I havn't named my topic yet

Student: Chiu, Yen-Chun

Advisor: Prof. Lin, Po-Ching

National Chung Cheng University

Ming-Hsiung, Chiayi 621, Taiwan

shuaichiou@gmail.com

Abstract

Software defined network (SDN) is a next-generation concept that allows network administrators to manage network flows a lot easier. It is programmable, centrally managed, and flexible with topology alteration. However, these new features also lead to new security problems. Applications, controllers, Openflow switches, topology managing mechanism, there are a lot of newly-introduced vectors for us to concern about. Although protection mechanisms such as TopoGuard and FortNox have been proposed, there are still more possibilities of attacks and countermeasures in different scenarios left to be discovered. Furthermore, as SDN evolves, it will definitely bring more new security issues. In this paper, we analysis several types of topology-related attacks, enhance the efficiency of an existing switch detection method, and create an method that is able to detect topology poisoning attacks by using the standard functionality of OpenFlow switch. To enhance the efficiency, we try to reduce the number of detection packets by aggregating the flow entries. In the newly created method, time measuring technique is used to judge if there is any malicious node inside the network. We also implement packet pair to ensure the time measuring method is stable under different network condition. The result of the experiment shows that XXXXXXXXXXXXXXXXXXXXXXXX

Contents

1 Introduction **1**

2 Background and related work **5**

2.1 SDN and OpenFlow 5

2.2 Topology discovery services 7

2.2.1 Switch discovery service and Host tracking service . . . 8

2.2.2 Link discovery service 8

2.3 SDN security 10

2.3.1 General security 12

2.3.2 Controller and control channel attack 13

2.3.3 Topology poisoning attacks 14

3 Method of malicious switch detection **17**

3.1 Data collection 20

3.1.1 Extract statistical signature 21

3.1.2 Feature definition 24

3.2 System Workflow 27

4 Experiment And Evaluation 30

4.1 Environment for experiment 30

4.2 Feature analysis 32

4.2.1 Message size distribution 32

4.3 Classification results 35

4.3.1 Classification with Similar Interactions in Each Run . . 36

4.3.2 Classification with Diverse Interactions in Each Run . . 39

4.3.3 Classification with Interactions from Multiple Users . . 40

4.3.4 Early Classification 41

4.4 Practice and Limitation 45

5 Conclusion and future work 47

List of Figures

2.1 SDN architectural structure 7

2.2 Columns of a flow entry 7

2.3 LLDP packet frame structure. [21] 10

2.4 A illustration of OFDP procedure 10

2.5 SDN threat overview. 11

3.1 System workflow of classification. 28

4.1 The message size distribution of each web application from two browsers. 33

4.2 Categorization of web applications. 34

4.3 The message size distribution for Dungeon Rampage. 43

4.4 The message size distribution for Google map. 44

List of Tables

2.1 Detail of each attack. 12

3.1 The scenario of each web application. 22

3.2 The judgment of the main connection. 25

3.4 A feature of editing by Google document. 26

3.5 A feature of downloading by Dailymotion. 26

3.6 A feature of downloading by Dropbox. 27

3.3 The contents of other connections 29

4.1 The average number of connections and the standard deviation (SD) for each web application. 31

4.2 True/false positive rates of classifying the interactive connections. 36

4.3 True/false positive rates of classifying the interactive connections with additional web applications. 37

4.4 True/false positive rates of classifying downloading connections. 38

4.5 True/false positive rates of classifying all connections. 40

4.6 True/false positive rates of classifying the interactive connections for multiple users. 41

4.7 Early classification true/false positive rate in all connections. . 44

Chapter 1

Introduction

Nowadays, as the popularity of public cloud like Google cloud, Microsoft Azure, Amazon EC2 increase, we can see how cloud computing offers a new way of deploying applications and services. While people try to move everything onto the cloud, the growth of volume and complexity of data center network (DCN) is getting out of hand, and server virtualization is becoming more and more common. Consequently, networking organizations are under the increasing pressure of being more efficient, agile, and maintainable. With the traditional network approach, it just seems impossible.

In traditional network, most of the network functions heavily rely on hardware because they are implemented in network devices, which are under the control of manufacturers. As a result, the evolution of those functions will be limited. Furthermore, implementing a network-wide policy requires configuring at the device level. And similarly, tasks such as provisioning, change management, and de-provisioning are also very time-consuming, error-prone and manpower-heavy. Considering the scalability problem, it is surely a formidable job for network administrators to manage a large scale traditional network.

Software defined network (SDN) is a dynamic, program manageable, cost-effective network structure that gains great popularity among enterprises as well as academia recent years. It is designed to deal with large scale, complex, dynamic data center network existing today. It is open standards-based and vendor-neutral, which make self-developing possible. It realizes centralized control with a controller and applications on the controller. The controller is able to control all network flow by managing flow entries inside the switches. Also, components of legacy network such as a regular switch is totally compatible within a SDN network structure.

Nevertheless, new technology often comes with new security problems. In addition to switches and hosts, SDN uses controller to realize centralized control, and there are applications on the controller. There are also some new mechanisms such as topology discovery, host managing, protocols and APIs for the communication between entities. With so many new elements introduced, there will be more potential security issues that need to be taken care of in different ways.

The motivation of our work is that, although lots of works has been done to deal with all these security problems in SDN, some aspect are not included in the works. For example, in [10], Hong et al. propose attacks that poison network visibility and its countermeasure, but they did not consider the situation that switches are compromised. And to the best of our knowledge, although some protection method are proposed, there has not been an effective way to detect if there is any switch being compromised in the network. With the program-configurable trait of SDN, we believe it is possible to implement defensive solution with the aids of SDN properties. We hope we are able to set an example to inspire others and draw more attention from the community to concern more about the security issues in SDN, and ultimately resulting in a more mature SDN environment.

