The datacenter is composed of 35 servers, 15 SDN switches, and some regular L2 switches. All the servers except the gateway are Dell PowerEdge 210 (2 cores 2.4 GHz processor, 4 GB RAM, 500 GB HDD) servers. Each server has at least two gigabit Ethernet ports. The SDN switches are Pica8 p-3922.

We set up two networks: a control network and a data network. In the control network, all management ports of SDN switches and the SDN controller (gateway) are connected through an L2 switch. SDN switches are configured as the out-of-band controller, which means the control plane and the data plane are separated. The dotted lines in Fig. 7.5 show the control network. Therefore, our control network is a star topology. In the data network, the data ports of the SDN switch and the gateway are connected. The topology is a three-level complete binary tree topology. The gateway is connected with the root SDN switch and other servers are connected to leaf SDN switches. We use OpenDaylight [71] as the SDN controller, which is installed in the Gateway. For flow rule generation, we use the L2Switch plugin with OpenDaylight. Next, we set up a Hadoop cluster on server 34 and 16 other servers (all even numbers). Server 34 is configured as the namenode and even-numbered servers are configured as data nodes. We configure the Hadoop file system to have three replications of files and set yarn as the resource manager.

Then, we generate some text files for input to a MapReduce program. The text files are generated by taking a word randomly from a dictionary. In our experiments, we generated

Table 7.3: Groups of nodes for Flows II using Algorithm 19

	Group 1	Group 2	Group 3	Group 4	Group 5	Group 6
K=2	26, 33, 34	rest of the nodes				
K=4	26, 33, 34	12, 14, 20	2, 20, 26	rest of the nodes		
K=6	26, 33, 34	12, 14, 20	2, 20, 26	10, 16, 18, 22	6, 8, 24, 28	rest of the nodes
# of packets	649,829	384,961, 2,282,555 (for K=2)	597,192	495,014, 1,300,402(for K=4)	465,218	340,170

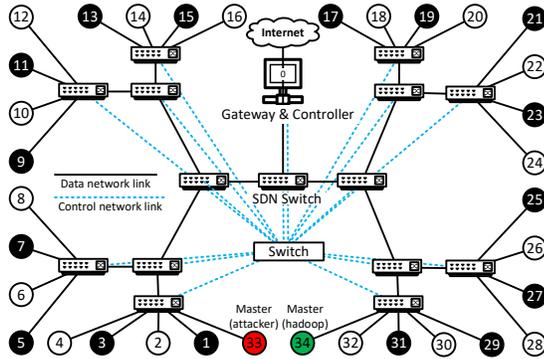


FIGURE 7.5: Datacenter topology.

100 text files each having a size of about 10 MB. Therefore, the total input is 1 GB of text files that are uploaded to the Hadoop file system. We run the WordCount or WordRank program on the input files to generate normal traffic. The MapReduce framework first splits the files by a new line and sends them to worker nodes for processing. This step creates much internal traffic. After processing parts of the data, the workers send their results to multiple workers depending on the keys of the result. This process also creates a lot of internal traffic. The final output from the reducer is also stored in the Hadoop file system. Because of three replication factor, writing to a file also produces much network traffic.

Then, we install the attacker programs in the 16 (odd numbered) servers. The attack master 33 runs a web service and the attacker program gets commands from it. The attacker also replies with the response to the attack master. The commands are usually Linux commands. We install popular attacking applications such as hping3 [72] and packETH [73]. Using hping3, we can launch SYN flood, UDP flood, and malformed packet attacks with different data rates. hping3 is not capable of randomizing the sending interval

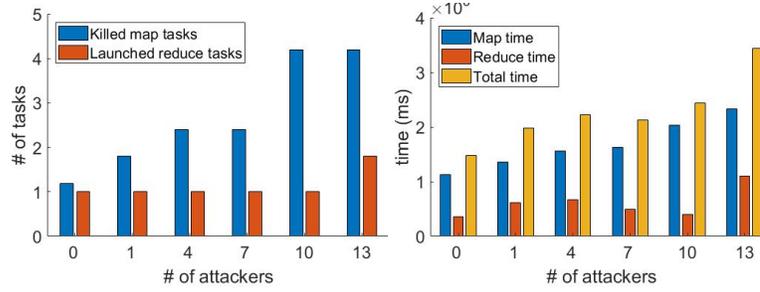
and packet size. With packETH, we can launch attacks with random sending interval and size. The attacks are targeted against the Hadoop master 34. The Hadoop master program listens on port 9000. Therefore, the attackers send the packets targeting port 9000.

We simultaneously run several WordCount programs and a mixed type of attack and capture incoming packets from the 34 servers (except the gateway) using tshark [74]. The packet capture files contain packets of all flows for 10 mins. The files contains about 30 million packets on average. We classify the flows from odd number servers to server 34 as DDoS flows. All other flows, including the flows due to the communication of attackers with the attack master, are considered as regular flows. Then, we train our ANN with the first 60% of the data (18 million packets). The rest of the data is grouped using our proposed grouping approach and relationship functions with the sampling rate are calculated. Finally, by solving an optimization problem, we find our desired sampling rate. We compare the uniform sampling rate with different numbers of groupings. We denote the flows with WordCount run as Flows I. We change the destination of the attack flows randomly and create another set of flows which is denoted by Flows II.

7.5.2 *Experiment results*

Firstly, we show a small scale DDoS attack targeting the Hadoop master (server-34). We start with one attacker and increase the number of attackers to sixteen. We run the MapReduce WordCount program on the generated 1 GB text files. Fig. 7.6 shows the effect of a SYN flood attack. We vary the number of attackers from 0 to 13 in 5 steps. The attack is produced using the hping3 tool. We keep the interval between two SYN packets as 1 millisecond (ms). Each packet is 64 bytes so that the data rate is 512 Kbps. All the measurements are averaged over 5 runs of the WordCount program.

Fig. 7.6(a) shows the number of killed map tasks and launched reduce tasks with different numbers of attackers. When the number of attackers increases, the number of killed map tasks increases. When there is no attack, the number of launched map tasks is



(a) Number of tasks vs. attackers. (b) MapReduce time vs. attackers.

FIGURE 7.6: Effect of small scale SYN flood attack.

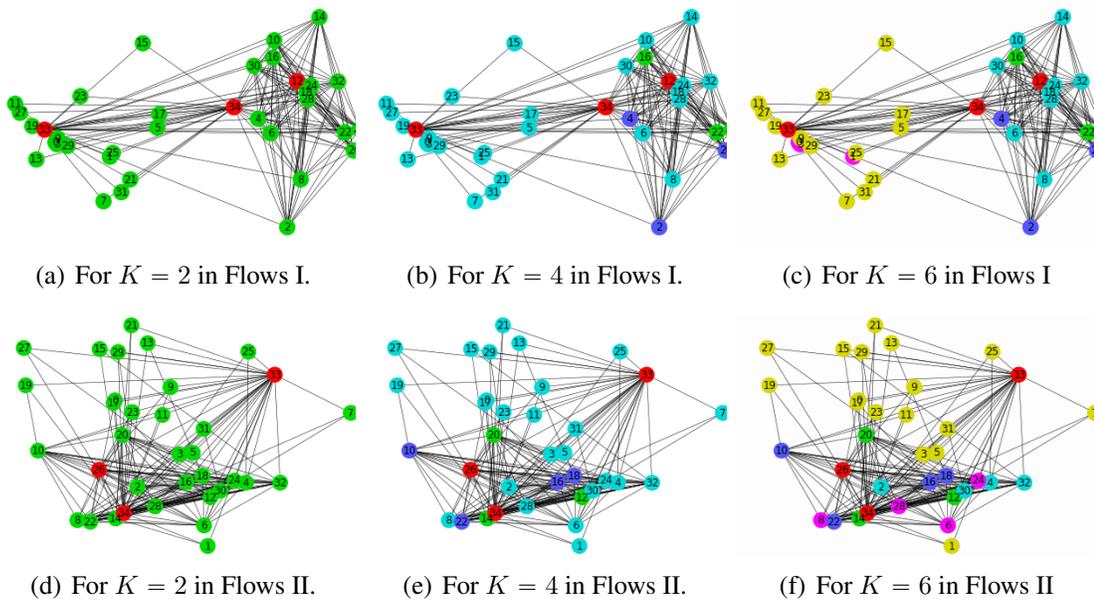


FIGURE 7.7: Grouping of nodes based on second level similarity.

101, and among them 1 task is killed. When the number of attackers is between 10 and 13, the number of launched map tasks is between 105 and 108, and among them the number of killed tasks is between 2 and 6. When the number of attackers is between 0 and 10, the number of launched reduce tasks is 1. When the number of attackers is 13, the number of launched reduce tasks is between 1 and 4 (1.8 on average).

Fig. 7.6(b) compares the time spent on map and reduce tasks with different numbers of attackers. When there is no attack, the total time spent on map tasks is 1,122,082

ms. When the number of attackers is 13, the total time spent on map tasks is 2,334,606 ms. So, the map time is more than double than that of with no attack. The reduce time without attack is 360,243 ms. Though the reduce time increases and then decreases when the number of attackers is between 1 and 10, it is normal. This is because the number of launched reduce tasks is 1 for 1 to 10 attackers. The variation of reduce time is due to the processing time of the data. When the number of attackers is 13, the number of launched reduce tasks is 2 and the reduce time is 1,106,689 ms, which is three times greater than that of with no attack. Therefore, due to 512 Kbps SYN flood attack, the map task is affected more than the reduce task. This is because there are more map tasks than reduce tasks. Due to the attack, the probability of communication failure is higher during the map period than during the reduce period. The MapReduce job never fails when the number of attackers is between 0 to 10. When the master is attacked by 13 attackers, the MapReduce job fails almost 50% of the time. When a MapReduce job fails in hadoop hosted by commercial service providers, it is least likely to fail because of back-up VMs which start on a failure. When this type of incident happens, the time to finish the job increases dramatically.

Fig. 7.7 shows the grouping of the nodes according to Algorithm 19. The groups of nodes are summarized in Tables 7.2 and 7.3. In Flows I, we observe that for $K = 2$, the attack master 33 and Hadoop master 34 are in the same group, which is expected. The other worker nodes except 12 and attackers belong to another group. Similar behavior is observed in other values of K . In Flows II, we also observe similar behavior for different values of K . Because of the large number of flows (total of 371 flows in the dataset), groups of flows are not shown here.

Next, we find the relationship between the sampling rate and the DDoS detection rate for each group. We train the ANN that we presented earlier with the first 60% of the data from each flow. Our ANN shows an accuracy of 93.38% with false positive and false negative rates of 0.8% and 5.79%, respectively. We keep $w = 10$ for the ANN. We vary

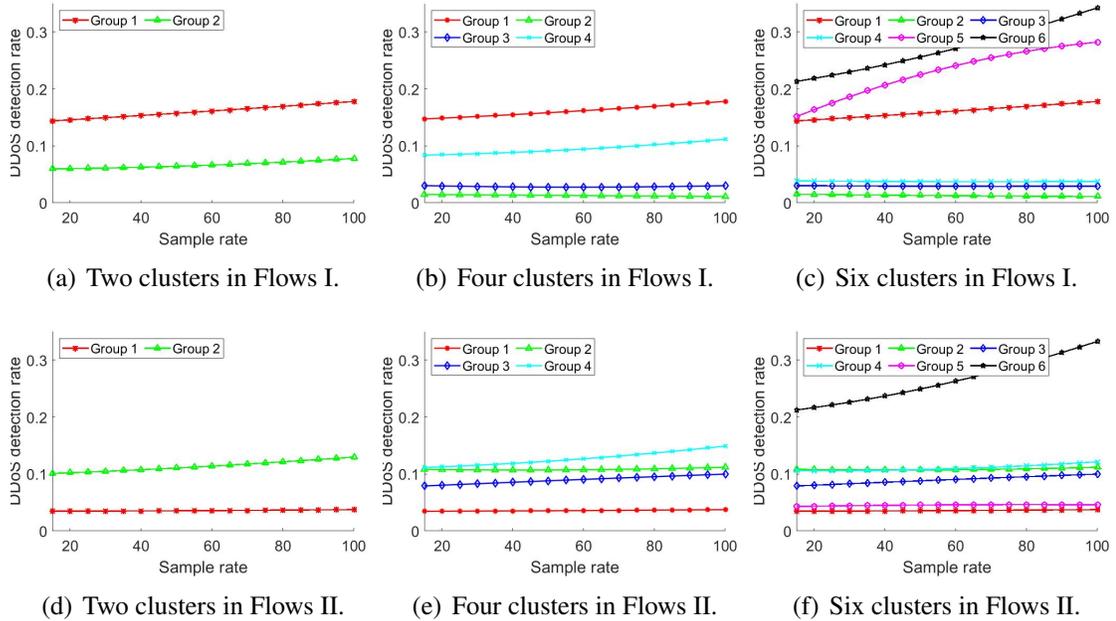


FIGURE 7.8: Sampling rate vs. DDoS detection rate.

the sampling rate from 20% to 100%. The DDoS detection rate for a group is the number of inputs that are classified as DDoS divided by the total number of inputs in that group. Using the DDoS detection rates and sampling rates, we find the relationship functions by second order polynomial regression.

Fig. 7.8 plots the relationship functions for different groups for different K values. Fig. 7.8(a) plots the relationship functions for $K = 2$ using Flows I. From the settings we know that all the attack flows belong to Group 1. Therefore, Group 1 shows the highest DDoS behavior. Group 2 also shows some DDoS behavior, which is false positive. Besides, some normal flows can also have similar behavior (entropy, variance of arrival time, etc.) to DDoS flows. It is impossible to build a model that detects DDoS with 100% accuracy. For $K = 2$ and $K = 4$, other groups (except Group 1) show DDoS behavior for the same reason. Most of the groups' (Groups 1 and 2 for $K = 2$, Groups 1 and 4 for $K = 4$, and Groups 1, 5, and 6 for $K = 6$) DDoS detection rates increase with the sampling rate. These flows are regular, having some similar characteristics to DDoS flows. Other groups' DDoS

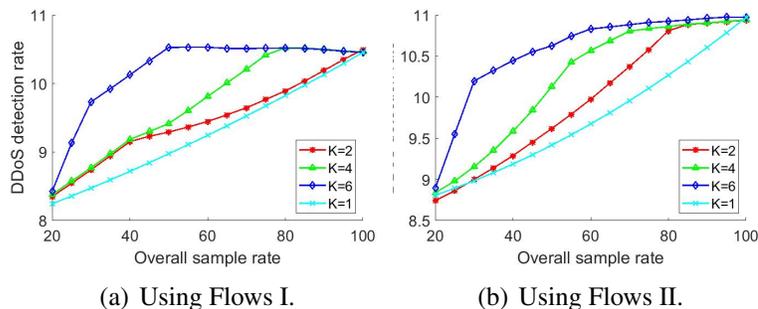
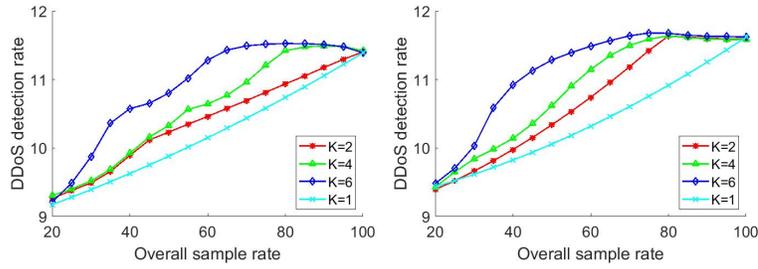


FIGURE 7.9: Uniform and grouped distribution.

detection rates decrease very slightly with the sampling rate. This is because more packets from a flow help the ANN find more accurate classification. These flows are regular and some packets are classified as DDoS packets by ANN. Fig. 7.8(d) shows the plot of the relationship functions for $K = 2$ using Flows II. As the destination of attack flows is no longer the server 34 in Flows II, Group 1 in Flows I has less DDoS behavior than Group 1 in Flows II.

Finally, we formulate the problem using the relationship functions in each group. We keep the minimum sampling rate at 20%. This is because a sampling rate less than 20% does not produce good accuracy. Using the CVXPY optimization library in python, we solve the problem. Part of the solution is summarized in Tables 7.4 and 7.5. We keep all the sampling rates as integers. Therefore, the overall sampling rate 20 refers to the sampling rate greater than or equal to 20 but less than 21. The groups that have higher DDoS detection rates are assigned higher sampling rates. For example, in Flows I when $K = 2$, to achieve an overall sampling rate of 50%, our approach assigns 100% and 32% to the flows in Group 1 and Group 2, respectively. This kind of assignment produces a better overall detection rate.

We compare the performances of the grouping approach with those of the uniform (no grouping) sampling approach. Fig. 7.9 shows the comparison between our flow grouping approach and the non-grouping approach. When $K = 1$, all the nodes belong to a group



(a) Attacks to the Spark master. (b) Attacks to random workers.

FIGURE 7.10: Uniform and grouped distribution in Spark.

Table 7.4: sampling rates for different K for Flows I.

Sample rate	Group sampling rates		
	K=2	K=4	K=6
20	23, 20	23, 20, 20, 20	20, 20, 20, 20, 68, 20
40	97, 20	97, 20, 20, 20	63, 20, 20, 20, 100, 100
50	100, 32	100, 20, 20, 40	100, 20, 20, 20, 100, 100
60	100, 46	35, 20, 20, 100	100, 21, 100, 21, 100, 100
80	99, 74	100, 20, 35, 100	100, 20, 35, 100, 100, 100
100	100, 100	100, 100, 100, 100	100, 100, 100, 100, 100, 100

Table 7.5: sampling rates for different K for Flows II.

