Enhancing network robustness using software-defined networking

by

Xin Li

B.S., Beihang University (Beijing University of Aeronautics and Astronautics), China, 2012

AN ABSTRACT OF A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Abstract

As today's networks are no longer individual networks, networks are less robust towards failures and attacks. For example, computer networks and power networks are interdependent. Computer networks provide smart control for power networks, while power networks provide power supply. Localized network failures and attacks are amplified and exacerbated back and forth between two networks due to their interdependencies. This dissertation focuses on finding solutions to enhance network robustness. Software-defined networking provides a programmable architecture, which can dynamically adapt to any changes and can reduce the complexities of network traffic management. This architecture brings opportunities to enhance network robustness, for example, adapting to network changes, routing traffic bypassing malfunction devices, dropping malicious flows, etc. However, as SDN is rapidly proceeding from vision to reality, the SDN architecture itself might be exposed to some robustness threats. Especially, the SDN control plane is tremendously attractive to attackers, since it is the "brain" of entire networks. Thus, researching on network robustness helps protect network from a destructive disaster.

In this dissertation, we first build a novel, realistic interdependent network framework to model cyber-physical networks. We allocate dependency links under a limited budget and evaluate network robustness. We further revise a network flow algorithm and find solutions to obtain a basic robust network structure. Extensive simulations on random networks and real networks show that our deployment method produces topologies that are more robust than the ones obtained by other deployment techniques.

Second, we tackle middlebox chain problems using SDN. In computer networks, applications require traffic to sequence through multiple types of middleboxes to accomplish network functionality. Middlebox policies, numerous applications' requirements, and resource allocations complicate network management. Furthermore, middlebox failures can affect network robustness. We formulate a mixed-integer linear programming problem to achieve a network load-balancing objective in the context of middlebox policy chain routing. Our global routing approach manages network resources efficiently by simplifying candidate-path selections, balancing the entire network and using the simulated annealing algorithm. Moreover, in case of middlebox failures, we design a fast rerouting mechanism by exploiting the remaining link and middlebox resources locally. We implement proposed routing approaches on a Mininet testbed and evaluate experiments' scalability, assessing the effectiveness of the approaches.

Third, we build an adversary model to describe in detail how to launch distributed denial of service (DDoS) attacks to overwhelm the SDN controller. Then we discuss possible defense mechanisms to protect the controller from DDoS attacks. We implement a successful DDoS attack and our defense mechanism on the Mininet testbed to demonstrate its feasibility in the real world.

In summary, we vertically dive into enhancing network robustness by constructing a topological framework, making routing decisions, and protecting the SDN controller.

Enhancing network robustness using software-defined networking

by

Xin Li

B.S., Beihang University (Beijing University of Aeronautics and Astronautics), China, 2012

A DISSERTATION

submitted in partial fulfillment of the
requirements for the degree

DOCTOR OF PHILOSOPHY

Department of Electrical and Computer Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2017

Approved by:

Co-Major Professor
Caterina Scoglio

Approved by:

Co-Major Professor
Don Gruenbacher

Copyright

Abstract

As today's networks are no longer individual networks, networks are less robust towards failures and attacks. For example, computer networks and power networks are interdependent. Computer networks provide smart control for power networks, while power networks provide power supply. Localized network failures and attacks are amplified and exacerbated back and forth between two networks due to their interdependencies. This dissertation focuses on finding solutions to enhance network robustness. Software-defined networking provides a programmable architecture, which can dynamically adapt to any changes and can reduce the complexities of network traffic management. This architecture brings opportunities to enhance network robustness, for example, adapting to network changes, routing traffic bypassing malfunction devices, dropping malicious flows, etc. However, as SDN is rapidly proceeding from vision to reality, the SDN architecture itself might be exposed to some robustness threats. Especially, the SDN control plane is tremendously attractive to attackers, since it is the "brain" of entire networks. Thus, researching on network robustness helps protect network from a destructive disaster.

In this dissertation, we first build a novel, realistic interdependent network framework to model cyber-physical networks. We allocate dependency links under a limited budget and evaluate network robustness. We further revise a network flow algorithm and find solutions to obtain a basic robust network structure. Extensive simulations on random networks and real networks show that our deployment method produces topologies that are more robust than the ones obtained by other deployment techniques.

Second, we tackle middlebox chain problems using SDN. In computer networks, applications require traffic to sequence through multiple types of middleboxes to accomplish network functionality. Middlebox policies, numerous applications' requirements, and resource allocations complicate network management. Furthermore, middlebox failures can affect network robustness. We formulate a mixed-integer linear programming problem to achieve a network load-balancing objective in the context of middlebox policy chain routing. Our global routing approach manages network resources efficiently by simplifying candidate-path selections, balancing the entire network and using the simulated annealing algorithm. Moreover, in case of middlebox failures, we design a fast rerouting mechanism by exploiting the remaining link and middlebox resources locally. We implement proposed routing approaches on a Mininet testbed and evaluate experiments' scalability, assessing the effectiveness of the approaches.

Third, we build an adversary model to describe in detail how to launch distributed denial of service (DDoS) attacks to overwhelm the SDN controller. Then we discuss possible defense mechanisms to protect the controller from DDoS attacks. We implement a successful DDoS attack and our defense mechanism on the Mininet testbed to demonstrate its feasibility in the real world.

In summary, we vertically dive into enhancing network robustness by constructing a topological framework, making routing decisions, and protecting the SDN controller.

Table of Contents

List of Figures

List of Tables

Acknowledgments

First of all, I am most grateful to my advisors, Dr. Caterina Scoglio and Dr. Don Gruenbacher for their guidance, encouragement, support, and patience. This work would not have reached completion without their help. They have provided numerous valuable suggestions, which are extremely helpful not only for my Ph.D. study but also for the rest of my life.

I sincerely thank my committee members, Dr. Dave Thompson and Dr. Todd Easton, for their precious time and helpful advice. I also appreciate Dr. Pavithra Prabhakar for donating her time to be the outside chairperson of my defense.

I would like to express my gratitude to my friends in the network science and engineering group: Heman Shakeri, Faryad Sahneh, Aram Vajdi, and all others. Thank you for the memorable time we spent together. I would also like to thank my friends at K-State: Futing Fan, Tianyu Lin, Qihui Yang, and Wenji Zhang for all the fun we had together.

Finally, I am thankful to my mother, Yanhua Zhang, and my father, Ning Li, for their continuous support over the years. Special thanks to my husband, Haotian Wu. You are my best friend and my best labmate. It is difficult to find the words to express my gratefulness for our being together.

Dedication

I dedicate my dissertation work to my dear husband, Haotian Wu, for his encouragement and patience, and to my daughter, Hannah Wu, for the happiness she brings to me. I also dedicate this work to my mother, Yanhua Zhang, and my father, Ning Li, for their constant support.

Preface

This dissertation, "Enhancing network robustness using software-defined networking," is submitted for the degree of Doctor of Philosophy in the Department of Electrical and Computer Engineering at Kansas State University. The research was conducted under the supervision of Professors Caterina Scoglio and Don Gruenbacher.

To the best of my knowledge, this work is original, except where acknowledgements and references are indicated. Part of the work has been presented in the following published peer-reviewed journals or is submitted for publication:

1. **Xin Li**, Haotian Wu, Caterina Scoglio, and Don Gruenbacher. "Robust allocation of weighted dependency links in cyberphysical networks." Physica A: Statistical Mechanics and its Applications 433 (2015): 316-327[2].

2. **Xin Li**, Haotian Wu, Don Gruenbacher, Caterina Scoglio, and Tricha Anjali. "Efficient routing for middlebox policy enforcement in software-defined networking." Computer Networks 110 (2016): 243-252[3].

3. **Xin Li**, Haotian Wu, Caterina Scoglio, and Don Gruenbacher. "A DDoS adversary model and defenses on the SDN control plane." Submitted (2017).

This research is supported by the Kansas State Electrical Power Affiliates Program (EPAP) and the Department of Electrical and Computer Engineering at Kansas State University.

Chapter 1

Introduction and background

Software-defined networking (SDN) is becoming increasingly important, bringing benefits that will potentially reform today's computer network. We propose to enhance network robustness using SDN. In this chapter, we first present our motivations by introducing the role of SDN and the importance of network robustness in section 1.1. Second, we introduce the concepts of network robustness, SDN, OpenFlow, and middlebox in section 1.2. Third, we highlight our contributions in section 1.3. Finally, the organization of this dissertation is introduced in section 1.4.

1.1 Introduction and motivation

Traditional computer networks rely on sophisticated protocols on legacy routers/switches to provide numerous services. This leads to Internet ossification, which means the current Internet has little space to scale. However, the population of network users is increasing dramatically, and users' requests are becoming more diverse. Data representing global internet traffic, in petabytes per month, is shown in Fig. 1.1[1].

Moreover, monthly active users of Facebook have been increasing dramatically over the last 13 years and reached 2 billion by the end of June 2017[4]. Monthly global mobile data traffic is projected to be 49 exabytes by 2021, with annual traffic exceeding half a zettabyte[5].

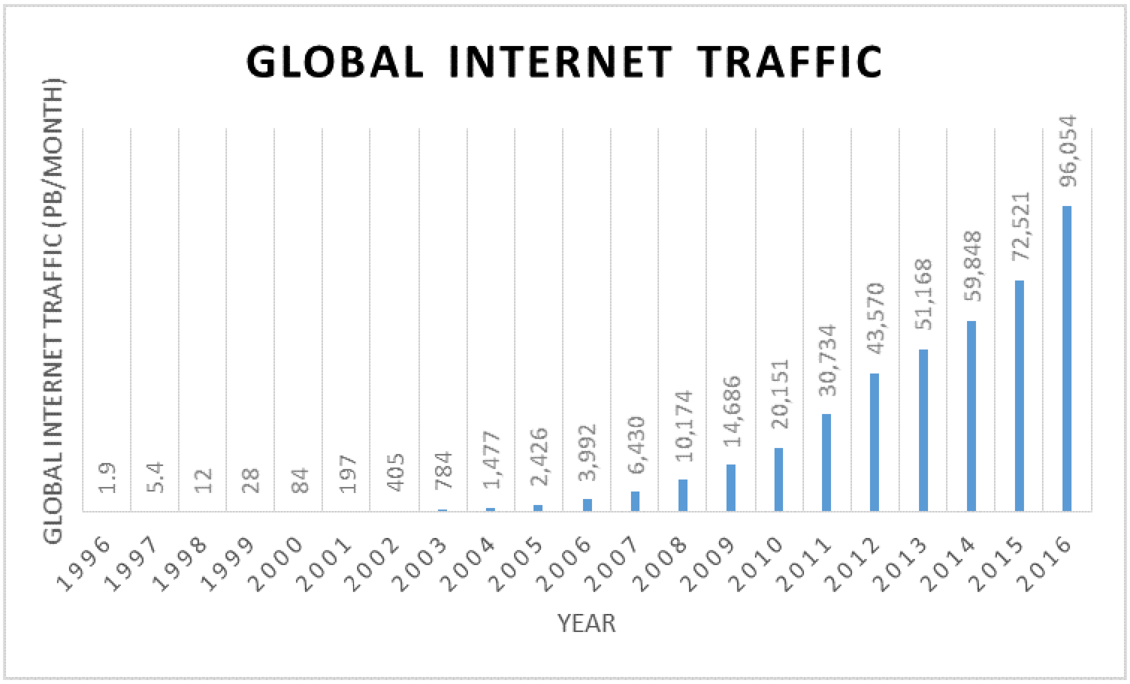


Figure 1.1: Global internet traffic (petabytes per month) over the past 11 years[1].

The United States will need to invest up to $150 billion in fiber infrastructure over the next five to seven years to support networking demand[6]. Furthermore, recent data, as reported by Sandvine, indicate more than 70 percent of North American traffic is now streaming video and audio[7]. Streaming traffic is delay-sensitive and consumes large bandwidth.

Security issues are another concern on today's network. Recently, credit-reporting firm Equifax revealed that hackers might have stolen financial and consumer data on at least 143 million customers[8]. Those data include birth dates, social security numbers, driver's licenses and addresses, which could lead to severe identity theft issues, and immeasurably harm the U.S. finance credit system in the long run. In 2016, major Internet platforms and services couldn't be accessed in Europe and North America, because Dyn servers were under DDoS attacks[9]. Such security threats have become even more severe in recent years. Microsoft cloud user accounts saw a 300 percent increase in cyberattacks in 2016[10]. Cisco's 2017 mid-year cybersecurity report indicates that 34 percent of service providers lost revenue from attacks. Moreover, the new destruction of service (DeOS) attacks are not aiming at attacking, but at destroying networks by preventing defenders from restoring systems and data[11].

Traditional computer networks may face problems in adapting to the dynamics and requirements of numerous applications. Fortunately, the concept of SDN makes this possible by assigning networks more flexibility and programmability. SDN refers to a network architecture enabling programmability and separating the control plane from the data plane. It reforms sophisticated legacy routers/switches as simple forwarding elements, and further supports network scalabilities and innovations. It highlights the importance of software and allows us to manage network operations via open interfaces to further reduce expenditures. For example, AT&T CTO expects SDN to reduce operational expenses by 40 percent by 2020[12], and CenturyLink's CEO, Glen Post, said his company remains on track to see at least $200 million in annual capital expenditures reduction[13].

SDN is rapidly moving from vision to reality. Tech news indicated an incomplete list of 42 vendors offering SDN products[14]. Moreover, a number of SDN startups have already been bought by larger companies, validating their potential. In 2012, VMware bought Nicira for $1.25 billion. Then Juniper bought Contrail Systems for $176 million[15].

Among its advantages, SDN brings opportunities to enhance a computer network's robustness. First, SDN provides centralized management with visibility of entire networks. One popular realization of SDN is OpenFlow[16]. The SDN controller supports acquiring flow statistics from OpenFlow switches in real time. This helps identify potential threats and speed up troubleshooting to guarantee network availability. Second, routing and rerouting flows are managed by the controller on the fly. For example, SDN incorporates middleboxes and dynamically routes flows along middlebox chains to accomplish network functionality. These middleboxes help improve network robustness, e.g., by use of a load balancer to circumvent overloads and an IDS to detect anomalies. Moreover, SDN can flexibly reroute traffic bypassing malfunction devices or drop malicious flows. Third, flow tables on OpenFlow switches are updated by running programs on the controller. Network operators are no longer required to remotely log in and configure each switch, which makes network more robust against any type of manual configuration errors. Moreover, SDN allows for eliminating policy conflicts and attacks by simply programming the OpenFlow controller.

When we enhance network robustness using SDN, we also need to study corresponding issues. Massive research has been exploiting opportunities and innovating promising applications under SDN architecture; however, the design of SDN itself is conversely exposed to security challenges. The issue of SDN robustness becomes a concern, before it can become a substitute for traditional networks. On one side, we exploit the flexibility of SDN to enhance network robustness. On the other side, the decoupled architecture itself might be exposed to some security threats.

In general, today's networks are independent networks. This leads to a situation where localized damage in one system will be extended to another system through their dependency links, triggering cascading failures and finally bringing large-scale damage[2;17;18]. For example, the 2003 Italy blackout affected a total of 56 million people. Consequently, network robustness is of great importance to ensure network availability, resilience, and adaption to changing scenarios.

1.2 Background

In this section, we introduce the concepts of network robustness and SDN, since SDN is the major technology to improve network robustness in this dissertation. Then, we present "OpenFlow," an SDN standard, and "middlebox," one type of network device.

1.2.1 Network robustness

Network robustness refers to a network's ability to withstand failures and attacks — a critical attribute to evaluate networks. It ensures network availability, resilience, and adaption to changing scenarios.

Current networks (e.g. biological, biosocial, electric, electronic, etc.) are no longer individual networks[19–21], because one network often depends on another. For example, the power network provides electricity to the computer network and, through the computer network, computers gather information reported from the power network, consequently controlling the power network. Localized damage in one network will be propagated to the other one through dependency links, triggering cascading failures and finally bringing large-scale damage[17;18]. A robust framework with proper allocation of dependency links can protect networks from these large-scale damages. Moreover, SDN can be the enabling technology in the computer network, thus affecting the robustness of the network framework. With flexibilities of SDN to be discussed in subsection 1.2.2, and functionalities of middleboxes to be discussed in subsection 1.2.3, attacks and failures could be detected and eliminated within computer networks before being propagated, thus enhancing network robustness.

1.2.2 Software-defined networking

SDN is an emerging networking architecture with a design of decoupling the control plane and the data plane, which, consequently, simplifies network management, lowers the cost of network devices' deployment, and potentially reforms today's networks.

With increasing needs of numerous applications and enormous user demand, traditional computer networks are becoming more complex. In traditional computer networks, there are a large number of network devices, e.g., routers and middleboxes 1.2.4, with complicated protocols running on them. Those devices are not only in charge of packet forwarding, but are also responsible for accomplishing network functions and conducting sophisticated management. Such integration of control logics and forwarding elements makes current IP networks very difficult to scale and evolve. We call it Internet ossification and its details are listed as follows:

1. Devices are vendor-specific. Network devices differ based on vendors and provide limited interfaces to be configured. This does not allow network operators to easily update network states. Network operators have to configure each network device separately. They also need to reconfigure them when there are network failures and policy changes. Moreover, such configurations are error-prone and very difficult to scale.

2. Protocols are sophisticated. To accomplish various policies, a set of protocols need to be implemented. As needs of numerous applications increase, these protocols become

more sophisticated and may conflict with each other. Network evolvement is caught in a dilemma as to whether to gain more intelligence and sophistication, or not.

3. Testing platforms are insufficient. New ideas should be tested in realistic settings to validate their practicality. However, research traffic cannot be isolated from production traffic and this might lead to severe consequences in current IP networks[16]. A new routing protocol can take five to 10 years to be fully designed, evaluated, and deployed in current IP networks[22].

SDN architecture consists of three planes and two interfaces, that is, application plane, control plane and data plane; and northbound interface and southbound interface, shown in Fig. 1.2. The biggest innovation is the separation of the control plane and the data plane. All control logics are defined in the control plane, and the data plane is only responsible for forwarding. That simplifies network management and further allows network to scale to adapt to increasing demands. The three planes are logically separated, but they might coexist with each other physically.

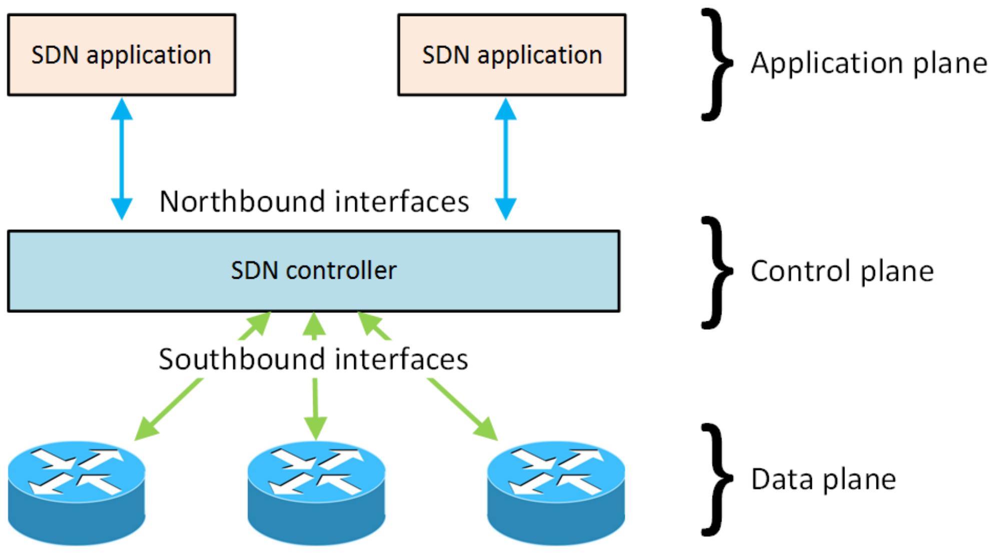


Figure 1.2: SDN architecture consists of three planes and two types of interfaces.

Compared with traditional computer networks, SDN has the following advantages:

1. Centralized management — an SDN controller has an overview of entire networks, simplifies traffic management to adapt to different application requirements, and provides better network supervision[23;24]. All control logics are managed by authenticated applications and dispatched through the control-to-data plane channel using an OpenFlow protocol.

2. Flexibility — SDN's data plane is only responsible for forwarding packets. It assigns more flexibilities on the networks without manually configuring legacy routers/switches. Once network policies or conditions change, switches receive updated commands from the controller and simply follow those instructions.

3. Programmability — SDN architecture introduces the ability of programmability. Network operators can control and update network behaviors by programming each network device dynamically via open interfaces. Such software programs are easier operated and less error-prone[25].

4. Innovation — SDN allows network innovations by isolating research traffic from production traffic, and helps with new protocol tests in realistic settings. It also provides functionalities of access control, load balancing, network monitoring, etc. to gradually substitute traditional computer networks, and further stimulates innovations.

1.2.3 OpenFlow

OpenFlow is the current SDN de-facto standard[26]. It is a communication protocol between the control plane and the data plane. Unless specified, switches in the data plane discussed in this dissertation are OpenFlow-enabled switches.

OpenFlow switch

An OpenFlow switch consists of one or more flow tables, a group table, and one or more OpenFlow channels connected to a remote controller[27].

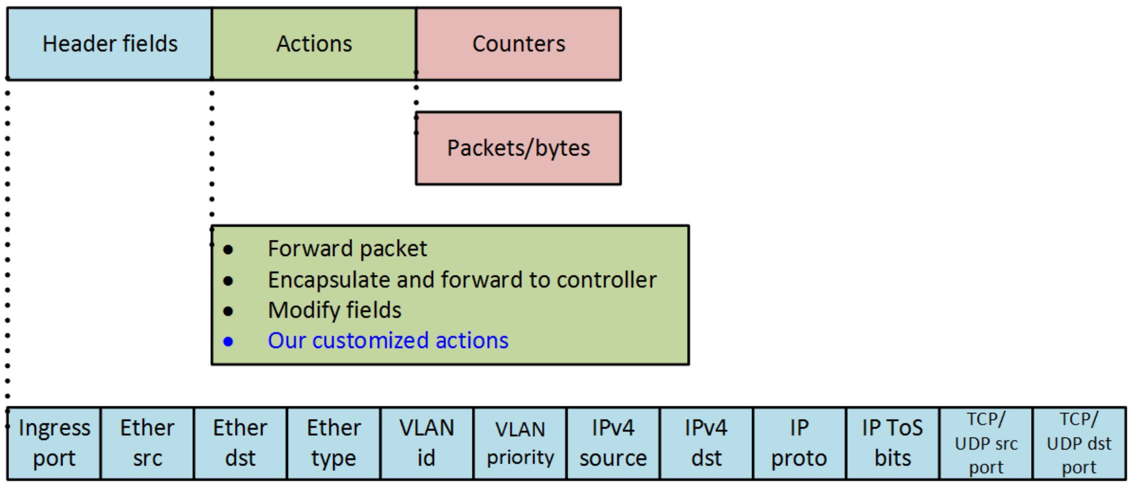


Figure 1.3: A flow entry consists of header fields, actions, and counters.

The flow table contains a list of flow entries 1.3, which perform packet matching and define forwarding actions. Each flow entry in the flow table consists of three fields: header fields, actions, and counters. In the header fields, 12 fields, defined in OpenFlow 1.0, classify a flow and provide various matching possibilities to meet different applications' requirements. Actions can be various and flexible — dropping the flow, forwarding the flow, or even changing flow content, etc. We can also customize actions. Counters are collected at the line rate, and further used for network monitoring and management.

The OpenFlow channel refers to the interface connecting the OpenFlow switch to the controller. OpenFlow defines a set of messages, e.g., symmetric messages "hello" and asynchronous messages "packet-in," to communicate or update status with the controller via this channel. The controller can also configure and manage OpenFlow switches via the channel. In addition, such communication is optionally secured by a transport layer security (TLS) protocol.

Flow forwarding

When a packet arrives at an OpenFlow switch, packet header fields are extracted to try to match the header fields of flow entries. When it matches the header field in a flow entry, it will conduct the corresponding actions. If a flow cannot find a matching flow entry, it will be encapsulated with an OpenFlow protocol header and forwarded to the controller to request a new flow-entry installation. After the new flow entry is installed, matched packets will be forwarded according to its actions.

The new flow entry can be installed either proactively by the controller or as requested, indicated above. The controller can also update and delete flow entries in flow tables.

1.2.4 Middlebox

Middlebox is a graphic description of an existing internet phenomenon providing network functions other than IP routing. Since it is implemented as an intermediary box between a source host and a destination host, it is called "middlebox"[28]. In traditional computer networks, all functions above the IP layer, except IP routing functions, can be considered as middlebox functions. For example, firewalls, proxies, DNS servers, and load balancers are middleboxes, while IP routers are not. In SDN networks, switches are not considered as middleboxes, while other functions are, as indicated above[29]. Today's network relies on these middleboxes to guarantee critical network functions, e.g., security inspection and performance improvement. For example, load balancers are used to circumvent overloads, and further improve network performance and robustness. RFC 3234 details taxonomy and issues about middleboxes[28].