During the research, we study the specification of components as well as the potential threats in SDN. After discussing about those attacks, we try to cover the situations that were left by others. In this work, there are *two main goals*: the first one is to improve an existing switch detection method. The detection method is able to detect whether a flow entry works as expected. However, it can only detect one flow entry at a time, which does not seem to be efficient enough. To solve this problem, we try to aggregate the match fields to reduce the number of detection packets. The second goal is to propose a new detection method that is able to detect a fabricated link caused by LLDP packets manipulation. The main idea behind it is to measure the round trip time of LLDP packets. If an intermediate switch is compromised and manipulates the LLDP packet, it is very likely to increase the round trip time. Nevertheless, time measuring will be significantly influenced by traffic jam. Therefore, we implement *packet pair* to deal with this problem. With two packets sent at the same time, we are able to estimate the delay of traffic crowdedness by using the arrival time of two packets.

In the experiments, we found that our method XXXXXXXXXXXXXXXXXXXXXXXXXXXX
Finally, we will discuss about the future expectation of this work.

The *main contributions of this paper* are as follow:

1. Analysis several types of attacks that influence the visibility of network.

2. Discuss about counter measurements in previous works.

3. Improve an existing switch entry validation method.

4. Create an innovative fake link detection method.

5. Evaluate and discuss about our methods.

The following chapters in this thesis will be: Chapter 2 gives detail background knowledge of the used technology, discuss about possible threats and countermeasure. Chapter 3 is about our threat model and the theory of our own detection method. XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX. ———————————————— Chapter 4 contains the experimental details including setup, considerations, simulations of attacks and evaluation methods. In Chapter 5, the proposed method will be evaluated under different conditions. Finally, the conclusion and future expectation of this work will be in Chapter 6.

Chapter 2

Background and related work

2.1 SDN and OpenFlow

SDN provides better visibility into network, reduces manual intervention, improves maintainability, and decreases hardware budget. Figure 2.1 simply shows the architectural components of SDN, which are applications, network devices, Northbound and Southbound interface. SDN applications consume the resources on controller and provide network functions. Routing, Intrusion Detection System (IDS), firewall, network balancer, network monitor are all good examples of SDN applications [17]. A controller is able to determine the path of packets by manipulating flow entries in forwarding tables of the OpenFlow switches. It also maintains the abstract view of the network, including network topology, host position and the state of network resources.

OpenFlow is the first and most popular standard southbound interface. The switch that support OpenFlow is called OpenFlow switches, they are connected to one another by OpenFlow ports. Aside from physical switches, there are also software implementations of virtual switch, such as *Open vSwtich*. Typically, OpenFlow switches separate OpenFlow traffic and non-OpenFlow traffic with OpenFlow instances, they do not interfere with each other [24].

When a packet comes in from the *ingress port*, it will go through flow tables, group table, and will be processed in corresponding to the matching flow entry. Each OpenFlow table contains multiple flow entries, figure 2.2 is the columns of a flow entry. Packets will be matched with *match field* of a flow entry. When a packet matches, it will modify the action set according to its instruction. There is an entry with lowest priority that matches all fields. It is for packets that cannot match with any other flow entry. Normally it will be encapsulated and sent to the controller, the controller decides how it should be processed, and a new flow entry will be added according to the result. Some examples of the action are output to a specific port, drop and change TTL in the packet. After the end of processing pipe line, all the actions in the action set will be executed. When ports are added or removed, the content of flow tables remain unchanged, so the controller should clean up the reference of a port if a port is deleted [19].

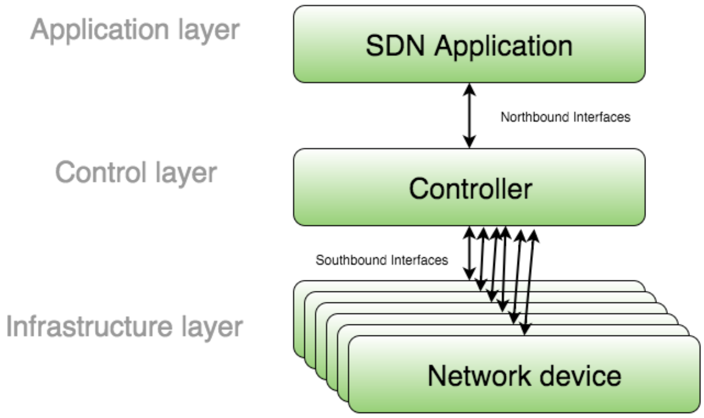


Figure 2.1: SDN architectural structure

Match Fields	Priority	Counters	Instructions	Timeouts	Cookie	Flags

Figure 2.2: Columns of a flow entry

2.2 Topology discovery services

To realize centralized control and high programmability, controllers need to maintain the visibility of the whole network. Therefore, topology discovery services play an important role in SDN. These services try to achieve auto adjustment when the topology alternate, resulting in reducing man effort significantly when it comes to network managing. The topology system includes three parts: switch discovery, host tracking, and internal link.

2.2.1 Switch discovery service and Host tracking service

Switch discovery is rather simple. When a switch try to initiate a connection with controller, the OpenFlow channel will be established, and the information of the switch will be sent to the controller and stored for future usage.

A controller maintains Host Profiles to keep track of the host location. When a Packet-Miss happens, a Packet_In message will be sent to the controller along with the packet's information and the controller will look up the Host Profiles it maintains. If the Host Profile of the host cannot be found, controller will assume a new host join the network and add the information of the host. But if there is a conflict between the Host Profile and the Packet_In message, the controller treats this as a host migration and update the location information of the Host Profile.

2.2.2 Link discovery service

When we refer to Link discovery, we mean the procedure of discovering the link between switches. Since there has not been a standard for the link discovery in OpenFlow controller, we will be using the term *OpenFLow Discovery Protocol* (OFDP) when mentioning it. Currently, although there are be some minor differences in detail, all the main stream controllers support OFDP.