Sample rate	Group sampling rates		
	K=2	K=4	K=6
20	23, 21	20, 20, 20, 22	20, 20, 20, 20, 68, 20
40	23, 46	20, 20, 20, 22	63, 20, 20, 20, 100, 100
50	22, 59	20, 20, 20, 67	100, 20, 20, 20, 100, 100
60	22, 72	20, 20, 21, 89	100, 21, 100, 21, 100, 100
80	21, 98	20, 90, 100, 100	100, 20, 35, 100, 100, 100
100	100, 100	100, 100, 100, 100	100, 100, 100, 100, 100, 100

and the sampling rate is distributed uniformly over all flows. When $S = 100\%$, the overall DDoS detection is the same for all K values. In this case, all the groups are assigned to a 100% sampling rate, which is uniform. In other cases, the higher the values of K , the higher the DDoS detection rate. This is because the nature of the flows in each group is different. Detection of DDoS behavior in some of the flows is more sensitive to the sampling rate. At some points, the overall DDoS detection rate is higher than the 100% overall sampling rate. For example, in Flows I for $K = 6$, the overall DDoS detection rate is 10.54%, which is higher than the detection rate at 100% sampling rate (10.45). If we look at the sampling rate distribution, we see that Groups 2, 3, and 4 are assigned the

lowest sampling rates, which produce some false positives and the total DDoS detection rate becomes higher. We observe similar behavior in Flows II.

We also conduct experiments by running WordCount in Spark. We set up Spark master on server 34. Workers are set up on all other even number servers (server-2, server-4,... server-32). We run the WordCount program, which comes with the documentation of Spark. Spark execution is more network intensive and WordCount takes longer than MapReduce WordCount. Fig. 7.10 shows the comparison between our flow grouping approach and the uniform grouping approach in Spark. We observe similar behavior to Hadoop in Spark.

We compare our proposed system with the uniform sampling distribution and k -means grouping approaches under the same settings. It is hard for our work to have the same settings and parameters as the existing works. If we change some of the parameters or settings, then the originality of them is hampered. Fig. 7.11 shows the comparison between the proposed flow grouping approach and the k -means clustering approach. We use Flows I for this experiment. We use in-degree and out-degree as features of the nodes. By using k -means clustering, we divide the nodes into K number of clusters. Then, incoming flows of each cluster are grouped together. We observe that the k -means grouping produces better DDoS detection rates than the proposed approach for $K = 2$ and 4. The k -means clustering cannot divide nodes into more than 4 clusters. Our proposed approach with $K = 6$ outperforms the k -means with $K = 4$.

The proposed group-based approach is always better than the uniform sampling rate distribution for different MapReduce applications and attack strategies. The grouping approach does not produce an optimal grouping of flows. Therefore, we can conclude that our sampling rate distribution approach works better than the uniform sampling rate distribution. The behavioral similarity-based approach can produce better grouping of the flows than the in-degree and out-degree based grouping approach when the number of groups is large.

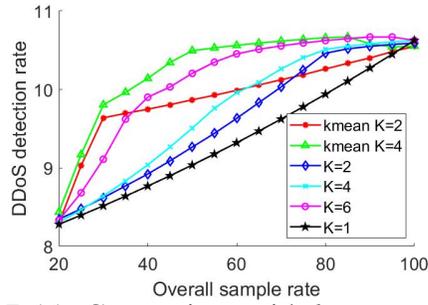


FIGURE 7.11: Comparison with k -means clustering.

7.6 Summary

The internal DDoS is less common than the external DDoS attack. Compared to the external DDoS attack, it is also harder to detect and protect against. We can use the functionalities of SDN switches to monitor internal network flows. In this work, we present a framework for flow monitoring in a datacenter. We present a flow grouping approach based on behavior similarities among the virtual machines. We formulate an optimization problem for sampling rate distribution and solve the problem with an optimization solver. We conduct all the experiments in our datacenter and compare performances with different granularity levels including the uniform sampling rate. Our experiments show that increasing the number of groups produces better detection rates than the uniform packet sampling approach. In the next chapter, we are going to explore more optimizations related issue in the context of link flooding attacks or regular congestion.

CHAPTER 8

MINIMIZING THE NUMBER OF RULES TO MITIGATE LINK FLOODING ATTACK IN SDN-BASED DATACENTERS

In this chapter, we focus on optimizing the network performance in the context of link flooding attacks and regular congestion in a network. We aim at minimizing the number of rule changes while redirecting some of the traffic from the congested link in a software defined network. We formulate two problems to minimize the number of rule changes to redirect traffic. The first problem is the basic and it considers a congested link and a flow to direct. We provide a Dijkstra-based and a rule merging based solution to the problems. The second problem considers multiple flows and we propose flow grouping and rule merging based solutions. Some of the parts of this chapter is published in [Related13] and submitted to [Submitted3].

8.1 Introduction

Congestion in link in a datacenter has become a common and serious problem nowadays. Link congestion can occur by regular traffic or a link flooding attack (LFA). In LFA, the bots generate traffic that is destined to decoy servers and the packets congest at least one of the important links on the paths to the victim. Nowadays, software defined network-

ing (SDN) switches are taking place of the regular routers in datacenters. In the SDN architecture, a centralised software, called the controller, controls all the actions of the SDN switches. This centralised architecture has opened opportunities to defend against the LFA and regular congestion easily and effectively.

Let us consider the network in Fig. 8.1(a) which consists of four SDN switches (a , b , c , and e), two regular switches/routers (x and y), three sources (s_1 , s_2 , and s_3), and two destinations (d_1 and d_2). There are three flows $f_1 = s_1 \rightarrow d_1$, $f_2 = s_2 \rightarrow d_2$, and $f_3 = s_3 \rightarrow d_1$. Link (c, e) is congested by other flows which are not shown in the figure. We need to redirect flow f_1 to prevent link congestion. There are multiple possible redirection points and ways to redirect f_1 . A redirection point is the farthest common SDN switch from the source between the new path (after redirection) and the old path. For f_1 , the redirection points can be nodes a and c . If we consider node a as a redirection point, then we can redirect f_1 through the path $\{a, b, y, e, d_2\}$ and we need new rules at node a and b . The forward rule of f_1 from y to e already exists at y . Therefore, in this way, we need to add two rules and the number of increased hop is zero. If we consider node c as a redirection point, then we can redirect f_1 through the path $\{a, x, c, y, e, d_2\}$ and we need a new rule at node c . Therefore, in this way we need to add a rule and the number of increased hops is one.

To observe the importance of the number of rules, we conduct a small experiment on a Pica8 P-3297 SDN switch at our datacenter. We observe that when the number of rules is greater than 4000 (see Fig. 8.1(b)), the transmission delay between two machines (2-hops away) jumps up about 1 ms. The delay keeps increasing with the number of rules after that. This small experiment shows the importance of minimizing the number of rules and motivates us to conduct research on this topic. We also conduct experiments to observe the interruption of flow due to changes in different numbers of rules, which are presented later in this paper.

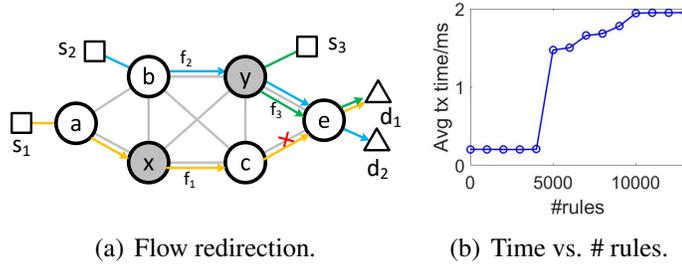


FIGURE 8.1: (a) Flow redirection due to LFA and (b) transmission time vs. # rules.

In this paper, we focus on mitigating LFA and regular congestion by redirecting some flows. When the number of rules are higher the controller spends more time to change resulting interruption in flow. Besides, if the number of rules becomes higher than the fast accessible memory, then the delay increases. To minimize the changes in rules the flow may travel a longer path which may increase delay but overall delay due to no overloaded switches and interruption is low. We formulate a basic problem to illustrate the basic solution and the concept of core-tree (CT). This problem considers one congested link and one flow to redirect. We provide a Dijkstra-based solution for this problem. We formulate another problem for multiple flows redirection considering one congested link. This problem is a NP-hard problem and we provide a grouping based solution. The main contributions of our paper are:

1. We study a basic problem of re-directing a flow by the alternate path with the minimum number of added rules and propose a Dijkstra-based solution.
2. We study another optimization problem to minimize the number of rules for the scenario where there are multiples flows and a congested link. We provide two solutions for the problem.
3. We provide extensive simulations and experimental results and compare it with existing approaches.

The remainder of this paper is arranged as follows. Section 8.2 presents some related works. In Section 8.3, we present the system and the attacker model. In Section 8.4, we present the basic problem of redirecting a flow and propose the Dijkstra-based solution. Section 8.5 contains the problem of redirecting multiple flows and solutions. Section 8.6 and Section 8.7 present the simulation and experimental and results. Finally, Section 8.8 concludes our paper.

8.2 Related Work

Unlike networks with regular topologies where re-rerouting under attacks and failures can be mitigated using inherit topology redundancy and structure [75], in regular networks with irregular topologies. There are three types of researches defending LFA and link congestion in SDN. Firstly, there are some statistical methods, including correlation, entropy, covariance, divergence, cross-correlation, and information gain-based system to detect attack traffic [8]. Other types of congestion mitigation systems include [41, 42, 43, 44] that are based on multi-path routing. These schemes either do not consider SDN switches or the number of rules.

Secondly, in [76], authors propose a hybrid and scalable SDN datacenter using both wireless and wired connections. They formulate a routing minimization problem to minimize the total number of installed rules in switches. They propose a heuristic algorithm to find a routing path, comprised of wired and wireless links with rule aggregation. In [77], authors propose an approach to ensure traffic reachability if any single link fails. They redirect traffic on the failed link to SDN switches via pre-configured IP tunnels. By configuring multiple backup paths they can mitigate congestion very fast. Some other work, such as [78], considers the problem of minimizing the number of rules in SDN switches and propose a heuristic algorithm that creates a reduced representation of rules in the SDN switches in network. They do not consider link congestion or attack scenario. In [79], authors propose a DoS attack on controller and link by dynamically rerouting potential

malicious traffic, adjusting flow timeouts, and aggregating flow rules. In [80], the authors proposed an efficient flow forwarding scheme called PASR which can learn the flow path information and implement flow aggregation. They present an intelligent encoding algorithm to minimize the number of rules.

Finally, in [81], the authors considers multiple failures of links and multi-flows rerouting in SDN environment. They propose a model for communication overhead between controller and switch during flow rerouting to minimize flow rules. They formulate the problem as a 0 – 1 nonlinear programming model and solve the model using decomposition based on Lagrange relaxation. In [82], authors propose a multicast routing model for multiple multicast requests to minimizes the number rules. They formulate the proposed model as an integer linear programming (ILP) problem and solve using traditional solvers. In [83], authors propose an ILP to minimize the total cost which is a combination of flow table space and average transmission delay.

We discussed three types of existing defense systems: (1) attack packet/flow detection at a router followed by packet drop, rate control, and redirection (2) rerouting and avoiding the congested link using statistical and heuristic methods, and (3) using redirecting some flows using linear programming. The first type increases the router computation overhead and the false positive creates great loss to the regular users. Besides, bots are smart to create traffic similar to users and are almost unidentifiable using statistical techniques. For the second type, the heuristically methods performs poorly in special cases. The linear programming approach is time consuming if the topology is complex and big. Therefore, a traffic routing where the number of new rules is minimized with a low time complexity is necessary.

8.3 System Model

In this section, we discuss the network model and the attacker model that we consider for this paper.

8.3.1 Network Model

Our network is composed of regular routers, SDN switches, sources, and destinations. We assume that the controller the congested/attacked links and the routing policy of both regular routers and SDN switches. Therefore, the controller has the global view of the topology and forwarding rules. A rule in a SDN switch is simply a packet forwarding policy. The properties of an incoming packet are matched with the criteria (source address, destination address, incoming port) of the rules and actions are taken according to the matched rule. For simplicity, we consider the destination address as the matching criteria of the rules. The controller can add, modify, or delete rules from a SDN switch but not from a regular switch. If one or multiple links are congested because of regular traffic or attack traffic, some of the flows going through those links must be redirected through any non-congested path. After redirection, none of the links should be congested.

To redirect a flow, the controller needs to add or modify rules in some of the SDN switches. Instead of modifying a rule, it is safe to add a new rule with higher priority in a SDN switch. The number of rules in a SDN switch is important because of the limited storage. When the number of rules is above a certain threshold, the packet forwarding delay increases dramatically. It also takes some time to add a rule in a SDN switch. If the number of additions or modifications is large, then the flow will be interrupted. Therefore, we aim at minimizing the number of rules to redirect some flows from congested links.

Let a flow $f = s \rightarrow d \in F$ from source s to destination d travels through path $P = \{n_1, n_2, \dots\}$. Here, n_i is the i th node on path and P is an ordered set. Each of the nodes in P has a rule for forwarding the packets in flow f . Let $R^f = \{r_1, r_2, \dots\}$ be the set of rules needed to forward the packets of f . $R^f \in R$ where R is the set of rules for all flows in the topology. We will use the superscript f when it is necessary, otherwise we will omit the superscript. Rule r_1 resides in SDN switch n_1 and forwards the packets of f to node n_2 through the link (n_1, n_2) . Some of the nodes in P are SDN switches and

some of them are regular routers. We denote $SDN(n_1) = 1$ if n_1 is an SDN switch and $SDN(n_1) = 0$ if it is regular router. After redirection, the flow f travels through a new path $P' = \{n'_1, n'_2, \dots\}$. The new set of rules is $R'^f = \{r'_1, r'_2, \dots\}$ is the set of rules needed to forward the packets in f through P' . Therefore, the newly added rules are $R' - R$ and they must reside in SDN switches. This is because the controller cannot add any rules in a regular router. The number of newly added rules is $|R' - R|$ which we aim to minimize.

Therefore, our system is a four phase system. In the first phase, the controller detects the congested links. In the second phase, it selects the flows and new paths to redirect. In the third phase, it adds/modifies the forwarding rules to the SDN switches. At the final phase, it removes the flows which are not necessary from the SDN switches. The network congestion can occur because of both regular traffic and attack traffic. Detecting the attack traffic is a slow process (sometimes it needs deep packet inspection and large number of packets) and redirection can be done very quickly. Therefore, the system redirects the flows as soon as the congestion is detected and followed by packet inspection by an intrusion detection system. The detection and blockage of attack traffic is out of scope of this paper and we focus on redirecting some traffic in a quick way to mitigate congestion without disrupting the continuity of the flows.

8.3.2 Attack Model

We assume that the attacker knows the topology and routing method of the network. By analyzing the topology and the routing of the network, the attacker selects a link as a target. Target links are selected based on two properties: the number of flows through a link and the utilization of the link. The target links should forward a large number of flows. And the remaining capacity of bandwidth must be lower or equal to the attacker's capacity. Otherwise, the link will not be overwhelmed and the attack will not be effective. After selecting the target link, the attacker selects the decoy servers and bots to generate traffic. Decoy servers are owned by the attacker and they are used to receive the attack

traffic from the bots. The bots are malicious programs residing in users' computers. The selection of a pair $\langle bot, decoy\ server \rangle$ is done in such a way that the traffic from the bot to the false decoy server passes through the targeted link. The traffic generated by a bot cannot congest a link, but when a large number of attack flows pass through a link it becomes overwhelmed. Consequently, the regular traffic that is passing through the target link suffers from packet drop and low data rate.

8.4 Redirecting a flow

Problem I: Find the route to redirect the flow so that the number of rules needed is the minimum.

In this problem, we assume that, the controller knows the flow to redirect. Let $P = \{n_0, n_1, \dots\}$ be the path of the flow $f = s \rightarrow d$. Let the link (n_c, n_{c+1}) on the path P be congested. After redirection, the new path is $P' = \{n'_1, n'_2, \dots\}$. R' denotes the set of rules needed to forward packets of the flow f through P' .

In this case, the problem can be expressed as the following optimization problem:

$$\begin{aligned}
 & \text{minimize} && |R' - R| \\
 & \text{subject to} && |P'| - |P| \leq \delta_0 \\
 & && \forall_{1 \leq i \leq |P'|} (n'_i, n'_{i+1}) \neq (n_c, n_{c+1}) \\
 & && \sum_{f \in F_0} r(f) \leq \delta_1
 \end{aligned} \tag{8.1}$$

Here, R denotes the set of existing rules in the network. The first constraint means the new path can be at most δ_0 longer than the old path. The second constraint ensures that the congested link does not appear on the new path. δ_1 denotes the maximum allowable bandwidth through the congested link. The third constraint ensures that none of the links traveled by redirected flows are congested after redirection.

8.4.1 A Dijkstra-based Solution

The problem is the basic of problem II (discussed in Section 8.5) and easy to solve. This kind of situation may appears when a heavy flow congest a link. We illustrate the solution as a basic of problem II.

To solve the problem, the controller needs to formulate a CT for the destination of the flow. Each node in the CT has the information of the next hop and the number of rules to needed to forward the flow through that node. The flow can be redirected from any upstream SDN switch that remain on the flow path. If the flow is redirected to any node of the CT, it is guaranteed to be delivered to the destination. To formulate a CT, the controller collects all the rules which are responsible for forwarding the packets of the flow we are redirecting. The complete algorithm for creating a CT is shown in Alg. 20. The complexity of Alg. 20 is same as the Dijkstra algorithm, which is $O(|V| + |E|\log|V|)$ ($|V|$ and $|E|$ are the number of vertices and edges, respectively).