Middlebox deployment

Middleboxes can be deployed in path 1.4 or off path 1.5. Off-path deployment is more robust and can ensure end-to-end connectivity when failures occur. It is also more efficient and flexible, since we only route flows to go through middleboxes as needed.

Middlebox policy chain

Network applications require traffic to sequence through multiple types of middleboxes to accomplish desired network functions. We call this "middlebox policy chain." For example, flows have to first go through a firewall and then an IDS for security purposes[29], shown in Fig. 1.6.

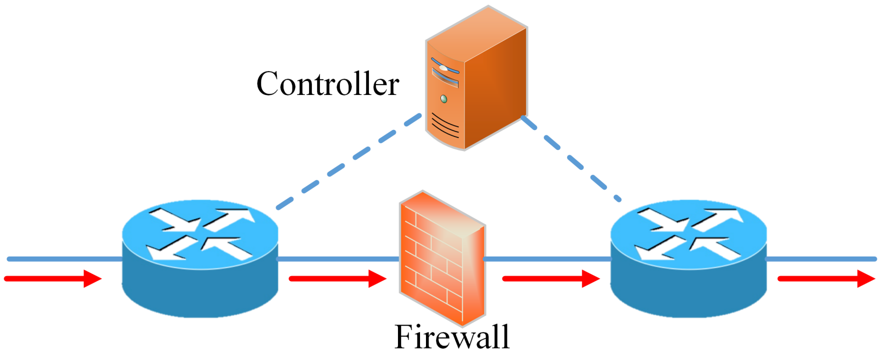


Figure 1.4: Middlebox in-path deployment — all flows on the red path are forced to be sent through the middlebox, such as a firewall (FW).

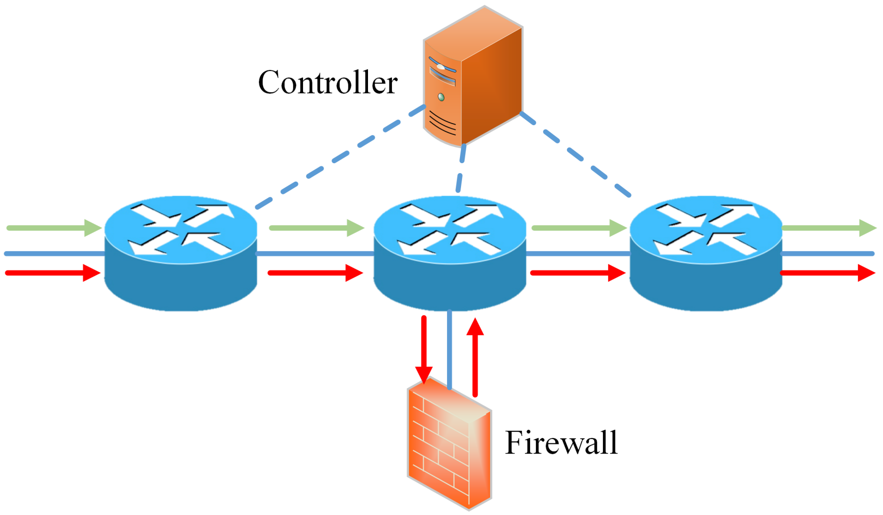


Figure 1.5: Middlebox off-path deployment — secure flows are sent along the green path, while others are sent through the firewall for inspection.

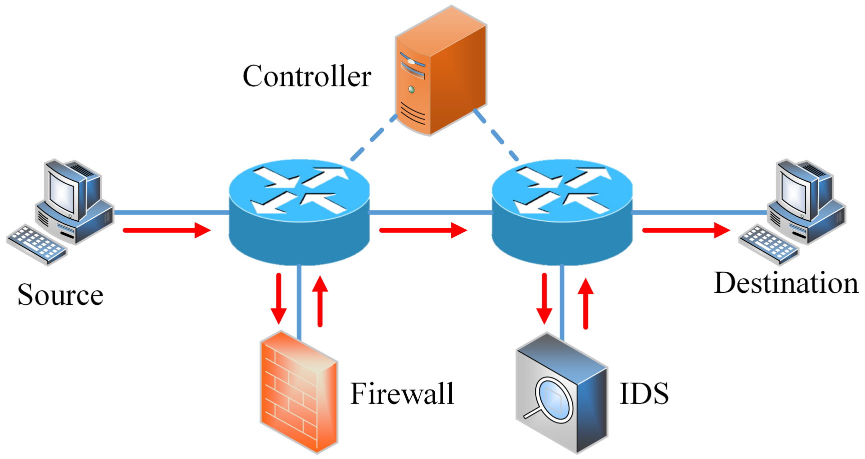


Figure 1.6: Example of a middlebox chain — flows sequenced through a firewall and then an IDS.

In a larger network with numerous middleboxes, flow routings along middlebox chains become complex[23], which might create false configurations. It is also difficult to dynamically update routing paths. It is even more challenging to enable middlebox chains within limited network resources.

1.3 Contributions

We vertically dive into enhancing network robustness by constructing a topological framework, making routing decisions, and protecting the SDN controller. Our major contributions can be summarized as follows:

1. We have designed and thoroughly tested an optimization-based scheme to allocate dependency links within a budget constraint, where the numbers of nodes in the two networks are not identical. To this end, we have built a realistic cyber-physical network framework with one-to-multiple dependencies, two unequal-size individual networks, and weighted dependency links. Cyber-physical networks are in essence interdependent networks, where one network supports and affects the other through dependency links. Localized network failures and attacks are amplified and exacerbated back and forth between the two networks due to their interdependencies. We allocate dependency links to obtain more network robustness under this realistic framework.

2. We have developed two routing schemes for middlebox policy enforcement to improve network robustness. This is the first work to handle failures in middlebox chain scenarios using OpenFlow. With our designs, network resources are managed efficiently to circumvent overloads, and the network can rapidly respond to middlebox failures. Consequently, network robustness is improved. Moreover, results of the optimization on a test topology include an increase up to 26.4 percent of the throughput, with respect to sequenced shortest-path routing.

3. We have explored vulnerabilities of the SDN controller from the attacker's point of view and have detailed the adversary model. We have presented preliminary results for several

strategies to protect the controller from saturation attacks. This work contributes fundamentally to detect and mitigate the SDN control plane's vulnerabilities, and to further enhance network robustness.

1.4 Organization

This dissertation is organized into chapters. In chapter 2, we present a thorough literature review on the issues of network robustness. In chapter 3, we propose a novel robust network framework and allocate dependency links to obtain more network robustness. In chapter 4, we further enhance network robustness by managing middlebox policies and handling middlebox failures using SDN. In chapter 5, we build a detailed adversary model to identify vulnerabilities of the SDN control plane, and present preliminary results on possible defense mechanisms. Finally, we conclude this dissertation and discuss future work in chapter 6.

Chapter 2

Literature review

Software-defined networking (SDN) is an emerging networking architecture that reduces complexities of network traffic management and presents a design of programmable networks that can adapt to changing application requirements[25]. SDN provides functionalities of access control, load balancing, network monitoring, etc. to supplement traditional computer networks, and further stimulates more innovations. In particular, SDN brings opportunities to network robustness, for example, fine-grained control over network-based security functions; flexibly rerouting traffic bypassing malfunction devices or dropping malicious flows; and other innovative applications such as protecting web servers from TCP SYN flooding, etc.[3;30;31] Massive research has been exploiting opportunities and innovating promising applications under SDN architecture; however, the design of SDN itself is conversely exposed to security challenges. Moreover, a computer network is always coupled with other networks, and a localized failure might destroy the entire network. It is of great importance to study network robustness. We define network robustness as the network ability to withstand possible network failures or attacks; and it ensures network availability, resilience, and adaption to changing scenarios.

SDN brings opportunities to enhance the computer network's robustness. First, it provides better network supervision. The controller has an overview of the entire computer network and can acquire flow statistics from OpenFlow switches on the fly. Thus, we can rapidly identify network failures or attacks. One common application is to identify attacking flow patterns from collected data[32;33]. Second, SDN allows dynamicality of complex, network-policy management. Data plane is only responsible for packet forwarding, to avoid complicated configurations. For example, SDN dynamically routes flows along a middlebox chain to accomplish network functionalities[3;34]. These middleboxes help improve network robustness, e.g., a load balancer is used to circumvent overloads and an IDS is used to detect anomalies. Third, SDN allows elimination of policy conflicts[35] and mitigates attacks by simply programming the OpenFlow controller.

We particularly outline how SDN helps solve the middlebox chain problem. Network applications require traffic to sequence through multiple types of middleboxes to accomplish desired network functions. For example, web traffic needs to go through a proxy and then a firewall[36]. To fulfill network functions, various types of middleboxes are utilized, and each type might have a hundred devices in a large network[29;37]. In traditional computer networks, traffic steering to meet the above goals is a critical problem[23], which might create false configurations. It is also difficult to dynamically update the routing policy. It is even more challenging to enable the stateful policy routing (middlebox policy chain) within limited network resources (network link bandwidth, middlebox-processing capability, and switch high-speed searching memory).

The problem of routing under middlebox sequence constraints has recently gained remarkable attention due to the role played by many network devices called middleboxes (e.g., firewalls, VPN gateways, proxies, intrusion detection systems (IDS), WAN optimizers) on network performance[38–42]. To enforce middlebox policies, a novel middlebox architecture was presented by Sekar et al. in[38]. In this paper, the authors designed a network-wide controller and a local coordinator to manage middlebox resources, resembling the architecture of SDN networks. As a matter of fact, a centralized SDN controller makes a network transparent and synchronous[39], as well as more efficient for network administrators to manage. Furthermore, Joseph et al. proposed a policy-aware switching layer to enforce middlebox policy and increase middlebox utilization[40]. They also presented an off-path middlebox deployment. However, under this off-path deployment, flows are often required to travel on one link multiple times, increasing the probability of link overload. Fayazbakhsh et al. further modified legacy middleboxes to support FlowTags used to differentiate flows with different policy requirements[41]. Alternatively, OpenFlow allows the identification of stateful policy flows using available fields in the packet header. Using OpenFlow, Qazi et al. elaborated on the complexity of selecting middleboxes and scheduling flows, and simplified the middlebox traffic steering problem by offline pruning some of the less-promising routing paths[29]. The proposed offline calculation is time consuming, and it was performed each time failures occurred or policy changed. This aspect is problematic, since networks should quickly respond toward middlebox/link overloads and failures. In summary, open issues are as follows: how to select possible routing paths with middlebox policy enforcement, and how to quickly assign flow routing paths to maintain network performance. In solving middlebox chain problems, resources on the middleboxes are another constraint. We take on solving these open issues in chapter 4.

Another topic of research in the field of middlebox management concerns how to deal with link failures and middlebox failures[43–46]. Research indicated that middleboxes contribute to 43 percent of high-severity incidents[28;46]. Thus, it is critical to study middlebox failures. Existing solutions are either to prevent the effect of middlebox failures beforehand[47], or react after middlebox failures; for example, reconstruct middlebox states after failures[48]. However, today's network relies on sequenced types of middleboxes to provide network functions, and different types of traffic go through different sequences — both of these being beyond the scope of existing approaches. Restarting middleboxes is a common approach to dealing with middlebox failures, but few articles considered the impact during the restarting period. In addition, when application requirements change, middlebox policy will be updated. To avoid misconfiguration during policy updates, SDN's centralized and programmable management helps solve this middlebox policy routing problem. Therefore, we are the first to consider middlebox failures in the middlebox chain problem, and then mitigate these failures' consequences using SDN. This will be introduced in chapter 4.

As SDN invokes huge interest from both academia and industry, it is rapidly proceeding from vision to reality. However, some researchers are arguing that SDN conversely brings several security threats due to nature of separation between the control plane and the data plane[49–53]. It is not clear whether SDN brings more benefits, or conversely, more threats, to today's networks.

Several components in SDN architecture are exposed to robustness issues. Attacks might occur on the application plane, the control plane, the data plane, the northbound interface, or the southbound interface as shown in Fig. 1.2[54–61]. First, attacks toward the controller can lead to a disaster for the entire network. Thus, the controller is a particularly attractive attack target. The controller might be exposed to unauthorized access and exploitation through open interfaces (northbound and southbound interfaces). Second, when multiple applications are deployed on the application plane, conflicting flow-rule problems may arise[35]. Third, the control plane and the data plane are both exposed to scalability issues. For example, the controller might be overwhelmed by an excess of flow-entry installation requests and switches might fail to buffer all new flow packets. This can be exploited by attackers to launch denial of service attacks.

OpenFlow is an enabler of SDN. In an OpenFlow network, when a packet arrives, a switch forwards the packet based on the matching flow entry. When no flow entry matches, the switch will generate a packet-in message to the controller for a flow-entry setup. Assume a large volume of table-miss packets are coming. The same number of requests will be sent from the switch to the controller[62]. Such requests will overwhelm the switch-to-control channel, exhaust the controller computation resources, and further lead to controller dysfunctionality.

As network size scales or user requirements vary, the controller's scalability issue will become more severe. Many efforts have been made to solve this scalability issue. Researchers proposed use of distributed controllers to decentralize the calculation burden of the single controller[63] and further protect the entire network from the single-point controller failure. Moreover, some research on wildcard rules is aiming at reducing the number of requests from the data plane[64;65]. All those solutions help improve the control plane scalability. They are also valuable for denial of service (DoS) attack elimination strategies, due to the fact that control plane scalability issues and control plane DoS attacks are both resource consumption issues. However, when DoS attacks are launched, those strategies are no longer sufficient.

To mitigate DoS attacks, we can detect a user with abnormal behaviors by simply setting a rate-limiting threshold. When a user sends more flows than the threshold, we can block all flows from that user[66]. However, those requests could come from legitimate users and simply blocking all suspicious flows unavoidably affects flows from legitimate users. Moreover, in the SDN architecture, attackers can generate excessive short new flows to overwhelm the controller, but absolutely skip rate limiting. In this case, total rate of those attacking flows is very small, and it's hard to block the attacks by setting a rate-limiting threshold.

Crafting a huge amount of short table-miss packets forces the switch to inquire the controller, leading to network dysfunctionalities. Such new table-miss packets' flooding is called "request flooding" for short in this dissertation. Many research groups are working on this topic[52;53;62;67–74]. Request flooding will lead to switch software components overload, switch-to-controller channel congestion, switch flow-table overflow, and controller's resource saturation[71]. Among them, controller-resource saturation is destructive to the entire network. Avant-Guard[72] and Lineswitch[73;74] proposed a proxy-like switch extension to shield the controller from attacks. Such a method is effective when it comes to TCP-SYN flooding. We are targeting a more general request flooding, also indicated in[53;62;67–71]. OF-GUARD[53] and Floodguard[62] built an additional data plane cache to temporarily hold excessive new packets to protect the controller. Zhang et al.[71] and Wei et al.[67] proposed to mitigate this request-flooding attack from the controller side, instead of adding any extra complexity on the data plane. Zhang et al. introduced weighted, fair-sharing queues to reduce the packets from attackers being served[71]. Wei et al. dynamically maintained a trust list and updated each user's trust value to block the attackers[67].

Thus, several challenges have arisen, as follows: Will the controller have enough CPUs and memory to hold queueing lists? Does the controller work to completely block attackers? How do we differentiate legitimate users from attackers? In this dissertation, we explore controller vulnerabilities from the attacker's point of view, and detail an adversary model and defenses. This will be introduced in chapter 5.

Regarding the robustness of cyber-physical systems, Buldyrev et al. provided a one-to-one node dependency interdependent network model to describe the cascading failures caused by the dependencies between two individual networks[18]. Cascading failures under random attack[18;75] and targeted attack[76;77] were further studied in order to develop suitable protection strategies. Not only node correlations[78] and clustering coefficients within one individual network[79], but also node coupling approaches of the nodes from two networks[20;77;80] influence network robustness. Schneider et al. indicated that choosing a fraction of nodes as autonomous beneficially increased robustness[81]. Mirzasoleiman et al. introduced weighted individual networks and studied link-load effects on robustness in[82]. Results in[83–85] comprehensively indicated and proved a coupling threshold existed for interconnected network structural transitions from two independent functioning networks to a whole system. Further research in edge attack rather than node attack was produced[86]. Gao et al. generalized interdependent networks with two individual networks to n individual networks[87]. However, all research is based on the one-to-one node dependency interdependent network model in[18]. The one-to-multiple dependency model brought up in[88] is more realistic than the one-to-one node dependency model in previous papers. Reis et al. extended this to study indegree-indegree and indegree-outdegree relationships, that is, one node can have multiple supports from another network[89]. Yağan et al. proved that a proper allocation of dependency links in their one-to-multiple dependency model would contribute to a more robust system[90]. Their dependency model is valuable, but in reality, it is always the case that one network has multiple dependencies from the other network. The numbers of nodes in two individual networks are not equal as well. Weights of dependency links are no longer identical. In chapter 3, we build a realistic robust interdependent network model and allocate dependency links to make the network more robust from cascading failures. With this allocation, we do not expect localized damage in the computer network to propagate to an uncontrollable disaster for the entire cyber-physical networks.

In summary, in this dissertation, we are analyzing and solving some challenging issues to make networks more robust toward failures and attacks.

Chapter 3

Robust network framework

In this chapter, we explore a robust cyber-physical network framework. Interdependent network models are often used to show how one network has an effect on another network through their dependencies. We propose a novel interdependent network model, which consists of two individual networks with unequal numbers of nodes and one-to-multiple weighted dependency links between the two networks. Based on realistic assumptions, this model differs from previous works that considered equal numbers of nodes in the two networks and identical under limited budgets. We formulate an optimization problem to allocate dependency links using least resources. This novel model enhances the practicability of traditional cyber-physical system structures, but it makes the dependency-link deployment problem more complex and the optimization problem cannot be solved in large networks. To overcome this problem, we propose a new algorithm based on a revised network flow method. Extensive simulations on random networks and real networks show that our deployment method produces topologies that are more robust than the ones obtained by other deployment techniques. Results indicate that our algorithm is efficient and cost-effective in designing robust interdependent networks, and our deployment method is suitable for networks of any size.

3.1 Interdependent network model

We build a novel network model for cyber-physical systems considering dependencies between individual networks. First, we introduce interdependent network models. Second, we present how one network affects the other in the cyber-physical systems. Third, we further discuss individual network models and their dependencies.

3.1.1 Classic interdependent network model

Buldyrev et al. presented an interdependent model for a blackout in Italy in 2003[18]. They considered two networks, A and B, with the same number of nodes. Network A is a power network, and the nodes of network A can be considered as power stations; network B is an Internet network, and the nodes of network B are internet servers. Functioning of a power station relies on control information provided by an internet server, and an internet server needs power supply from a power station. The two networks are one-to-one coupled with each other and rely on the other to provide critical resources. Connections between two individual networks are presented by their dependency links.

3.1.2 Realistic interdependent network model

In real-world interdependent networks, a node in a network is often supported by multiple nodes in another network. A one-to-one node dependency model is no longer suitable for further interdependent network studies. And, it is always the case that numbers of nodes in two individual networks are not equal in reality. Our novel interdependent network model can be considered as one of our contributions. Moreover, costs of allocating dependency links rely on geographical features, corresponding node loads, etc. Thus, dependency links should be weighted rather than being assumed equal. Consequently, a one-to-one node-dependency-link model is not realistic, but one-to-multiple dependency links with various weights must be studied. Furthermore, dependency-link allocation designs can greatly affect the robustness (R) of interdependent networks.

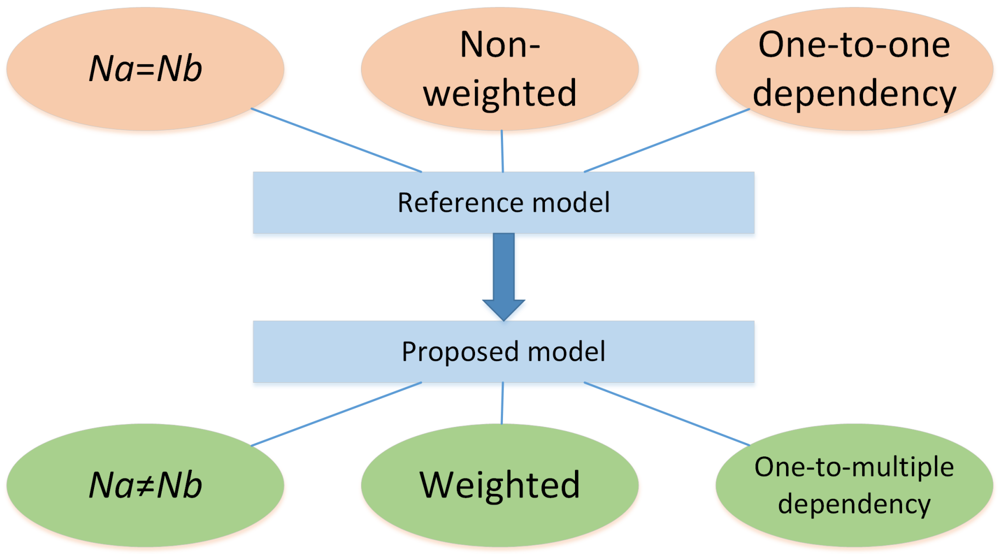


Figure 3.1: Compared with previous models, we build a realistic cyber-physical network framework with one-to-multiple dependencies, two unequal-size individual networks, and weighted dependency links.

Our model consists of three parts: two individual networks (A and B) and their dependencies[91]. Individual network models will be introduced in subsection 3.1.4, and their dependencies are discussed in subsection 3.1.5. Two nodes in the same individual network are connected by connected links, while two nodes from different individual networks are interdependent by dependency links. Numbers of nodes in network A and B are denoted by N_A and N_B, respectively, and dependency links are weighted. If all dependency links are identical, we call them non-weighted links or weighted links with weight equal to 1. The largest component[18;88;92], i.e., largest connected subgraph, represents a functioning component in each individual network. We allow for one-to-multiple dependencies in the interdependent network, rather than one-to-one dependency. Moreover, N_A and N_B are not necessarily equal in our model. The improvement of our model from previous models is shown in Fig. 3.1.

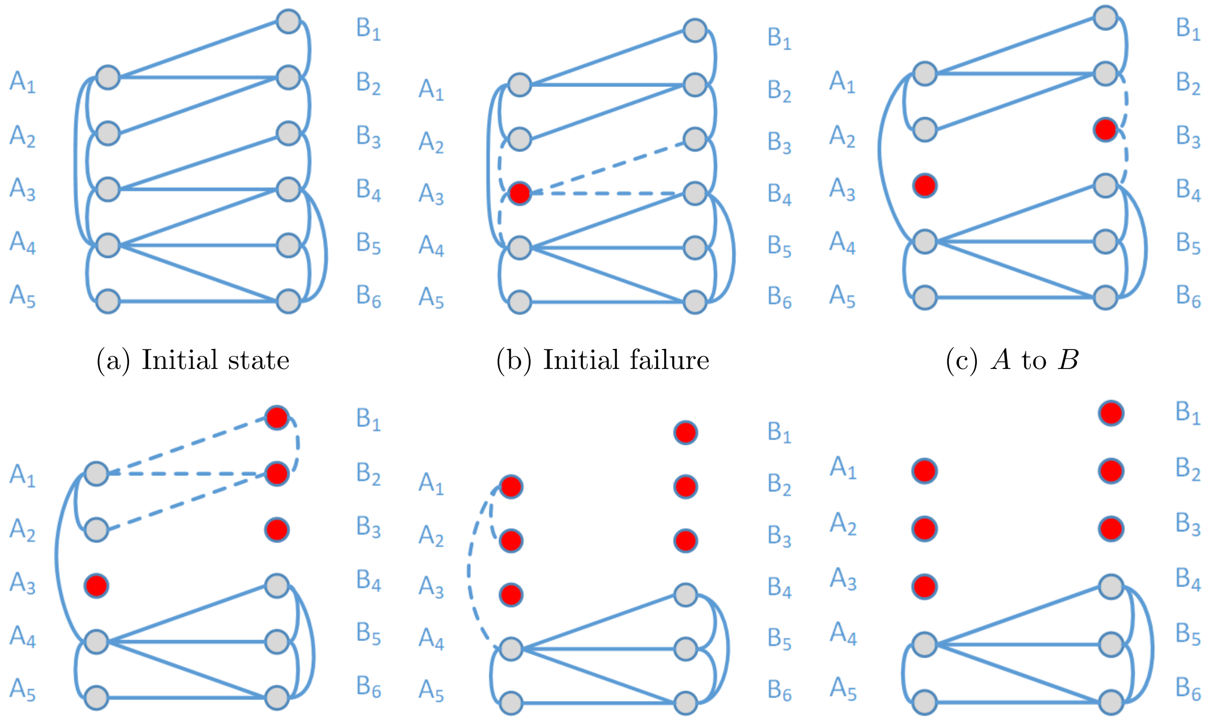
3.1.3 Cascading failures

Cyber-physical systems are more likely to fail because of interdependence between cyber networks (e.g., computer networks) and physical systems (e.g., power networks). If a power station fails, the assumption is made that it will no longer supply power to the linked computers in computer networks. In addition, other power stations lose connections with that station and must redistribute their loads. Therefore, we have an assumption that if some nodes fail, the links (connected links and dependency links) connected to these nodes fail. Then we obtain several disconnected groups of power stations after failures appearing in the power network, and assume the largest component in the power network works as the functioning component, rather than other small connected components[88]. The same thing happens in the computer network, that is, we only consider the largest component. The second assumption is that nodes not in the largest component (of its own network) fail. Moreover, if no power station supplies power to a computer, the computer will not operate, and vice versa, leading to the third assumption: nodes with no dependency links fail. N_A^* and N_B^* represent the numbers of nodes in two stable networks when cascading failures come to an end. Based on initial and stable network states, robustness is defined as:

$$R = 0.5 \times (\frac{N_A^*}{N_A} + \frac{N_B^*}{N_B}) \tag{3.1}$$

Cascading failure example

In Fig. 3.2, A_i indicates the i^{th} node in one network A, while B_j represents the j^{th} node in another network B. The connection between A_i and B_j is represented by m_{ij}. The connection between A_i and A_j is represented by a_{ij}, and the connection between B_i and B_j is represented by b_{ij}, respectively. In Fig. 3.2a, we see an initial state of the interdependent networks with one-to-multiple dependencies and unequal numbers of nodes in the two networks. We consider the initial state in (a) as $N_A = 5$ and $N_B = 6$. In Fig. 3.2b, the initial failure rate happening to network A is 0.2. It illustrates when A_3 is attacked, the connected links a_{32}, a_{34} and dependency links m_{33}, m_{34} fail. Nodes A_1, A_2, A_4, and A_5 are in the largest component. Network A influences Network B through m_{33} and m_{34}. In Fig. 3.2c, B_3 has no dependency links, so B_3 fails and the links connected to B_3 fail. Node B_1 and B_2 fail as they become disconnected from the functioning component of network B. Network B influences Network A through m_{11}, m_{12}, and m_{22}, as shown in Fig. 3.2d. Fig. 3.2e indicates that A_1 and A_2 have



(a) Initial state (b) Initial failure (c) A to B

(d) Largest component in B (e) B to A (f) Stable state

Figure 3.2: A cascading failure process is illustrated.

no dependency links. In Fig. 3.2f, we have a stable state after cascading failures. A_4A_5 and $B_4B_5B_6$ are existing nodes, i.e., $N_A^* = 2$ and $N_B^* = 3$ at this time. According to Eq. (3.1), $R = 0.5 \times (2/5 + 3/6) = 0.45$. It is noted that robustness (R) in the following simulations is calculated by Eq. (3.1).