OFDP leverages the Link Layer Discovery Protocol (LLDP) with subtle modification to perform topology discovery in an OpenFlow network. LLDP is originally implemented by Ethernet switch to exchange its identity and capabilities with a adjacent layer 2 peers. In legacy network, LLDP packets are sent regularly via each port of switches [21]. The information learned from LLDP packets sent by neighbor is stored by all the switches and the packets will not be forwarded after a single hop. Figure 2.3 shows the structure of LLDP Ethernet frame structure. Each LLDP Data Unit contains a sequence of type-length-value (TLV).

However, OFDP operates quite differently. The topology information is kept by the controller instead of OpenFlow switch, and an OpenFlow switch will do nothing more than forward the LLDP packet. The simplified process is shown in Figure 2.4. All switches have a pre-installed rule in their flow table, sending any LLDP packet received from any port, except the controller port, back to controller via Packet_In. Initially, controller creates an individual LLDP packet for each port on every switches via Packet_Out message. After receiving LLDP packet from controller, S1 sends it out on Port 1 and received by S2 on Port 3. With the pre-installed forwarding rule, switch S2 forwards the received LLDP packet to the controller via a Packet_In message. This Packet_In message contains meta-data such as the ID of the switch and the ingress port via which the packet was received. Thus, the controller can now infer that there exists a link between Port 1 of S1 and Port 3 of S2, and this information will be added to controller's topology database. After running this process through all ports on all the switches, controller is able to obtain all link between switches in the network. The entire discovery process is performed periodically with a typical default interval size of 5 seconds [8].

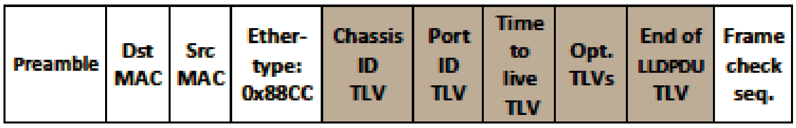


Figure 2.3: LLDP packet frame structure. [21]

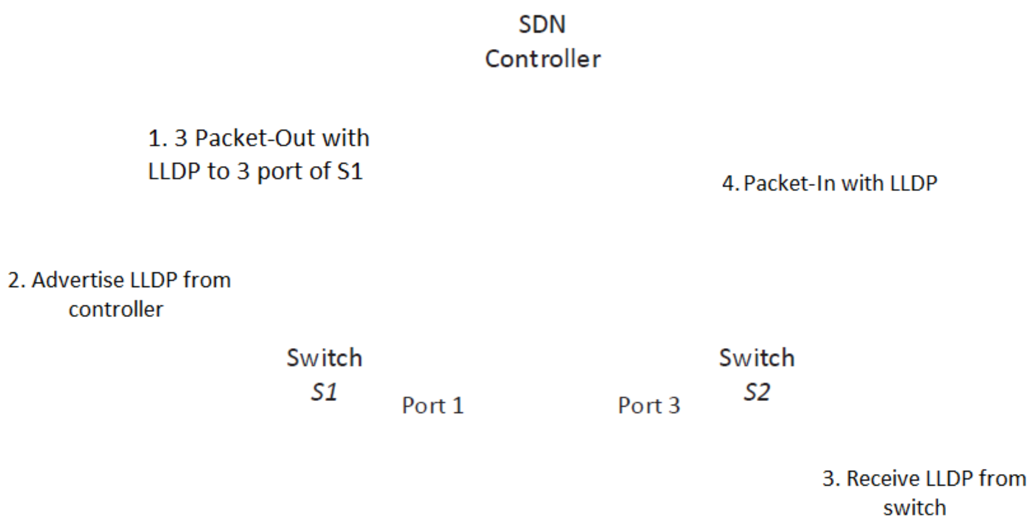


Figure 2.4: A illustration of OFDP procedure

2.3 SDN security

As SDN brings fascinating development and introduces new factors to network technology, there are a lot more vectors for us to concern about. For instances, applications inside the controller might have flaws, control channel should be secured carefully, what a malicious host inside the network can achieve should be considered, and a lot more. And don't forget, beside those new elements, attacks may also happen in non-SDN-specific ways, such as system vulnerability exploit, brute forcing, or even a physical break in etc. Figure 2.5 is a whole picture of attacks we discuss and where they might take place inside SDN, and Table 2.1 contains further description of these threats. In this chapter, we will talk about SDN security-related issues. We will focus on some of them and discuss about the attack scenarios, possible consequences and countermeasures.