8.4.2 An Example

Let us consider the example in Fig. 8.2(a). Let us assume that link (z, x) is congested and the flow $s_1 \rightarrow d_2$ needs to redirect. Flow $s_1 \rightarrow d_2$ can be redirected from either node b or a . To find the best location of redirection we need to formulate the CT for destination node d_2 . Fig. 8.2(b) shows the CT for destination d_1 . We first remove the congested link. Then, we calculate the number of hops and number of rules needed to deliver any flows to d_2 . The number of rules gets a higher priority over number of hops. We use the Dijkstra algorithm to calculate the smallest rule path from each node. We do not consider the links that becomes congested if they pass $s_1 \rightarrow d_2$ flow. For regular nodes, we do not update distance if the next hop is different from the previous next hop. Construction of CT also considers the data rate of the flow. For example, if we are constructing a CT for redirecting a flow of 10 MBps, then links that can forward additional 10 Mbps traffic without being congested are considered. From the CT, we can see that if we redirect from node b , it will

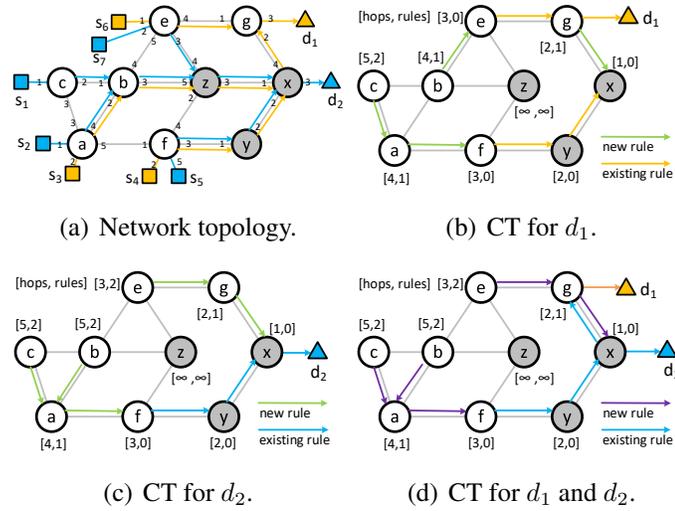


FIGURE 8.2: An example for Problems I and II.

need 2 new rules. If we redirect from node a , it will need 1 new rule. Therefore, we add a new rule at node a that indicates if the source and destination are s_2 and d_2 , respectively, then forward the traffic to node f .

8.5 Redirecting multiple flows

Problem II: Find a set of flows to redirect from all the flows through a congested link so that the number of rules needed to redirect is the minimum.

In this problem, we assume that the controller knows the congested link. Let F be the set of flows passing through the congested link (n_c, n_{c+1}) . We split F into F_0 and F_1 . Flows in F_0 will continue using the congested link. Flows in F_1 will be redirected. Let P'^f be the new path of the flow f ($f \in F$). In this case, the problem can be expressed as

Algorithm 20 Generate CT

Input: $G(V, E, R)$, congested links L_c , destination d .

Output: A CT of G for destination d .

```
1: Procedure: FIND-CT( $G, L_c, d$ )
2:    $\forall_{n \in V} C[n] = \infty, D[n] = \infty, S[n] = \emptyset$ 
3:    $Q \leftarrow \{d\}, C[d] \leftarrow 0, E \leftarrow E - L_c$ 
4:   while  $Q \neq \emptyset$  do
5:      $n \leftarrow$  vertex in  $Q$  with min  $C[n]$ 
6:     REMOVE( $Q, n$ )
7:     for  $n' \in$  NEIGHBOR( $n$ ) do
8:        $r \leftarrow (d, n)$ 
9:       if  $r \in R$  then
10:         $C[n'] \leftarrow \text{MIN}(C[n'], C[n]), S[n'] \leftarrow S[n]$ 
11:         $D[n] \leftarrow D[\text{ARGMIN}(C[n], C[n'])] + 1$ 
12:       else if  $n'$  is SDN then
13:         $C[n'] \leftarrow \text{MIN}(C[n'], C[n] + 1), S[n'] \leftarrow S[n] \cup \{r\}$ 
14:         $D[n] \leftarrow D[\text{ARGMIN}(C[n], C[n'])] + 1$ 
15:   return  $G_{ct}(V, E, C, S, D)$ 
```

the following optimization problem:

$$\begin{aligned} & \text{minimize} && \left| \bigcup_{f \in F_1} R'^f - R \right| \\ & \text{subject to} && \forall_{f \in F_1} |P'^f| - |P^f| \leq \delta_0 \\ & && \forall_{1 \geq i \geq n'} (n_i'^f, n_{i+1}'^f) \neq (n_c, n_{c+1}) \\ & && \sum_{f \in F_0} r(f) \leq \delta_1 \end{aligned} \tag{8.2}$$

Here, $\bigcup_{f \in F_1} R'^f - R$ is the set of all rules needed to redirect the flows. If a rule is used to redirect multiple flows, then it appears on multiple rule sets. Therefore, we need to take union of the set of rules of all flows. δ_1 denotes the maximum allowable bandwidth through the congested link. The first constraint means the new path can be at most δ_0 longer than the old path. The second constraint ensures that the congested link does not appear on the new path. The third constraint ensures that none of the links traveled by redirected flows are congested after redirection.

Algorithm 21 Combine CTs

Input: $G(V, E, R)$, congested links L_c , destinations set D .

Output: A CT of G for destination d .

```
1: Procedure: MULT-DEST-CT( $G, L_c, D$ )
2:    $\forall_{d \in D} \quad CT[d] \leftarrow \text{FIND-CT}(G, L_c, d)$ 
3:    $\forall_{n \in V} C[n] = \infty, D[n] = \infty, S[n] = \emptyset$ 
4:    $Q \leftarrow D, E \leftarrow E - L_c$ 
5:   while  $Q \neq \emptyset$  do
6:      $n \leftarrow$  vertex in  $Q$  with min  $C[n]$ 
7:     REMOVE( $Q, n$ )
8:      $X \leftarrow \bigcup_{d \in D} CT[d].S, C[n] \leftarrow |X|$ 
9:     for  $n' \in \text{NEIGHBOR}(n)$  do
10:       $r \leftarrow (D, n')$ 
11:       $S[n] \leftarrow d \in DCT[d].S[n]$ 
12:      if  $r \in \bigcup_{d \in D} CT[d].S[n]$  then
13:         $C[n] \leftarrow \text{MIN}(C[n], C[n'])$ 
14:         $D[n] \leftarrow D[\text{ARGMIN}(C[n], C[n'])] + 1$ 
15:      else
16:         $C[n] \leftarrow \text{MIN}(C[n], C[n'] + 1), S[n] \leftarrow S[n] \cup \{r\}$ 
17:         $D[n] \leftarrow D[\text{ARGMIN}(C[n], C[n'])] + 1$ 
18:   return  $G_{ct}(V, E, C, S, D)$ 
```

8.5.1 A Flow Grouping Solution

The problem is NP-hard and we provide a heuristic solution for this problem. The basic idea is to group the flows and add some rules that can work on the group of flows. The small number of groups helps to reduce the run-time with some additional rules. We group the flows based on common upstream links. We consider the paths of the flows up-to k hops from the congested link. Flows having the same path goes to the same group. Therefore, each group has the same upstream path up-to length k . Each node on the common path is eligible to be a redirect point. After that, CT for the destinations in each group is created. Creating a CT for a group is basically combining the CTs for the destinations in that group. To combine multiple CTs, we need compute the number of rules needed and the hop-counts for the destinations. A node can forward the packets that belong to the destination group in two different ways:

Table 8.1: Flows and bandwidth

	Flow	Bandwidth		Flow	Bandwidth
f_0	$s_1 \rightarrow d_2$	40 Mbps	f_4	$s_4 \rightarrow d_1$	10 Mbps
f_1	$s_2 \rightarrow d_2$	10 Mbps	f_5	$s_5 \rightarrow d_2$	10 Mbps
f_2	$s_3 \rightarrow d_1$	20 Mbps	f_6	$s_7 \rightarrow d_2$	20 Mbps
f_3	$s_6 \rightarrow d_1$	10 Mbps			

1. **Separate forwarding:** The node forwards the packets of each flows to separate next hops that offers the least number of rules. The combined number of rules needed is the total of the number of unique rules of all CTs.
2. **Aggregated forwarding:** The node forwards all the packets that are destined to any of the destinations in the group to a next hop. The combined number of rules needed is one plus the minimum of separate or aggregated forwarding of all CTs of the next hop nodes.

However, we select the option that provides the minimum number of rules. The number of CTs are the same as the number of groups. For each group, we calculate the minimum number of rules needed to redirect the flows from one of the redirect points. Then, the group with the minimum number of rules is taken out for redirection. After that, we update the CT of the rest of the groups. We need to update the CTs because after redirection, the capacity of some links will change. As a result, the CT for the rest of the groups may change. This process continues until the congested link becomes non-congested. Alg. 21 shows the complete procedure of combining multiple CTs.

The value of k plays an important role in grouping. A small k produces a small number of large size groups. This reduces the running time of the algorithm, but a large group of flows may need more rules to redirect.

8.5.2 An Example of Common k Links Grouping

Let us consider the example in Fig. 8.2(a). Each link has a maximum capacity of 100 Mbps. Flows and their bandwidths are shown in Table 8.1. We can observe that flows f_1 ,

f_2 , f_3 , and f_6 are passing through link (z, x) . Their total bandwidth is $40 + 10 + 20 + 20 = 90$ Mbps. We assume that any link utilization above 70% (70 Mbps) is considered congested. Therefore, only link (z, x) is congested in our example. We need to redirect some of the links among f_0 , f_1 , f_2 , and f_6 to make link (z, x) usage less than or equal 70 Mbps.

Next, we group the flows according to the Common k Links policy. Let us consider $k = 1$ first. The the upstream path of the flows of length 1 is following:

$$f_0 : \{b, z\}, f_1 : \{b, z\}, f_2 : \{b, z\}, f_6 : \{e, z\}.$$

We can observe that f_0 , f_1 , and f_2 have the same upstream path $\{b, z\}$. Therefore, flows f_0 , f_1 , and f_2 goes to group 1 and f_6 goes to group 2.

Next, we construct the CT for the groups ($k = 1$). The CT for group 1 contains routes for destinations d_1 and d_2 . The total bandwidth of the group is $40 + 20 + 20 = 70$. Therefore, any link having existing flows cannot redirect all the traffic. As a result, group 1 cannot be redirected. The CT for group 2 contains routes for destination d_2 only. Therefore, the CT is the same as in Fig. 8.2(c). The redirect points of group 1 is e . The number of rules needed to redirect from e is 2. Therefore, the minimum number of rules to redirect the flows in group 1 is 2. At this point, we redirect the flows in group 2 and the number of rules needed is 2. After redirection, the congested link usage becomes $90 - 20 = 70$ Mbps and is no longer considered congested.

If we consider $k = 2$, the upstream paths of the flows of length 2 are following:

$$f_0 : \{c, b, z\}, f_1 : \{a, b, z\}, f_2 : \{a, b, z\}, f_6 : \{e, z\}.$$

Therefore, flows f_0 goes to group 1, f_1 , and f_2 go to group 2, and f_6 goes to group 3. The CT for group 1 contains routes for destination d_2 only. Therefore, the CT is the same as in Fig. 8.2(c). The redirect points of group 1 are c and b . The number of rules needed to redirect from b and c is 2. The minimum number of rules to redirect the flows in group 1 is 2. The CT for group 2 contains routes for destination d_1 and d_2 . Therefore, the CT is the same as in Fig. 8.2(d). The redirect points of group 2 are a and b . The number of rules

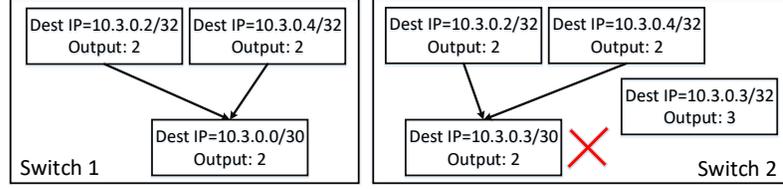


FIGURE 8.3: Merging rules.

needed to redirect from a and b is 1 and 2, respectively. So, the minimum number of rules to redirect the flows in group 2 is 1.

The CT for group 3 contains routes for destination d_2 only. Therefore, the CT is the same as in Fig. 8.2(c). The redirect point of group 3 is e only. This is because, z is a regular router and cannot be redirected from there. The number of rules needed to redirect from e is 2. Therefore, the minimum number of rules to redirect the flows in group 3 is 2.

Therefore, redirecting group 2 adds the least number of rules. We redirect the flows f_1 and f_2 from node a . After redirection, the link capacity of the congested link comes down to $90 - (10 + 20) = 60$ Mbps, which is below 70% of the link utilization and the algorithm stops.

Theorem 21. *The complexity of Alg. 21 is $O(|V| + |E|\log|V|)$.*

Proof. To calculate the complexity of Alg. 21, we need to calculate the complexity of Alg. 20. Alg. 20 is basically the Dijkstra algorithm. Therefore, the complexity is $O(n + m\log n)$ where $n = |V|$ and $m = |E|$. Step 2 in Alg. 21 takes $O(|D| \times (|V| + |E|\log|V|))$. Step 11 takes $O(|D|)$ times because at any node the number of added rules can be at most as the number of destination nodes. Therefore, the while loop in Step 5 takes $O(n \times |D|)$ time. So, the total run time of the Alg. 21 is $O(|D| \times (|V| + |E|\log|V|)) + O(|V| \times |D|)$ which is $O(|D| \times (|V| + |E|\log|V|))$ □

8.5.3 Greedy Rule Merging Solution

In this subsection, we consider that two rules can be merged if it does not conflict with others. We define conflict of rules for merging if all of the following conditions are satisfied:

1. Forwarding ports of the rules to be merged must be same.
2. Merged rule matching criteria cannot overlap with the criteria of any other rules except the rules to be merged.
3. Merged rule matching criteria must match all the criteria of rules to be merged.

Let us consider the example in Fig. 8.3 to explain the merging of rules. Switch 1 has two rules; *if destination IP is 10.3.0.2/32 then output to port 2*, and *if destination IP is 10.3.0.4/32 then output to port 2*. These two rules have the same output ports so they may be merged (condition 1 satisfied). We combine the matching criteria to 10.3.0.0/30 that covers both rules (condition 2 satisfied). As there are no other rules, condition 3 is satisfied. Therefore, we can merge the rules to *if destination IP is 10.3.0.0/30 then output to port 2*. In switch 2, we have three rules. Rule 1 and Rule 2 have same output ports. We combine their matching criteria to 10.3.0.0/30, which covers the matching criteria of rule 3 (violation of condition 3). Therefore, the rules cannot be merged.

When we add new rules and the new rules can be merged with any of the existing rules, then the number of rules does not increase. Using this principle, we design a greedy algorithm to solve problem II. The greedy algorithm is simple; we calculate the minimum number of rules for each flow through alternative possible paths with merging. Then, we pick the flow with the minimum number of unseen rules and redirect it through the path. After that, we mark the taken unseen rules as seen. We continue until the congested link becomes non-congested.

Table 8.2: Topology Parameters

Number of	Topology I (T_1)	Topology-II (T_2)
Nodes	73	121
Sources	13	16
Destinations	15	24
SDN switches	45	81
Links	184	296

8.5.4 An example

Let us consider the example in Fig. 8.2(a) again. We first consider f_0 through path $\{s_1, a, f, y, x, d_2\}$. There are already rules to forward packets to d_2 at node f , y , and x . We need to add two new rules at a and c . None of the existing rules output to port 3 at c and port 5 at a . Therefore, we need 2 rules to redirect f_0 through the path. If we consider path $\{s_1, c, b, e, g, x, d_2\}$, then we need new rules at nodes b , e , and g to output the packets to ports 4, 4, and 2, respectively. There are no rules at b and g to output to port 4 and 2. The existing rule at e that outputs to port 4 can be merged with the new rules. Therefore, we also need 2 new rules to forward f_0 through this path. Similarly, considering other paths, we find that the cost of redirecting f_0 is 1 ($\{s_1, c, b, z, f, y, x, d_2\}$). Similarly, we calculate the cost of redirecting f_1 , f_2 , and f_6 is 1. Therefore, we can pick any of the flows to redirect. Let us pick f_0 to redirect. After redirection we calculate the cost of the remaining flows and continue the process until link (z, x) becomes non-congested.

Theorem 22. *The complexity of Greedy Rule Merging Solution is $O(|F||R_n|(|V| + |E|))$.*

Proof. Finding the cost of redirection needs to traverse the topology graph once. To find whether a new rule can be merged or not takes $O(|R_n|)$, where R_n is the set of existing rules at node n . Therefore, each iteration of the greedy algorithm takes $O(|R_n|(|V| + |E|))$. The iteration can run at most $|F|$ times. Therefore, the complexity of the algorithm is $O(|F||R_n|(|V| + |E|))$ \square

Theorem 23. *The complexity of Greedy Rule Merging Solution has an approximation ratio of $\eta(1 - \frac{1}{e})$.*

Proof. To find the upper bound of the greedy approximation, we assume that all rules that output to the same port can be merged. We denote a rule by r_{np} where n is the node and p is the output port. We denote the minimum set of rules needed to redirect a flow f by S_f . Now, the problem II can be expressed as finding k elements from the set $S = \{S_1, S_2, \dots, S_{|F|}\}$ so that the number of unique elements is the minimum. Now we convert the problem to a maximum coverage problem by replacing S with S' and S_f with S'_f . Here $S' = \{S'_1, S'_2, \dots, S'_{|F|}\}$ and $S'_f = U - S_f$. U is the universal set ($U = \cup_{f=1}^{|F|} S_f$). Therefore, a maximum coverage solution represents a solution to Problem II. The maximum coverage greedy solution has an approximation ratio of $(1 - \frac{1}{e})$. Let η be the merging ratio of the rules. If there are $|R_n|$ number of rules before merging and $|R'_n|$ number of rules that output to a specific port at node n , then $\frac{|R_n|}{\eta} = |R'_n|$. Therefore, the number of rules for greedy rule merging solution must be less than $\eta(1 - \frac{1}{e})$ times of the minimum number of rules. \square

8.6 Simulations and Experiments

8.6.1 Simulation Settings

We conduct the experiments with our custom built java simulator. We want to count the number of rules needed for redirecting the flows. We do not need to analyze transmission time, actual link bandwidth, or packet drop issues. The network topologies we consider in this simulation contain 70 – 120 SDN switches, sources, and destinations. Using NS3 or other similar simulators for this kind of simulation would take a long time to produce results. That is the reason for building our own java simulator to get the results fast.