3.1.4 Individual network model

Erdős-Rényi (ER)[93] and Barabśi-Albert (BA)[94] are both random graph models. In the ER model, each vertex has the same probability with a fixed number of edges. The BA model is used to generate random scale-free networks using a preferential attachment mechanism. It represents some real networks with the feature of containing few nodes (a.k.a., hubs) with unusually high degree. We use these two typical random network models as the individual network models to generalize our results. We also apply our strategies with real network topologies in section 3.7.

3.1.5 Discussion of dependency links

This work designs how to allocate dependency links with prior information of link weights. The weights show the differences among dependency links. A larger weight represents a higher cost to allocate this dependency link. We use a limited budget (total weights: tw) to build all dependency links. To the author's best knowledge, this is the first work to apply the concept of limited budget to arrange dependency links from an engineering perspective. Link allocation in this work shows how nodes in two individual networks are dependent and is performed as an extensive study of node coupling[75–77]. In the following sections, we focus on designing allocation strategies of dependency links.

3.2 Dependency-link allocation

Our design of dependency-link allocations is divided into the following two steps:
Step 1: Basic connectivity — each node has at least one dependency link with least total weights.
Step 2: Augmented connectivity — we use the remaining budget to set up additional dependency links.

3.2.1 Problem formulation

In Step 1, each node in one network requires power or information from the node in another network through dependency links. Therefore, each node in the interdependent networks should have at least one dependency link in our design. Using least total weights to achieve basic connectivity becomes an optimization problem. Different weights of dependency links complicate this problem. We adapt a revised network flow algorithm[95] to solve it.

In our model, the numbers of nodes in two networks are unequal. The number of links should be no less than the larger number of nodes in two networks, so that each node in that network has one dependency link, while each node in the opposite network has at least one dependency link. In this chapter, we assume the number of nodes in the right side network

(N_B) is larger than the number of nodes in the left side (N_A). Therefore, a total of N_B links is needed to achieve basic connectivity using the least weights. We generally describe the Step 1 as follows:

$$min \sum_{i,j} m_{ij} \times W_{ij} \tag{3.2}$$

$$\sum_{i} m_{ij} = 1 \tag{3.3}$$

$$\sum_{j} m_{ij} \geq 1 \tag{3.4}$$

Matrix m is a 0-1 matrix, which means that m_{ij} could only equal 0 or 1. If the link between A_i and B_j is selected, m_{ij} is equal to 1. Matrix W is the matrix containing all dependency-link weights in the network, and W_{ij} represents the weight of edge m_{ij}.

3.2.2 Existing tools limitation

Solving the basic connectivity is an optimization problem. However, due to the large numbers of variables in large networks (e.g., if 1,000 nodes are in each network, 1,000,000 variables will be in our problem 3.2.1), this problem cannot be solved by traditional tools such as LINGO or MATLAB.

3.3 Revised network flow algorithm

We revise the network flow algorithm[96;97] to solve Step 1. This is another highlight of this chapter. The new algorithm is the minimum cost maximum flow with lower and upper bounds (cost flow for abbreviation in the following).

3.3.1 Notations of cost flow

$G(V, E)$ is a finite directed graph in which every edge $(u, v) \in E$ has a non-negative, real-valued capacity $c(u, v)$, lower flow bound $c_0(u, v)$, and a real-valued cost $p(u, v)$. If $(u, v) \notin E$, we assume that $c(u,v) = 0$, $c_0(u,v) = 0$, and $p(u,v) = 0$. The three properties of edge (u,v) can be written as $(c_0(u,v), c(u,v); p(u,v))$ for simplicity.

3.3.2 Basic constraints in cost flow

Let $N = G(V,E)$ be a network with $s,t \in V$ being the source and sink, respectively.

A flow in a flow network is a real function $f : V \times V \rightarrow R$ with the following properties for all nodes u and v. Notation $f(u,v)$ is the network flow from u to v.

Capacity and lower flow bound constraints: $c_0(u,v) \leq f(u,v) \leq c(u,v)$. The flow along an edge is larger than c_0 but cannot exceed its capacity c.

Skew symmetry: $f(u,v) = -f(v,u)$. The net flow of link (u,v) must be opposite of the net flow of (v,u).

Flow conservation: $\sum_{v \in V} f(u,v) = 0$, unless $u = s$ or $u = t$. The net flow of a node is zero, except for the source, which "produces" flow and the sink which "consumes" flow.

Cost notation: $p(u,v)$ denotes the unit cost of flow from node u to node v, and $P(G)$ denotes total cost of flow from s to t. Therefore, $P(G) = \sum_{(u,v) \in E} p(u,v) \times f(u,v)$.

3.3.3 Algorithm process

In a basic connectivity problem, we construct the network as follows: for a weight matrix W with size $N_A \times N_B$, we build N_A nodes denoted by l_1 through l_{N_A}, and N_B nodes denoted by r_1 through r_{N_B}. An edge (l_i, r_j) has properties denoted by $(0, 1; W_{ij})$, which means for any integer $1 \leq i \leq N_A$ and $1 \leq j \leq N_B$, there is an edge between node l_i and r_j, and the lower flow bound is 0, capacity is 1, and cost is the corresponding value in the weight matrix, W_{ij}. In addition, we add an edge (s, l_i) with $(1, N_B; 0)$ for every l_i, and an edge (r_j, t) with $(1, 1; 0)$ for every r_j, as shown in Fig. 3.3, and then run the minimum cost maximum flow algorithm on this network. All these edges are directed edges.

Residual capacity of an edge is denoted by $c_f(u,v)$ and cost of this edge is denoted $p_f(u,v)$. If $(u,v) \in E$, then $c_f(u,v) = c(u,v) - f(u,v)$, $p_f(u,v) = p(u,v)$; $c_f(v,u) = f(u,v) - c_0(u,v)$, $p_f(v,u) = -p(u,v)$. These edges construct a residual network denoted

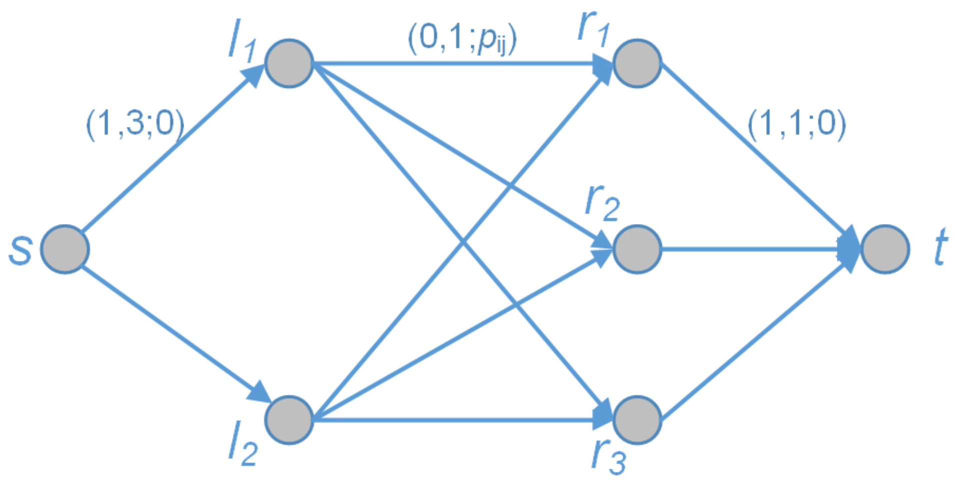


Figure 3.3: Cost flow algorithm. Three properties are on each link, denoted by $(c_0, c; p)$. The leftmost node is the source (s), and the rightmost node is the sink (t). The N_A nodes $l_1, l_2, \cdots, l_{N_A}$ and the N_B nodes $r_1, r_2, \cdots, r_{N_B}$ represent N_A rows and N_B columns in matrix W, respectively.

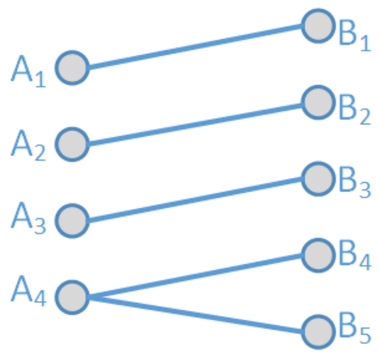
$G_f(V, E_f)$, representing the amount of available capacity. This residual network is a new directed network compared to the original network with two properties on each edge: residual capacity and cost. Augmenting is expected to happen in this residual network. A path can be observed from u to v in the residual network, even though no path is evident from u to v in the original network. If $(u, v) \in E$, we point out $p(v, u) = 0$ in the original network; however, we have $p_f(v, u) = -p(u, v)$ in the new network. Since flows in opposite directions cancel out, decreasing the flow from v to u and increasing the flow from u to v are identical.

An augmenting cycle is a negative-weight cycle in the residual network. Weight in the network is the cost; sending flow around the cycle strictly decreases total cost and preserves feasibility. A feasible flow f is optimal if and only if there are no augmenting cycles.

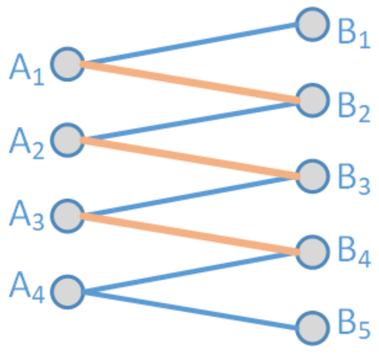
In a regular minimum cost maximum flow problem, first we should find maximum flow with a feasible solution, and then find an augmenting cycle in order to identify minimum cost. However, in this problem, all edges from r_j to t are identical; every edge has both lower bound and capacity equal to 1. Consequently, $f(r_j, t)$ is equal to 1 for any j, and thus maximum flow of this network is N_B. Therefore, the procedure of finding maximum flow is unnecessary. By initially setting any feasible flow and finding an augmenting cycle repeatedly, minimum cost can finally be found. A negative-weight cycle can be found by utilizing a shortest-path algorithm, for example, shortest path faster algorithm (SPFA).

3.3.4 Relationship between flow graph and link selection

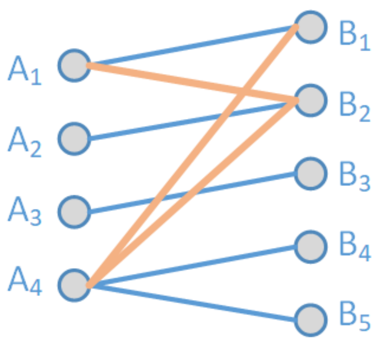
We propose this cost flow algorithm to solve the following problem: in a bipartite graph, we select the minimum number of links (equal to N_B), such that all nodes are covered and have the minimum total weights. Every feasible flow in the graph corresponds one-to-one to a set of links, which satisfy the constraints in Eq. (3.2), (3.3), and (3.4).



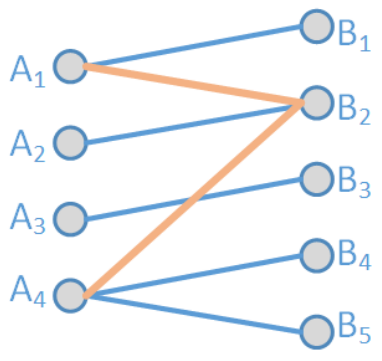
(a) The basic connectivity



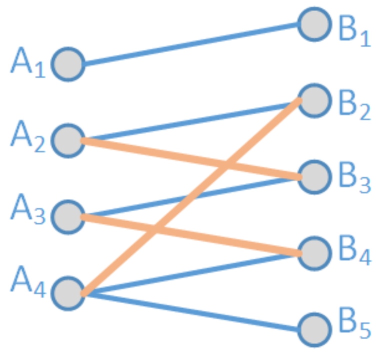
(b) Uniform distribution



(c) Random distribution



(d) An example of weighted networks



(e) Another weighted networks

Figure 3.4: Non-weighted link and weighted link allocation example. Specifically, dependency links, in Fig. 3.4d and 3.4e, are allocated according to matrices $W_{non-weighted}$ and $W_{weighted}$, respectively.

The capacity of link (r_j, t) is 1, which means that at most one node can provide node r_j one unit of flow. Also, the lower bound is 1, which means node r_j needs at least one unit of flow. These two constraints guarantee that node r_j will acquire exactly 1 unit of flow; therefore, the constraint in Eq. (3.3) is guaranteed.

Similarly, the capacity and lower bound of link (s, l_i) guarantee the constraint in Eq. (3.4).

Matrix f can be any feasible flow matrix, and the selection of f does not influence the final result; it is only an initial guess. The initial guess of f in our simulation is:

Algorithm 1 Cost Flow

```
Input: $W$ ($N_A \times N_B$ matrix)
Output: $f$ (0-1 Matrix)
 1: function MinCostMaxFlow($W$)
 2:     $p \leftarrow \begin{bmatrix} 0_{1\times 1} & 0_{1\times N_A} & 0_{1\times N_B} & 0_{1\times 1} \\ 0_{N_A\times 1} & 0 & W & 0 \\ 0_{N_B\times 1} & 0 & 0 & 0 \\ 0_{1\times 1} & 0 & 0 & 0 \end{bmatrix}$
 3:     $c \leftarrow \begin{bmatrix} 0_{1\times 1} & \infty_{1\times N_A} & 0_{1\times N_B} & 0_{1\times 1} \\ 0_{N_A\times 1} & 0 & 1 & 0 \\ 0_{N_B\times 1} & 0 & 0 & 1 \\ 0_{1\times 1} & 0 & 0 & 0 \end{bmatrix}$
 4:     $c_0 \leftarrow \begin{bmatrix} 0_{1\times 1} & 1_{1\times N_A} & 0_{1\times N_B} & 0_{1\times 1} \\ 0_{N_A\times 1} & 0 & 0 & 0 \\ 0_{N_B\times 1} & 0 & 0 & 1 \\ 0_{1\times 1} & 0 & 0 & 0 \end{bmatrix}$
 5:     $f \leftarrow$ any matrix that is a feasible initial flow
 6:     while true do
 7:         $Dis \leftarrow 0$ with the same size as $p$
 8:         for all $i$, $j$ do
 9:             if $f_{ij} < c_{ij}$ then $Dis_{ij} \leftarrow p_{ij}$
10:             if $f_{ij} > c_{0_{ij}}$ then $Dis_{ji} \leftarrow -p_{ij}$
11:         end for
12:         $Cycle \leftarrow$SPFA($Dis$)            ▷ Use SPFA algorithm to find negative cycle.
13:         if Cycle not found then return $f$
14:         for each edge $(u, v)$ on the cycle do
15:             $f_{uv} \leftarrow f_{uv} + 1$
16:             $f_{vu} \leftarrow f_{vu} - 1$
17:         end for
18:     end while
19: end function
```

$$f \leftarrow \begin{bmatrix} 0_{1\times1} & K & 0_{1\times N_B} & 0_{1\times1} \\ 0_{N_A\times1} & 0 & D & 0 \\ 0_{N_B\times1} & 0 & 0 & 1 \\ 0_{1\times1} & 0 & 0 & 0 \end{bmatrix}$$

$$K \leftarrow \begin{bmatrix} 1_{1\times(N_A-1)} & N_B - N_A + 1 \end{bmatrix}$$

$$D \leftarrow \begin{bmatrix} 1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & 1 & 1 & \cdots & 1 \end{bmatrix}$$

3.4 Dependency-link allocation example under limited budget

We use an example to describe various strategies of non-weighted link and weighted link allocations as shown in Fig. 3.4. In Fig. 3.4, blue links are allocated in basic connectivity and yellow links show augmented connectivity by utilizing the remaining budget. Fig. 3.4a shows the basic connectivity.

3.4.1 Parameters of the example

Here, we have parameters $tw = 8$, $N_A = 4$, $N_B = 5$.

$$W_{non-weighted} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

$$W_{weighted} = \begin{pmatrix} 1 & 2 & 2 & 2 & 2 \\ 2 & 1 & 1 & 2 & 2 \\ 2 & 2 & 1 & 1 & 2 \\ 2 & 1 & 2 & 1 & 1 \end{pmatrix}$$

3.4.2 Adding non-weighted links

Comparing Fig. 3.4b and Fig. 3.4c, the numbers of dependency links are identical using the limited budget, while distributions of these links differ. We offer simulation results and analysis on different distributions of links in section 3.5.

3.4.3 Adding weighted links

With matrix $W_{weighted}$, we compare various sequences of dependency-links allocation in Fig. 3.4d and Fig. 3.4e, resulting in different numbers of links using an equal limited budget. We discuss the sequence of allocating dependency links and analyze simulation results in section 3.6.

3.5 Non-weighted dependency-link model simulation and comparison

Non-weighted dependency links, which can be assumed as links with the same weight 1, can largely simplify the Step 1 talked about in section 3.2. In Step 2, considering the remaining budget, no matter which links are selected, the number of links is equal because of the identical weight of all links. However, the network robustness differs based on its deployment. Many studies have indicated that node degree distribution is critical for robustness in the interdependent networks. The aim of augmented connectivity is to achieve minimum variance of node degrees, referred to as uniform distribution deployment in Fig. 3.4b. Also, instead of adding links in this pattern, randomly adding additional links, known as random distribution deployment, is shown in Fig. 3.4c.

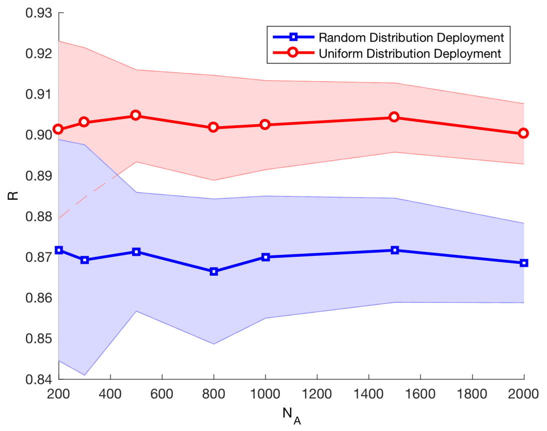
We apply ER networks to generate the topologies of two individual networks; however, ER networks generated in each run in the simulation are not identical. Due to the stochastic nature of ER networks, we simulate multiple times in order to obtain the average performance of both deployment methods. The relationship between numbers of nodes in two individual networks is formulated as $N_B = (1 + \alpha) \times N_A$. Therefore, node numbers N_A and N_B have identical magnitude and ratio, making simulation results comparable. These descriptions are also suitable for the weighted model in section 3.6.

Two non-weighted dependency-link deployments are simulated in Fig. 3.5a and Fig. 3.5b. In these two distributions, we have the same average node degree. Also, in the basic connectivity, we guarantee every node is connected to at least one node on the other side (left side or right side). However, performances of these two distributions are not identical. The simulations indicate uniform distribution is more robust than random distribution, as shown in Fig. 3.5a and Fig. 3.5b. In[81], the authors indicated similar conclusions under equal numbers of nodes in two networks. Our simulation results can be considered as the extensions of results in[81].

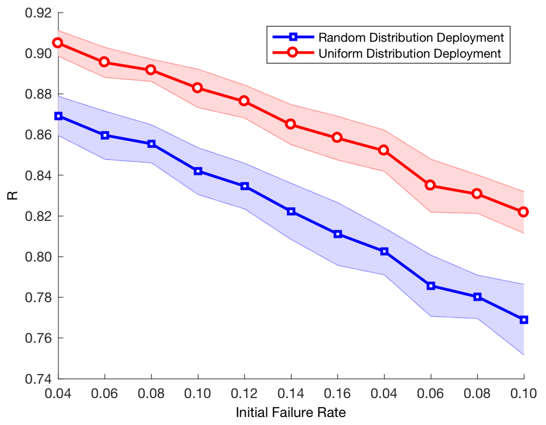
In Fig. 3.5c, each line represents a number of nodes (N_A). Every line shows the relationship between initial failure rate and robustness (R), where we use uniform distribution deployment. Simulation results show a threshold of the sharp decrease is between 0.4 and 0.5. We call this threshold critical failure rate (p_c). The sharp decrease of robustness will be the focus of further study. When initial failure rate is larger than 0.5, networks are very likely to be completely destroyed.

3.6 Weighted dependency-link model simulation and comparison

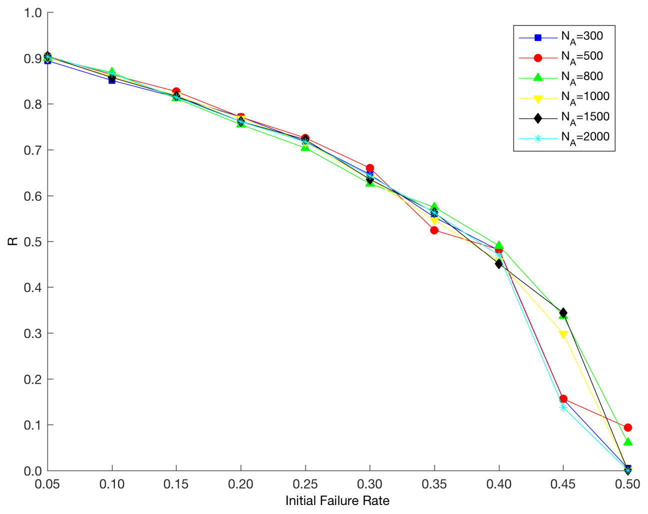
The concept of limited budget is first introduced by us from engineering perspectives. Given the limited budget of dependency-links deployment, how to design the links' deployment? We have discussed how to allocate weighted dependency links using least total weights to achieve



(a) The R of two strategies at different N_As



(b) The R of two strategies at different initial failure rates



(c) Each N_A's corresponding R at different initial failure rates

Figure 3.5: Non-weighted networks comparisons.

basic connectivity in section 3.3. From a traditional viewpoint, maximizing the number of dependency links always performs better statistically in the case where the dependency-link weight distribution is independent of the structures of the two individual networks. Maximizing the number of dependency links in Step 2 might give us more robustness. Maximizing the number of dependency links deployment and randomly adding dependency links deployment are included in our comparison.

In randomly choosing deployment, if we choose links randomly from the beginning, a majority of nodes will not have a dependency link. This is not fair to random distribution. Therefore, we implement the random selection algorithm after the basic connectivity. We solve the basic connectivity by first using a revised network flow algorithm and then compare robustness of the maximum number of link model and random links deployment model with a limited budget. After the optimal allocation of dependency links in Step 1 is realized by the revised network flow algorithm, we need to allocate additional links in augmented connectivity using the remaining weights. There are two ways to realize Step 2.