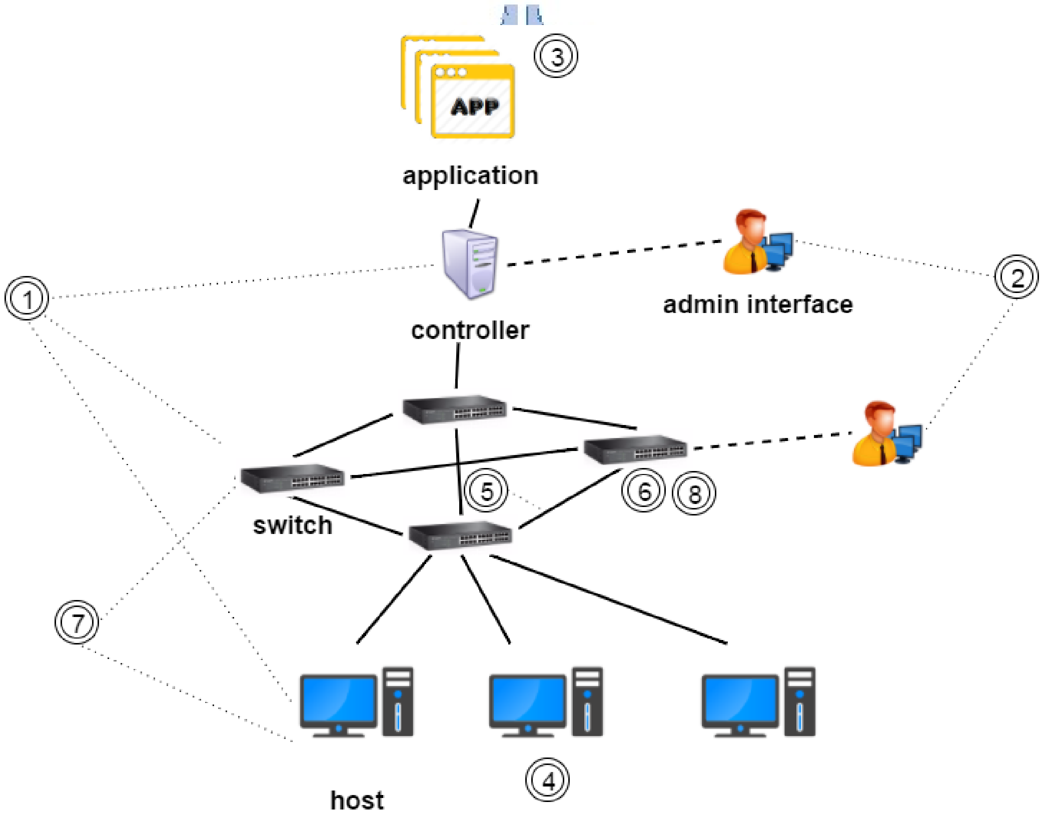


Figure 2.5: SDN threat overview.

Table 2.1: Detail of each attack.

ID	Threat name	Location	Consequences
1	System Vulnerability	all devices	Device compromised
2	Administrative Interface Compromised	all administrative interfaces	Device compromised
3	Application Vulnerability	SDN application	Malicious command to controller
4	Host Hijacking	host	Attacker receives target host's traffic
5	Link Fabrication	link between switch	Add non-existing link to controller's view
6	Flow Entry Modification	switch	Unwanted packet redirection or drop
7	Malicious Packet Controller DOS	host, switch	DOS to controller
8	Control-channel Hijacking	switch	Manipulate control traffic

2.3.1 General security

There are quite a few devices in SDN network, such as controllers, switches and hosts. All these devices possibly contain system vulnerabilities like Heartbleeding, Poodle and Shellshock [25–27] found in recent years. Moreover, one might be able to gain unauthorized access to the network physically or virtually, by using trivial methods like brute forcing the administrative interface or a physical break in. To deal with these kind of threats, all the best practices for hardening servers are applicable. Autonomic trust management technique should be used to harden these components in the network [16]. Also, Diego et al. propose replication, dynamic device association and self-healing concepts in their work to reinforce the security [12]. As to the administrative interface, organizations should implement Role-Based Access Control (RBAC) policies and event logging [13]. With all these methods, the damage should be reduced if controller compromise should happen.

2.3.2 Controller and control channel attack

Controllers are in charge of the SDN network. Therefore, it is known as the most severe threat in SDN if a controller is taken over by an attacker. By manipulating the flows, one may cause Deny Of Service (DOS) between the desired connections or Man In The Middle (MITM) with spoofed Southbound API message to redirect flow to host they have access. Besides, all the sensitive information including information of devices, network topology and all the cryptographic keys, which reside in controller, will fall into attacker's hand, resulting in damage expansion. It is also hard to detect such a threat, since with what one can do with a controller, it is possible to avoid many intrusion detection methods. The possibilities of controller being compromised including malicious applications that send unwanted command through northbound API, vulnerabilities on controller or administrative stations etc.

With a compromised switch, one may also reconfigure it to use an attacker-controlled controller than the one it should. This type of attack is called Control-channel hijacking attack. By manipulating the control traffic, an attacker is able to spoof messages to the target controller. An attacker can also perform a DOS to the controller by deliberately crafting malicious packets for controller to process slowly with a switch or host. Nevertheless, this kind of attack depends on the design and implementation of the controller heavily [5].

Phillip et al introduced FortNox, the security policy enforcement kernel, as a countermeasure to malicious applications. It implements a rule detection engine and role-based authentication to mediate all OpenFlow rule insertion requests [18]. As to securing the control channel, using out-of-band network for control traffic can reduce the chance of undesired control channel manipulation. Also, one should always use cryptographic methods such as SSL/TLS to secure the channels. However, it is not enough, vulnerabilities of SSL/TLS have been proposed and proven [15]. Kreutz et al. propose the concept of dynamic trust model and replication to maintain a trustful relationship among the connections between controller and design a secure and dependable control platform to enforce security [12].

2.3.3 Topology poisoning attacks

So far, the topology poisoning attacks have been discussed in many papers. The main idea of this type of attack is to trick controller into believing the existence of a non-existing link to host or switch by exploiting traits of topology management service. One can initiate such type of attack with either a switch or a host.

In Host Location Hijacking Attack, attacker exploits the trait of Host Tracking Service. The attacker impersonates a target host by sending spoofed packet with the host's information with PACKET_IN, and the controller will think that the target host has moved to a new location. But the truth is, the traffic of the target host is now redirected to the attacker's host [10]. Another type of host-initiated attack is called Link Fabrication Attack. The attack is caused by the fact that OpenFlow controllers accept LLDP from all switch ports, even if it is connected to a host. After an attacker receives LLDP packet with a host, he can modify specific contents like DPID or port number of LLDP packets to launch a LLDP packet injection attack. To deal with these two types of host-initiated attack, Hong et al. present TopoGuard, a new security extension to OpenFlow controller. They verify the legitimacy of Host Migration by further inspecting sign of host migration, and manage port property in order to avoid any host residing inside the LLDP propagation [10].