We generate random topologies by dividing an area of 500×500 square unit into 50×50 blocks. In each block, a certain number of nodes are placed randomly. We randomly select some nodes and attach a source or destination with some probability. We set the link capacities as 100 Mbps. Although in a real network the link capacity is different for different links (100Mbps/1Gbps/10Gbps), we keep them the same for simplicity and ease

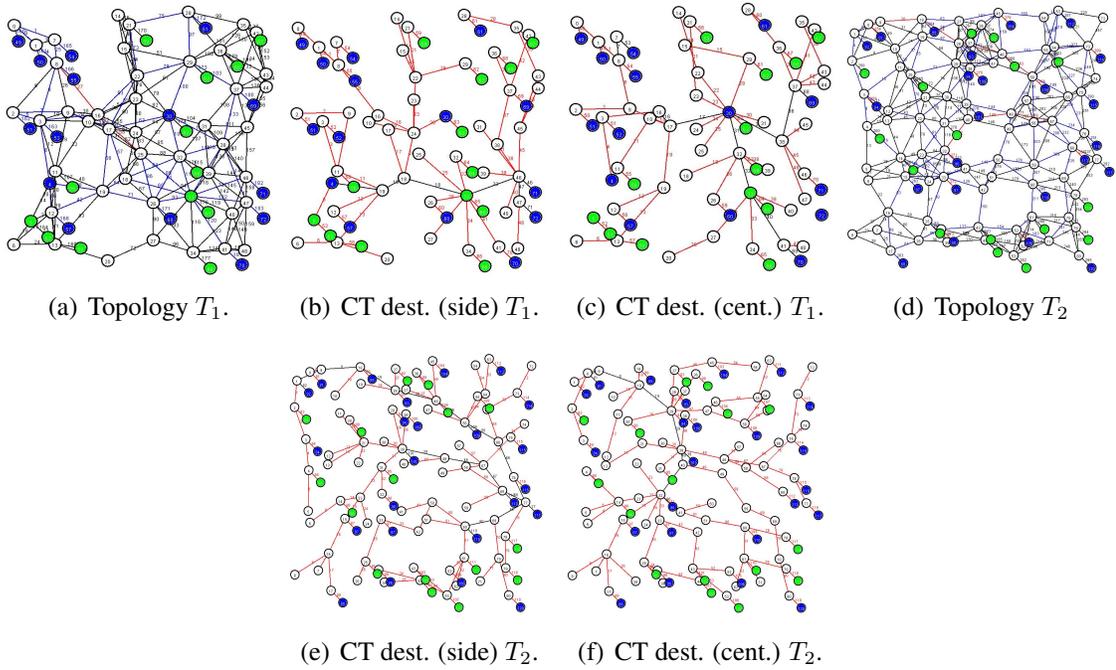


FIGURE 8.4: CT of a randomly generated topology for different destinations (green node: source, blue node: destination, red link: new rules added, black link: existing rule).

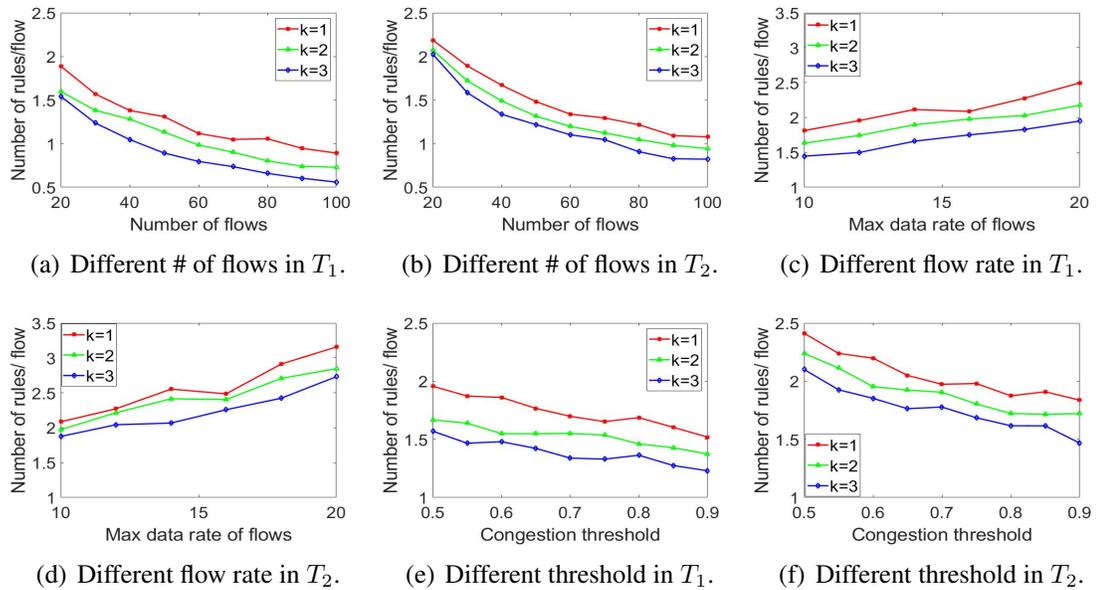


FIGURE 8.5: Number of rules per redirected flow with different settings.

of comparison. Table 8.2 shows details properties of the topologies. Next, we generate the desired number of flows by randomly selecting a source and a destination with a random rate from a range. We set the minimum rate to 1 Mbps. We select different maximum rates for different simulations.

We measure the number of rules increased/flow and the number of hops increased after redirection for different number of flows, maximum data rate, and congestion threshold. The congestion threshold is defined by $\frac{\delta_1}{\text{link capacity}}$. We compare the performance of our approaches with shortest alternative path redirection approach. All the results in the plot are the average of 200 runs with three different k values.

8.6.2 Simulation result

Firstly, we observe the number of increased rules per flows by changing different parameters. Fig. 8.5(a) shows the number of increased rules per flow for different number of flows in T_1 . We vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. For all values of k , the number of rules per flow decreases with the increase of number of flows. A higher value of k , produces lower number of rules per flow. This is because when the number of flows are higher, the network needs a higher number of rules to forward them. As a result, the existing rules help to forward the packet through an alternative way and we need fewer number of new rules. If k is higher, then the number of groups increases and the selection of flows to redirect is better. When the number of flows is 20 and $k = 1$, the average number of needed rules to redirect a flow is 1.88. When $k = 3$, the average number of needed rules to redirect a flow is 1.54. The number of needed rules increased by about 18%. Fig. 8.5(b) presents the number of increased rules per flow for different number of flows in T_2 . We observe similar behavior to T_1 . The overall number of rules per flow is higher in T_2 than T_1 . This is because T_2 is larger and average number of hops between source and destination is higher than that of T_1 .

Fig. 8.5(c) shows the number of increased rules per flow for different maximum data rate of flows in T_1 . We vary the maximum data rate of flows from 10 to 20 Mbps and keep the number of flows as 20. For all values of k the number of rules per flow increases with the increase of maximum data rate. Similar to the previous simulation, a higher value of k produces lower number of rules per flow. This is because when the max data rate is higher, the network observes more congested links. Because of a higher number of congested or soon to be congested links, there remains less options for redirecting a flow. As a result, a higher number of new rules is needed to forward the flow. When the maximum data rate is 10 and $k = 1$, the average number of needed rules to redirect a flow is 1.81. When $k = 3$, the average number of needed rules to redirect a flow is 1.44. The number of needed rules increased by about 20%. Fig. 8.5(d) plots the number of increased rules per flow for different maximum data rate of flows in T_2 . We keep the same parameters as the previous simulation and observe the similar behavior. The overall number of rules /flow is higher in T_2 than T_1 .

In Fig. 8.5(e), we show the number of increased rules per flow for different congestion threshold of links in T_1 . We vary the congestion threshold of links from 0.5 to 0.9 and keep the number of flows at 20. For all values of k , the number of rules per flow decreases with the increase of congestion threshold of links. This is because when the congestion threshold is higher, the network observes less congested links. Because of the lower number of congested or soon to be congested links there are more options for redirecting a flow. As a result, the number of rules needed to forward the flow is smaller. Similar to all previous simulations, a higher value of k produces lower number of rules per flow. When the congestion threshold is 0.5 and $k = 1$, the average number of needed rules to redirect a flow is 1.95. When $k = 3$, the average number of needed rules to redirect a flow is 1.56. The number of needed rules decreased by about 20%. Fig. 8.5(f) presents the number of increased rules per flows for different congestion threshold of links in T_2 . We keep the same setting to the previous simulation and observe the similar behavior in T_2 .

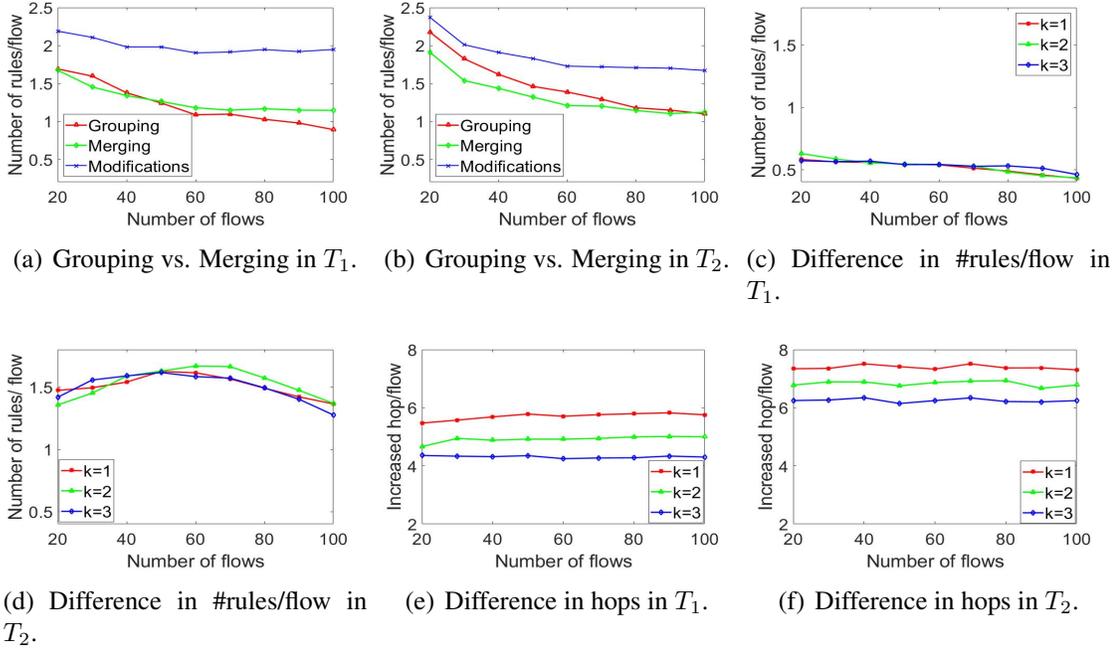


FIGURE 8.6: Comparison with shortest alternative path routing.

Figs. 8.6(a) and 8.6(b) show the comparison between the grouping and merging based approaches in T_1 and T_2 . We vary the number of flows from 20 to 100, keep the maximum data rate as 10 Mbps, and set $k = 1$. We observe that the merging based approach needs a smaller number of rules but a higher number of modifications than the grouping based approach. When the number of flows is 20 in T_1 , the average number of needed rules to redirect a flow is by grouping and merging based approaches are 1.69 and 1.67. The number of modification is needed in the merging based approach is 2.18 per flow. The merging based approach needs less (or more) rules when the number of flows is less (higher) than 50. The number of modifications for merging is always higher than grouping based approach. We observe similar behavior for T_2 . Therefore, when we need to save storage on SDN switch, we need to use the merging based approach. When we prioritize interruption over SDN storage, then we need to use the grouping based approach.

Fig. 8.6 compares the alternate shortest path routing with redirection with k grouping in terms of number of increased hops/flow and number of increased rules per flow. For

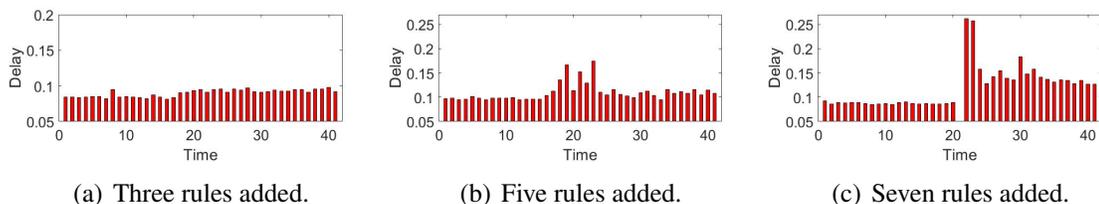


FIGURE 8.7: Effects on ping delay.

these simulations, we vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. Figs. 8.6(c) to 8.6(f) are executed with this settings. Fig. 8.6(c) shows the number of increased rules per flow for different number of flows in T_1 . When the number of flows is 20 and $k = 1$, the average number of needed rules to redirect a flow in our approach is 0.58 less than the shortest alternate path approach. When $k = 2$ and $k = 3$, the average number of needed rules to redirect a flow in our approach is 0.62 and 0.57 higher than shortest alternate path approach. The difference is similar for all k values. The difference slightly decreases with the increase of number of flows. In Fig. 8.6(d), we present the number of increased rules per flow for different numbers of flows in T_2 . The difference slightly increases then decreases with the increase of number of flows.

Fig. 8.6(e) shows the number of increased hops/flow for different numbers of flows in T_1 . When the number of flows is 20 and $k = 1$, the average number of increased hops after redirecting a flow in our approach is 4.47 higher than the shortest alternate path approach. When $k = 2$ and $k = 3$, the average number of needed rules to redirect a flow in our approach is 4.66 and 4.35 higher than the shortest alternate path approach. The difference remains unchanged for all numbers of flows. Fig. 8.6(f) lists the number of increased hops/flow for different numbers of flows in T_2 . We observe similar behavior as in T_1 .

Therefore, based on the simulation result, we can conclude that our redirection approach with k links grouping can redirect by adding a lower number of rules than the shortest alternate path approach.

Table 8.3: Shortest and alternative path rules

Shortest path rules (101-103)			Alternative path rules (101-103)					
SDN	IN	OUT	SDN	IN	OUT	SDN	IN	OUT
9	11	3	9	11	1	12	11	1
12	2	11	5	2	1	6	3	1
12	11	2	3	2	3	3	3	2
9	3	11	6	1	3	5	1	2
			12	1	11	9	1	11

8.7 Experiments in Datacenter

8.7.1 Experimental Settings

We conduct experiments to evaluate the effect on a flow when we change the routing. To conduct the experiments we use our small datacenter with fifteen SDN switches. We do not use any regular routers because they only increase hops without any effect on the changes in rules. The partial topology of the datacenter is shown in Fig. 8.8. Nodes 2 to 16 are Pica 8 (P-3297) SDN switches. The numbers at the ends of a link denotes the port number where it is plugged. There are four servers (101-104) that are connected at the leaf level switches. We create three flows $\{101 \rightarrow 102, 101 \rightarrow 103, 101 \rightarrow 104\}$ and continuously record the ping delays (round trip delay). The initial flows travel the shortest path and we suddenly change the route by adding some rules. We compare the delays before, after, and during the rule change period.

We use ONOS (2.2.2) as the controller and REST api to add/delete rules. We develop a separate Java program that can monitor the link status and add/delete rules. We also develop a flow generator program that continuously sends ping requests to the destination and records the ping delay. The rule modifier program first removes all previously added rules using REST api. Then it adds the rules necessary to follow the shortest path for the flow we are considering (generated by our flow generator program) with a priority of 4000. Table 8.3 shows the shortest and alternative path rules for the flow from server 101 to server 103. After 15 seconds it adds the rules necessary to follow the alternate path with a higher priority of 5000. Therefore, the flow starts passing through the alternate path

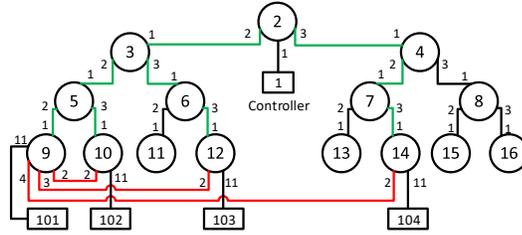


FIGURE 8.8: Datacenter topology (partial).

and we observe the changes in ping delay. Fig. 8.7 presents the ping delays for the three experiments.

8.7.2 Experimental Results

Fig. 8.7(a) shows the ping delay of 40 times of flow servers 101 to 102. The flow during time 1 to 16 follows the shortest path $\{101, 9, 10, 102\}$ and the average ping delay is 0.84 ms. After changing the rules by the rule modifier program, the flow travels the alternative path $\{101, 9, 5, 10, 102\}$. The flow during times 17 to 40 follows the alternative path and the average ping delay is 0.93 ms. The ping delay increases 0.09 ms for an increase of 1 hop. No other changes are observed for the addition of three new rules.