3.6.1 Maximizing the number of dependency links

To maximize the number of links according to the greedy algorithm, the least weight link should be added until the budget has been achieved. We sort weights using an unstable sorting algorithm, quick sort, to make the process efficient and random[98]. For the purpose of this study, "random" can be illustrated by saying if we have 5 units of budget left, but we have 10 links with weight 1 to choose from, we should choose 5 links from 10 randomly, instead of selecting links from left to right, up to down in the matrix.

$$\sum_{i,j} m_{ij} \times W_{ij} \leq cb$$

3.6.2 Randomly choosing dependency links

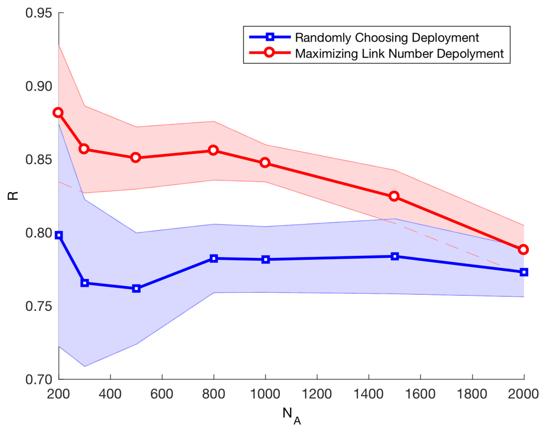
Randomly (with equal probability) let $m_{ij} = 1$

$$\sum_{i,j} m_{ij} \times W_{ij} \leq cb$$

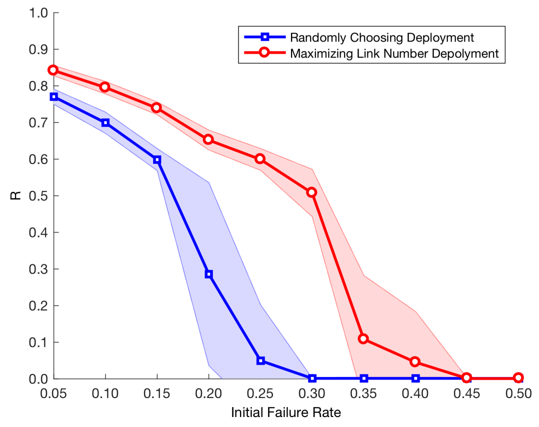
In regular random selection, k items should be picked up from n items; however, in this problem, we have a budget and we must guarantee the chosen links do not exceed the budget limit; therefore, how many links will be chosen is unknown in advance. When we pick up a link with a weight larger than the remaining budget, we cannot simply discard that link and randomly choose another link, because this is very likely to happen when the remaining budget is low, and therefore the process will be very time-consuming. Consequently, we design a more efficient algorithm for the random deployment problem. Our algorithm is shown in Algorithm 2. Since our algorithm is exactly the same as the naive algorithm until the remaining budget decreases to a number that is less than C, we will focus on the algorithm complexity analysis when the remaining budget is less than C. In the simulation, elements in the weight matrix follow the uniform distribution $\mathcal{U}\{1, C\}$. Our algorithm is polynomial with the complexity of O($N$lg$N$), where $N = N_A \times N_B$; while the naive algorithm is a pseudo-polynomial algorithm with the complexity of O(C).

Algorithm 2 Randomly Choosing Edges

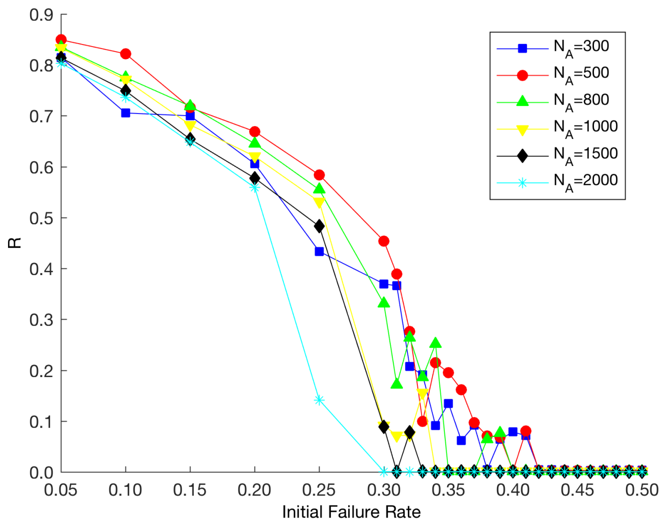
```
Input: $W$ ($N_A \times N_B$ matrix), $cb$
Output: $ch$ (0-1 Matrix)
 1: function RandomChoose($W$)
 2:     $ch \leftarrow$MinCostMaxFlow($W$)
 3:     Create array $s$ whose element is {int,int,int}
 4:     for all $i$, $j$ do
 5:         if $f_{ij} = 1$ then $cb \leftarrow cb - p_{ij}$
 6:         else if $p_{ij} \neq 0$ then $s.append(\{i, j, p_{ij}\})$
 7:     end for
 8:     sort $s$ by the third dimension
 9:     $k \leftarrow s.size$
10:     $chosen[1..k] = 0$
11:     while $k > 0$ do
12:         $r \leftarrow$ random int from 1 to $k$
13:         if $cb > s[r].third$ && $chosen[r] = 0$ then
14:             $cb \leftarrow cb - s[r].third$
15:             $choosen[r] = 1$
16:         else
17:             $k \leftarrow r - 1$
18:     end while
19:     for all $i$ in $chosen$ do
20:         if $chosen[i] = 1$ then
21:             $ch[s[i].first][s[i].second] = 1$
22:     end for
23:     return $ch$
24: end function
```



(a) The R of two strategies at different N_As



(b) The R of two strategies at different initial failure rates



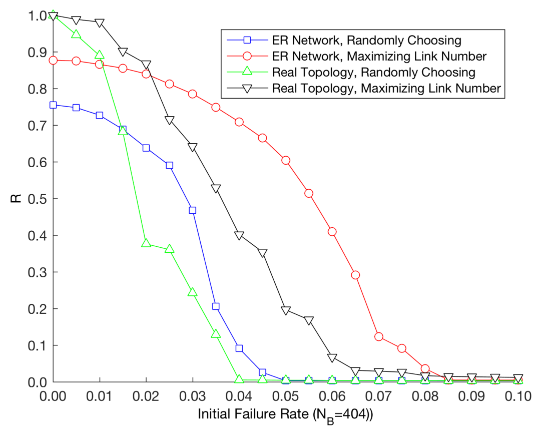
(c) Each N_A's corresponding R at different initial failure rates

Figure 3.6: Weighted networks comparisons.

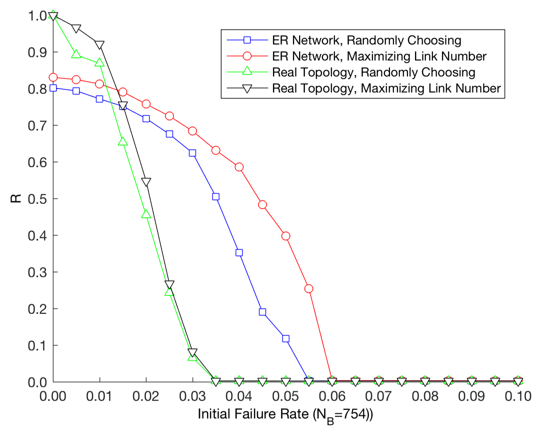
3.6.3 Comparison

We obtain an adjacency matrix of dependency links between nodes from two networks based on Step 1 and Step 2. Results of maximizing the number of links deployment show more robust than results obtained by randomly choosing deployment, as shown in Fig. 3.6a and Fig. 3.6b. Results indicate that various designs of weighted dependency links greatly affect network robustness.

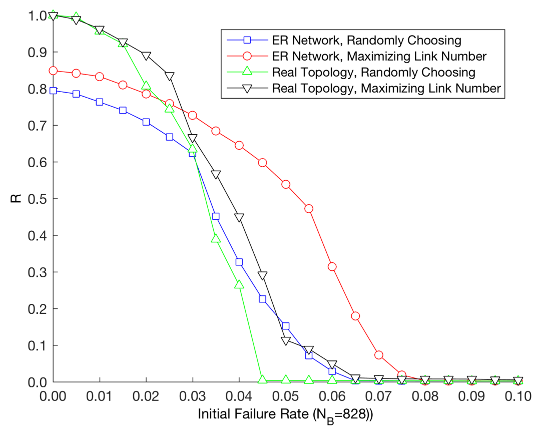
Similar to Fig. 3.5c, Fig. 3.6c illustrates the relationship between failure rate and robust-



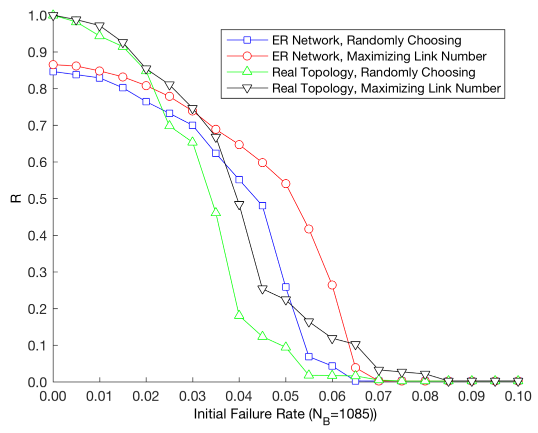
(a) $N_A = 300$ and $N_B = 404$



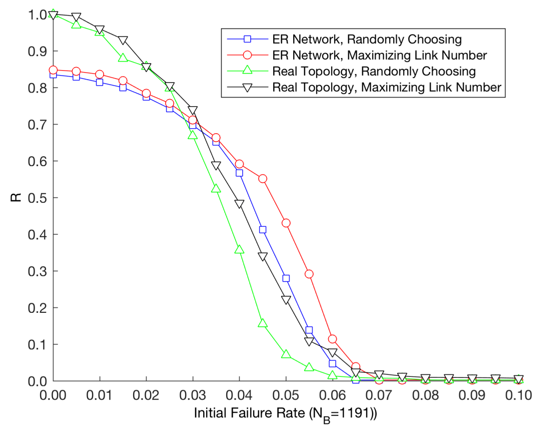
(b) $N_A = 300$ and $N_B = 754$



(c) $N_A = 300$ and $N_B = 828$



(d) $N_A = 300$ and $N_B = 1085$



(e) $N_A = 300$ and $N_B = 1191$

Figure 3.7: Models simulations in real topologies.

ness. We use a maximizing number of links deployment in the simulation. Zigzag occurs due to randomness of simulation, but a falling trend can be observed in every line. Critical failure rate p_c resides between 0.25 and 0.35, at which point the curves have a sharp decrease.

3.7 Extensive simulations on real network topologies and Barabási-Albert networks

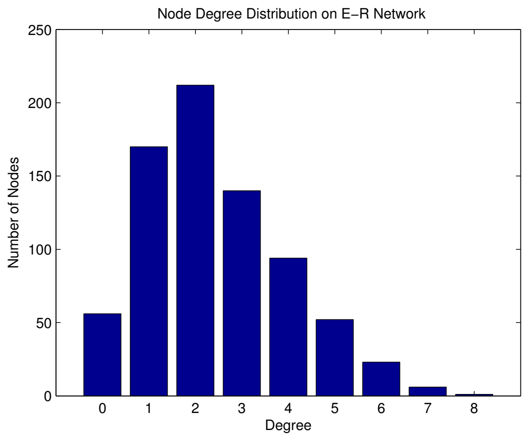
We have applied ER network topologies into previous simulations, thereby achieving meaningful results regarding allocations of dependency links. Strategies now need to be applied into our simulations with real network topologies in order to achieve the acceptability of our design.

3.7.1 Real network topology simulation and comparison

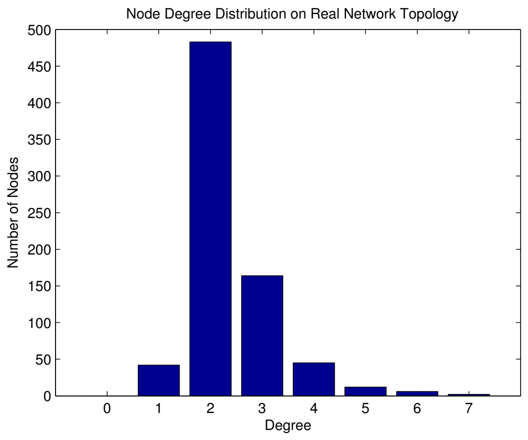
In the simulation, we use the IEEE 300-bus system as the power network topology, which is obtained from a power systems test case archive[99]. Computer network topologies are obtained from Caida[100] and Topology Zoo[101]. Dependency-link weights are generated in a uniform distribution. Here, we should notice that real dependency-link weights are affected by various factors, and exact data cannot be obtained by simply measuring and calculating. For research purposes, studying mathematical models to achieve the estimated weight matrix is an open issue, which is beyond the scope of our work.

With real network topology of two individual networks, the robustness of maximizing the number of links deployment is greater than the robustness of randomly choosing deployment as shown in Fig. 3.7. The simulation results coincide with previous simulation results with ER networks. However, the curves of real topologies in Fig. 3.7 decrease more sharply than those with ER topologies. The p_c of real network topology is less than the p_c of ER network topology because of different node-degree distributions of two networks as shown in Fig. 3.8.

Versatility and conciseness are the primary advantages of ER network topology. Simulations on ER networks show which deployment of interdependent networks is more robust. Robustness simulation results based on ER networks are fit with results based on real topologies. Furthermore, the existence of p_c is shown in the simulation results. This rate is critical to the protection of interdependent networks, because it gives us a threshold, under which the fraction of nodes failed can be tolerated. Thus, a proper model equals the meaning of millions of real data simulations.



(a) The R of two strategies at different N_As



(b) The R of two strategies at different initial failure rates

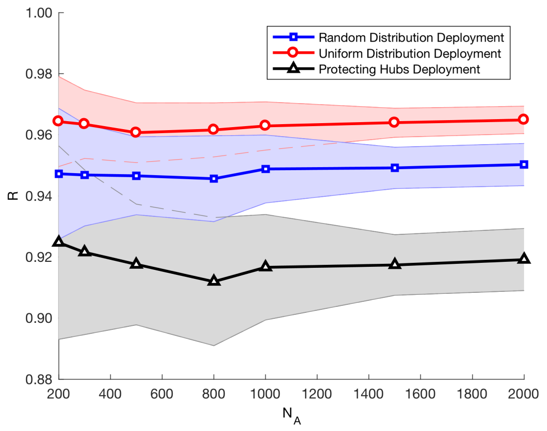
Figure 3.8: Different degree distributions.

3.7.2 Dependency-link allocation on Barabási-Albert networks

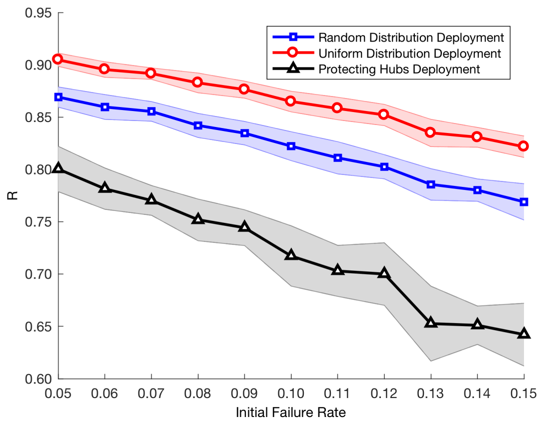
Discovering an optimal solution to Step 2 for both weighted and non-weighted dependency links is in general challenging. To achieve more results, we explore more strategies on non-weighed dependency link design. Here, we extend network topologies from ER networks to Barabási-Albert networks, and protect the hubs by assigning them more dependency links. We call this method protecting hubs deployment.

We simulate and compare random distribution deployment, uniform distribution deployment, and protecting hubs deployment with Barabási-Albert networks as the two individual networks.

Hub failures always break a single network into several small components. In the interdependent networks, if we assign more dependency links to the hubs of network A, the hubs always have dependency links survivable from the dependent nodes failing in network B, and won't fail easily. Consequently, network A will be protected from breaking into several small components. However, less dependency links will be assigned to non-hubs due to a limited budget. Now, we consider a case where the average number of dependency links is two for a node in network B. When we assign more dependency links to the hubs, many non-hubs will have only one dependency link, and they are at highly vulnerable states. Therefore, this new protecting hubs deployment might not perform as well as the first two deployments. Fig. 3.9 indicates uniform distribution deployment has better performance, which is coincident with our analysis.



(a) The R of three strategies at different N_As



(b) The R of three strategies at different initial failure rates

Figure 3.9: Three deployments within Barabási-Albert networks.

3.8 Contributions

In this chapter, we explore a more robust cyber-physical network framework. First, we design a more realistic interdependent network model and demonstrate its cascading failure's process. Second, we allocate dependency links to make networks more robust, and we adapt a revised network flow algorithm to solve this dependency-link allocation problem. Third, we conduct extensive simulations on different network topologies to validate the effectiveness of our deployment methods.

Our contributions are listed as follows:

1. We propose a realistic cyber-physical network framework with one-to-multiple dependencies, two unequal-size individual networks, and a weighted dependency link.

2. We deploy dependency links under a limited budget to obtain more robustness.

3. We adapt a revised network flow algorithm to obtain a basic network structure.

4. We conduct extensive simulations on different dependency-link deployments and individual network topologies. Robustness tends to have a sharp decrease at a certain initial failure rate.

Chapter 4

Middlebox policy enforcement using SDN

Network applications require traffic to sequence through multiple types of middleboxes to enhance network functionalities, e.g., load balancers are used to circumvent overloads and IDSes are used to detect anomalies. Sequenced-middlebox policy routing on top of regular layer 2/3 flow routing is challenging to be flexibly managed by network administrators. In addition, various types of middlebox resources concurrently obtained by numerous applications complicate network-resource management. Fortunately, SDN helps solve these problems flexibly. Because of the existence of redundancy in the network, effects of middlebox failures could be eliminated if we are able to quickly reroute those affected flows to middleboxes with enough processing capabilities. Therefore, the challenge is to find such backup middlebox and reroute flows quickly when a middlebox failure occurs. SDN also helps quickly identify the failures and find alternative paths in order to minimize failure effects. In this chapter, we develop a global load-balancing routing approach and a local rerouting approach to handle different scenarios in the middlebox chain problem using SDN.

4.1 Middlebox policy enforcement problem

4.1.1 Problem statement

We are aiming at solving open issues in the middlebox chain problem. First, we propose a candidate-path generation method to quickly select possible flow-routing paths. Second, we propose to consider middlebox resources as one of the constraints in the middlebox chain problem. Third, we are seeking to improve network performances with constraints of numerous limited network resources. Fourth, the network can quickly respond to network failures and changes.

4.1.2 Candidate paths generation

Candidate paths are used to balance network loads. We propose a set of routing paths called "middlebox-by-middlebox shortest routing paths" (*m-by-m* routing paths) to simplify candidate-path selection. When the middlebox policy is specified by network administrators, candidate paths are determined. M_i denotes the set of the i_{th} type middleboxes, e.g., firewalls, and $|M_i|$ denotes the number of middleboxes of the type i. A middlebox policy chain is "$Source \rightarrow M_1 \rightarrow M_2 \rightarrow ... \rightarrow M_n \rightarrow Destination$." Each candidate path is determined by finding the shortest path to or from each middlebox: $Source \rightarrow M_1$, $M_1 \rightarrow M_2$, ..., and $M_n \rightarrow Destination$. An example is shown in Fig. 4.1. There is a directed demand from S_1 to S_6 with the logical policy "$S_1 \rightarrow$ firewall $\rightarrow$ intrusion detection system $\rightarrow S_6$." Step 1 is to find the shortest path from S_1 to FW_1 (or FW_2). Step 2 is determination of the shortest path from FW_1 (or FW_2) to IDS. Step 3 is to find the shortest path from IDS to S_6. Since there are two firewalls and one IDS, there will be at least two candidate paths for the demand "$S_1 \rightarrow S_6$" to route along. To be more general, if there are $|M_{FW}|$ firewalls and $|M_{IDS}|$ IDSes, and the flow must go through a firewall then an IDS, there will be at least $|M_{FW}| \times |M_{IDS}|$ possible paths for this flow to choose from.

We show three different *m-by-m* routing paths in Table 4.1. The difference between path a and path b is a different firewall selection; the difference between path b and path c is a

Table 4.1: Routing path (S_1 → Firewall → IDS → S_6)

Path \ Step	1	2	3	4	5	6	7	8	9	10	11
a	S_1	S_2	S_5	S_8	FW_1	S_8	S_5	IDS	S_5	S_4	S_6
b	S_1	S_3	FW_2	S_3	S_6	S_4	S_5	IDS	S_5	S_4	S_6
c	S_1	S_3	FW_2	S_3	S_1	S_2	S_5	IDS	S_5	S_4	S_6

different shortest-path selection among multiple shortest paths.

We propose the *m-by-m* routing paths as candidate paths for the following reasons:

1. The *m-by-m* approach allows the simple generation of many candidate paths, and one of them will be chosen by the centralized controller to achieve network load balancing. On the chosen path, we can easily record link loads and middlebox loads. Therefore, we are able to balance link and middlebox loads at the same time.

2. To avoid network congestion, the flows may be routed through a longer *m-by-m* shortest path. However, within a certain step, the shortest path is always chosen to save network resources.

3. Using this approach, flow-level routing can largely reduce the number of flow entries

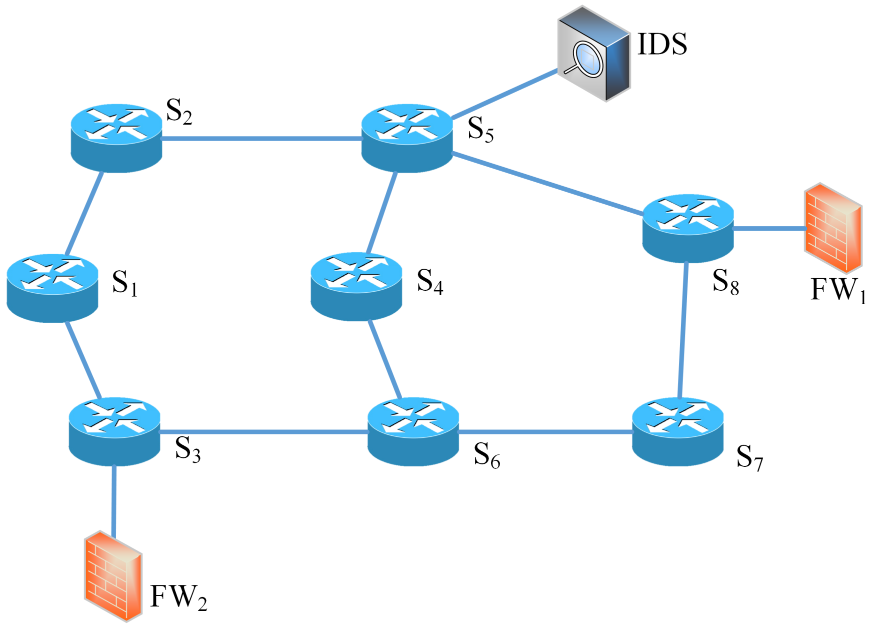


Figure 4.1: Middlebox-by-middlebox shortest paths.

installed on each switch, compared with flow-splitting routing.

4. The *m-by-m* approach can reduce congestion on a middlebox. Considering a routing example with the policy "Source → Firewall → Destination," the directed pairwise demand list is $\{S_1 \to S_5 : 10\text{Mbps}, S_6 \to S_5 : 10\text{Mbps}, S_7 \to S_5 : 10\text{Mbps}, S_8 \to S_5 : 10\text{Mbps}\}$, shown in Fig. 4.2. FW_1 will be overloaded if all flows choose the sequenced, shortest source-destination path (SP routing approach). We use *m-by-m* shortest paths instead, with which a middlebox of the same type (FW_2) can help reduce congestion on the highly used middlebox FW_1 (because it is connected with a high-betweenness switch S_2).

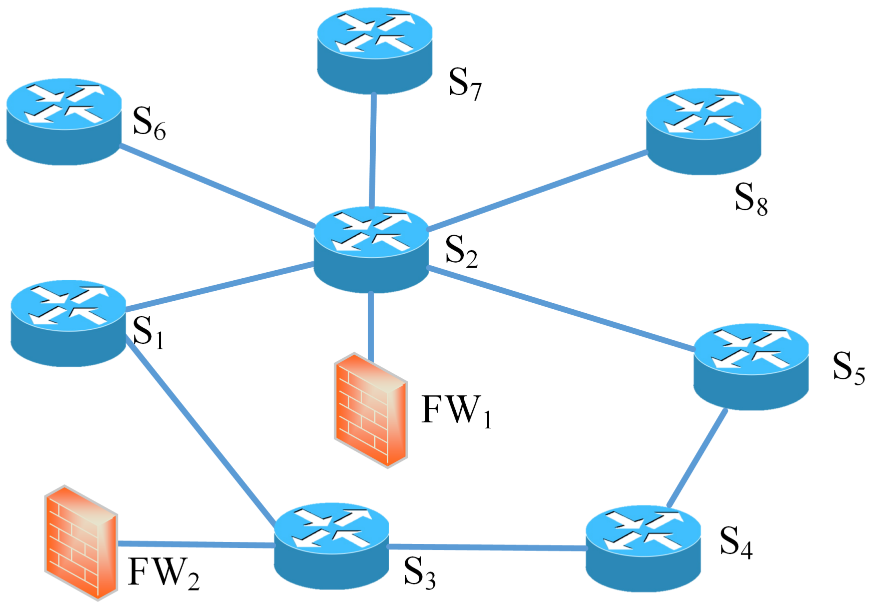


Figure 4.2: Middlebox overloaded example.