However, LLDP packets are passed around with the aid of switches. The Link Fabrication Attack can also initiate by compromised switches, which is not included in the scenario of TopoGuard. Bui gives three different attack scenarios of Link Fabrication Attack with compromised switches, which are two-switch tunnel attack extended two-switch tunnel attack and single-switch attack, and evaluates their consequence under different routing algorithms and network topologies [7]. This attack is caused by the lack of authentication of LLDP. However, simply adding authenticator inside LLDP packet will not help against LLDP relay attack [10]. Alharbi et al. implement HMAC based mechanism with a little modification to static secret key, which is able to detect the injection of any fabricated LLDP packets, with only an acceptable of amount of overhead added [11].

Besides link fabrication attacks, attackers can also modify the flow entries inside the flow tables of the compromised switch to perform MITM, eavesdropping or DOS attack [5]. The detection method proposed in [9] is able to detect whether a switch is forwarding the packets in an unexpected way. After selecting a flow entry as the detecting target, they install new entry on its neighbors. With the match field selected by their algorithm, they are able to let every packet that matches the new flow entry matches the target flow entry. A packet containing the match field of the new flow entry will be sent from Packet_Out to a neighbor of the target switch, forwarded to the target switch, and should be sent back to the controller. Finally, they will check if the packet comes back to the controller as expected and remain unchanged. However, this method will take a longer time to run if we want to scan through lots of flow entries. Pre-detection method to narrow down the potential target is needed.

Chapter 3

Method of malicious switch detection

A compromised switch can also drop the packet it receives and cause DOS. The detection of this kind of attack is pretty straight-forwawrd,

we will focus on the MITM types of attack

control-channel hijacking [5].

Compromised switches not only have the same capabilities as the malicious hosts, but they are also capable of performing more dynamic and severe attacks. First, a compromised switch can be used for traffic eavesdropping. Both data and control traffic passing through the compromised switch can be replicated and sent to the attacker for further processing. Furthermore, the attacker can interfere with the control traffic passing through the compromised switches to perform man-in-the-middle attacks [8]. By doing so, the attacker can act as the controller to some target switches. The attacker can also spoof control messages to the controller on behalf of the target

Below is our attack scenario assumption:

- All switches support the OpenFlow - protocol and are controlled by one OpenFlow controller - All switch follows the controller - Cloud service provider is trusted - potential flaw is unintentional z - Adversary either gain unfair resource, dos, info stealing - The switch has access to internet - Attacker can add,remove, modify entries - Attacker cant change the way of switch processing - Attacker has no physical access - The network uses OpenFlow as southbound protocol - The control channel is properly protected with TLS protocol, meaning that it provides confidentiality for the control traffic as well as mutual authentication between the controller and switches. - Non-compromised switch operate normally

Attacker sure can drop LLDP, resulting in DOS. However, its very noticeable.

Packet pair

- The dispersion in normal vs attack scenario, regardless of congestion

Manet protection

- Protection against packet dropping =¿ Keep trust schema according to neighbors

Possible solutions

- Auth the host

- Implement public-private key structure - Bring space/computation overhead, not very practical - Verify legitimacy of migration (Adopted)

- Controller receive Port_Down signal before host migration - Host become unreachable from previous location - Add light overhead to pack-in - Auth for LLDP packets - Add TLV authenticator to packet - Calculated by DPID and Port number - Fail to protect against LLDP relay

The network-wise visibility is one of a key point we can use to solve some problem, comparing to the ordinary network.

Most of the existing works assume that all of the openvswitches are trustable Scenario of switch compromised

- 0days - login password guessed - Not the main purpose of this paper. We will be discussing the hazards, detection and prevention

The capabilities of the attacker are largely determined by two factors: how TLS is used to protect control-plane communication, and whether the network uses in-band or out-of-band control channels. Additionally, some attacks are only possible if the attacker can perform control-channel hijacking instead of only being able to modify the flow-table of the compromised switch.

Packet forwarded to non-existent ports are just dropped

-==

Precise traffic classification relies on a clear understanding of web traffic. However, as application protocols and development techniques of web applications keep evolving (particularly, the HTTP/2 protocol [57] and new development techniques such as Ajax and node.js in recent years), the traffic characterization of web applications in prior studies may not reflect the state-of-the-art. Therefore, it is necessary to re-examine the traffic characteristics from typical web applications, and find out effective features for precise classification. Typical features in traffic characterization usually include application payloads, packet size, connection length and duration, packet inter-arrival time, and so on. We argued in [32] that the features may be unstable due to the variations of Internet traffic. In this work, we focus primarily on the application-level features that are not subject to the conditions in underlying networks. We study the traffic characteristics in popular web applications, including office applications (Google document), maps (Google maps), file sharing (Google drive and dropbox), video streaming (Youtube, Dailymotion, Tudou), and online games (Tetris battle and Dungeon Rampage on Facebook), in terms of the features on various browsers. We then extract the features from the main connection described in next section, so we do not need to compute the average and the standard deviation of the request/response lengths like [32].

3.1 Data collection

A web application usually involves highly user interactions between the browser and the web server. Developers often use techniques such as Ajax to improve user experiences, e.g., by actively pushing web content to the browser before the user requests it. Moreover, new application protocols, particularly SPDY (`www.chromium.org/spdy`) primarily developed at Google and HTTP/2 based on SPDY, support features to reduce the latency of loading web pages for efficient web browsing. Major browsers such as IE, Firefox and Chrome have supported SPDY and HTTP/2, and most of them have enabled the support by default at the time of this thesis writing. The web traffic from the Google services covered in this work is all sent over SPDY. However, the traffic from the other web applications (see Table 3.1) also use SPDY, except that from Facebook, Dailymotion and Tudou. Thus, the traffic studied in this work reflect the latest status in the development of web applications.