Fig. 8.7(b) depicts the ping delay of 40 times of flow from 101 to 103. The flow during times 1 to 14 follows the shortest path $\{101, 9, 12, 103\}$ and the average ping delay is 0.94 ms. After changing the rules, the flow travels the alternative path $\{101, 9, 5, 3, 6, 12, 103\}$. The flow during times 15 to 40 follows the alternative path. During time 15 to 23 we observe a lift in delay. This lift in delay occurs because of the addition of 5 new rules. The average ping delay becomes stable during times 24 to 40 and the average delay is 1.11 ms. The delay increases 0.17 ms for an increment of 3 hops.

Fig. 8.7(c) presents the ping delay of 40 times of flow servers 101 to 104. The flow during times 1 to 19 follows the shortest path $\{101, 9, 14, 104\}$ and the average ping delay is 0.87 ms. After changing the rules by the rule modifier program, the flow travels the alternative path $\{101, 9, 5, 3, 2, 4, 7, 14, 104\}$. The flow during times 20 to 40 follows the alternative path. At time 20 the destination remained unreachable for 1 s. During times 21

to 22 we observe a high lift in delay. This lift in delay occurs because of an addition of 7 new rules. The average average ping delay becomes stable during times 22 to 40 and the average delay is 1.38 ms. The ping delay increases 0.51 ms for an increase of 5 hops.

Therefore, from these experimental results, we can conclude that the more the change in number of rules, the more the interruption in flow. Changes in up to 5 rules might not cause noticeable interruption to delay sensitive applications. Changes in 7 or more rules might cause interruption to delay sensitive applications.

8.8 Conclusion

The link flooding attack and link congestion are common issues in a datacenter. A larger number of rules plays an important role for transmission delay. When the number of rules in an SDN switch is higher than its fast accessible rules storage, the delay increases dramatically. Besides, changing rules in a higher number of SDN switches takes a long time, which causes interruption in flows. We consider this important issue for mitigating link flooding attack or congestion by redirecting some flows with less number of new rules. We propose and evaluate performances of k links flow grouping and merging-based approaches. Our extensive simulation supports that the proposed approaches can minimize the number of added rules by increasing the number of hops.

CHAPTER 9

EFFICIENT SWITCH MIGRATION FOR CONTROLLER LOAD BALANCING IN SOFTWARE DEFINED NETWORKING

In the previous chapters, we have explored different problems and proposed multiple solutions for different network attacks and attack related optimization in both regular and software defined networks. In this chapter, completely focus on optimizing network performance in a large scale software defined networks. We consider multiple controllers and formulate problems for initial and incremental load balancing. The initial assignment process is executed at the beginning of the network deployment. After initial deployment, the incremental assignment process is executed periodically. The incremental process migrates some of the switches to another controller to improve the performance of the network. We propose greedy and clustering-based solutions for initial switch-controller assignment. We also propose a greedy solution for incremental assignment. Our proposed solutions are evaluated using both synthetic and real datasets, and the parameters are driven by experiments at a data center. The contents of this chapter is submitted to [Submitted4].

9.1 Introduction

Software-defined networking (SDN) technology is an approach to manage networks dynamically and programmatically. Unlike the traditional network system where a human (network administrator) manages the network components including switches and routers, a software called a controller manages the network components. The SDN system also enables the feasibility of decoupling the controller network and data network. An in-band controller system enables the SDN switches to communicate with the controller via the data network. On the other hand, the out-band controller system uses the control network to communicate with the controller.

The forwarding of packets in an SDN switch is defined by the rules installed in it. There are two types of rules: wildcard and re-active rules. When a packet arrives at any ports of an SDN switch, it first tries to use the wildcard rules. A wildcard rule is stored in the SDN switch and its usage for forwarding packets is not reported to the controller. When there exist no rules to match the packet, the SDN switch asks for a forwarding decision to the controller. The controller replies with the decision (output to one or multiple ports, drop, broadcast, packet modifications, etc.). The requests from a switch leave some workload on the controller. For example, when the first SDN switch asks for a forwarding rule for the packet of a flow, the controller needs to construct the routing path for the flow, creates the corresponding rules, and reply to the rule for forwarding the packet. This process takes longer than processing other types of requests. For example, when an SDN switch asks for a forwarding rule for a packet of the flow, the controller does not need to re-calculate the path. It replies to the rules already created for that flow. Therefore, for different types of requests, the controller load is different. If the number of switches and the flows in an SDN network is large, then one controller cannot handle all of the requests. In this situation, the network is divided into domains and a controller is assigned to each domain. The network needs to be divided wisely to balance the load of the controllers.

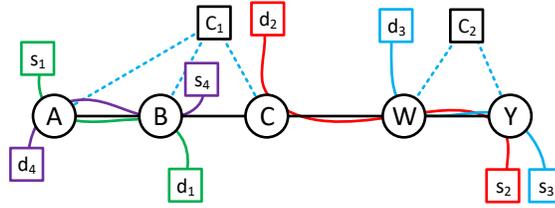


FIGURE 9.1: An Example of switch migration.

For example, in Fig. 9.1, there are four SDN switches (A , B , C , X , and Y) and four flows ($f_1 : s_1 \rightarrow d_1$, $f_2 : s_2 \rightarrow d_2$, $f_3 : s_3 \rightarrow d_3$, and $f_4 : s_4 \rightarrow d_4$) in the network. Let us consider that switches A , B , and C are controlled by controller C_1 and switches W and Y are controlled by controller C_2 . In this assignment, C_1 becomes overloaded with three path construction and two intermediate query requests. On the other hand, C_2 is underutilized because of two path construction and two intermediate query requests. If we migrate the switch B to controller C_2 , then it will increase one more intermediate query to C_2 , but C_1 will receive one less path construction request. Usually, path constructions induce almost ten times the load than intermediate query requests do. Switch migration is challenging for several reasons. Firstly, to the best of our knowledge none of the commercially available SDN switches supports migration. However, it is possible to implement the migration feature by using SSH and Linux commands provided by some of the switches. The downside of this type of migration is that it will block packet forwarding for a period.

According to the SDN architecture, the controllers know the links, their usage, and SDN switches. Each of the controllers knows a portion of the topology and they report their knowledge to the assignment manager (AM) server. The AM gets the global view of the topology and uses that information to assign switches to the controllers so that the load is balanced. As we know, the topology and flows are subject to change over time. Therefore, the AM needs to adjust the assignment based on new topology and flow information changes. Therefore, we needed to consider two types of switch-controller assignment processes: initial assignment and incremental assignment. The initial assignment process

is executed at the beginning of the network deployment and the incremental assignment process migrates some switches from one controller to another. These problems of the initial and incremental assignment are NP-hard and we provide greedy and clustering-based solutions. The main contributions of our paper are:

1. We study a switch-controller assignment problem to minimize network overhead and propose a solution for initial deployment.
2. We investigate incremental switch-controller assignment problems and provide greedy solutions.
3. We conduct extensive simulations and experimental results and compare them with existing approaches.

The remainder of this paper is arranged as follows. Section 9.2 presents some related works. In Section 9.3, we present the system and the cost model. In Section 9.4, we present the problem of switch controller assignment and propose two solutions. Section 9.5 contains the problem of incremental switch-controller assignment and proposed greedy solution. Section 9.6 and Section 9.6 present the simulation and experimental and results. Finally, Section 9.7 concludes our paper.

9.2 Related Works

There exists several works on switch-controller assignment and load balancing. In [84], authors formulate the dynamic controller assignment problem as an online optimization to minimize the total cost. The cost includes response time and maintenance on the cluster of controllers. They propose a two-phase algorithm that integrates key concepts from both matching theory and coalitional games to solve it. In [85], authors propose an effective switch-controller mapping scheme for distributed controllers and distributes flow setup requests among them in order to minimize flow setup time, and obtain resilience. In [86], authors show that dynamic mapping between switches and controllers improves efficiency

in traffic load balancing. They propose two balanced controllers (BalCon) and BalConPlus that are switch migration schemes to achieve load balance among SDN controllers with minimum migration cost. BalCon is limited to the scenarios where the network can process switch requests out of order, otherwise BalConPlus is more suitable.

In [87], authors propose a switch migration-based decision-making (SMDM) scheme that is aware of the load imbalance. They consider the tradeoff between migration costs and the load balance rate for producing a switch migration trigger. They propose a greedy method that utilizes the migration efficiency to produce possible migration actions. A game theory based decision mechanism for switch migration is proposed in [88]. Their system dynamically migrates switches from heavily loaded controllers to lightly loaded controllers based on a centralized available resource utilization maximization problem. The system takes controllers and switches as game players and how to migrate switches among the control plane is considered as the player policy. In [89], a framework for deploying multiple controllers within an WAN is proposed. The framework dynamically adjusts the number of active controller. This system only considers the flow setup latency involved in path construction in controllers and formulate an optimization problem and solve using Integer Linear Program. Other works that uses ILP for solving controller load balancing includes [90, 91, 92, 93]. In [94], authors propose a dynamic load balancing method based on switch migration mechanism for clustered controllers. The mechanism can also handle controller failover without switch disconnection. In [95], authors propose mechanism to maximize resource utilization. It detects imbalance in load and the controller randomly selects a switch for migrating. It also maintains integrity by broadcasting the migration activity to the controller's neighbors.

Some other works that proposed distributed controller system includes [96, 97, 98, 99, 100, 101, 102, 103]. However these system does not consider the cost of migration and other performance metrics including packet forwarding delays while load balancing. Most of the works discussed here considers controller load as the main load balancing/switch

migrating parameters. None of them considered the control plane communication overhead and intermediate queries for deciding switch migration that are an important parameters for network performance. Therefore, a system that considers controller load and control plane communication overhead for deciding switch migration to obtain a load balanced state of the controller is necessary.

9.3 Network Model

Our network is composed of SDN switches, controllers, sources, destinations, and flows. We assume that the controller knows the links, their usage, and SDN switches. Each of the controllers knows a portion of the topology. The controllers report their knowledge with the assignment manager (AM) server. Therefore, the AM has a global view of the topology and the flows in the network. The AM also has knowledge about the control plane and thus can infer the number of hops and delays between switches and controllers.

The AM is responsible for the assignment of switches to the controllers. By analyzing the topology and the flows, it finds out the switch-controller assignment that minimizes the overall transmission network delay. As we know, the topology and flows are subject to change over time. Therefore, the AM needs to adjust the assignment based on new topology and flow information. So, there are two types switch-controller assignment processes: initial assignment and incremental assignment. The initial assignment process is executed at the beginning of the network deployment. After initial deployment, the AM executes the incremental assignment process. The incremental process migrates some of the switches to another controller to improve the performance of the network.

A switch migration refers to changing the controller of a switch. The switch migration is a complex process that includes removing the existing controller, adding the new controller, and initialization of the switch with the controller. Switch migration is challenging for several reasons. Firstly, to the best of our knowledge, none of the commercially available SDN switches supports migration. However, it is possible to implement the migration

feature by using SSH and Linux commands provided by some of the switches. The downside of this type of migration is that it will block packet forwarding for a period. The replacement of the controller can be done in two ways: direct replacement and switch to the backup. Direct replacement refers to the process of deleting the primary active controller and adding a new controller. The switch to backup approach refers to adding the new controller as a backup controller followed by the removal of the primary controller.

Therefore, our system is a two phase system. In the first phase, the switches are assigned according to the initial deployment methods. After that the system enters into second phase which is basically adjusting the assignment with the dynamic changes in flows. In the second phase, the adjustments are done periodically using the incremental deployment approach. The period of adjustment is determined based on the dynamic nature of the SDN network.

9.3.1 Cost Model

Unlike the cost metric in a single controller system [104] where only the number of rules are considered, the cost of a switch-controller assignment is incurred by the three parameters: the number of path construction requests, the number of intermediate query request, and the number of hops from the controller to the switches. The response delays from a controller depends on the number of hops from the controller to the switches. $P(A, c)$ denotes the number of path construction requests at controller c for A assignment. Therefore, $P(A, c)$ can be expressed as the following:

$$P(A, c) = \sum_{f \in F} \left| \bigcup_{v \in P_f, A(v)=c} A(v) \right| \quad (9.1)$$

$Q(A, c)$ denotes the number of intermediate query requests at controller c for A assignment. $Q(A, c)$ is the number of all forwarding nodes that are controlled by c on the path of all flows minus the number of path construction requests. Therefore, $Q(A, c)$ can be expressed as the following:

$$Q(A, c) = \sum_{f \in F} |P_f, A(v) = c| - P(A) \quad (9.2)$$

$D(A, c)$ denotes the total number of hops traveled (in control plane) by all requests at controller c for A assignment. That why we multiply the distance from the controller to the nodes ($d_{v,A(v)}$) with the number of requests. Therefore, $D(A, c)$ can be expressed as the following:

$$D(A, c) = \sum_{f \in F} \sum_{v \in P_f, A(v)=c} d_{v,A(v)} \times (P(A) + Q(A)) \quad (9.3)$$

We define the cost of a switch-filter assignment as an weighted summation of the above three metrics. Therefore the cost of assignment A can be expressed as the following:

$$C(A, c) = \omega_1 P(A) + \omega_2 Q(P) + (1 - \omega_1 - \omega_2) D(A) \quad (9.4)$$

In the next sections, we formulate problems to minimize this cost. This cost represents the overall transmission delays in a SDN network. This is because, when the number of path construction requests and intermediate query requests are high, the controller needs a long time to produce a decision and response the requests. The delay to receive the responses at a switch also depends on the number of hops to reach the controller. As a result, when the number of hops to the controller is high, a packet faces delay on each intermediate node. The SDN switches usually catches the forwarding rules locally in their limited memory which makes the forwarding fast. In this case, only the first packet of the flow encounters delays on each hop. When the number of rules in a switch is very high, then the catching cannot provide the best performance. In that case, not only the first packets but also the following packets encounters delays.

9.4 Initial Deployment

In this section, we formulate the problem of switch assignment so that the maximum cost for all flows is minimal.

Problem I: Find switch to controller mapping so that the maximum delay of all flows is minimum.

Let the topology be $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_N\}$ is the set of N SDN switches and E is the set of links. Let the set of controllers $C = \{c_1, c_2, \dots, c_M\}$ contains M controllers. The set of flows in the network is denoted by $F = \{f_1, f_2, \dots, f_K\}$. Therefore, the problem can be expressed as the following:

$$\begin{aligned} & \text{minimize} && \max_{\forall c \in C} C(A, c) \\ & \text{subject to} && \forall_{v \in V} d_{vA}(v) \leq Th_1 \end{aligned} \tag{9.5}$$

Here, Th_1 is the threshold, the maximum allowable distance from a switch to a controller.

9.4.1 Solution Approaches

In this subsection, we will present several proposed solutions for problem I. The problem is NP-Hard and its NP-Hardness can be proved by converting the problem into the famous graph partitioning problem. We propose three approaches based on greedy and clustering methods.

9.4.2 Greedy Controller Assignment

In this approach, we assign a controller to each switch in a greedy way. We consider that each controller is a bucket and each switch is an element that will eventually put in a bucket. The process is composed of two parts: initialization of bucket and incremental growth of bucket. The initialization of a bucket is very important because it directs the rest of the assignments. If the initially assigned switches are very close to each other, then there will be fewer options to grow each bucket. Therefore, we select a switch for each

Algorithm 22 Find a switch assignment

Input: Topology G , set of controller C , set of flows F .

Output: A set of links to block from L_c .

```
1: Procedure: FIND-ASSIGNMENT( $G$ )
2:    $\forall c \in C$   $B[c] \leftarrow$  initial switch.
3:   for  $c \in C$  do
4:      $Candidate \leftarrow$  NEI( $B[C]$ )
5:     for  $s \in Candidate$  do
6:        $cost[s] \leftarrow$  COST( $s, B, F, G$ )
7:      $B[c] = B[c] \cup$  ARGMIN( $cost$ )
8:      $A(\text{ARGMIN}(\text{COST})) \leftarrow c$ 
9:   return  $A$ 
```

bucket in such a way that they are at least a certain number of hops away from each other.

We also prioritize the distance from the controller to the switch.

After the initialization of the buckets, we consider some candidates for the extension of each bucket. The candidate switches for extension of a bucket include the neighbors of each switch in that bucket. We pick the switch that adds a minimum amount of cost. The process continues until all of the switches are covered. The complete algorithm is shown in Alg. 22.

9.4.3 An Example to Greedy Controller Assignment

Let us consider the topology and flows in Fig. 9.2. There are eight SDN switches (A, B, C, D, W, X, Y , and Z) and two controllers (C_1 , and C_2). There are four flows (f_1, f_2, f_3 , and f_4) originating at s_1, s_2, s_3 , , and s_4 and destined to d_1, d_2, d_3 , , and d_4 , respectively. Distances from C_1 to the switches A, B, C , and , D

According to Alg. 22, we first need to initialize the buckets for the controllers. Let the minimum distance among the initial items in the bucket be 2. We consider $\omega_1 = 0.08$ and $\omega_2 = 0.8$ There are several options for choosing the initial items of the buckets. Let us choose $\{\{A\}, \{W\}\}$ where the first and second elements correspond to the bucket related to C_1 and C_2 , respectively.

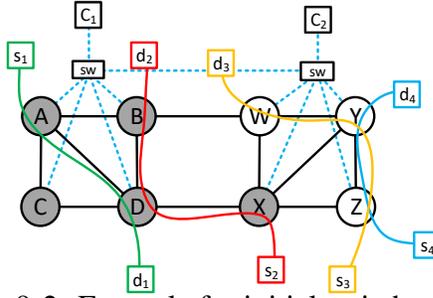


FIGURE 9.2: Example for initial switch migration.