4.2 Global load-balancing routing

In the middlebox policy routing problem, we need to achieve network load balancing not only on the links but also on the middleboxes. In this part, we formulate a mixed-integer linear programming (MILP) problem to describe the network[102] and accomplish our goal, given the set of candidate paths. We call this global load-balancing routing approach (global

LB approach). Considering multiple network-resource constraints to globally make routing decisions is one of our contributions in this chapter.

4.2.1 Notations

Traffic matrix D_t represents demands of traffic types t, $t \in T$. Each traffic type has a different policy requirement. For example, traffic matrix D_1 represents HTTP traffic demands, which need to route through a firewall middlebox then an IDS middlebox; traffic matrix D_2 represents all other traffic demands, which need to route through a firewall middlebox only. Traffic demands are directed pairwise demands. P_d denotes the set of candidate paths for a given pairwise demand d. Given a network topology, E denotes the set of links and M denotes the set of middleboxes. The cardinality of a set is denoted by $|\ |$. For example, $|D_t|$ denotes the number of demand pairs of a traffic type t.

Constants:

n_{tdpe} :	the number of times link e occurs in path p of demand pair d of traffic type t.
δ_{tdpm} :	1 if middlebox m belongs to path p of demand pair d of traffic type t; 0, otherwise.
h_{td} :	volume of demand pair d of traffic type t.
c_e :	capacity of link e.
c_m :	capacity of middlebox m.

Variables:

x_{tdp} :	flow allocated to path p of demand pair d of traffic matrix t.
u_{tdp} :	binary variable associated with x_{tdp}.
θ_e :	utilization of link e.
θ_m :	utilization of middlebox m.
θ :	maximum utilization of links and middleboxes.

4.2.2 Formulations

The demand satisfaction constraints are shown in Eq. 4.1. From the candidate paths, if we choose path p to route the flow, the corresponding binary variable u_{tdp} equals 1; otherwise 0.

The demand volume can be routed on only one path out of all the candidate paths, shown in Eq. 4.2. Here, Eqs. 4.3 and 4.4 represent link-capacity constraints and middlebox-capacity constraints, respectively. We use the variable θ to constrain link and middlebox utilization. Both link and middlebox utilizations should be no greater than 1. The goal is to minimize the maximum link or middlebox utilization, therefore achieving network load balancing. We call θ the network utilization in the following sections.

P1:

minimize $\quad\theta$

subject to

$$x_{tdp} = h_{td}u_{tdp}, t \in T, d \in D_t, p \in P_d. \tag{4.1}$$

$$\sum_p u_{tdp} = 1, t \in T, d \in D_t. \tag{4.2}$$

$$\sum_t \sum_d \sum_p n_{tdpe}x_{tdp} \leq \theta_e c_e, e \in E. \tag{4.3}$$

$$\sum_t \sum_d \sum_p \delta_{tdpm}x_{tdp} \leq \theta_m c_m, m \in M. \tag{4.4}$$

$$\theta_e \leq \theta \leq 1, e \in E. \tag{4.5}$$

$$\theta_m \leq \theta \leq 1, m \in M. \tag{4.6}$$

We substitute x_{tdp} by $h_{td}u_{tdp}$ using Eq. 4.1, which largely reduces the number of variables and simplifies $P1$. Also, we can directly use variable θ, so the variables θ_e and θ_m are omitted. $P2$ is the same problem derived from $P1$.

P2:

minimize θ

subject to

$$\sum_p u_{tdp} = 1, t \in T, d \in D_t. \tag{4.7}$$

$$\sum_t \sum_d h_{td} \sum_p n_{tdpe} u_{tdp} \leq \theta c_e, e \in E. \tag{4.8}$$

$$\sum_t \sum_d h_{td} \sum_p \delta_{tdpm} u_{tdp} \leq \theta c_m, m \in M. \tag{4.9}$$

4.2.3 Other use cases

Today, computer network demands are increasing dramatically, so network resources are limited. Our LB routing approach improves network performance by balancing all network loads. Our approach relies on the accuracy of estimated traffic demands, which can be guaranteed using the approaches in[103–105]. By minimizing the maximum utilization, the approach is effective to balance all network resources well.

Other network features (cost, delay, congestion, etc.) can also be formulation objectives. We can slightly modify the load-balancing formulation to meet the new requirements. For example, we formulate a problem $P3$ to minimize network cost in a simplified case of a single traffic type. y_e denotes the load of link e. y_m denotes the load of middlebox m. ξ_e represents the unit cost of link e. ξ_m represents the unit cost of middlebox m.

P3:

minimize $\quad F = \sum_e \xi_e y_e + \sum_m \xi_m y_m$

subject to

$$x_{dp} = h_d u_{dp}, d \in D, p \in P_d. \tag{4.10}$$

$$\sum_p u_{dp} = 1, d \in D. \tag{4.11}$$

$$\sum_d \sum_p n_{dpe} x_{dp} = y_e, e \in E. \tag{4.12}$$

$$y_e \leq c_e, e \in E. \tag{4.13}$$

$$\sum_d \sum_p \delta_{dpm} x_{dp} = y_m, m \in M. \tag{4.14}$$

$$y_m \leq c_m, m \in M. \tag{4.15}$$

4.2.4 Complexity analysis

$P2$ is formulated as an MILP problem. It's challenging for the controller to make routing decisions in a short time, when the network is large and the topology or policy is updated. We will illustrate how to solve this problem in section 4.3.

For the complexity of management, we have an estimation of the number of flow entries. Upper bounds of the number of flow entries on each switch in the simplified case of a single traffic type: $\frac{|D|(|Len|+1)\varnothing}{\#switch}$. $|Len|$ denotes the number of distinct types of middleboxes in the middlebox policy sequence of that single traffic type. $\varnothing$ represents the diameter of the network. $\#switch$ indicates the number of switches in the network.

4.3 Solutions of the global load-balancing routing

We are going to solve the load-balancing optimization problem in this section. First, we use the branch-and-bound algorithm (BBA)[102] to find the optimal solution of this problem.

In the optimization problem, there are $\prod_{t \in T} (\prod_i |M_i|)^{|D_t|}$ possible combinations of variables. Though BBA is an optimized algorithm, running time grows exponentially with the number of variables. Therefore, the problem cannot be solved by BBA in a large network. Then, we use the simulated annealing algorithm (SAN)[102] to obtain near-optimal solutions in a faster way, and then compare the results acquired from BBA and SAN algorithms in subsection 4.3.3.

4.3.1 Branch-and-bound algorithm

We apply BBA to solve this optimization problem, shown in Algorithm 3. BBA is a search algorithm designated for discrete optimization problems, and it gives us the optimal solution much faster than a brute-force approach.

Here we are applying BBA to a binary integer programming problem, so each binary variable has two branches. Function SOLUTION(N_U,N_0,N_1) returns the optimal solution θ^* and the corresponding variable vector u of the relaxed LP subproblem. The following constraints hold:

$$\begin{aligned} &0 \le u_j \le 1 (\text{continuous}) && \text{for } j \in N_U \\ &u_j = 0 && \text{for } j \in N_0 \\ &u_j = 1 && \text{for } j \in N_1 \end{aligned}$$

However, because of the searching nature of BBA, its execution time is, in general, as exponential as the number of binary variables. If we have X binary variables and function SOLUTION runs in $O(S)$ time, the worst-case, overall running time is $O(2^X S)$. It is far beyond our computing capability when it comes to a larger network.

4.3.2 Simulated annealing algorithm

We also apply the SAN algorithm as a substitute for the BBA. The algorithm is shown in Algorithm 4. SAN is a general optimization technique for solving combinatorial optimization problems, based on randomization techniques[106]. SAN is a heuristic algorithm and gives us

Algorithm 3 Branch-and-Bound Algorithm (BBA)

```
Input: $N_U$, $N_0$, $N_1$
Output: $\theta_{BBA}^{best}$
 1: function BBA($N_U$, $N_0$, $N_1$)
 2:     $\theta, u \leftarrow$SOLUTION($N_U$,$N_0$,$N_1$)
 3:     if $N_U = \emptyset$ or $\forall i \in U$, $u_i$ are binary then
 4:         if $\theta < \theta_{BBA}^{best}$ then
 5:             $\theta_{BBA}^{best} \leftarrow \theta$
 6:             $u^{best} \leftarrow u$
 7:     else
 8:         if $\theta \geq \theta_{BBA}^{best}$ then return                    ▷ Bounding
 9:         else                                                              ▷ Branching
10:             Choose $i \in N_U$ such that $u_i$ is fractional
11:             BBA($N_U \backslash \{i\}$,$N_0 \cup \{i\}$, $N_1$)
12:             BBA($N_U \backslash \{i\}$,$N_0$, $N_1 \cup \{i\}$)
13: end function
```

an acceptably good solution; and, moreover, it is much faster than search-based algorithms.

The realization of SAN is very straightforward. Our stopping criterion is either the outer loop counter reaches K (in our case $K = 1000$), or θ_{SAN} haven't updated for 10 outer loops. Initial temperature T^0 represents the ability of jumping out of a local minimum of the algorithm. We reduce the temperature every L inner loops. $L = 200$.

Since in our algorithm, running time of computing $F(x)$ is $O((|E| + |M|)X)$, where X is the number of binary variables, the worst-case overall running time of SAN is $O(KL(|E| + |M|)X)$.

4.3.3 Algorithm test and comparison

We test the running time of these two algorithms as the number of binary variables increases. Our test topology is shown in Fig. 4.3. There are two types of traffic: one is HTTP traffic, which needs to route through a firewall then an IDS; the second is OTHER traffic, which needs to route through a firewall only. The number of binary variables u_{tdp}s is related to the number of demand pairs, and the number of candidate paths of each demand pair. In the matrix of HTTP traffic, there are five demand pairs, and each demand pair has four candidate paths; while in the matrix of OTHER traffic, there are two demand pairs, and each demand

Algorithm 4 Simulated Annealing (SAN) Algorithm

```
Input: A feasible solution $x$, $T \leftarrow T^0$ and $L$
Output: $\theta_{SAN}$
 1: $x^{best} \leftarrow x$, $\theta_{SAN} \leftarrow F(x^{best})$
 2: while $stopping\_criterion$ not true do
 3:     $l \leftarrow 0$
 4:     while $l < L$ do
 5:         $z \leftarrow random\_neighbor(N(x))$
 6:         $\Delta\theta \leftarrow F(z) - F(x)$
 7:         if $\Delta\theta \leq 0$ then
 8:             $x \leftarrow z$
 9:             if $F(x) < \theta_{SAN}$ then
10:                 $\theta_{SAN} \leftarrow F(x)$, $x^{best} \leftarrow x$
11:         else if $random(0,1) < e^{-\Delta\theta/T}$ then
12:             $x \leftarrow z$
13:         $l \leftarrow l + 1$
14:     end while
15:     $reduce\_temperature(T)$
16: end while
```

pair has two candidate paths. The number of binary variables is "$5 \times 4 + 2 \times 2 = 24$." We get a group of test cases by increasing the number of demand pairs. The number of combinations of variables is calculated based on the expression at the beginning of this section. We keep

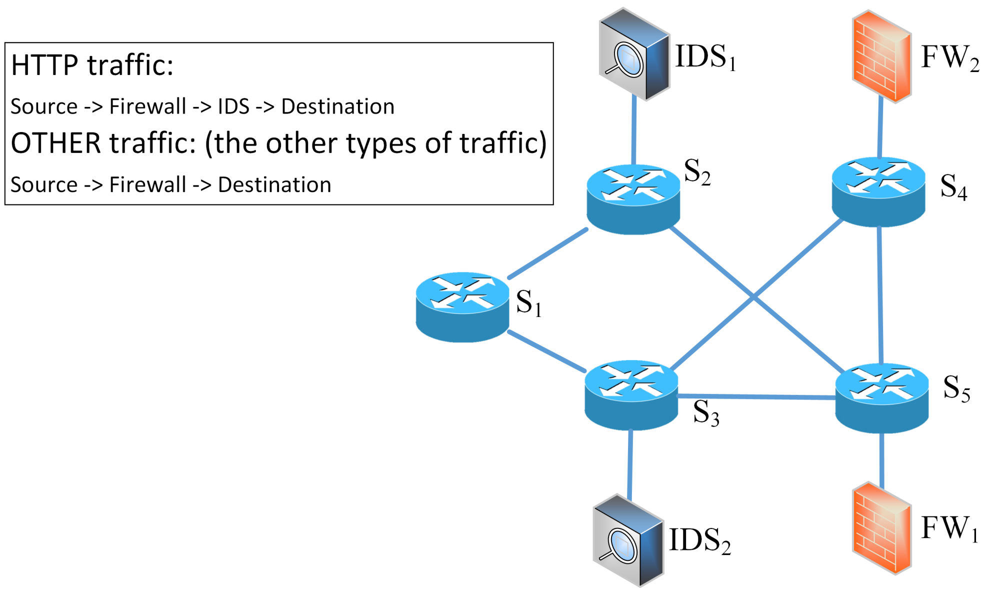

HTTP traffic:
Source -> Firewall -> IDS -> Destination
OTHER traffic: (the other types of traffic)
Source -> Firewall -> Destination
IDS$_1$
FW$_2$
S$_2$
S$_4$
S$_1$
S$_3$
S$_5$
IDS$_2$
FW$_1$

Figure 4.3: SAN and BBA test network

Table 4.2: Test cases and results

Test Case / Results	1	2	3	4	5	6	7	8
Binary variables	24	26	28	30	32	34	42	46
Combinations of Variables	4,096	8,192	16,384	32,768	65,536	131,072	2,097,152	8,388,608
Running Time Ratio (BBA / SAN)	88	157	876	974	1,534	2,109	12,884	23,891
θ_{BBA}^{best}	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85
θ_{SAN}	0.90	0.90	0.90	0.90	0.85	0.85	0.85	0.85

the total traffic volume in the network identical for all eight test cases. Test results are listed in Table 4.2. We have two observations from the results:

1. The ratio is defined as the BBA algorithm's actual running time divided by the SAN algorithm's actual running time. The ratio grows exponentially as the number of variables increases.

2. The SAN algorithm can always achieve a near-optimal solution. Specifically, the SAN algorithm can achieve the optimal solution θ in all eight test cases.

4.4 Evaluation of our global load-balancing routing on a Mininet testbed

In this section, we implement our global LB approach using OpenFlow on a Mininet testbed, and evaluate its effectiveness, compared with SP routing. We use the topology shown in Fig. 4.4, where the number of middleboxes is comparable to the number of nodes[38 107].

The *m-by-m* approach provides several choices of paths to balance network link and middlebox utilization. Each flow is assigned to only one candidate path according to the LB optimization problem, and all candidate *m-by-m* shortest routing paths are evaluated and stored at the same time. Flow entries on the switches are installed by the centralized POX controller. 'Iperf' is used to generate traffic at constant rates and measure network performances. Normalized throughput is defined as the ratio of received packets over sent packets, that are given by 'Iperf'. End-to-end latency is measured by sending ICMP packets in addition to the regular traffic.

4.4.1 Experiment setup

In our test, there are two types of traffic: HTTP and OTHER traffic, as described in subsection 4.2.1. Here we use traffic destined to port 5001 to denote 'HTTP' traffic, and port 5002 to denote 'OTHER' traffic. We test our approach with homogeneous and heterogeneous traffic matrices. The homogeneous traffic matrix is a demand matrix where all directed pairwise traffic demands are identical, and the data rate of each pair ranges from 1.5Mbps to 3.6Mbps. We have a total of 182 directed pairwise traffic demands. Heterogeneous traffic matrix means all but one outgoing traffic from a source node have identical data rates, and the exceptional one has a much higher data rate than the others. The exceptional node is chosen at random for each source node. For both homogeneous and heterogeneous matrices, it is the case that half of the traffic is HTTP traffic, and the other half is OTHER traffic. Link capacity is 115.0 Mbps and middlebox capacity is 93.0 Mbps. For end-to-end latency measurement, during each 10-second trial of regular traffic, default size ICMP packets with

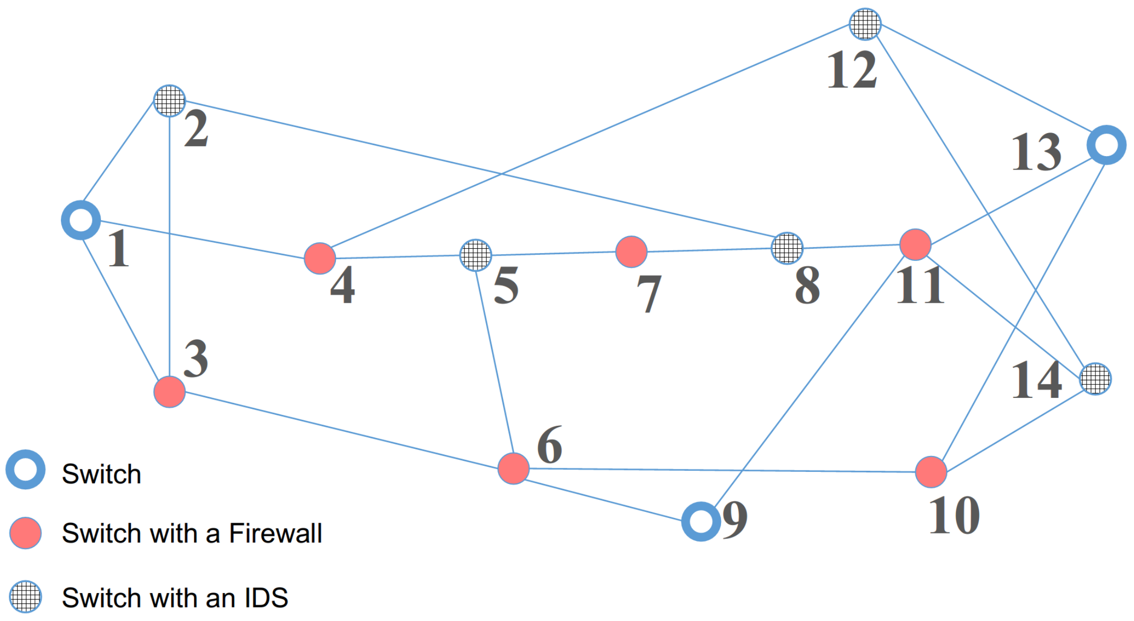


Figure 4.4: Test topology

an interval of 50 ms are injected between seconds 8 and 9, so that the network has an opportunity to stabilize. An average of 50 independent trials is used for each total traffic volume.

4.4.2 Observations from experiment results

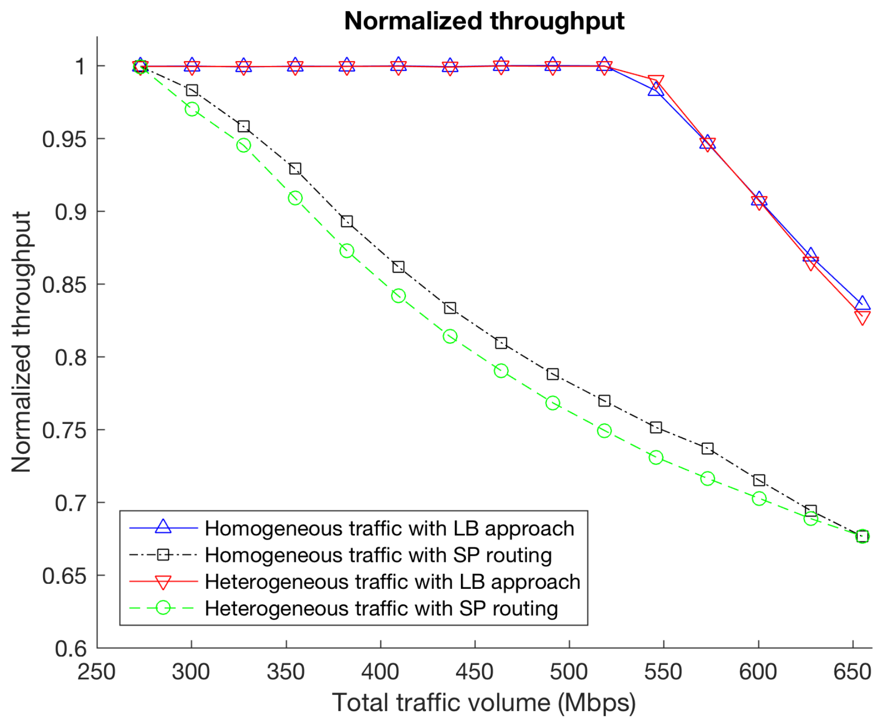


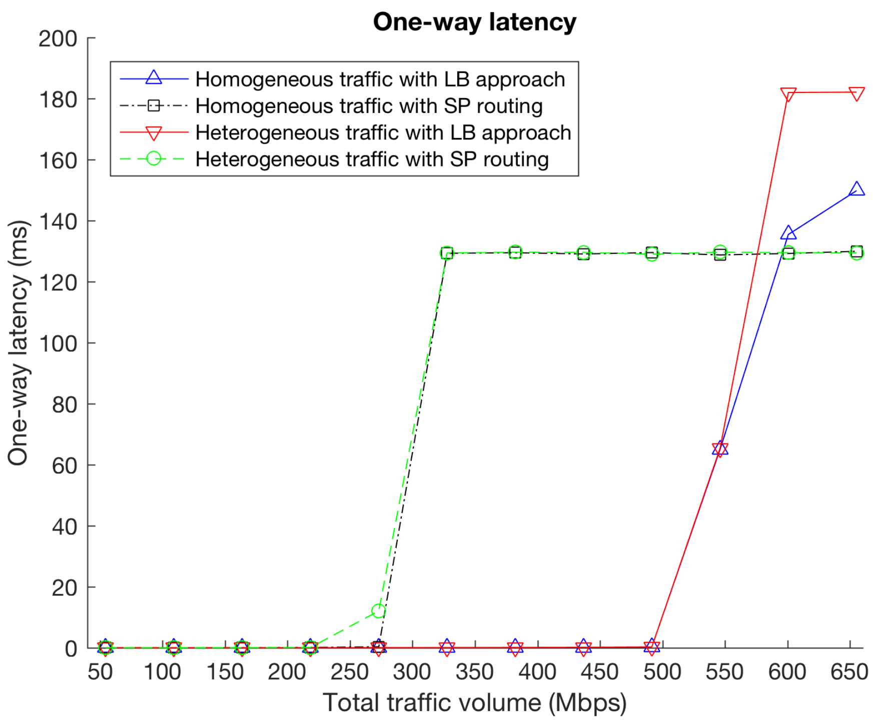
Figure 4.5: Evaluation of global LB routing

In Fig. 4.5, our global LB routing achieves almost the highest normalized throughput, when the total traffic volume equals 546.0 Mbps. At this point, network utilization is 1. When total traffic volume exceeds 546.0 Mbps, network resources are no longer sufficient to accommodate all flows and the normalized throughput decreases. Our LB approach shows an increase up to 26.4 percent on the throughput, when compared with the SP approach discussed in subsection 4.1.2. Throughput varies little between homogeneous traffic and heterogeneous traffic in both approaches. We also measure overall packet losses. Since results of normalized throughput and normalized packet losses are complementary, we don't show results of overall packet loss.

In Fig. 4.6, end-to-end latency and loss rate from node 1 to node 14 are shown as total traffic increases. Node 1 to node 14 is a representative node pair, and the shortest path between them is the diameter of the test topology. When total traffic volume is between 273 Mbps and 546 Mbps, network resources are relatively sufficient for each demand pair, and the LB approach achieves much lower end-to-end latency and loss rate than the SP routing. When the total traffic volume is greater than 546 Mbps, there is a bottleneck link on all possible paths from node 1 to node 14, i.e., both LB approach and SP routing are running on a congested network. Though the network is congested, the LB approach tries to balance the traffic so each flow is on a less congested path, while SP routing leads to very congested paths. This is the reason why the LB approach achieves lower end-to-end loss rate than SP routing. As expected, latency for any paths which are not shortest, including those of the LB approach, is greater than SP routing when the entire network becomes fully congested (the total traffic exceeds 546 Mbps), as shown in Fig 4.6.

4.5 Fast local rerouting

When there are long-term changes, recalculating routing paths of the entire network is unavoidable but should be done quickly. When there is a transient disturbance (temporary failure) on the network, we can centrally recalculate the MILP problem according to the working subnetwork (global LB approach). However, this process takes more time and forces the unaffected flows to reroute. Consequently, it increases the delay, packet loss, and use of network resources. Therefore, the entire network recalculation is not a good way to handle the transient disturbance. We have already evaluated the efficiency of solving the MILP problem when dealing with long-term network changes in the previous section. In this section, we design a local rerouting strategy to deal with network transient disturbance. With regard to the transient disturbance, a redundant backup path is used until the disturbance has ended.