Since a browser may keep prior web content in a local cache to speed up web browsing, we use the guest mode of a browser (in which the web content in prior browsing will not be preserved) when interacting with a web application to ensure a complete set of packets during the interaction can be collected. We browse only a specific web application at a time to ensure the web traffic is all from that application, and use Wireshark (`www.wireshark.org`) to collect the packets.

In this work, we consider several typical scenarios of using web applications, and capture the web traffic from them. This work covers totally five types of 9 web applications, as listed in Table 3.1. We implore users to operate each web application on either Chrome or Firefox in the scenarios described in this table. A user is requested to interact with the applications for a period from one to two minutes as usual. We do not use Internet Explorer for several reasons. First, the browser is not supported on many operating systems, e.g., Mac OS and Android [58]. Second, the usage of IE is reported to be dropped to only 7.1% [59]. Third, Windows 10 no longer supports IE.

3.1.1 Extract statistical signature

A web application server may offer two or more services, so identifying the application based solely on the server's IP address is unreliable. For example, Google offers all its services on the same back-end server infrastructure (e.g., Google document and Google map can be provided by the same IP address 74.125.23.102 in our observation.), so users can reuse existing TCP connections to Google servers to access the other services [60]. Thus, classification with statistical features to distinguish the traffic from various web

Table 3.1: The scenario of each web application.

no.	type	application	scenario
1	Document	Googledoc	arbitrarily typing and editing
2	Map	Googlemap	typing a location name, arbitrarily browsing the map and zooming-in and zooming-out
3-5	Video	Youtube/ Tudou/ Dailymotion	typing a video name and arbitrarily moving to a specific time position during watching
6-7	File transfer	Google drive/ Dropbox	up/downloading
8-9	Gaming	Facebook : Tetris Battle/ Dungeon Rampage	arbitrary operation

applications is necessary.

There are several requirements for the training packet traces to acquire precise application-level statistical features:

1. The collected packet traces must be generated only from the targeted web application. We set the filter of Wireshark to capture the packets from or to port 80 and 443, and ask the user runs only a web application on the browser for every time of packet collection.

2. The packets in the beginning of connections should be preserved for TCP state tracking, which is necessary for packet reassembly to recover

the application-level features.

3. The collected traffic should be sufficient and diverse because it may affect the accuracy and reliability. The collected traffic must contain the packet traces from the user interactions from the targeted web application. Moreover, we performed various interactions on a same web application for every collection to ensure the diversity of packet traces. For example, we changed the frequency of typing extremely every time when we collected the traffic from Google document.

We set the target IP address and ports and follow the scenarios listed in Table 3.1 to run the web applications when collecting the training packet traces. The quantity of collected traffic is important to reflect practical user behavior precisely. We take advantage of this to observe whether diverse user behaviors will impact on the classification or the characterization. Not only the user interactions, but also the environment will influence the results. For example, some browsers support the SPDY protocol, but Mozilla Firefox did not until 2011. The difference will influence the patterns of some flows when the packets are carried over different protocols.

After collecting a new set of packet traces for a web application in the training set, we find the main connection by using the developer tool affiliated to the browser (e.g., Firebug on Firefox), which allows developers to view information about the transmitted messages. Hence we confirm which connection is the most representative of the user interactions by referring to the parameters in the HTTP messages summarized in Table 3.2. Nevertheless, many simultaneous connections may load web information in Table 3.3 to provide smooth user experiences, so we choose the longest one as the main connection. The only exception is the main connection for Tetris Battle, which transfers the elements related to the game platform instead of the game interactions. Thus, our mechanism takes advantage of this property to decide the main connection when a set of packet traces are analyzed. We can effectively segregate the noise, such as embedded advertising, with the method. This work uses the `libnids` library (`libnids.sourceforge.net`), which offers IP defragmentation, TCP stream reassembly and TCP port scan detection, to reconstruct the request-response streams of all the connections. The tool needs to track the states of TCP connections, so the beginning time of packet capturing is crucial to ensure important state transitions, e.g., 3-way handshaking, are not missing. We extract the source/destination IP addresses and source/destination ports to recognize each flow.

3.1.2 Feature definition

We assume that f_m is the main connection when a user interacts with a certain web application, and extract all n messages to be analyzed (after processed by `libnids`). Let s_i be the i-th message size in the request connection, where $i = 1 \ldots n$. We do not adopt the features in the bi-direction because including the features from the responses will introduce more ambiguity between some web applications. For example, the features generated from file download applications are sometimes similar to those from games and maps, and will decrease accuracy by 6% according to our preliminary experiment (not shown in Chapter 4). This work counts the occurrence frequency of each message size, and takes the top five frequent sizes to be represented as a vector v_i for every main connection.

Table 3.4 presents an example, in which we arbitrarily typed in a Google document on Chrome for around two minutes. Thus, the feature to char-

Table 3.2: The judgment of the main connection.

	Referred parameter	Judgment
Document	bundles	contain some arrays to store the characters we typed
Map	content-type	(1)image/png : comparing whether the preview picture and the map showed on the website are the same or not (2)application/vnd.google.octet-stream-compressible;charset=x-user-defined : this message continuously appear as we arbitrarily send actions to the map
Video	content-type	For example, audio/mp4, video/webm and video/f4v.
File transfer	content-disposition content-type	(1)checking the filename in the content-disposition match with the file we transfer (2)comparing the content-type match with the file we transfer
Game	content-type	the content-type is application/x-shockwave-flash game applications always use .swf to transfer files

acterize this interaction is (38, 34, 220, 224, 116). However, the number of different message sizes may be less than five to form a five-tuple vector. We take two ways to solve this problem. Table 3.5 is generated by watching a video clip on Dailymotion with Chrome. It describes that there is a vacancy within the vector. We can fill up the missing tuple with the mode (i.e, the most common value) of the other message sizes [61], and the vector becomes (636, 665, 658, 589, 636). Table 3.6 is generated by downloading a file from

Dropbox on Firefox. The occurrence time of all message sizes are the same in this table. We choose the first one to fill up the vacancies, and the vector is (753, 202, 162, 753, 753). The message sizes mentioned in these three tables are sorted by the occurrence times in the decreasing order. The features are stable and the testing results are showed in Section 4.3.