Now, for the first and second buckets, we have three candidates for extension $\{B, C, D\}$ and $\{B, X, Y\}$, respectively. Now we need to calculate the cost of extension for each candidate element. The cost of adding C to the first bucket is $0.8 \times 0 + 0.08 \times 0 + 0.12 \times 2 = 0.24$. This is because there is no new path construction requests or intermediate query requests. The cost of adding B to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 2 = 1.04$. This is because there is one new path construction requests and no intermediate query requests. The cost of adding D to the first bucket is $0.8 \times 1 + 0.08 \times 2 + 0.12 \times 2 = 1.2$. This is because there is one new path construction requests and two intermediate query requests.

Next, we need to calculate the cost of extension for each candidate element of the second bucket. The cost of adding B to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 3 = .26$. This is because there is no new path construction requests or intermediate query requests. The cost of adding W to the first bucket is $0.8 \times 1 + 0.08 \times 0 + 0.12 \times 2 = 1.04$. This is because there is one new path construction requests and no intermediate query requests. The cost of adding Y to the first bucket is $0.8 \times 2 + 0.08 \times 0 + 0.12 \times 2 = 1.2$. This is because there is one new path construction requests and two intermediate query requests. The cost of adding X to the first bucket is $0.8 \times 1 + 0.08 \times 2 + 0.12 \times 2 = 1.8$. This is because there is one new path construction requests and two intermediate query requests.

Therefore, adding switch C to the first bucket produces the lowest cost increase. Therefore, the new buckets are $\{\{A, C\}, \{W\}\}$. Similarly, we calculate the costs for each can-

	A	B	C	D	W	X	Y	Z	C ₁	C ₂
A	0	1	1	1	2	2	3	3	2	3
B	1	0	2	1	1	2	2	3	2	3
C	1	2	0	1	3	2	3	3	2	3
D	1	1	1	0	2	1	2	2	2	3
W	2	1	3	2	0	1	1	2	3	2
X	2	2	2	1	1	0	1	1	3	2
Y	3	2	3	2	1	1	0	1	3	2
Z	3	3	3	2	2	1	1	0	3	2
C ₁	2	2	2	2	3	3	3	3	0	3
C ₂	3	3	3	3	2	2	2	2	3	0

FIGURE 9.3: Distance matrix for clustering.

didate in the candidate sets and pick the one that produces the lowest cost. Finally, the buckets are $\{\{A, C, D, X\}, \{W, Y, Z\}\}$ which represents that switches A, B, C, D, and X will be assigned to controller C_1 and switches W, Y, and Z will be assigned to controller C_2 . The total cost for this assignment is 24.6.

Theorem 24. *The complexity of Alg. 22 is $O(|C|(|V|^2 + |V||F|))$.*

Proof. To calculate the complexity of Alg. 22, we need to calculate complexity of $\text{COST}(s, B, F, G)$. If we pre-compute the number of path construction requests and intermediate queries, then $\text{COST}(s, B, F, G)$ can be obtain in constant time. The pre-computation takes $O(|V||F|)$. Step 5 to 8 takes $O(|V|^2)$ and the loop at Step 4 runs for $|C|$ times. Therefore, the Alg. 24 takes $O(|C|(|V|^2 + |V||F|))$. \square

9.4.4 Hierarchical Clustering

We group the switches using the hierarchical clustering algorithm. We formulate the distance matrix from the distance between two nodes in topology. The distance values between a pair of nodes are normalized by dividing with the maximum distance. The normalised distances are used for hierarchical clustering. The distance matrix is represented as follows:

$$D[u, v] = \frac{d_{u,v}}{\max_{u,v \in V} d_{u,v}} \quad (9.6)$$

Next, we use D to cluster the nodes into $|C|$ groups. We use the hierarchical clustering algorithm to partition the nodes. We set the maximum class to be the number of controllers ($|C|$). This clustering will group the most similar nodes in a cluster. We pick a cluster and calculate the average number of hops to each controller. We assign the closest controller to the nodes in that cluster. Then, we continue the process with the unassigned clusters and controllers. The complete approach is shown in Algorithm 23.

9.4.5 An Example of Hierarchical Clustering Solution

We are going to use the topology in Fig. 9.3 to explain this solution. We group the nodes using the hierarchical clustering algorithm. We formulate the distance matrix from the distance matrix. The similarity values between a pair of nodes are subtracted from the maximum of the similarity values (except the similarity value of a node with itself) to get dissimilarities. The dissimilarities are used as distances for hierarchical clustering. The distance matrix is represented as follows:

Using Equation 9.6, we calculate $D'[A, B]$. Therefore, $D[A, B] = d_{A,B}/4 = 0.25$. Similarly, we calculate that $D'[B, C] = 2$ and $D'[B, D] = 1$. So, we can see that nodes 1 and 2 are more similar than nodes 1 and 4 or 2 and 7. The similarity score between 1 and 3 is 0. This is because they are different in type and they do not have any similar neighbors. Finally, by using hierarchical clustering we found that the clusters are $\{\{A, C, D, X, C_1\}, \{W, Y, Z, C_2\}\}$ which represents that switches A, B, C, D, and X will be assigned to controller C_1 and switches W, Y, and Z will be assigned to controller C_2 . The total cost for this assignment is also 24.6.

Theorem 25. *The complexity of Alg. 23 is $O(|V|^3)$.*

Proof. To calculate the distance matrix we need $O(|V|^2)$. The the rest of the part is determined by the complexity of the hierarchical clustering. The standard algorithm for

Algorithm 24 Find a switch assignment

Input: Topology G , set of controller C , set of flows F , current Assignment A .

Output: A set of links to block from L_c .

```
1: Procedure: FIND-ASSIGNMENT( $G, C, F, A$ )
2:   while DIFF( $A, A'$ ) <  $K$  do
3:     Divide  $C$  into  $C_o$  and  $C_u$ 
4:     for all  $c \in C_o$  do
5:       for  $v, u \in V : A(v) = c \ \& \ A(u) \in \text{NEI}(c)$  do
6:          $B[u, v] \leftarrow \text{BEN}(v, u)$ 
7:        $u, v \leftarrow \text{ARGMAX}(B)$ 
8:        $A'(v) \leftarrow A(u)$ 
9:   return  $A'$ 
```

Let the topology be $G = (V, E)$ where $V = \{v_1, v_2, \dots, v_N\}$ is the set of N SDN switches and E is the set of links. Let the set of controllers $C = \{c_1, c_2, \dots, c_M\}$ contains M controllers. The set of flows in the network is denoted by $F = \{f_1, f_2, \dots, f_K\}$. Therefore, the problem can be expressed as the following:

$$\begin{aligned} & \text{minimize} \quad \max_{c \in C} C(A, c) \\ & \text{subject to} \quad \forall_{v \in V}, d_{vA(v)} \leq Th_1, \\ & \quad \quad \quad \text{DIFF}(A, A') \leq Th_2 \end{aligned} \tag{9.7}$$

Here, Th_2 is the threshold of changes in an assignment. $\text{DIFF}(A, A')$ is the number of changes between the current assignment A' and the next assignment A . Therefore, $\text{DIFF}(A, A')$ can be expressed as the following:

$$\begin{aligned} \text{DIFF}(A, A') &= \sum_{v \in V} x(A(v), A'(v)) \\ x(a, b) &= \begin{cases} 1, & \text{if } a = b \\ 0, & \text{otherwise} \end{cases} \end{aligned} \tag{9.8}$$

9.5.1 Solution Approaches

In this subsection, we present our proposed solutions for problem II. Though we have added one more constraints, the problem is still NP-Hard We propose three approaches based on greedy methods.

9.5.2 Greedy Incremental Deployment

To find the new assignment, we first calculate the load of each controller. By using the equation 9.3 we calculate the load of the controllers in C . Then, we divide the C into two parts C_o and C_u ($C = C_o \cup C_u$) based on the load of the controller. If a load of a controller is higher (or lower) than a threshold then C_o (or C_u) will contain it. Therefore, C_o represents the set of the overloaded controller and C_u represents the set of the depleted controllers. The part of the topology that is controlled by an overloaded controller is called the overloaded domain and the part of the topology that is controlled by a depleted controller is called the depleted domain.

Next, for each overloaded domain, we consider the depleted neighboring domains as the candidate domains. Then, we need to find a switch that will be migrated to the neighboring domain. Let $A(v) \in C_o$ and $A(u) \in C_u$ such that there is a link between u and v . In this case, v may be considered to migrate to $A(u)$. We consider a parameter called migration benefit for determining the priority of migration in this greedy approach. The migration benefit is the ratio of increased total load and the amount of transferred load. Let A be the current assignment and A' is the new assignment after migrating v to the neighboring domain ($A'(v) = A(u)$).

$$\begin{aligned} \text{BEN}(v, u) = \\ C(A, A(v)) + C(A, A(u)) - C(A', A'(v)) - C(A', A'(u)) \end{aligned} \tag{9.9}$$

Among all eligible migrations we pick the pair that provides the maximum benefit. The process continues until the number of migrations is equal to K .

9.5.3 An Example of Incremental Assignment

Let us consider the current topology and flows in Fig. 9.4 and the previous assignment $\{A, B, C, D, X\}$ to C_1 and $\{W, Y, Z\}$ to C_2 . Because of three new flows (f_5, f_6 and f_7) and an expired flow f_4 in Fig. 9.4, controller C_1 gets overwhelmed. Therefore, the $C_o\{C_1\}$ and $C_u\{C_2\}$. There are two possible migration options here: migrate B to C_2 or migrate X to C_2 . Next, we will calculate the benefit of migration for both options.

For the first option (migrate B to C_2) the benefit is $\text{BEN}(v, u) = (4.48 + 3.36 - 3.84 - 4.96) = -0.96$. If we migrate X to C_2 then the benefit will be $\text{BEN}(v, u) = (4.48 + 3.36 - 3.68 - 3.68) = 0.48$. Therefore, we get higher benefit of migrating the switch X to C_2 . Fig. 9.4 shows the new assignment after migration.

Theorem 26. *The complexity of Alg. 24 is $O(|F||V|K)$.*

Proof. To calculate the complexity of Alg. 24, we need to calculate complexity of $\text{BEN}(u, v)$. According to the Eq. 9.4, it takes $O(|F||V|)$ to calculate cost. Therefore, the complexity of $\text{BEN}(u, v)$ is $O(|F||V|)$. The nested loops in Step 4-8 take $O(|V||C|)$. The loop at Step 2 executes at most K times. Therefore, the Alg. 24 takes $O(|F||V|K)$. \square

9.6 Simulation and Experiments

In this section, we present our experimental and simulation results comparing some existing works.

9.6.1 Experimental Settings

We conduct the experiments in our data center with fifteen SDN switches. We use the regular switches in the control plane. The partial topology of the data center is shown in Fig. 9.5(a). Nodes 2 to 16 are Pica 8 (P-3297) SDN switches. The numbers at the ends of a link denote the port number where it is plugged. There are four servers (101-104) that are connected at the leaf level switches. We create three flows $\{101 \rightarrow 102, 101 \rightarrow 103, 101 \rightarrow 104\}$ and record the ping delays (round trip delay). Initially, switches between

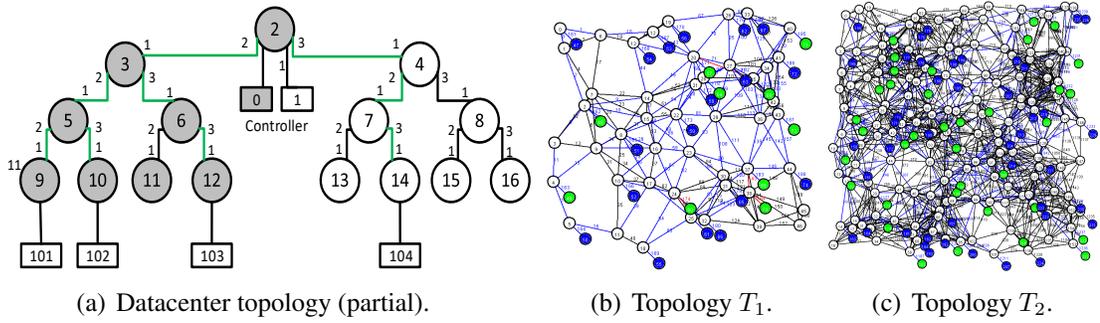


FIGURE 9.5: Generated topologies.

2 and 12 are controlled by controller 0, and switches between 4 and 16 are controlled by controller 1.

We use ONOS (2.2.2) as the controller software. We develop a Java program that can connect the switches through SSH and change the controller of any switch. The program does not need to run on the same machine as the controller because it communicates with the controller through SSH. We also develop a flow generator program that keeps generating flows to random destinations. The controller modifier program first uses the direct replacement method and observes the delay to restore the communication of $101 \rightarrow 104$ flow. We observe an average delay of 5.2 seconds for the direct replacement of controllers. The controller modifier program is used to apply the switch to backup methods. We observe an average delay of 5.6 seconds for this method. Therefore, from these experiments, we conclude that the direct replacement method works better than the switch to backup method.

After that we run some experiments to obtain the values of the weights (ω_1 , and ω_2). To do so, we needed to change the control topology to make the different number of hops reach the controller. We change the number of hops between switches and the controllers from 2 to 5 and observe the round trip time of the three flows mentioned above. We observe the first round trip time is higher than the following round trip times. This is because of the path establishment overhead of the controller. This delay is used to calculate the weight of

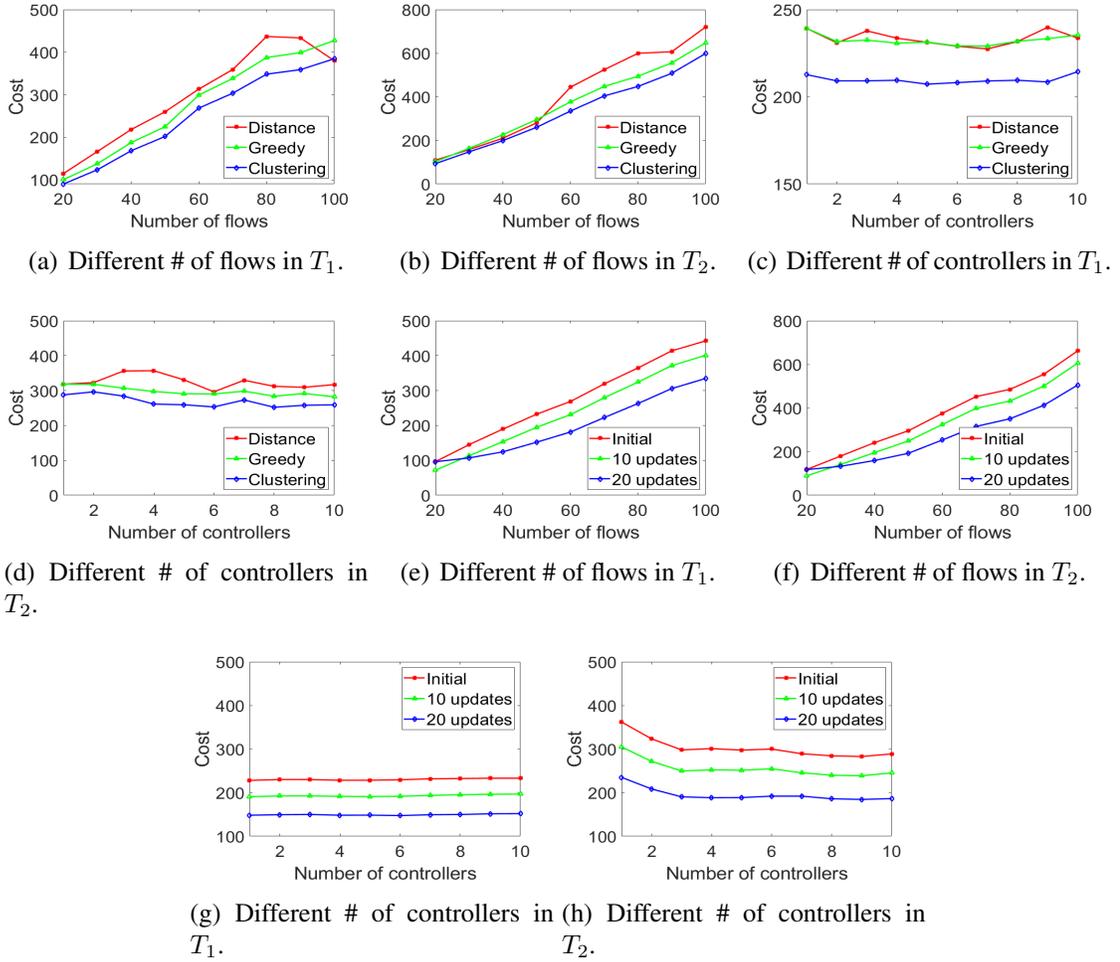


FIGURE 9.6: Simulation results.

path construction cost (ω_1). From the delays, we calculate the delay of the response from the controller for different hops (ω_2). We use these values in the simulations presented in the next subsections.

9.6.2 Simulation Settings

We conducted all the experiments with a custom-built Java simulator. The main reason for using a custom-built simulator is its scalability and fast execution time. We do not need to analyze transmission time or natural packet drop issues rather to calculate cost of switch controller assignment, migration costs, and benefits. The network topologies we considered contain 100 – 300 switches. Based on our experience, using NS3 or other

similar simulators for this kind of simulation would take several days. That is why we built our own Java multi-threaded simulator to get the results quickly.