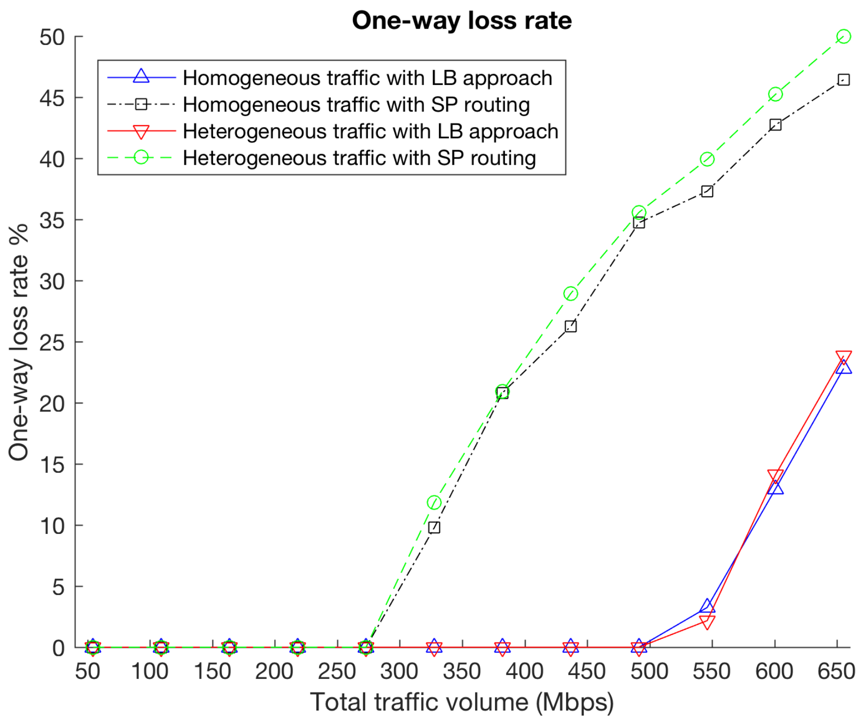


Figure 4.6: End-to-end latency and loss rate

Table 4.3: Alternative paths ($S_1 \rightarrow FW \rightarrow IDS \rightarrow S_2$ with the topology in Fig. 4.3)

	Virtual path	Physical path
Path 1	$S_1 \rightarrow FW_1 \rightarrow IDS_1 \rightarrow S_2$	$S_1 \rightarrow S_2 \rightarrow S_5 \rightarrow FW_1 \rightarrow S_5 \rightarrow S_2 \rightarrow IDS_1 \rightarrow S_2$
Path 2	$S_1 \rightarrow FW_1 \rightarrow IDS_2 \rightarrow S_2$	$S_1 \rightarrow S_2 \rightarrow S_5 \rightarrow FW_1 \rightarrow S_5 \rightarrow S_3 \rightarrow IDS_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_2$
Path 3	$S_1 \rightarrow FW_2 \rightarrow IDS_1 \rightarrow S_2$	$S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow FW_2 \rightarrow S_4 \rightarrow S_5 \rightarrow S_2 \rightarrow IDS_1 \rightarrow S_2$
Path 4	$S_1 \rightarrow FW_2 \rightarrow IDS_2 \rightarrow S_2$	$S_1 \rightarrow S_3 \rightarrow S_4 \rightarrow FW_2 \rightarrow S_4 \rightarrow S_3 \rightarrow IDS_2 \rightarrow S_3 \rightarrow S_1 \rightarrow S_2$

4.5.1 Middlebox failures

Network failures may happen on links, switches, or middleboxes. Link and switch failures are mostly studied; therefore, we only consider failures on middleboxes. We call the flows affected by the failures "affected flows," while flows not affected by the failures are called "unaffected flows." Middlebox failures belong to the transient disturbance, since one of the approaches of handling a middlebox failure is to restart it[28]. We propose a fast-recovery mechanism to handle middlebox failures by rerouting the affected flows during the restarting period. More importantly, this mechanism is realized locally, and therefore does not disturb the unaffected flows. To the best of our knowledge, this is the first work to handle failures in middlebox chain scenarios using OpenFlow.

4.5.2 Backup middlebox selection

Let's look at the example in Table 4.3, which lists all the candidate paths. These paths are saved before failures by solving the MILP, and thus we can find alternative paths immediately when there are failures on the middleboxes. For example, path 1 is chosen for routing a directed demand from S_1 to S_2 when no failures occur. If FW_1 fails, paths 3 and 4 can be the alternative paths. The challenge is to quickly check whether the alternative path has enough resources to accommodate the affected flows.

4.5.3 Rerouting strategy analysis

We next consider a network scenario in which one middlebox fails. Flows processed by this middlebox lose some functionality and need to be routed through another middlebox with equivalent functionality during the restarting period. We work on finding a rerouting mechanism to achieve higher speed and better performances toward failures without focusing on the routing path update process. The process is referred to as the network convergence process, during which the network responds to the failures and network performance decreases. Routing paths will come back to the original states when the failed middlebox resumes work after restarting. To not influence the unaffected flows, we have two approaches discussed below.

Flow modification

One of the approaches is to assign lower priority to the affected flows than the unaffected flows. A similar approach is also discussed in[108]. That is, the switch routes the flows based on priority. The affected flows can be routed only when all unaffected flows with higher priority are delivered. This approach works well to route the unaffected flows first; however, it does not take limited network resources into consideration. This may lead to packets from the affected flows being dropped due to queuing-buffer overflow.

Flow accommodation

We propose a local fast-recovery mechanism without modifying the flows. The switch considers the affected and unaffected flows equally. The affected flows compete with the unaffected flows for the link bandwidth and middlebox-processing capability. Our approach is to seek an alternative path with enough bandwidth, and a backup middlebox with enough processing capability, to accommodate the affected flows. In this way, there will be no failure effect on the unaffected flows. Once several paths are available, we will choose the one that makes the entire network as balanced as possible. We measure the network balance feature using θ and expect to choose the path with the smallest θ.

4.5.4 Traffic demand fluctuation

Our fast local rerouting approach is also applicable to the case where there are minor fluctuations of some flows. For instance, one flow with a minor increase might overwhelm the middleboxes and links on its routing path. We can seek an alternative path that can accommodate this flow. It can be solved by the local rerouting approach as middlebox-failure scenarios.

4.6 Evaluation of our fast local rerouting on a Mininet testbed

In this section, we test our local rerouting approach when there are middlebox failures. We use the same topology shown in Fig. 4.4. Experiment settings are also the same as those in section 4.2 unless otherwise specified.

4.6.1 Experiment setup

We measure our fast local rerouting approach, and find how many flows are reallocated with a new path and how much normalized throughput is increased when failures occur. Based on our experiment settings, firewalls act as the network bottleneck and can be considered as critical resources. IDSes are considered as non-critical resources since IDS resources are relatively sufficient. We also consider three other scenarios for comparisons: global rerouting flows, using the global LB approach proposed in section 4.2, when failure occurs; dropping affected flows when failure occurs; and no middlebox failures. We test four scenarios with the homogeneous traffic matrix.

4.6.2 Major observations from experiment results

When a device with critical resources fails, e.g., FW2, our fast local rerouting requires a maximum of 21.1 percent flow reallocations of those required in the global rerouting approach. When a device with non-critical resources fails, e.g., IDS2, our fast local rerouting requires a maximum of 9.1 percent flow reallocations.

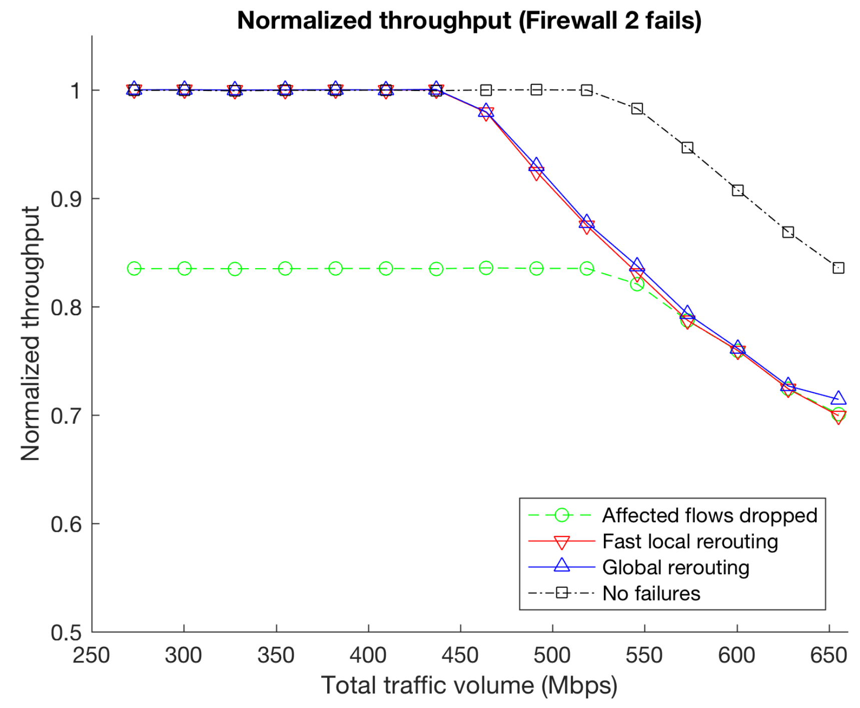
Moreover, our local rerouting approach achieves good network throughput, compared with the global rerouting approach. In Fig. 4.7, two subfigures represent the normalized throughput with a failure on FW2 and IDS2, as examples, respectively. When the failure occurs on the critical-resource device FW2, our fast local rerouting achieves almost the same normalized throughput. Results of our fast local rerouting approach reside between the global rerouting approach and the affected-flows-dropped approach, as expected. Our local rerouting increases the throughput up to 16.8 percent, compared with the affected-flows-dropped approach.

4.6.3 Further analysis on experiment results

We test all possible failures occurring on each middlebox, respectively. We obtain a group of results with the same trend on critical resource devices, namely, firewalls; and the other group of results with the same trend on non-critical resource devices, namely, IDSes. Thus, we select one failed firewall and one failed IDS, respectively, as examples in Fig. 4.7.

When the total traffic volume is greater than or equal to 546.0 Mbps, the local rerouting approach and the global LB approach cannot find better routing paths, since no resource is available to allow rerouting. When there are failures on IDSes with non-critical resources and total traffic volume is larger than 546.0 Mbps, global rerouting routing and fast local rerouting can still find alternative paths with IDS resources, shown in Fig. 4.7. When network resources are scarce (total traffic volume is 655.2 Mbps), normalized throughput of any curve converges. As the global rerouting approach achieves near-optimal results, there is a shift on the normalized throughput, that is, the global rerouting approach dealing with failures achieves a little bit higher throughput than the one with no failures.

Failures on middleboxes can also overwhelm the links. Let's examine HTTP traffic. When a firewall fails, each affected flow arrives at one of the working firewalls and selects one IDS. Thus, there are $(6 - 1) \times 5 = 25$ working candidate paths, and each affected flow chooses



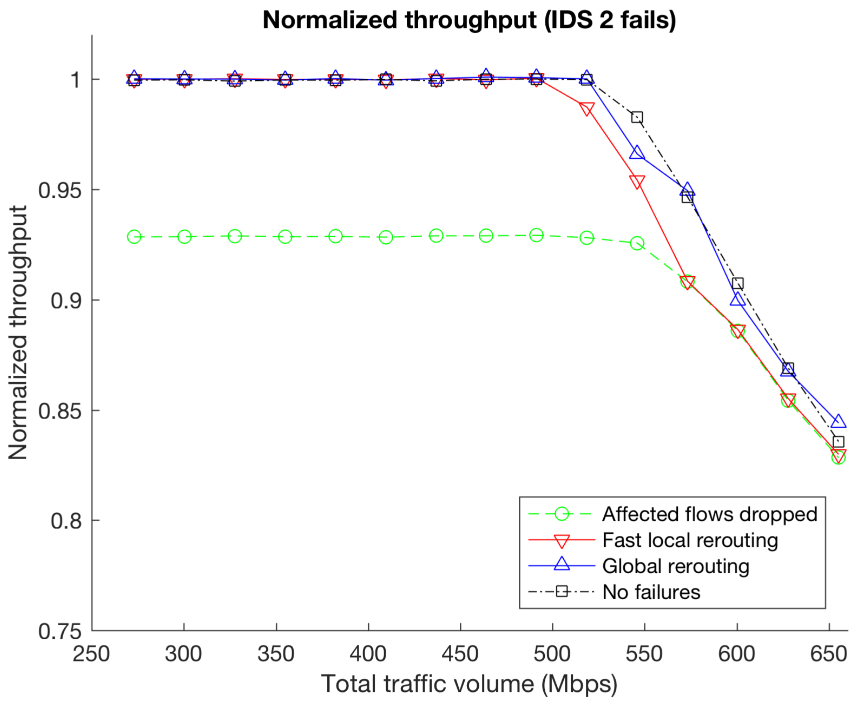


Figure 4.7: Failures on different middleboxes. Firewall 2 is connected to Switch 4, and IDS 2 is connected to Switch 5. Firewall 2 provides critical resources while IDS 2 does not.

the best one. Near-optimal routing paths can always be found. However, an IDS's failure is different from a firewall's, as the IDS is the last-type middlebox in the policy sequence. When an IDS fails, each affected flow arrives at one working IDS and then routes along the shortest path to its destination. There are fewer alternative paths ($5 - 1 = 4$); thus, some links can be more easily overloaded.

4.7 Discussion of network scalability

We discuss network scalability to verify the practicability of our approaches. First, we narrowed the choices of candidate paths to accelerate the routing path selection procedure. Then we demonstrated the upper bounds of the number of flow entries in subsection 4.2.4. The number of flow entries on each switch scales linearly as the number of distinct types of middleboxes in a middlebox chain increases. It also linearly depends on the number of demand pairs. The number of flow entries on each switch is within the switch-processing capability, since $\varnothing$ and $\#switch$ increase simultaneously and are canceled out. Furthermore, the SAN algorithm approximates the optimal solution effectively. In addition, our design of fast local rerouting indicated only affected flows need to be rerouted during the restarting period of the failed middlebox. This mechanism is realized locally, which does not disturb the working subnetwork. This is a great feature that supports network topology's scalability.

Possible limitations of scaling the network reside in the controller, which include the difficulty of synchronized communication between the controller and all switches, the controller's limited processing capability, etc. Distributed controller techniques can help reduce the burden on the controller[109]. The problem on how to exploit the benefits of the centralized SDN controller and scale the controller's processing capability remains an open question in SDN research and is beyond the scope of our work.

4.8 Contributions

In this chapter, we formulate a mixed-integer linear programming problem to achieve a network load-balancing objective in the context of sequenced-middlebox policy routing. Our global routing approach manages network resources efficiently by simplifying candidate-path selections, balancing the entire network, and using the simulated annealing algorithm. Moreover, in the case of middlebox failures, we design a fast recovery mechanism by exploiting the remaining link and middlebox resources locally. To the best of our knowledge, this is the first work to handle failures in the middlebox chain scenarios using OpenFlow. Finally, we implement proposed routing approaches on a Mininet testbed and evaluate the experiments' scalability, assessing the effectiveness of the approaches. Results of the optimization on a test topology include an increase up to 26.4 percent of the throughput, with respect to a sequenced shortest-path routing.

Our contributions are listed as follows:

1. We design an efficient routing strategy for middlebox policy enforcement.
2. We consider limited network resources (link loads, middlebox loads, and switch capabilities) and formulate a novel flow management problem.
3. We design a fast local rerouting approach to handle middlebox failures.
4. We implement branch-and-bound, simulated annealing, and greedy algorithms to make efficient routing decisions.
5. We implement our flow routing approaches on a Mininet testbed to validate its practicality.

Chapter 5

Robustness of SDN control plane

In this chapter, we evaluate and improve the robustness of the SDN control plane. First, we discuss DoS attacks and SDN vulnerabilities. Second, we build an adversary model to describe in detail how to launch DoS attacks to overwhelm the SDN controller. Third, we implement a successful distributed DoS attack on the Mininet testbed. In section 5.4, we discuss possible defense mechanisms and present our preliminary results. Finally, we summarize our contributions at the end of this chapter.

5.1 DoS attacks

5.1.1 Definition

Attackers send a large number of requests to exhaust server resources, so that servers cannot serve requests from legitimate users. This is called the DoS attack. Classic DoS attacks target services such as a web server, a DNS server, etc.[66]

5.1.2 Severe consequences

DoS attacks lead to severe problems. In 2016, major websites such as Twitter, Netflix, Spotify, Airbnb, Reddit etc., couldn't be accessed for more than two hours, because Domain Name System (DNS) provider Dyn was under DDoS attacks. Furthermore, the most powerful DDoS attack in the first half of 2016 consumed 579 gigabits per second (Gbps) of bandwidth[110].

DDoS attacks pose an immense threat to the Internet, resulting in millions of dollars in losses for companies. Attackers are constantly exploiting possible Internet vulnerabilities to launch DDoS attacks, and researchers are improving their defense mechanisms to tackle those attacks. Mirkovic et al. provided a thorough survey on the classification of existing DDoS attacks and defense mechanisms[66].

5.1.3 Mitigation of DoS attacks using SDN

As SDN emerges, it provides more possibilities to mitigate DoS attacks. For example, SDN helps acquire flow information, and then pattern matching and machine learning techniques are used to identify DDoS attacks[32;33]. Moreover, the SDN controller gains an overview of the entire network and monitors network anomalies[111;112]. In addition, SDN is able to flexibly route flows through security devices to identify attacks, as discussed in chapter 4. After identifying attacks, SDN can react toward potential threats by simply dropping malicious flows.

5.1.4 SDN vulnerabilities

In a traditional network, DoS attacks normally only affect one service. However, in an SDN network, the DoS of the controller is destructive to the entire network, since the SDN controller centrally provides services to all the flows. As today's network is a hybrid of both traditional and SDN networks, attackers first investigate which network is using SDN and then launch a controller DoS attack. Shin et al. presented an SDN scanner to test whether the targeted network is likely to be an SDN network[68]. Other works are focusing on mitigating the controller DoS attack, given an SDN network[62;67;71]. In our work, we assume an SDN network is given.

The OpenFlow controller can behave proactively or reactively. The proactive behavior means forwarding rules are installed beforehand, while the reactive one means the controller installs forwarding rules requested by switches on the fly[62]. The proactive behavior reduces communication overhead between the controller and switches. However, the reactive behavior enables more flexibility, and flow entries are installed as requested in order to save flow-table space. We are focusing on the reactive behavior, as it is more widely used in today's OpenFlow networks. When a new flow arrives at the network, the switch will ask the controller first, and then the controller will calculate the forwarding rule for this packet. This feature enables the dynamics of OpenFlow; however, it requires the controller's fast and massive calculations. Such calculations might overwhelm the controller, so that the controller provides slow responses or no responses to the requests. This will give the attackers an opportunity to launch the controller DoS attack.

5.2 Detailed adversary model

In this section, we build a controller DoS attack model from the attacker's point of view. The most successful attack is to break down the controller, consequently destroying the entire network.

5.2.1 Problem statement

In the header field of an OpenFlow flow entry, 12 different fields are aiming at providing fine-grained control and QoS guarantee, as defined in OpenFlow 1.0. Wildcard rules on the switch can help reduce the number of requests from the data plane. However, the attacker can always send numerous table-miss packets. Thus, we assume 12 fields need to be one-to-one equivalent when packets match flow entries. To launch a DoS attack, attackers can craft arbitrarily many random values to each field. For example, the destination IP address can be crafted with up to 2^{32} possibilities, though many of them are reserved IP addresses or invalid ones. Moreover, one of the IP addresses can be combined with another field, such as a destination port number, to represent another packet. The number of crafted packets grows exponentially. When a switch receives a packet, the switch looks up a matching flow entry for it. However, those crafted packets are very unlikely to match a flow entry and force the switch to send requests to the controller. Then, the controller receives excessive attack requests and fails to respond to legitimate requests. Consequently, the entire network is destroyed. This is called “controller saturation attack”.

5.2.2 Attack scenarios

Attackers expect to break down the controller with a small cost. Ideally, a single attacker can generate a huge number of table-miss packets and force the switch to inquire the controller. We define “request rate” as the number of packet-in messages sent from the switch to the controller in a unit of time. As long as the request rate is larger than the controller’s processing capability, the controller cannot serve requests from legitimate users in a timely manner. The attack scenario is shown in Fig. 5.1. The switch is a simple Open Vswitch, and the controller is a POX l2 learning controller.

Now, we briefly discuss how attackers bypass current security systems in controller saturation attacks. First, they craft a huge number of short packets, and the aggregation flow rate is very small, thus bypassing the rate-limiting method. Second, they can create packets regardless of a specific protocol to make a protocol-based defense mechanism, say proxy, ineffective. Third, they can replay some real packets captured in the past to avoid security devices’ pattern inspection.

5.3 Implementation of DoS attacks

Our emulations are conducted on the Mininet testbed. We use “hping3” command to craft packets[113]. The command is —

```
hping3 10.0.0.1 -i u200 --rand-source -d 80 -c 2000
```

The parameters are described as follows:

- `u200`: send one packet every 200 millisecond.

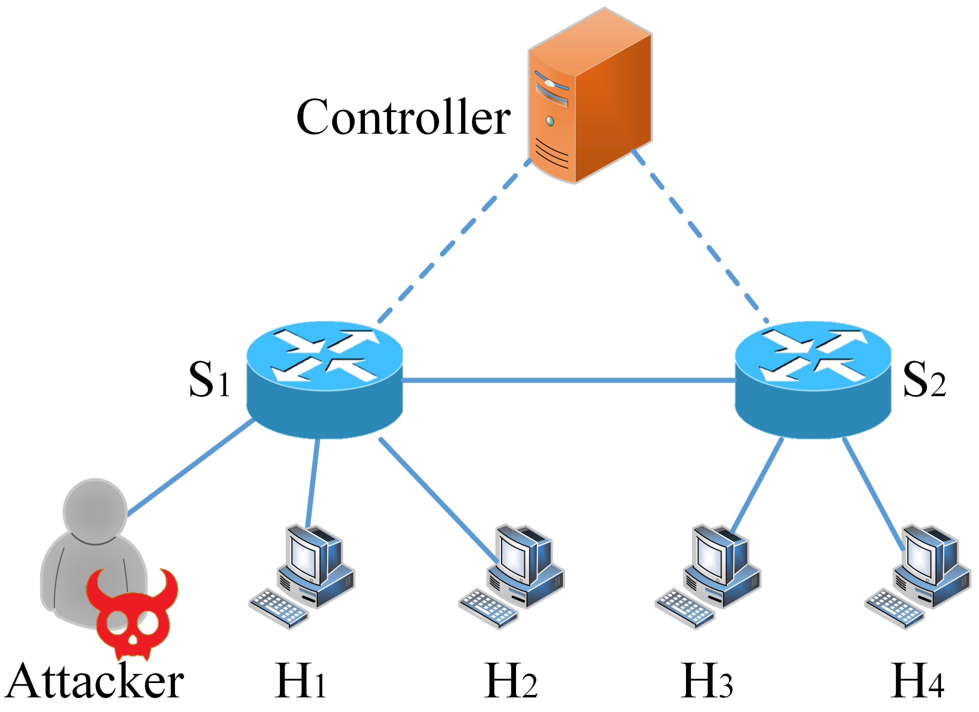


Figure 5.1: Test scenario 1: The attacker, switch S_1, and the controller are involved in the DoS attack. The attacker conducts attacks through S_1 toward the controller. Without loss of generality, we add an additional switch S_2 and four hosts to represent the test scenario. Network topologies can consist of many switches and thousands of hosts.

- `rand-source`: a random source IP address is assigned for ease of crafting table-miss packets.
- `d 80`: craft packets with the size of 80 bytes each.
- `c 2000`: send 2000 packets in total. This is used together with the "u" parameter to set the test duration.

5.3.1 Measurement of DoS effects

We notice there are no standard methods to measure DoS effects. Some measurements are used in the above-mentioned papers, for example, how fast will attacks succeed[73], monitor CPU/memory usage[62], or the delay that a legitimate user will experience[71]. We agree on measuring DoS effects based on controller availability[71], that is, sending a probe to test the delay the probe experiences. This measurement is more intuitive than monitoring CPU's usages, because a CPU's 100 percent usage indicates the controller is either best utilized or congested. Moreover, we are arguing probes should be sent at a proper rate: fast enough to get better time resolution; slow enough to avoid unnecessary burden on the controller.

5.3.2 DoS attack from a single attacker

Experiment setup

The attack scenario is shown in Fig. 5.1. The attacker, switch S_1, and the controller are involved in the DoS attack. The attacker conducts attacks through S_1 toward the controller. Without loss of generality, we add an additional switch S_2 and four hosts to represent the test scenario. Network topologies can consist of many switches and thousands of hosts. Host H_1 sends ICMP packets to H_2 in order to measure the round-trip time (RTT), which includes the controller processing time and switch S_1 processing time. When an ICMP packet arrives at the switch, the switch does not have any matching flow entry and then inquires of the controller for actions. The controller and switch S_1 are both involved during this process. If the controller or the switch is congested, the round-trip time becomes large. In our experiments, H_1 sends an ICMP packet every one second for 10 seconds. The controller instructs the ICMP packet with an action to its destination, but does not install any flow entry. This ensures each measured RTT includes both the switch and controller processing time. Otherwise, the subsequent ICMP packets will find a matching flow entry and follow its actions without asking the controller.