Table 3.4: A feature of editing by Google document.

	s_1	s_2	s_3	s_4	s_5	s_6	...	s_n
Message size (bytes)	38	34	220	224	116	199	...	m_n
Count	177	177	37	27	13	12	...	c_n

The vector is (38, 34, 220, 224, 116).

Table 3.5: A feature of downloading by Dailymotion.

	s_1	s_2	s_3	s_4	s_5
Message size (bytes)	636	665	658	589	
Count	4	3	2	1	

The vector is (636, 665, 658, 589, 636).

Table 3.6: A feature of downloading by Dropbox.

	s_1	s_2	s_3	s_4	s_5
Message size (bytes)	753	202	162		
Count	1	1	1		

The vector is (753, 202, 172, 753, 753).

3.2 System Workflow

Figure 3.1 illustrates the workflow of the classification process. A user may use either Mozilla Firefox or Google Chrome to run a web application as mentioned above. Simultaneously, the capture filter for the web application traffic is set to port 80 and 443 on the PC. The developer tool and Wireshark is employed to find the main connection that reflects user interactions most. Next, `libnids` will reassemble the captured packets to extract application-level messages. In the last step, these features extracted from the messages are fed into Weka (`http://www.cs.waikato.ac.nz/ml/weka`) for traffic classification with four machine learning methods: NBTree, Random Forest, J48graft and Naive Bayes.

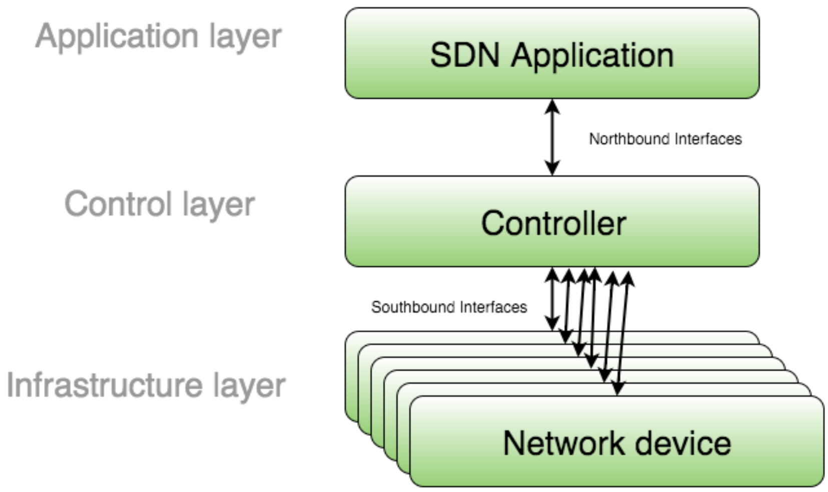


Figure 3.1: System workflow of classification.

Table 3.3: The contents of other connections

Label	Web application	Content-Type
Document	Google doc	application/json, text/html, text/javascript, font/woff, image/png, image/gif, etc.
Map	Google map	application/json, text/javascript, image/gif, image/png, etc.
Game	Dungeon Rampage	text/html, application/x-javascript, image/png, application/x-shockwave-flash, etc.
	Tetris Battle	image/gif, image/png, application/xml, application/x-shockwave-flash, audio/mpeg, etc.
File Transfer	Google Drive	text/javascript, application/json, text/css, text/html, image/gif, etc.
	Dropbox	application/x-javascript, image/gif, image/png, application/octet-stream, text/css, etc.
Video Stream	Youtube	audio/mp4, video/mp4, text/javascript, text/css, application/x-shockwave-flash, image/gif, etc.
	Dailymotion	application/x-shockwave-flash, text/css, text/xml, image/jpeg, application/x-javascript, etc.
	Tudou	application/x-shockwave-flash, image/jpeg, application/octet-stream, text/xml, image/gif, etc.

Chapter 4

Experiment And Evaluation

In this chapter, we are going to talk about the implementation of our experiment to evaluate the methods we proposed in chapter 3. First, we will list the software and config setting in detail.

4.1 Environment for experiment

VM tutorial provided by official ryu_manager 3.6 mininet 2.1.0

We analyzed the traffic from the web applications with the developer tools and Wireshark. The traffic characteristics are summarized as follows. In the office application, the characters from user typing are sent in short messages in the same connection from both Chrome and Firefox, and the length of the short massage size for each browser is either 34 bytes or 46 bytes. When we enter the website of the map application, it will display the map of the user's location according to the source IP address. Furthermore, we search

Table 4.1: The average number of connections and the standard deviation (SD) for each web application.