We generate random topologies by dividing an area of 500×500 square unit into 50×50 blocks. In each block, a certain number of nodes are placed randomly to make an uniform distribution of nodes. We randomly select some nodes and attach a source or destination. We set the link capacities as 100 Mbps for simplicity and ease of comparison. After that, we generate the desired number of flows by randomly selecting a source and a destination. We set the data rate randomly. The initial routing of the flows are done using shortest path routing. Fig. 9.5 shows the randomly generated topologies. Topology T_1 is small and sparse and contains 75 nodes. Topology T_2 is large and dense and contains 233 nodes.

We measure the cost and compare with different approaches. In the distance based approach each switch is assigned to the closest available controller. All of the results in the plot are the average of 1000 runs.

9.6.3 Simulation Results

Firstly, we observe the cost of flow assignment by changing different parameters such as the number of rules and number of controllers. Fig. 9.6(a) shows the cost of assignment for different number of flows in T_1 . We vary the number of flows from 20 to 100 and keep the maximum data rate as 10 Mbps. We set the number of controllers as 5. For all assignment methods, the cost increases with the increase of number of flows. This is because when the number of flows are higher, the controllers get a higher number of requests from the switches. The distance-based clustering-based method produces the highest and the lowest amount of costs for all numbers of flows, respectively. The cost produced by greedy assignment method remains in between them. When the number of flows is 20 and distance-based method is used, the average cost is 114.78. When the greedy method is used, the average cost is 100.42. When the clustering-based method is used, the

average cost is 90.14. The cost decreased by about 12% and 21% in greedy and clustering methods. When the number of flows is 80 and distance-based method is used, the average cost is 407.34. When the greedy method is used, the average cost is 366.25. When the clustering-based method is used, the average cost is 328.95. The cost decreased by about 10% and 19% in greedy and clustering methods.

Fig. 9.6(b) shows the cost of assignment for different number of flows in T_2 . We keep the same settings as the previous simulation. We observe a higher overall cost in T_2 than T_1 . This is because the topology is large and the average number of hops between the sources and the destinations are also higher in T_2 . The distance-based clustering-based method produces the highest and the lowest amount of costs for all numbers of flows, respectively. The cost produced by greedy assignment method remains in between them. When the number of flows is 20 and distance-based method is used, the average cost is 109.39. When the greedy method is used, the average cost is 103.28. When the clustering-based method is used, the average cost is 1.54. The cost decreased by about 5.5% and 14.5% in greedy and clustering methods. When the number of flows is 100 and distance-based method is used, the average cost is 719.63. When the greedy method is used, the average cost is 647.03. When the clustering-based method is used, the average cost is 1.54. The cost decreased by about 10% and 17% in greedy and clustering methods.

Next, we observe the cost of flow assignment by changing the number of controllers. Fig. 9.6(c) shows the cost of assignment for different number of controllers in T_1 . We vary the number of controllers from 2 to 10 and keep the number of flows as 50. Other settings are kept same as the previous simulation. For all assignment methods, the cost decreases slightly with the increase of number of controllers. This is because when the number of controllers are higher, there are more options of assignment and the average distance between switches and controllers reduces. The distance-based clustering-based method also produces the highest and the lowest amount of costs for all numbers of controllers, respectively. The cost produced by greedy assignment method is lightly lower than the

distance based method. When the number of controllers is 2 and distance-based method is used, the average cost is 230.92. When the greedy method is used, the average cost is 231.63. When the clustering-based method is used, the average cost is 209.10. The cost decreased by about 0.3% and 9% in greedy and clustering methods. When the number of controllers is 10 and distance-based method is used, the average cost is 233.70. When the greedy method is used, the average cost is 235.36. When the clustering-based method is used, the average cost is 214.37. The cost increased by about 0.85% and 8.15% in greedy and clustering methods.

Fig. 9.6(d) shows the cost of assignment for different number of controllers in T_2 . We keep the same settings as the previous simulation. We also observe a higher overall cost in T_2 than T_1 . For all assignment methods, the cost decreases slightly with the increase of number of controllers. We also observe similar behavior as in T_1 . When the number of controllers is 2 and distance-based method is used, the average cost is 322.72. When the greedy method is used, the average cost is 317.95. When the clustering-based method is used, the average cost is 296.22. The cost decreased by about 1.5% and 8% in greedy and clustering methods. When the number of controllers is 10 and distance-based method is used, the average cost is 317.12. When the greedy method is used, the average cost is 281.97. When the clustering-based method is used, the average cost is 259.08. The cost increased by about 11% and 18% in greedy and clustering methods.

Next, we observe cost of flow assignment by using the incremental switch assignment. Fig. 9.6(e) shows the cost of assignment for different number of controllers in T_1 . We vary the number of flows from 20 to 100 and keep the number of controllers at 5. We first assign the switches by using the clustering-based method, then we randomly delete 10 (or 20) flows and add 10 (or 20) flows. For all changes, the cost increases with the increase of number of flows. When the number of flows is 20, we observe similar cost for all changes. When the number of flows is 100, the cost decreases by about 10% and 24% for 10 and 20 flow changes, respectively. We also observe similar behavior in T_2 shown in

9.6(f). When the number of flows is 20, we observe similar cost for all changes. When the number of flows is 100, the cost decreases by about 8% and 23% for 10 and 20 flow changes, respectively.

Figs. 9.6(g) and 9.6(h) show the cost of assignment for different number of controllers in T_1 . We vary the number of controllers from 2 to 10 and keep the number of flows at 50. For all changes, the cost decreases slightly with the increase of number of controllers. For all changes, the cost decreases slightly with the increase of number of controllers. When the number of controllers is 2, we observe that the cost decreases by about 16% and 35% for 10 and 20 flow changes, respectively. We also observe similar behavior in T_2 . When the number of controllers is 10, we observe similar cost difference as 2 controllers. Therefore, from the above simulations we can conclude that the clustering based approach works better than other two approaches.

9.7 Summary

Controller load balancing for large-scale SDN datacenters is very important for ensuring good performance. To balance the load of the controllers, we need to migrate SDN switches from overloaded controller to relatively lightly loaded controller. Currently, switch migration functionality is not available in commercially available switches. Therefore, we investigate the possibility of switch migration by accessing the system using ssh. We have formulated two problems for switch-controller assignments that consider initial deployment and incremental modification for load balancing. We proposed several solutions to the problems. Our extensive simulations by using parameters driven from experiments, show great support for the solutions. In the next chapter, we are going to focus more on SDN network optimization than defending network attacks. We will explore an interesting and new topic called switch migration which is an important aspect to achieve load balancing in a large scale SDN network.

CHAPTER 10

CONCLUSION AND FUTURE WORKS

10.1 Summary of Contributions

The DDoS attack is the most powerful attack to make a service unavailable to users. As the most important component in a network, routers are upgraded as filter routers and we use them for protecting DDoS attacks. In the proposed four-phase DDoS protection system, we contributed to the most important phase which is filter assignment. We formulated multiple problems and provide greedy and optimal solutions. We compare the performances of proposed scheduling policies with some of the existing approaches. Simulation results show that our proposed approaches work better than the other approach. Both the source-based and destination-based filters have some advantages and limitations.

We extended our tree based solution to more general directed acyclic graph based topology scenarios which is more practical. We considered two types of attacks with different objectives for attackers and formulate the defense mechanism as optimization problems. We provide an approximation filter assignment solution to block the maximum amount of attack traffic with a limited number of filters. We propose a dynamic programming-based optimal solution to assign a moving target defense approach to some

nodes. We conducted extensive simulations to observe behaviors of the defense mechanisms for different settings.

We have also proposed a transit-link DDoS attack mitigation system. Regular DDoS defense system cannot protect the transit-link DDoS attack because the attack packets do not go to the victim. We proposed an optimal filter assignment policy to redirect the legitimate traffic through non congest path. We also compare performances of proposed assignment policy with some existing approaches. Simulation results show that our proposed approaches work much better than the other existing approaches. After that we focused on defending against the internal attacks which is harder to detect and protect against

We have used SDN switches and virtual machines to monitor internal network flows. We presented two problems for assigning flows to monitors and selecting the best locations for the monitors. We compare the performances of the proposed placement policy with an existing greedy approach. We extended the monitoring system to the scenario where the available number of virtual machines is less than the required virtual machines for monitoring all the flows. We sub-sampled the flows and proposed a sampling rate distribution approach. We implemented the system partially and our experimental results supported greatly our system.

We have explored some of the optimization areas of software defined networks. We have proposed a framework for redirecting some of the flows of a congested link in such a way that introduces minimum number of rules changes. When the number of rules in an SDN switch is higher than its fast accessible rules storage, the delay increases dramatically. Besides, changing rules in a higher number of SDN switches takes a long time, which causes interruption in flows. We propose and evaluate performances of our proposed system and simulation supports that the proposed approaches can minimize improve the performance of an SDN network.

Finally, we have studied large scale SDN networks where multiple controller controls the network. We have formulated two problems for switch-controller assignments that consider initial deployment and incremental modification for load balancing and proposed several solutions to the problems. Our extensive simulations are driven by parameters obtained from real system showed great support for the solutions.

10.2 Future Extensions

We have several future plan for extending some of the solution addressed in this dissertation. Our future plans are categorised into three areas that we are going to discuss in the following subsections.

10.2.1 Defending Network Attacks

Nowadays the internet service providers (ISPs) usually do not use the shortest path routing. The routing is influenced by congestion and relation with other ISPs. The routing and network engineering are designed to optimize the congestion based on the demand of their customers. Our proposed FR-based model can easily adopt any routing protocol easily but changing the route of a destination may cause some overhead to the ISP. They should be aware of this issue and should not accept filters that lower their quality of service. Further research is needed to identify whether or not a filter is harmful.

In our research, we consider a victim in a network. In reality, there can be multiple victims. According to our assumptions, all the victims protect their user by deploying filters separately. There is no victim to victim communication. This method is not efficient when multiple victims together want to protect a group of users. Then a filter may affect traffic to multiple victims. If the filter's condition is a prefix of address instead of an address, then the victims having the prefix in their address will be able to direct traffic which are coming to any of the victim having that prefix in address. This system requires that the victims having the prefix will cooperate and may be owned by an entity. Existing internet service providers or autonomous systems have similar IP allocation. Therefore, in

the future we can extend the problem described in this paper to solve the multiple victim scenarios.

10.2.2 Optimization in Software Defined Networks

Traffic engineering is one of the important parts in a datacenter that can improve the performance of an SDN network. Because of the limited space on SDN switches, it is important to keep the number of rules as small as possible while ensuring a guaranteed minimal performance. This performance include the delay requirements and reliability/downtime requirements requested by the users. If the number of rules in a switch is higher than a threshold, then the forwarding delay jumps up. We plan to implement a virtual tunnel based approach which will help to reduce the number of rules in an SDN switch while ensuring the quality-of-service requirements. A virtual tunnel is basically a conceptual structure in the controller of an SDN network where a group of flows follow a common path. A wise formulation of tunnels can reduce the number of rules needed dramatically by forwarding multiple flows through the same tunnels with the tunnel's common rules. Therefore, in future we plan to extend our proposed solution to adopt the quality-of-service requirements while ensuring protection from different attacks.

10.2.3 Improving Reliability in Service Function Chain

Nowadays, the quality-of-service requirements of a user is not only limited to delay and connection reliability, it can include different network functions on the way with some dependency relations. We call this requirement as service function chain requirement. Currently, we considered our system without any network function. In the future, we can explore to extend our systems to adopt ensuring service function chain requirements. Additionally, we plan to explore more traffic engineering issues to address the service function chain requirement including security and optimization.

10.3 Concluding Remarks

This dissertation has (1) presented defense strategies of multiple variant of DDoS attack in different scenarios; (2) proposed multiple optimization techniques for software defined networks considering different types of attacks including link flooding attacks; and (3) explored some new performance related issues encountered by modern software defined networks including controller load balancing.

At a high level, the goal of this dissertation is to design and build efficient strategies to defend network attacks, optimize network performance, which is very complicated because of a diverse network equipments, traffic engineering strategy, relation among the ISPs, and evolving nature of network attacks. We leverage the emerging existing network technologies to solve these challenges with efficient algorithms and experiments in our system. We believe this is an interesting research area and are useful to obtain improved and secured society. We are excited about future work on designing new effective algorithms and implementing them using the future generation equipment.

Related Publications

- [Related1] R. Biswas and J. Wu, “Preserving source and destination location privacy with controlled routing protocol,” *International Journal of Security and Networks*, vol. 13, no. 3, pp. 187–198, 2018.
- [Related2] R. Biswas, J. Wu, and X. Du, “Mitigation of the spectrum sensing data falsifying attack in cognitive radio networks,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
- [Related3] R. Biswas, J. Wu, X. Du, and Y. Yang, “Mitigation of the spectrum sensing data falsifying attack in cognitive radio networks,” *Cyber-Physical Systems*, vol. 0, no. 0, pp. 1–20, 2020.
- [Related4] R. Biswas, J. Wu, and X. Li, “A capacity-aware distributed denial-of-service attack in low-power and lossy networks,” in *2019 IEEE 40th Sarnoff Symposium*, 2019, pp. 1–6.
- [Related5] R. Biswas and J. Wu, “Co-existence of lte-u and wi-fi with direct communication,” in *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, 2019, pp. 1–6.
- [Related6] —, “Minimizing the number of channel switches of mobile users in cognitive radio ad-hoc networks,” *Journal of Sensor and Actuator Networks*, vol. 9, no. 2, 2020. [Online]. Available: <https://www.mdpi.com/2224-2708/9/2/23>
- [Related7] —, “Filter assignment policy against distributed denial-of-service attack,” in *2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS)*, 2018, pp. 537–544.
- [Related8] —, “Optimal filter assignment policy against distributed denial-of-service attack,” *IEEE Transactions on Dependable and Secure Computing*, pp. 1–1, 2020.

- [Related9] R. Biswas, J. Wu, and A. Srinivasan, “Cost-aware optimal filter assignment policy against distributed denial-of-service attack,” in *2019 Resilience Week (RWS)*, vol. 1, 2019, pp. 57–63.
- [Related10] R. Biswas, J. Wu, W. Chang, and P. Ostovari, “Optimal filter assignment policy against transit-link distributed denial-of-service attack,” in *2019 IEEE Global Communications Conference (GLOBECOM)*, 2019, pp. 1–6.
- [Related11] R. Biswas, J. Wu, and Y. Chen, “Optimal monitor placement policy against distributed denial-of-service attack in datacenter,” in *2019 Resilience Week (RWS)*, vol. 1, 2019, pp. 64–70.
- [Related12] R. Biswas, S. Kim, and J. Wu, “Sampling rate distribution for flow monitoring and ddos detection in datacenter,” *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 2524–2534, 2021.
- [Related13] R. Biswas and J. Wu, “Traffic engineering to minimize the number of rules in sdn datacenters,” *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.

Under Revision/Submitted

- [Submitted1] R. Biswas and J. Wu, “Protecting resources against volumetric and non-volumetric network attacks,” in *ITC 33 - Networked Systems and Services*, 2021.
- [Submitted2] R. Biswas, J. Wu, W. Chang, and P. Ostovari, “Optimal filter assignment policy against transit-link distributed denial-of-service attack,” in *IEEE Transactions on Dependable and Secure Computing*, 2021.
- [Submitted3] R. Biswas and J. Wu, “Minimizing the number of rules to mitigate link flooding attack in sdn-based datacenters,” in *15th International Conference on Networking, Architecture, and Storage*, 2021.
- [Submitted4] —, “Efficient switch migration for controller load balancing in software defined networking,” in *IEEE Conference on Communications and Network Security*, 2021.

BIBLIOGRAPHY

- [1] “CISCO Annual Report,” <https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html>.
- [2] J. Caballero, C. Grier, C. Kreibich, and V. Paxson, “Measuring Pay-per-install: The Commoditization of Malware Distribution,” in *Proceedings of the 20th USENIX Conference on Security*. Berkeley, CA, USA: USENIX Association, Aug 2011.
- [3] M. S. Kang, V. D. Gligor, and V. Sekar, “Defending Against Evolving DDoS Attacks: A Case Study Using Link Flooding Incidents,” in *Security Protocols XXIV*, J. Anderson, V. Matyáš, B. Christianson, and F. Stajano, Eds. Springer International Publishing, Apr 2017.
- [4] “Cloudflare,” blog.cloudflare.com.
- [5] V. Corey, C. Peterman, S. Shearin, M. S. Greenberg, and J. Van Bokkelen, “Network forensics analysis,” *IEEE Internet Computing*, vol. 6, no. 6, Nov 2002.
- [6] T. Winter, P. Thubert, A. Brandt, J. Hui, R. Kelsey, P. Levis, K. Pister, R. Struik, J.-P. Vasseur, and R. Alexander, “RPL: IPv6 routing protocol for low-power and lossy networks,” Tech. Rep., Mar 2012.
- [7] US Code: Title 18,1030, “Fraud and related activity in connection with computers — government printing office,” www.gpo.gov, 2014.
- [8] J. Wang and I. C. Paschalidis, “Statistical traffic anomaly detection in time-varying communication networks,” *IEEE Transactions on Control of Network Systems*, vol. 2, no. 2, Jun 2015.
- [9] W. Wei, F. Chen, Y. Xia, and G. Jin, “A rank correlation based detection against distributed reflection dos attacks,” *IEEE Communications Letters*, vol. 17, no. 1, Jan 2013.
- [10] A. Kulkarni and S. Bush, “Detecting distributed denial-of-service attacks using kolmogorov complexity metrics,” *J. Netw. Syst. Manage.*, vol. 14, no. 1, Mar 2006.
- [11] T. A. Ahanger, “An effective approach of detecting ddos using artificial neural networks,” in *2017 International Conference on Wireless Communications, Signal Processing and Networking*, Mar 2017.