Experiment 1

In Fig. 5.2, the X axis represents the attacking rate increase from 500 packets per second (PPS) to 1500 PPS from the attacker. The Y axis is the RTT between H_1 and H_2. We send repeated ICMP packets with an interval of 1s for 10 seconds. Each dot in Fig. 5.2 is the average of RTTs for 100 runs. When the attacking rate is between 1,100 and 1,200, the network becomes congested.

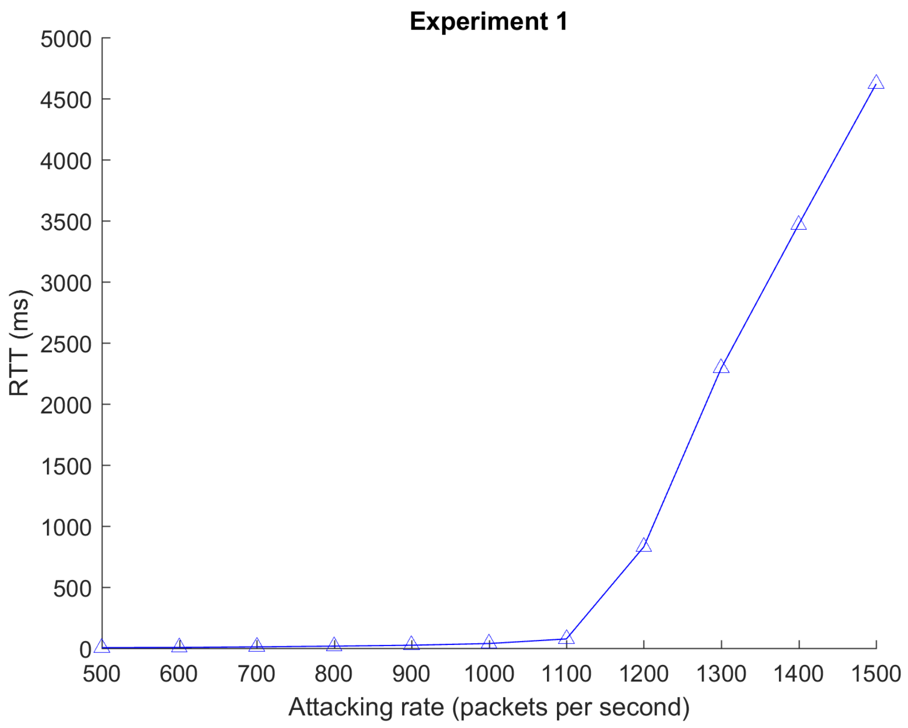


Figure 5.2: The network becomes congested when attacks are launched.

Experiment 2

We measure the total number of packets received on switch S_1 and the total number of packet-in packets received by the controller, shown in Fig. 5.3. The green curve indicates the ideal amount of packets sent from the attacker. The red curve represents total packets received on switch S_1. And the blue curve shows total packets received on the controller. When the attacking rate is larger than 1,000 PPS, total packets received by the controller remain the same. This might indicate the switch reaches its capability of generating packet-in packets, shown in Fig. 5.2, or that the controller cannot receive and process those inquires. Another experiment will be conducted to conclude.

Experiment 3

We further measure controller availability from another switch S_2 to circumvent the limitation of switch S_1. The network is indicated in Fig. 5.4. Now S_2 and its two hosts, H_3 and H_4, are used for testing controller availability. H_3 sends ICMP packets to H_4. At the

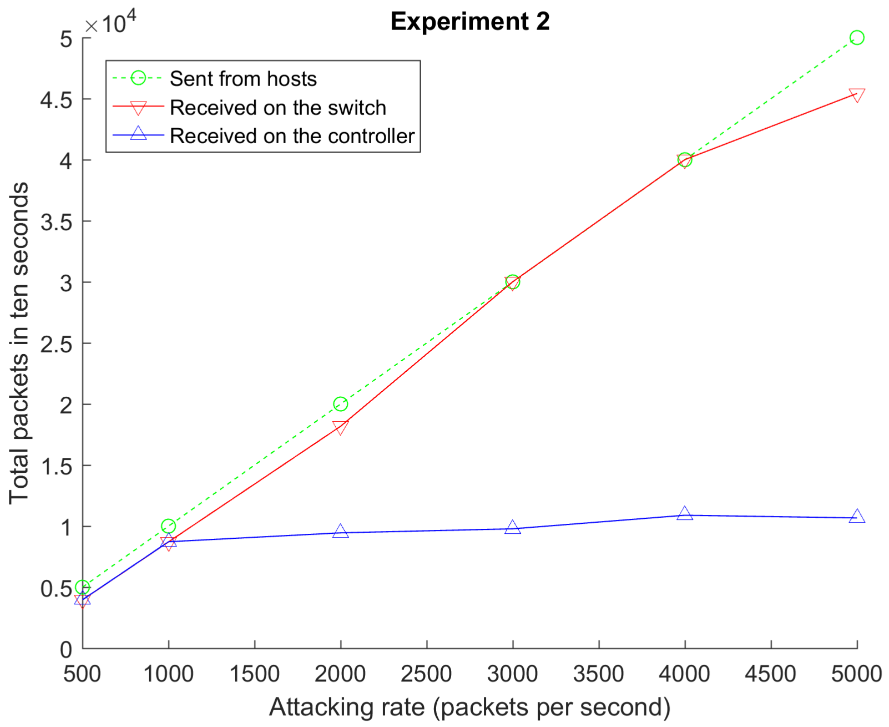


Figure 5.3: There is only one attacker in the network. The green curve indicates the ideal amount of packets sent from the attacker. The red curve represents total packets received on the switch. The blue curve shows total packets received on the controller.

beginning, we are launching a 1,000 PPS attack from a single attacker. All those packets from the attacker are expected to be sent through its connected switch S_1, and then S_1 will inquire the controller. However, we've noticed no matter how high the sending rate is, the controller can still be accessed by probe requests. More results will be presented and discussed in subsection 5.3.3.

We conclude the controller is never congested when there is only one attacker. The controller cannot be successfully overloaded, no matter how many attackers are connected with one switch. Combined with results from experiment 2, we can conclude that the switch reaches its capability of generating packet-in packets. We can also conclude the switch's low processing capability protects the controller from overload to some extent.

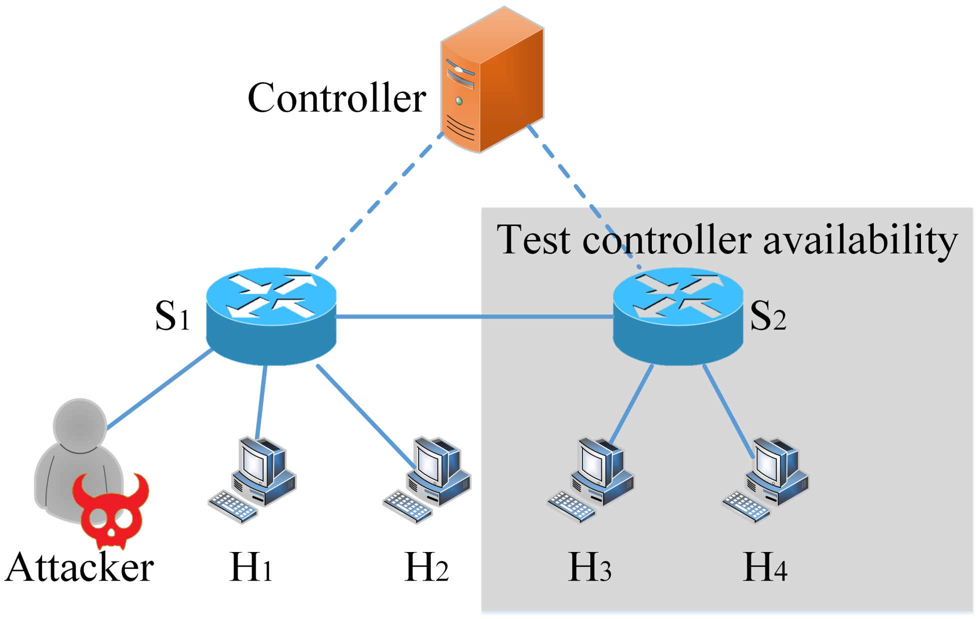


Figure 5.4: Test scenario 2: S_2, and its two hosts H_3 and H_4, only work for testing controller availability.

5.3.3 Distributed DoS attacks

Compared with DoS attacks, DDoS attackers will first compromise lots of hosts, then launch attacks toward the controller using all these compromised hosts.

Launching a DDoS attack

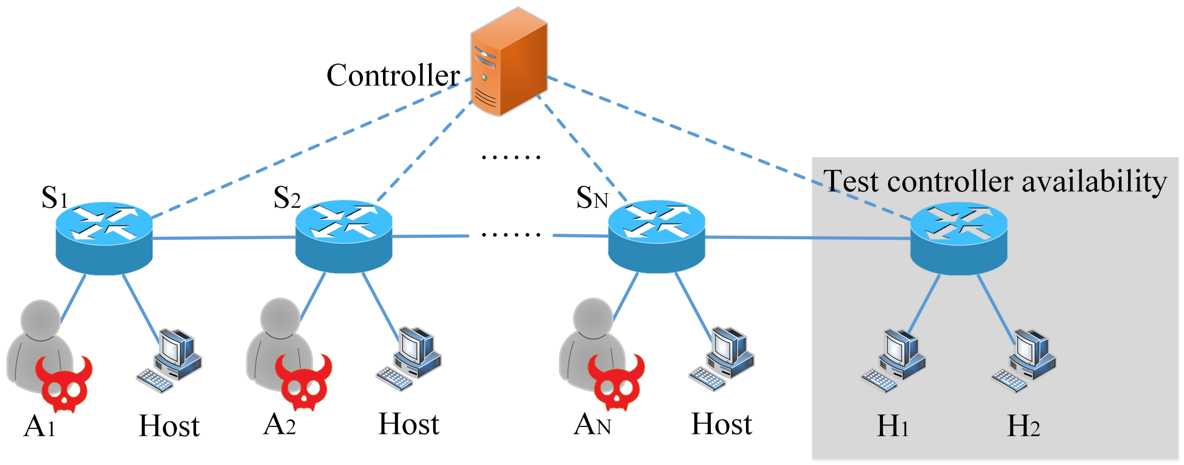


Figure 5.5: Test scenario 3: N is the number of attackers. Each attacker has to be connected with a separated switch. We measure the controller availability using a test switch, and its two hosts H_1 and H_2.

An attack scenario is indicated in Fig. 5.5. We launch DDoS attacks when there is one attacker, 10 attackers, and 20 attackers, respectively. Each attacker has to be connected with a separated switch to avoid reaching the switch's processing limitation. We measure controller availability using a test switch and two hosts, H_1 and H_2, connected to it. We measure the round-trip time (RTT) using ICMP packets sent from H_1 to H_2. Results are shown in Fig. 5.6. Generally, as the attacking rate increases, legitimate requests become more difficult to be served. Ideally, RTT should keep increasing all the way when the attacking rate is increasing. However, in the 10-attacker results, the curve becomes flat when the attacking rate is greater than 150 PPS per attacker. This is because, when the aggregation attacking rate is around 1500 PPS, the controller's CPU is overwhelmed. Compared with the green curve, we conclude as the number of attackers increases, the controller availability decreases.

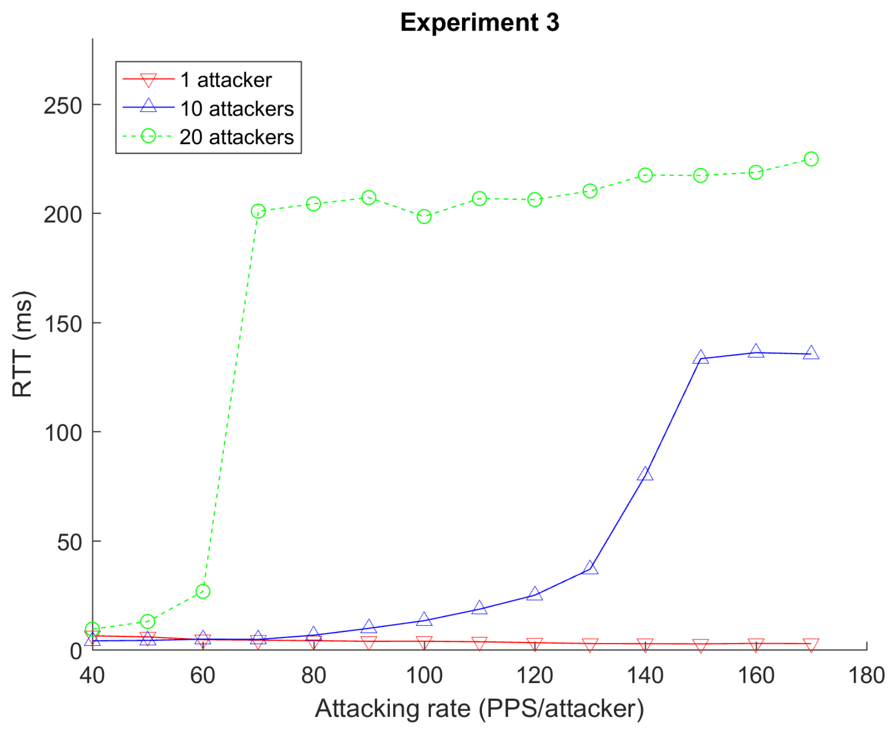


Figure 5.6: Results of RTTs represent controller availability. We launch DoS attacks when there is a single attacker, 10 attackers, and 20 attackers, respectively. Each attacker must be connected with a separated switch.

Observations

We make the following observations:

The limitation on the switch bothers us when testing controller availability from the discussion above. Such a limitation can be considered as packet-in generation ability. In Mininet, the generation ability of a software switch is about 500 packets/second[62]. Hardware switches have similar performances[114]. However, the controller processing ability is larger than the switch packet-in generation ability in reality, which is validated through our experiments. Thus, one attacker cannot break down the controller by creating massive table-miss packets. Instead, the attacker can only overload the switch software components, and the hosts connected with that switch are affected. We can also conclude the switch's low processing capability protects the controller from overload in real implementations. Thus, the simulation results, regardless of real implementation in[67], are no longer sufficient. Moreover, with DDoS attacks, we've successfully decreased the controller's availability.

5.4 Discussion of countermeasures

5.4.1 Challenges

To design defense mechanisms, a list of considerations should be noted. First, how do we measure DoS effects? On one side, we need to know when the controller is under a DDoS attack. On the other side, it helps measure effectiveness of defense mechanisms. Second, how do we differentiate legitimate requests from attack requests? We do not expect to block the requests from legitimate users accidentally. Third, defense mechanisms do not include any switch modification and are able to mitigate the general request-flooding attacks, as indicated in chapter 2. We expect to mitigate such attacks from the controller's side, which is flexible and does not include data plane's modifications[67;71]. However, the controller, which is supposed to mitigate attacks, is also a victim in the request-flooding attack scenario. This is strikingly different from current DDoS defense mechanisms, which are independent from the victim. We should guarantee the controller is still able to identify and respond to attacks when it is under attacks.

5.4.2 Proposed work and preliminary results

Differentiating legitimate users from attackers

The differentiation between legitimate users and attackers is challenging in a DoS attack. Mirkovic et al. argued that once IP spoofing is solved, many other kinds of DDoS attacks could be solved through fair-sharing among hosts[66]. Zhang et al. proposed a multi-layer fair queueing method to guarantee legitimate users are served. However, whether the controller has enough CPU and memory to manage the smart queueing is questionable. Instead, blocking mechanisms can be used to protect the network indefinitely. We propose to slow down suspicious attacks temporarily for further verification. Once attacks are verified, we block them. Simultaneously, we can protect burst requests from legitimate users and a reasonable delay will be added in exchange. The processing diagram of our proposed defense mechanism is presented in Fig. 5.7. In this diagram, *"queue"* does not refer to packet queueing buffers at the controller. Instead, it is a data structure to store packets' arriving time.

Protect the controller

For controller-side defense mechanisms[67;71], a striking challenge is whether the controller is still able to respond to attacks when it is under attacks. Consequently, our fundamental goal is to make the controller work all the time.

1. Sliding-window method

   Normally, we measure the attacking rate by counting the total number of attacking packets per time slot. Let's consider an attack scenario where the request rate is far more than the controller's processing rate. Before we get the rate of requests, the controller will be saturated no matter how small the time slot is set. Our first step is to use a sliding-window method. For example, if we want to block all users with the new request rate larger than

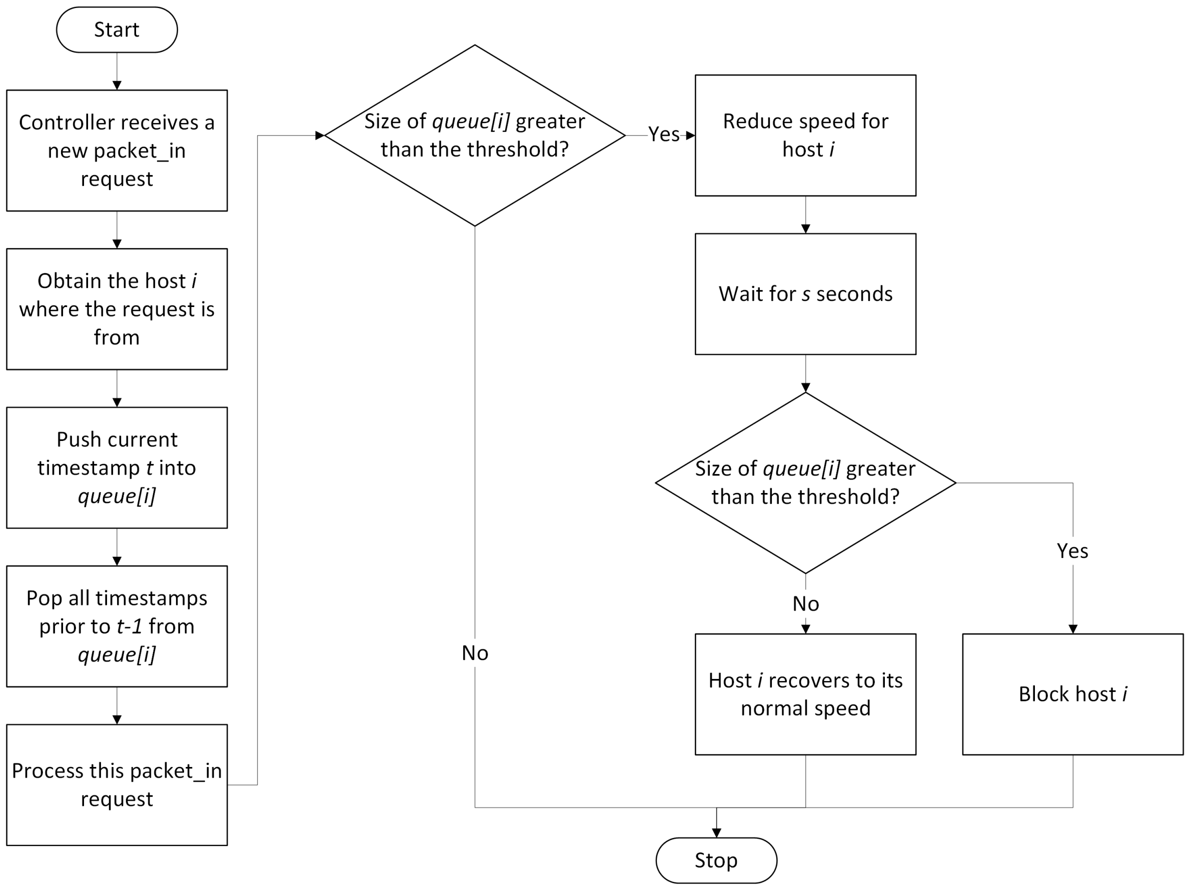

Start
Controller receives a new packet_in request
Obtain the host i where the request is from
Push current timestamp t into queue[i]
Pop all timestamps prior to t-1 from queue[i]
Process this packet_in request
Size of queue[i] greater than the threshold?
Yes
No
Reduce speed for host i
Wait for s seconds
Size of queue[i] greater than the threshold?
Yes
No
Host i recovers to its normal speed
Block host i
Stop

Figure 5.7: Processing diagram of defense mechanisms.

100 PPS. We set the sliding-window size as 100. Once the window is full within one second, we start our defense mechanism. Ideally, as the attacking rate increases, our response is faster.

2. Block suspicious hosts

   Once attacks are detected, the controller sends a traffic control command to the switch and the switch blocks the attackers. We can easily implement traffic control commands on Mininet, since Mininet depends on the Linux kernel. Moreover, this operation has also been supported by hardware switches. For example, the controller can run shell or CLI vendor extension commands on the switch in the Arista's OpenFlow implementation[115].

3. Drop coming packets

   Lots of packets may be on the way before traffic control is effective. The controller drops the packets on itself, and also adds a flow entry on the switch to match and drop all packets from suspicious users. It's quite straightforward to identify the attacker's real IP address, since its IP address is associated with its MAC address, and we can get the physical port of the attacker as well in the OpenFlow network.

4. Virtual functionality separation

   We propose to virtually separate the controller into two sections. One section is used to conduct general processing of requests. The other section conducts defense mechanisms to mitigate attacks.

Preliminary results

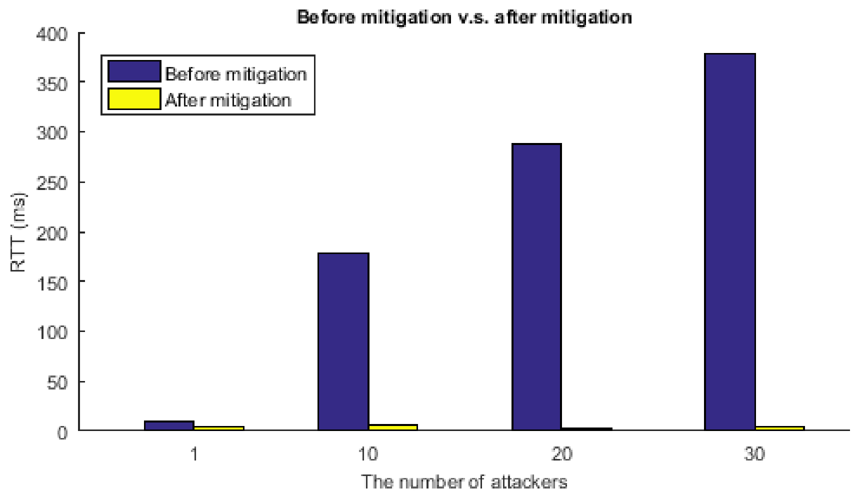


Figure 5.8: Attacking rate is 130 PPS/attacker. X-axis indicates number of attackers. Blue histogram represents controller availability under attacks. Yellow histogram represents controller availability after applying the blocking mechanism.

We modify a remote POX controller to accomplish blocking DDoS attacks using the sliding-window method as an example. We set the window size as 100. After identifying the attacks, we block the attackers. We still use the ICMP request to measure the RTTs. In Fig. 5.8, we use the 9th-second RTT result at which the network becomes stable.

This blocking mechanism works well when the network becomes stable, as shown in Fig. 5.8. However, during our tests, we've noticed the controller is still congested before the blocking mechanism is applied. Thus, this creates possibilities for attackers to circumvent our defense mechanism when the attacking rate is high enough, and a great many compromise hosts are connected with different switches. We have also implemented the method to differentiate legitimate users from attackers. Our detailed adversary model provides a fundamental and thorough analysis of SDN controller's vulnerability. Our preliminary defense mechanism can inspire and encourage more works for this topic.

5.5 Contributions

In this chapter, we study robustness of the control plane by evaluating its vulnerabilities. By implementing DoS attacks from the attackers' point of view, we demonstrate how attackers explore the control plane's vulnerabilities. In fact, implementations indicate that limited switch capability protects the controller. We thoroughly discuss possible defense mechanisms from the controller side, and present our preliminary results. This chapter represents a fundamental research step for studying the SDN control plane's vulnerabilities and inspires more research on this topic. Our contributions are listed as follows:

1. We build a detailed DDoS adversary model.

2. We perform a thorough analysis on defense mechanisms.

3. We design a scalable and lightweight DDoS attack defense mechanism to protect the controller from saturation attacks, and present preliminary results.

4. We implement and evaluate both DDoS attacks and defense mechanisms on the Mininet testbed.

Chapter 6

Conclusion and future work

6.1 Conclusion

In this dissertation, we work on enhancing network robustness. As defined in chapter 1, network robustness is the network ability to withstand possible network failures or attacks; and it ensures network availability, resilience, and adaption to changing scenarios. Our work stresses the importance of studying network robustness, which is a complex topic attracting researchers' attention. Use of SDN creates many possibilities to address network robustness issues. First, we present a robust cyber-physical network framework in chapter 3. Second, we solve the middlebox chain problem, which is a critical network robustness issue addressable with SDN in chapter 4. It's important to note that SDN networks also have vulnerabilities. Third, we demonstrate how attackers can exploit those vulnerabilities on the SDN control plane and present preliminary results on defense mechanisms in chapter 5.