- [12] X. Ma and Y. Chen, "Ddos detection method based on chaos analysis of network traffic entropy," *IEEE Communications Letters*, vol. 18, no. 1, Jan 2014.
- [13] Z. Tan, A. Jamdagni, X. He, P. Nanda, and R. P. Liu, "A system for denial-of-service attack detection based on multivariate correlation analysis," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 2, Feb 2014.
- [14] P. Ning and S. Jajodia, "Intrusion detection techniques," 2004.
- [15] N. Ye, S. Emran, Q. Chen, and S. Vilbert, "Multivariate statistical analysis of audit trails for host-based intrusion detection," *IEEE Transactions on Computers*, vol. 51, no. 7, Jul 2002.
- [16] V. Matta, M. D. Mauro, and M. Longo, "DDoS Attacks With Randomized Traffic Innovation: Botnet Identification Challenges and Strategies," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, Aug 2017.
- [17] H. Luo, Z. Chen, J. Li, and A. V. Vasilakos, "Preventing Distributed Denial-of-Service Flooding Attacks With Dynamic Path Identifiers," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, Aug 2017.
- [18] J. Zheng, Q. Li, G. Gu, J. Cao, D. K. Y. Yau, and J. Wu, "Realtime DDoS Defense Using COTS SDN Switches via Adaptive Correlation Analysis," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 7, Jul 2018.
- [19] G. Somani, M. S. Gaur, D. Sanghi, M. Conti, and M. Rajarajan, "Scale Inside-out: Rapid Mitigation of Cloud DDoS Attacks," *IEEE Transactions on Dependable and Secure Computing*, 2017.
- [20] C. Wang, T. T. N. Miu, X. Luo, and J. Wang, "SkyShield: A Sketch-Based Defense System Against Application Layer DDoS Attacks," *IEEE Transactions on Information Forensics and Security*, vol. 13, no. 3, Mar 2018.
- [21] B. Rashidi, C. Fung, and E. Bertino, "A Collaborative DDoS Defence Framework Using Network Function Virtualization," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 10, Oct 2017.
- [22] D. Seo, H. Lee, and A. Perrig, "PFS: Probabilistic filter scheduling against distributed denial-of-service attacks," in *2011 IEEE 36th Conference on Local Computer Networks*, Oct 2011.
- [23] ———, "APFS: Adaptive Probabilistic Filter Scheduling against distributed denial-of-service attacks," *Computers & Security*, vol. 39, Nov 2013.
- [24] "Autonomous systems AS-733," <https://snap.stanford.edu/data/as-733.html>.
- [25] M. Zareapoor, P. Shamsolmoali, and M. A. Alam, "Advance DDOS detection and mitigation technique for securing cloud," *International Journal of Computational Science and Engineering*, vol. 16, no. 3, 2018.

- [26] B. K. Joshi, N. Joshi, and M. C. Joshi, "Early Detection of Distributed Denial of Service Attack in Era of Software-Defined Network," in *2018 Eleventh International Conference on Contemporary Computing*, Aug 2018.
- [27] D. Almomani, M. Alauthman, F. Albalas, O. Dorgham, and A. Obeidat, "An On-line Intrusion Detection System to Cloud Computing Based on Neucube Algorithms," *International Journal of Cloud Applications and Computing*, vol. 8, 04 2018.
- [28] M. Suk Kang, V. D. Gligor, and V. Sekar, "SPIFFY: Inducing Cost-Detectability Tradeoffs for Persistent Link-Flooding Attacks," in *Network and Distributed System Security Symposium*, Jan 2016.
- [29] J. Ioannidis and S. M. Bellovin, "Implementing Pushback: Router-Based Defense Against DDoS Attacks," in *Network and Distributed System Security Symposium*, Mar 2002.
- [30] S. B. Lee, M. S. Kang, and V. D. Gligor, "CoDef: Collaborative Defense Against Large-scale Link-flooding Attacks," in *Proceedings of the Ninth ACM Conference on Emerging Networking Experiments and Technologies*. New York, NY, USA: ACM, Dec 2013.
- [31] J. M. Smith and M. Schuchard, "Routing Around Congestion: Defeating DDoS Attacks and Adverse Network Conditions via Reactive BGP Routing," in *IEEE Symposium on Security and Privacy*, May 2018.
- [32] N. Polatidis, M. Pavlidis, and H. Mouratidis, "Cyber-attack path discovery in a dynamic supply chain maritime risk management system," *Computer Standards & Interfaces*, vol. 56, pp. 74–82, 2018.
- [33] X. Liu, "A network attack path prediction method using attack graph," *Journal of Ambient Intelligence and Humanized Computing*, pp. 1–8, 2020.
- [34] M. Ge, J. B. Hong, S. E. Yusuf, and D. S. Kim, "Proactive defense mechanisms for the software-defined internet of things with non-patchable vulnerabilities," *Future Generation Computer Systems*, vol. 78, pp. 568–582, 2018.
- [35] M. Touhiduzzaman, A. Hahn, and A. K. Srivastava, "A diversity-based substation cyber defense strategy utilizing coloring games," *IEEE Transactions on Smart Grid*, vol. 10, no. 5, pp. 5405–5415, 2019.
- [36] R. Biswas and J. Wu, "Filter assignment policy against distributed denial-of-service attack," in *2018 IEEE 24th International Conference on Parallel and Distributed Systems*, Dec 2018.
- [37] B. B. Gupta and O. P. Badve, "Taxonomy of DoS and DDoS attacks and desirable defense mechanism in a Cloud computing environment," *Neural Computing and Applications*, vol. 28, no. 12, Dec 2017.

- [38] A. Kanevsky, "Finding all minimum-size separating vertex sets in a graph," *Networks*, vol. 23, no. 6, pp. 533–541, 1993.
- [39] D. Seo, H. Lee, and A. Perrig, "PFS: Probabilistic filter scheduling against distributed denial-of-service attacks," in *2011 IEEE 36th Conference on Local Computer Networks*, Oct 2011.
- [40] —, "APFS: Adaptive Probabilistic Filter Scheduling against distributed denial-of-service attacks," *Computers and Security*, vol. 39, Nov 2013.
- [41] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris, "Resilient Overlay Networks," in *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*. ACM, Oct 2001.
- [42] K. Argyraki and D. R. Cheriton, "Loose Source Routing As a Mechanism for Traffic Policies," in *Proceedings of the ACM SIGCOMM Workshop on Future Directions in Network Architecture*. ACM, Aug 2004.
- [43] M. Motiwala, M. Elmore, N. Feamster, and S. Vempala, "Path Splicing," in *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 4. ACM, Aug 2008.
- [44] W. Xu and J. Rexford, "MIRO: Multi-path Interdomain Routing," in *Proceedings of the 2006 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*. ACM, Sep 2006.
- [45] X. Yang and D. Wetherall, "Source Selectable Path Diversity via Routing Deflections," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, Aug 2006.
- [46] S. B. Lee and V. D. Gligor, "FLoc : Dependable Link Access for Legitimate Traffic in Flooding Attacks," in *2010 IEEE 30th International Conference on Distributed Computing Systems*, Jun 2010.
- [47] S. Acid and L. M. De Campos, "An Algorithm for Finding Minimum D-separating Sets in Belief Networks," in *Proceedings of the Twelfth International Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., Aug 1996.
- [48] R. Biswas, J. Wu, W. Chang, and P. Ostovari, "Optimal filter assignment policy against transit-link distributed denial-of-service attack," in *IEEE Global Communications Conference*, Dec 2019.
- [49] R. Biswas, J. Wu, and A. Srinivasan, "Cost-aware optimal filter assignment policy against distributed denial-of-service attack," in *Resilience Week*, Nov 2019.
- [50] S. K. Fayaz, Y. Tobioka, V. Sekar, and M. Bailey, "Bohatei: Flexible and Elastic DDoS Defense," in *USENIX Security Symposium*, 2015.
- [51] A. Bates, K. Butler, A. Haeberlen, M. Sherr, and W. Zhou, "Let SDN Be Your Eyes: Secure Forensics in Data Center Networks," in *Workshop on Security of Emerging Networking Technologies*, Jan 2014.

- [52] M. A. Lopez and O. C. M. B. Duarte, “Providing elasticity to intrusion detection systems in virtualized Software Defined Networks,” in *2015 IEEE International Conference on Communications*, Jun 2015.
- [53] P. Lin, C. Wu, and P. Shih, “Optimal Placement of Network Security Monitoring Functions in NFV-Enabled Data Centers,” in *2017 IEEE 7th International Symposium on Cloud and Service Computing*, Nov 2017.
- [54] T. Alharbi, A. Aljuhani, and H. Liu, “Holistic DDoS mitigation using NFV,” in *2017 IEEE 7th Annual Computing and Communication Workshop and Conference*, Jan 2017.
- [55] Amazon, “Amazon web services: Overview of security processes,” May 2017.
- [56] A. Marshall, M. Howard, G. Bugher, B. Harden, C. Kaufman, M. Rues, and V. Bertocci, “Security best practices for developing windows azure applications,” *Microsoft Corp*, Jun 2010.
- [57] N. Jain and S. Kamara, “Attacking data center networks from the inside,” Jun 2015.
- [58] R. K. Ahuja, *Network Flows: Theory, Algorithms, and Applications*, 1st ed. Pearson Education, 2017.
- [59] A. Procopiou, N. Komninos, and C. Douligeris, “Forchaos: Real time application ddos detection using forecasting and chaos theory in smart home iot network,” *Wireless Communications and Mobile Computing*, vol. 2019, 2019.
- [60] C. Hu, S. Wang, J. Tian, B. Liu, Y. Cheng, and Y. Chen, “Accurate and efficient traffic monitoring using adaptive non-linear sampling method,” in *The 27th Conference on Computer Communications*, Apr 2008.
- [61] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, “Sampling and filtering techniques for ip packet selection”, rfc 5475,” 2009.
- [62] E. A. Hernandez, M. C. Chidester, and A. D. George, “Adaptive sampling for network management,” *Journal of Network and Systems Management*, vol. 9, no. 4, Dec 2001.
- [63] J. M. Silva, P. Carvalho, and S. Rito Lima, “A multiadaptive sampling technique for cost-effective network measurements,” *Computer Networks*, vol. 57, p. 3357–3369, 12 2013.
- [64] Y. Yu, C. Qian, and X. Li, “Distributed and collaborative traffic monitoring in software defined networks,” in *Proceedings of the Third Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: ACM, Aug 2014.
- [65] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, “Deep learning approach for network intrusion detection in software defined networking,” in *2016 International Conference on Wireless Networks and Mobile Communications*, Oct 2016.

- [66] X. Yang, B. Han, Z. Sun, and J. Huang, "Sdn-based ddos attack detection with cross-plane collaboration and lightweight flow monitoring," in *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*, Dec 2017.
- [67] R. Biswas, J. Wu, and Y. Chen, "Optimal monitor placement policy against distributed denial-of-service attack in datacenter," in *Resilience Week*, Nov 2019.
- [68] M. A. S. Saber, M. Ghorbani, A. Bayati, K. Nguyen, and M. Cheriet, "Online data center traffic classification based on inter-flow correlations," *IEEE Access*, vol. 8, pp. 60 401–60 416, 2020.
- [69] M. Newman, *Networks: an introduction*. Oxford university press, 2010.
- [70] C. Tsallis, "Possible generalization of boltzmann-gibbs statistics," *Journal of Statistical Physics*, vol. 52, no. 1, Jul 1988.
- [71] "OpenDaylight," <https://www.opendaylight.org>.
- [72] "hping3," <https://linux.die.net/man/8/hping3>.
- [73] "packETH," <https://github.com/jemcek/packETH>.
- [74] "tshark," <https://www.wireshark.org/docs/man-pages/tshark.html>.
- [75] J. Wu, "Adaptive fault-tolerant routing in cube-based multicomputers using safety vectors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 4, 1998.
- [76] C. Chuang, Y. Yu, A. Pang, and G. Chen, "Minimization of tcam usage for sdn scalability in wireless data centers," in *Global Communications Conference*, 2016, pp. 1–7.
- [77] C. Chu, K. Xi, M. Luo, and H. J. Chao, "Congestion-aware single link failure recovery in hybrid sdn networks," in *Conference on Computer Communications*, 2015, pp. 1086–1094.
- [78] I. S. Petrov, "Algorithm for reducing the number of forwarding rules created by sdn applications," *Modelirovanie i Analiz Informatsionnykh Sistem*, vol. 26, no. 1, 2019.
- [79] L. Dridi and M. F. Zhani, "Sdn-guard: Dos attacks mitigation in sdn networks," in *5th International Conference on Cloud Networking*, 2016.
- [80] Z. Li and Y. Hu, "Pasr: An efficient flow forwarding scheme based on segment routing in software-defined networking," *IEEE Access*, vol. 8, 2020.
- [81] M. Sun, K. Shao, and L. Wang, "Minimizing flow rules for rerouting multi-flows in multi-failure recovery over sdn," in *8th International Conference on Software and Computer Applications*. Association for Computing Machinery, 2019, p. 555–559.
- [82] S. Kotachi, T. Sato, R. Shinkuma, and E. Oki, "Multicast routing model to minimize number of flow entries in software-defined network," in *20th Asia-Pacific Network Operations and Management Symposium*, 2019.

- [83] Z. Zhao, W. Yang, and B. Wu, "Flow aggregation through dynamic routing overlaps in software defined networks," *Computer Networks*, vol. 176, 2020.
- [84] T. Wang, F. Liu, and H. Xu, "An efficient online algorithm for dynamic sdn controller assignment in data center networks," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2788–2801, 2017.
- [85] V. Sridharan, M. Gurusamy, and T. Truong-Huu, "On multiple controller mapping in software defined networks with resilience constraints," *IEEE Communications Letters*, vol. 21, no. 8, pp. 1763–1766, 2017.
- [86] Y. Xu, M. Cello, I.-C. Wang, A. Walid, G. Wilfong, C. H.-P. Wen, M. Marchese, and H. J. Chao, "Dynamic switch migration in distributed software-defined networks to achieve controller load balance," *IEEE Journal on Selected Areas in Communications*, vol. 37, no. 3, pp. 515–529, 2019.
- [87] C. Wang, B. Hu, S. Chen, D. Li, and B. Liu, "A switch migration-based decision-making scheme for balancing load in sdn," *IEEE Access*, vol. 5, pp. 4537–4544, 2017.
- [88] G. Cheng, H. Chen, H. Hu, and J. Lan, "Dynamic switch migration towards a scalable sdn control plane," *Int. J. Commun. Syst.*, vol. 29, no. 9, p. 1482–1499, Jun. 2016.
- [89] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proceedings of the 9th International Conference on Network and Service Management*, 2013, pp. 18–25.
- [90] V. Eramo, E. Miucci, M. Ammar, and F. G. Lavacca, "An approach for service function chain routing and virtual function network instance migration in network function virtualization architectures," *IEEE/ACM Transactions on Networking*, vol. 25, no. 4, pp. 2008–2025, 2017.
- [91] F. He and E. Oki, "Load balancing model against multiple controller failures in software defined networks," in *ICC 2020-2020 IEEE International Conference on Communications*, 2020, pp. 1–6.
- [92] X. Zhang, L. Li, and C.-b. Yan, "Robust controller placement based on load balancing in software defined networks," in *2020 IEEE International Conference on Networking, Sensing and Control*, pp. 1–6.
- [93] L. Li, N. Du, H. Liu, R. Zhang, and C. Yan, "Towards robust controller placement in software-defined networks against links failure," in *2019 IFIP/IEEE Symposium on Integrated Network and Service Management*, 2019, pp. 216–223.
- [94] C. Liang, R. Kawashima, and H. Matsuo, "Scalable and crash-tolerant load balancing based on switch migration for multiple open flow controllers," in *2014 Second International Symposium on Computing and Networking*, 2014, pp. 171–177.

- [95] G. Cheng, H. Chen, Z. Wang, and S. Chen, “Dha: Distributed decisions on the switch migration toward a scalable sdn control plane,” in *2015 IFIP Networking Conference*, 2015, pp. 1–9.
- [96] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, “Onix: A distributed control platform for large-scale production networks,” in *9th USENIX Symposium on Operating Systems Design and Implementation*. Vancouver, BC: USENIX Association, Oct. 2010.
- [97] A. Tootoonchian and Y. Ganjali, “Hyperflow: A distributed control plane for openflow.” USA: USENIX Association, 2010, p. 3.
- [98] S. Hassas Yeganeh and Y. Ganjali, “Kandoo: A framework for efficient and scalable offloading of control applications,” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: Association for Computing Machinery, 2012, p. 19–24.
- [99] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, “Heuristic approaches to the controller placement problem in large scale sdn networks,” *IEEE Transactions on Network and Service Management*, vol. 12, no. 1, pp. 4–17, 2015.
- [100] J. C. Mogul and P. Congdon, “Hey, you darned counters! get off my ASIC!” in *Proceedings of the First Workshop on Hot Topics in Software Defined Networks*. New York, NY, USA: Association for Computing Machinery, 2012, p. 25–30.
- [101] A. Tootoonchian, S. Gorbunov, Y. Ganjali, M. Casado, and R. Sherwood, “On controller performance in software-defined networks,” in *2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*, Apr. 2012.
- [102] D. Erickson, “The beacon openflow controller,” in *Proceedings of the Second ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking*. New York, NY, USA: Association for Computing Machinery, 2013, p. 13–18.
- [103] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, “Devoflow: Scaling flow management for high-performance networks,” in *Proceedings of the ACM SIGCOMM 2011 Conference*. New York, NY, USA: Association for Computing Machinery, 2011, p. 254–265.
- [104] R. Biswas and J. Wu, “Traffic engineering to minimize the number of rules in sdn datacenters,” *IEEE Transactions on Network Science and Engineering*, pp. 1–1, 2021.