In chapter 3, we propose a novel interdependent network model to pursue increased robustness of cyber-physical systems by properly allocating dependency links. We have considered one-to-multiple weighted dependency links between two individual networks, rather than simply adopting the one-to-one node dependency model. This enhanced realistic model highly complicates the design. Moreover, we formulate two optimization problems to allocate weighted dependency links under a limited budget, and a revised network flow algorithm is adapted to obtain a basic network structure in an effective way. To the author's best knowledge, this is the first work to apply the concept of limited budget to arrange dependency links from an engineering perspective. Simulations on dependency-link allocation strategies in both random networks and real network topologies are performed to validate our strategies and to identify the critical failure rate.

In chapter 4, we have explained how the middlebox policy chain can help networks enhance robustness and accomplish various network functionalities. We demonstrate how to solve critical issues in the middlebox chain problem. The middlebox policy makes the network routing problem more difficult. To solve this problem, we formulate an MILP optimization problem to allocate limited network resources and then use the simulated annealing algorithm to find a near-optimal solution. The solution of the optimization problem selects one path out of the candidate paths to be assigned to each flow for load balancing. The global load-balancing routing not only distributes network loads well, but also keeps the number of flow entries on each switch within the range of its processing capability. Furthermore, we propose a fast local rerouting approach to tackle middlebox failures. The rerouting has no effect on the working part of the network and can respond to middlebox failures quickly. Finally, our experiments on Mininet validate the efficiency and effectiveness of our approach, and attest to the feasibility of applying our approaches to real networks. On our test topology, our load-balancing (LB) approach shows an increase up to 26.4 percent on the throughput and much lower end-to-end latency, when compared with the shortest-path (SP) approach. Our fast local rerouting approach achieves similar results to the global rerouting approach: both approaches increase the throughput up to 16.8 percent, compared with the affected-flows-dropped approach.

In chapter 5, we build an adversary model to describe in detail how to launch DDoS attacks to overwhelm the SDN controller. This stresses the importance of studying the control plane's vulnerability to protect the network from a destructive disaster. We discuss DoS attacks in both traditional and SDN networks. Furthermore, we implement a successful DoS attack on the Mininet testbed to demonstrate its achievability in the real world. Finally, we highlight the challenges to mitigate such attacks and present our preliminary results on possible defense mechanisms. Results indicate a mechanism based on a sliding-window method can successfully mitigate DoS attacks. In summary, this chapter represents a fundamental research step for studying SDN control vulnerabilities and can inspire more research on this topic.

6.2 Future work

In this dissertation, we research many aspects of enhancing network robustness. However, open questions still require further investigation.

As the cascading failure is destructive to cyber-physical systems, protecting critical nodes can prevent the network from triggering cascading behaviors. For example, we can assign more resources to hubs or high-load nodes, but the effectiveness of such approaches needs to be researched. How SDN techniques can be incorporated to dynamically allocate those resources and distribute loads in interdependent-network scenarios also needs further investigation.

Moreover, middleboxes provide network functionalities to balance loads and detect network anomalies. However, we can always face the situation where middlebox resources are inadequate. Sometimes, we have to route flows quite far away to be served by the middlebox with adequate resources. This issue needs to be addressed, because it consumes more link bandwidth and produces additional delay. Network function virtualization (NFV) could be a good solution by allocating resources with more flexibility. NFV is an emerging technology aiming at using software components to gradually substitute for hardware middleboxes. Multiple functionalities can be deployed on one workstation and share resources. Therefore, resources are utilized with more flexibility and effectiveness. Associated problems should be addressed, for example, location selection and resource allocation of NFV functions.

The final issue is control plane vulnerability. In chapter 5, we have presented a thorough analysis on this issue. In the future, a comprehensive solution regarding the proposed challenges listed in section 5.4 is required to fully protect an SDN network from controller saturation attacks. Additionally, the controller must respond *rapidly* to attacks, so efficient solutions are needed to this end.

Bibliography

[1] Wikipedia. Internet traffic. [Online]: https://en.wikipedia.org/wiki/Internet_traffic, 2017.

[2] Xin Li, Haotian Wu, Caterina Scoglio, and Don Gruenbacher. Robust allocation of weighted dependency links in cyber–physical networks. *Physica A: Statistical Mechanics and its Applications*, 433:316–327, 2015.

[3] Xin Li, Haotian Wu, Don Gruenbacher, Caterina Scoglio, and Tricha Anjali. Efficient routing for middlebox policy enforcement in software-defined networking. *Computer Networks*, 110:243–252, 2016.

[4] Josh Constine. Facebook now has 2 billion monthly users... and responsibility. [Online]: https://techcrunch.com/2017/06/27/facebook-2-billion-users/, 2017.

[5] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 20162021 white paper. [Online]: https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html, 2017.

[6] Dan Meyer. Nfv and sdn only slightly offset costs needed for u.s. fiber investment. [Online]: https://www.sdxcentral.com/articles/news/nfv-and-sdn-only-slightly-offset-costs-needed-for-u-s-fiber-investment/2017/07/, 2017.

[7] Sandvine. 2016 - global internet phenomena - latin america & north america. [Online]: https://www.sandvine.com/downloads/general/global-internet-phenomena/2016/global-internet-phenomena-report-latin-america-and-north-america.pdf, 2016.

[8] Equifax. Equifax announces cybersecurity incident involving consumer information. [Online]: https://investor.equifax.com/news-and-events/news/2017/09-07-2017-213000628, 2017.

[9] Kyle York. Dyn statement on 10/21/2016 ddos attack. [Online]: https://dyn.com/blog/dyn-statement-on-10212016-ddos-attack/, 2016.

[10] Conner Forrest. Microsoft cloud cybersecurity attacks up 300says. [Online]: http://www.techrepublic.com/article/microsoft-cloud-cybersecurity-attacks-up-300-in-last-year-report-says/, 2017.

[11] Cisco. Cisco 2017 midyear cybersecurity report. [Online]: https://www.cisco.com/c/dam/m/digital/elq-cmcglobal/witb/1456403/Cisco_2017_Midyear_Cybersecurity_Report.pdf, 2017.

[12] Sue Marek. At&t cto expects sdn to reduce opex costs by 40 [Online]: https://www.sdxcentral.com/articles/news/att-cto-expects-sdn-reduce-opex-costs-40/2016/06/, 2016.

[13] Dan Meyer. Verizon sees sdn as a pillar of network streamlining, cost reduction. [Online]: https://www.sdxcentral.com/articles/news/verizon-sees-sdn-as-a-pillar-of-network-streamlining-cost-reduction/2017/05/, 2017.

[14] Ethan Banks. Thinking about sdn? here are 42 vendors that offer sdn products. [Online]: http://searchsdn.techtarget.com/news/2240212374/Thinking-about-SDN-Here-are-42-vendors-that-offer-SDN-products, 2014.

[15] Marcia Savage. 10 sdn startups on the cutting edge. [Online]: http://www.networkcomputing.com/networking/10-sdn-startups-cutting-edge/2105793252, 2015.

[16] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation

in campus networks. *ACM SIGCOMM Computer Communication Review*, 38(2):69–74, 2008.

[17] Andrea Bobbio, G Bonanni, Ester Ciancamerla, R Clemente, Alessandro Iacomini, Michele Minichino, Alessandro Scarlatti, Roberta Terruggia, and Emilio Zendri. Unavailability of critical scada communication links interconnecting a power grid and a telco network. *Reliability Engineering & System Safety*, 95(12):1345–1357, 2010.

[18] Sergey V Buldyrev, Roni Parshani, Gerald Paul, H Eugene Stanley, and Shlomo Havlin. Catastrophic cascade of failures in interdependent networks. *Nature*, 464(7291):1025–1028, 2010.

[19] Jianxi Gao, Sergey V Buldyrev, H Eugene Stanley, and Shlomo Havlin. Networks formed from interdependent networks. *Nature physics*, 8(1):40–48, 2012.

[20] Jianwei Wang, Chen Jiang, and Jianfei Qian. Robustness of interdependent networks with different link patterns against cascading failures. *Physica A: Statistical Mechanics and its Applications*, 393:535–541, 2014.

[21] Quanyan Zhu and Linda Bushnell. Networked cyber-physical systems: Interdependence, resilience and information exchange.

[22] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[23] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *Communications Magazine, IEEE*, 51(2):114–119, 2013.

[24] Chaithan Prakash, Jeongkeun Lee, Yoshio Turner, Joon-Myung Kang, Aditya Akella, Sujata Banerjee, Charles Clark, Yadi Ma, Puneet Sharma, and Ying Zhang. Pga: Using graphs to express and automatically reconcile network policies. *ACM SIGCOMM Computer Communication Review*, 45(4):29–42, 2015.

[25] Evangelos Haleplidis, Kostas Pentikousis, Spyros Denazis, J Hadi Salim, David Meyer, and Odysseas Koufopavlou. Software-defined networking (sdn): Layers and architecture terminology. Technical report, 2015.

[26] Bruno Astuto A Nunes, Marc Mendonca, Xuan-Nam Nguyen, Katia Obraczka, and Thierry Turletti. A survey of software-defined networking: Past, present, and future of programmable networks. *IEEE Communications Surveys & Tutorials*, 16(3):1617–1634, 2014.

[27] OpenFlow Switch Specification Version. 1.5. 1 (protocol version 0x06), december, 2014.

[28] Brian Carpenter and Scott Brim. Middleboxes: Taxonomy and issues. Technical report, 2002.

[29] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 27–38. ACM, 2013.

[30] Phillip A Porras, Steven Cheung, Martin W Fong, Keith Skinner, and Vinod Yegneswaran. Securing the software defined network control layer. In *NDSS*, 2015.

[31] Silvia Fichera, Laura Galluccio, Salvatore C Grancagnolo, Giacomo Morabito, and Sergio Palazzo. Operetta: An openflow-based remedy to mitigate tcp synflood attacks against web servers. *Computer Networks*, 92:89–100, 2015.

[32] Rodrigo Braga, Edjard Mota, and Alexandre Passito. Lightweight ddos flooding attack detection using nox/openflow. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 408–415. IEEE, 2010.

[33] Javed Ashraf and Seemab Latif. Handling intrusion and ddos attacks in software defined networks using machine learning techniques. In *Software Engineering Conference (NSEC), 2014 National*, pages 55–60. IEEE, 2014.

[34] Seungwon Shin and Guofei Gu. Cloudwatcher: Network security monitoring using openflow in dynamic cloud networks (or: How to provide security monitoring as a service in clouds?). In *Network Protocols (ICNP), 2012 20th IEEE International Conference on*, pages 1–6. IEEE, 2012.

[35] Ahmed Khurshid, Wenxuan Zhou, Matthew Caesar, and P Godfrey. Veriflow: Verifying network-wide invariants in real time. *ACM SIGCOMM Computer Communication Review*, 42(4):467–472, 2012.

[36] Seyed Kaveh Fayazbakhsh, Luis Chiang, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags. In *Proc. USENIX NSDI*, 2014.

[37] Vyas Sekar, Norbert Egi, Sylvia Ratnasamy, Michael K Reiter, and Guangyu Shi. Design and implementation of a consolidated middlebox architecture. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, pages 24–24. USENIX Association, 2012.

[38] Vyas Sekar, Sylvia Ratnasamy, Michael K Reiter, Norbert Egi, and Guangyu Shi. The middlebox manifesto: enabling innovation in middlebox deployment. In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 21. ACM, 2011.

[39] Ying Zhang, Neda Beheshti, Ludovic Beliveau, Gregoire Lefebvre, Ravi Manghirmalani, Ravishankar Mishra, Ritun Patneyt, Meral Shirazipour, Ramesh Subrahmaniam, Catherine Truchan, et al. Steering: A software-defined networking for inline service chaining. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[40] Dilip A Joseph, Arsalan Tavakoli, and Ion Stoica. A policy-aware switching layer for data centers. *ACM SIGCOMM Computer Communication Review*, 38(4):51–62, 2008.

[41] Seyed Kaveh Fayazbakhsh, Vyas Sekar, Minlan Yu, and Jeffrey C Mogul. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In

Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking, pages 19–24. ACM, 2013.

[42] Peng Wang, Julong Lan, Xiaohui Zhang, Yuxiang Hu, and Shuqiao Chen. Dynamic function composition for network service chain: Model and optimization. *Computer Networks*, 92:408–418, 2015.

[43] Deepak Ganesan, Ramesh Govindan, Scott Shenker, and Deborah Estrin. Highly-resilient, energy-efficient multipath routing in wireless sensor networks. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(4):11–25, 2001.

[44] Hui Yang, Lei Cheng, Jian Yuan, Jie Zhang, Yongli Zhao, and Young Lee. Multipath protection for data center services in openflow-based software defined elastic optical networks. *Optical Fiber Technology*, 23:108–115, 2015.

[45] Weidong Cui, Ion Stoica, and Randy H Katz. Backup path allocation based on a correlated link failure probability model in overlay networks. In *Network Protocols, 2002. Proceedings. 10th IEEE International Conference on*, pages 236–245. IEEE, 2002.

[46] Rahul Potharaju and Navendu Jain. Demystifying the dark side of the middle: A field study of middlebox failures in datacenters. In *Proceedings of the 2013 Conference on Internet measurement conference*, pages 9–22. ACM, 2013.

[47] Shriram Rajagopalan, Dan Williams, and Hani Jamjoom. Pico replication: A high availability framework for middleboxes. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 1. ACM, 2013.

[48] Justine Sherry, Peter Xiang Gao, Soumya Basu, Aurojit Panda, Arvind Krishnamurthy, Christian Maciocco, Maziar Manesh, João Martins, Sylvia Ratnasamy, Luigi Rizzo, et al. Rollback-recovery for middleboxes. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*, pages 227–240. ACM, 2015.

[49] Sakir Sezer, Sandra Scott-Hayward, Pushpinder Kaur Chouhan, Barbara Fraser, David Lake, Jim Finnegan, Niel Viljoen, Marc Miller, and Navneet Rao. Are we ready for sdn? implementation challenges for software-defined networks. *IEEE Communications Magazine*, 51(7):36–43, 2013.

[50] Sandra Scott-Hayward, Gemma O'Callaghan, and Sakir Sezer. Sdn security: A survey. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN For*, pages 1–7. IEEE, 2013.

[51] Rajat Kandoi and Markku Antikainen. Denial-of-service attacks in openflow sdn networks. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1322–1326. IEEE, 2015.

[52] Rowan Kloti, Vasileios Kotronis, and Paul Smith. Openflow: A security analysis. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–6. IEEE, 2013.

[53] Haopei Wang, Lei Xu, and Guofei Gu. Of-guard: A dos attack prevention extension in software-defined networks. *The Open Network Summit (ONS)*, (2014), 2014.

[54] Ehab Al-Shaer and Saeed Al-Haj. Flowchecker: Configuration analysis and verification of federated openflow infrastructures. In *Proceedings of the 3rd ACM workshop on Assurable and usable security configuration*, pages 37–44. ACM, 2010.

[55] Mark Reitblatt, Nate Foster, Jennifer Rexford, and David Walker. Consistent updates for software-defined networks: Change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, page 7. ACM, 2011.

[56] Kevin Benton, L Jean Camp, and Chris Small. Openflow vulnerability assessment. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 151–152. ACM, 2013.

[57] Philip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson, and

Guofei Gu. A security enforcement kernel for openflow networks. In *Proceedings of the first workshop on Hot topics in software defined networks*, pages 121–126. ACM, 2012.

[58] Andrzej Kamisiński and Carol Fung. Flowmon: Detecting malicious switches in software-defined networks. In *Proceedings of the 2015 Workshop on Automated Decision Making for Active Cyber Defense*, pages 39–45. ACM, 2015.

[59] Jon Matias, Jokin Garay, Alaitz Mendiola, Nerea Toledo, and Eduardo Jacob. Flownac: Flow-based network access control. In *Software Defined Networks (EWSDN), 2014 Third European Workshop on*, pages 79–84. IEEE, 2014.

[60] Wenjuan Li, Weizhi Meng, and Lam For Kwok. A survey on openflow-based software defined networks: Security challenges and countermeasures. *Journal of Network and Computer Applications*, 68:126–139, 2016.

[61] Sungmin Hong, Lei Xu, Haopei Wang, and Guofei Gu. Poisoning network visibility in software-defined networks: New attacks and countermeasures. In *NDSS*, 2015.

[62] Haopei Wang, Lei Xu, and Guofei Gu. Floodguard: A dos attack prevention extension in software-defined networks. In *Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on*, pages 239–250. IEEE, 2015.

[63] Amin Tootoonchian and Yashar Ganjali. Hyperflow: A distributed control plane for openflow. In *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, pages 3–3, 2010.

[64] Richard Wang, Dana Butnariu, Jennifer Rexford, et al. Openflow-based server load balancing gone wild. *Hot-ICE*, 11:12–12, 2011.

[65] Andrew R Curtis, Jeffrey C Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: Scaling flow management for high-performance networks. *ACM SIGCOMM Computer Communication Review*, 41(4): 254–265, 2011.

[66] Jelena Mirkovic and Peter Reiher. A taxonomy of ddos attack and ddos defense mechanisms. *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004.

[67] Lei Wei and Carol Fung. Flowranger: A request prioritizing algorithm for controller dos attacks in software defined networks. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5254–5259. IEEE, 2015.

[68] Seungwon Shin and Guofei Gu. Attacking software-defined networks: A first feasibility study. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 165–166. ACM, 2013.

[69] Sejun Song, Sungmin Hong, Xinjie Guan, Baek-Young Choi, and Changho Choi. Neod: network embedded on-line disaster management framework for software defined networking. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 492–498. IEEE, 2013.

[70] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Nishanth Devarajan. Amplified distributed denial of service attack in software defined networking. In *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*, pages 1–4. IEEE, 2016.

[71] Peng Zhang, Huanzhao Wang, Chengchen Hu, and Chuang Lin. On denial of service attacks in software defined networks. *IEEE Network*, 30(6):28–33, 2016.

[72] Seungwon Shin, Vinod Yegneswaran, Phillip Porras, and Guofei Gu. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*, pages 413–424. ACM, 2013.

[73] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: Efficiently managing switch flow in software-defined networking while effectively tackling dos attacks. In *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security*, pages 639–644. ACM, 2015.

[74] Moreno Ambrosin, Mauro Conti, Fabio De Gaspari, and Radha Poovendran. Lineswitch: tackling control plane saturation attacks in software-defined networking. *IEEE/ACM Transactions on Networking*, 25(2):1206–1219, 2017.

[75] Roni Parshani, Sergey V Buldyrev, and Shlomo Havlin. Interdependent networks: Reducing the coupling strength leads to a change from a first to second order percolation transition. *Physical review letters*, 105(4):048701, 2010.

[76] Xuqing Huang, Jianxi Gao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Robustness of interdependent networks under targeted attack. *Physical Review E*, 83 (6):065101, 2011.

[77] Fei Tan, Yongxiang Xia, Wenping Zhang, and Xinyu Jin. Cascading failures of loads in interconnected networks under intentional attack. *EPL (Europhysics Letters)*, 102 (2):28009, 2013.

[78] Di Zhou, H Eugene Stanley, Gregorio D'Agostino, and Antonio Scala. Assortativity decreases the robustness of interdependent networks. *Physical Review E*, 86(6):066103, 2012.

[79] Xuqing Huang, Shuai Shao, Huijuan Wang, Sergey V Buldyrev, H Eugene Stanley, and Shlomo Havlin. The robustness of interdependent clustered networks. *EPL (Europhysics Letters)*, 101(1):18002, 2013.

[80] Sergey V Buldyrev, Nathaniel W Shere, and Gabriel A Cwilich. Interdependent networks with identical degrees of mutually dependent nodes. *Physical Review E*, 83(1): 016112, 2011.

[81] Christian M Schneider, Nuri Yazdani, Nuno AM Araújo, Shlomo Havlin, and Hans J Herrmann. Towards designing robust coupled networks. *Scientific reports*, 3, 2013.

[82] Baharan Mirzasoleiman, Mahmoudreza Babaei, Mahdi Jalili, and MohammadAli Safari. Cascaded failures in weighted networks. *Physical Review E*, 84(4):046114, 2011.

[83] Filippo Radicchi and Alex Arenas. Abrupt transition in the structural formation of interconnected networks. *Nature Physics*, 9(11):717–720, 2013.

[84] Faryad Darabi Sahneh, Caterina Scoglio, and Piet Van Mieghem. Exact coupling threshold for structural transition in interconnected networks. *arXiv preprint arXiv:1409.6560*, 2014.

[85] Filippo Radicchi. Driving interconnected networks to supercriticality. *Physical Review X*, 4(2):021014, 2014.

[86] Jian-Wei Wang and Li-Li Rong. Robustness of the western united states power grid under edge attack strategies due to cascading failures. *Safety science*, 49(6):807–812, 2011.

[87] Jianxi Gao, SV Buldyrev, S Havlin, and HE Stanley. Robustness of a network formed by n interdependent networks with a one-to-one correspondence of dependent nodes. *Physical Review E*, 85(6):066134, 2012.

[88] Jia Shao, Sergey V Buldyrev, Shlomo Havlin, and H Eugene Stanley. Cascade of failures in coupled network systems with multiple support-dependence relations. *Physical Review E*, 83(3):036116, 2011.

[89] Saulo DS Reis, Yanqing Hu, Andrés Babino, José S Andrade Jr, Santiago Canals, Mariano Sigman, and Hernán A Makse. Avoiding catastrophic failure in correlated networks of networks. *Nature Physics*, 2014.

[90] Osman Yagan, Dajun Qian, Junshan Zhang, and Douglas Cochran. Optimal allocation of interconnecting links in cyber-physical systems: Interdependence, cascading failures, and robustness. *Parallel and Distributed Systems, IEEE Transactions on*, 23(9):1708–1720, 2012.

[91] J Martín-Hernández, H Wang, P Van Mieghem, and G D'Agostino. Algebraic connectivity of interdependent networks. *Physica A: Statistical Mechanics and its Applications*, 404:92–105, 2014.

[92] Strategic Engineering Research Group. Matlab tools for network analysis. [Online]: http://strategic.mit.edu/downloads.php?page=matlab_networks, 2011.

[93] P ERDdS and A R&WI. On random graphs i. *Publ. Math. Debrecen*, 6:290–297, 1959.

[94] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of modern physics*, 74(1):47, 2002.

[95] Jack Edmonds and Richard M Karp. Theoretical improvements in algorithmic efficiency for network flow problems. *Journal of the ACM (JACM)*, 19(2):248–264, 1972.

[96] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[97] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows: theory, algorithms, and applications. 1993.

[98] Donald Ervin Knuth. *The art of computer programming: sorting and searching*, volume 3. Pearson Education, 1998.

[99] Richard Christie. Power systems test case archive. [Online]: http://www.ee.washington.edu/research/pstca/, 1999.

[100] Caida. Macroscopic internet topology data kit (itdk). [Online]: http://www.caida.org/data/internet-topology-data-kit/, 2014.

[101] The internet topology zoo. [Online]: http://www.topology-zoo.org/dataset.html, 2011.

[102] Michal Pióro and Deepankar Medhi. *Routing, flow, and capacity design in communication and computer networks*. Elsevier, 2004.

[103] Antonio Nucci, Ashwin Sridharan, and Nina Taft. The problem of synthetically generating ip traffic matrices: initial recommendations. *ACM SIGCOMM Computer Communication Review*, 35(3):19–32, 2005.

[104] Matthew Roughan, Mikkel Thorup, and Yin Zhang. Traffic engineering with estimated traffic matrices. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 248–258. ACM, 2003.

[105] Anja Feldmann, Albert Greenberg, Carsten Lund, Nick Reingold, Jennifer Rexford, and Fred True. Deriving traffic demands for operational ip networks: Methodology and experience. In *ACM SIGCOMM Computer Communication Review*, volume 30, pages 257–270. ACM, 2000.

[106] Peter J Van Laarhoven and Emile H Aarts. *Simulated annealing: theory and applications*, volume 37. Springer Science & Business Media, 1987.

[107] Navendu Jain and Rahul Potharaju. Middlebox reliability, June 28 2012. US Patent App. 13/536,782.

[108] Dan Li, Yunfei Shang, and Congjie Chen. Software defined green data center network with exclusive routing. In *INFOCOM, 2014 Proceedings IEEE*, pages 1743–1751. IEEE, 2014.

[109] Ian F Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71:1–30, 2014.

[110] Warwick Ashford. Ddos attack size up 73 [Online]: http://www.computerweekly.com/news/450300564/DDoS-attack-size-up-73-from-2015, 2016.

[111] Mohan Dhawan, Rishabh Poddar, Kshiteej Mahajan, and Vijay Mann. Sphinx: Detecting security attacks in software-defined networks. In *NDSS*, 2015.

[112] Chu YuHunag, Tseng MinChi, Chen YaoTing, Chou YuChieh, and Chen YanRen. A novel design for future on-demand service and security. In *Communication Technology (ICCT), 2010 12th IEEE International Conference on*, pages 385–388. IEEE, 2010.

[113] Salvatore Sanfilippo. Hping3 (8)-linux man page, 2005.

[114] An Wang, Yang Guo, Fang Hao, TV Lakshman, and Songqing Chen. Scotch: Elastically scaling up sdn control-plane using vswitch based overlay. In *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, pages 403–414. ACM, 2014.

[115] Arista. Openflow. [Online]: https://www.arista.com/assets/data/pdf/user-manual/um-eos/Chapters/OpenFlow.pdf, 2017.