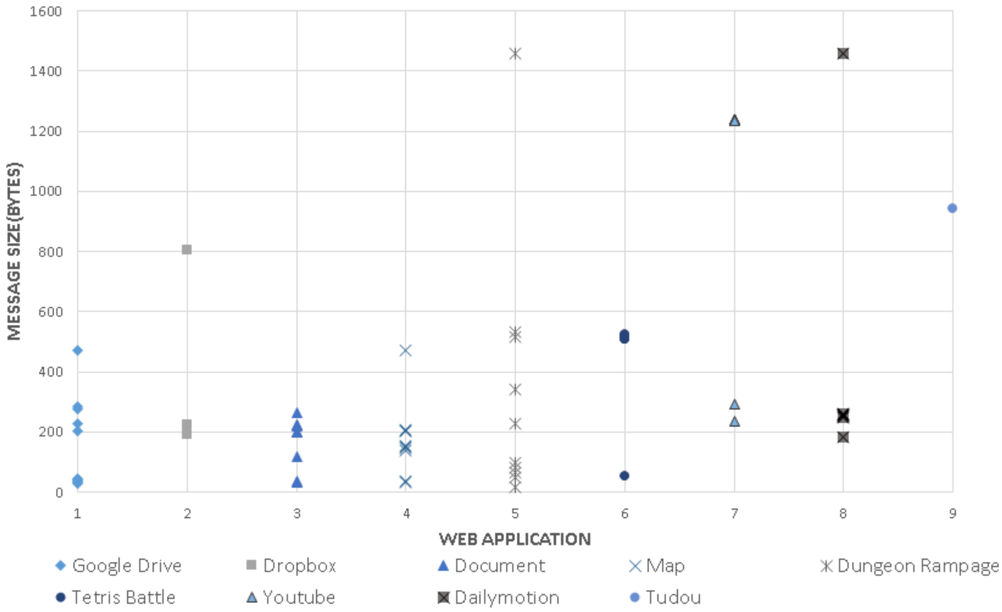
		Chrome		Firefox	
Label	Web application	Avg.	SD	Avg.	SD
Document	Google doc	37.30	4.69	33.70	2.71
Map	Google map	27.60	2.50	19.40	1.65
Game	Dungeon Rampage	125.40	10.01	108.30	7.45
	Tetris Battle	218.30	14.77	174.70	8.65
File Transfer	Google Drive	37.50	3.17	31.70	0.95
	Dropbox	68.00	11.26	61.30	3.30
Video Stream	Youtube	52.3	6.02	23.70	2.63
	Dailymotion	380	118.08	130.50	8.67
	Tudou	150.80	16.60	196.30	25.19

a new location, the application will transfer all the data associated with the region to the browser at a time, so no further packets are transferred when we slightly zoom-in and zoom-out the map. When we play Tetris Battle, we find that around 17.28% of the packets in our each collection contain the TCP PSH flags to "push" the packets from the receive buffer to the server application because this game is highly interactive. In the file sharing applications, the client keeps transferring short messages during downloading a file on Google drive. In contrast, there are few messages transferred from the client during file downloading on Dropbox. In video applications except Youtube, the connection that downloads the video clip will be reset when the user changes the time position.

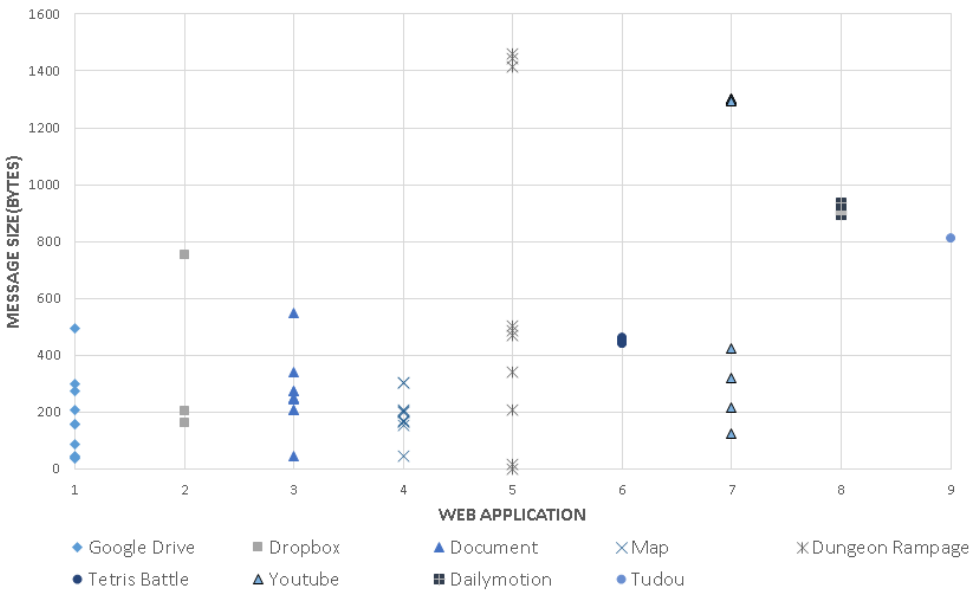
4.2 Feature analysis

4.2.1 Message size distribution

Figure 4.1 presents the top 10 frequent message sizes from the two browsers in each scenario. The messages are reassembled packet content in the main connections by `libnids`, and their sizes are ordered by the occurrence frequency. This figure demonstrates that the message sizes in different web applications vary significantly, and the occurrence frequencies of each message size also vary with the applications. The significant variations in different applications imply that the feature of message sizes should be effective for classification. Moreover, the features in the same web applications on different browsers are similar, meaning that the classification is expected to label the applications correctly even though they are running on different browsers.



(a) Chrome



(b) Firefox

Figure 4.1: The message size distribution of each web application from two browsers.

Performing the actions on similar web applications will result in different features. For example, the message size distribution of video streaming applications (Youtube, Dailymotion, Tudou) differ obviously according to Figure 4.1. However, we do not need to classify individual applications in practice because the management policies (e.g., bandwidth management or QoS) for similar applications are likely to be the same. Thus, we argue that it suffices to classify similar applications into the same category.

Figure 4.2 shows the categorization procedure. We divide the web applications into five categories: file sharing, office, map, game and video streaming. A category consists of one or multiple similar applications. If an application is correctly classified, it is also classified into the correct category. Even though the features are occasionally ambiguous between the applications in the same category (e.g., Dailymotion and Tudou), the applications can be still classified into the category of video streaming applications.

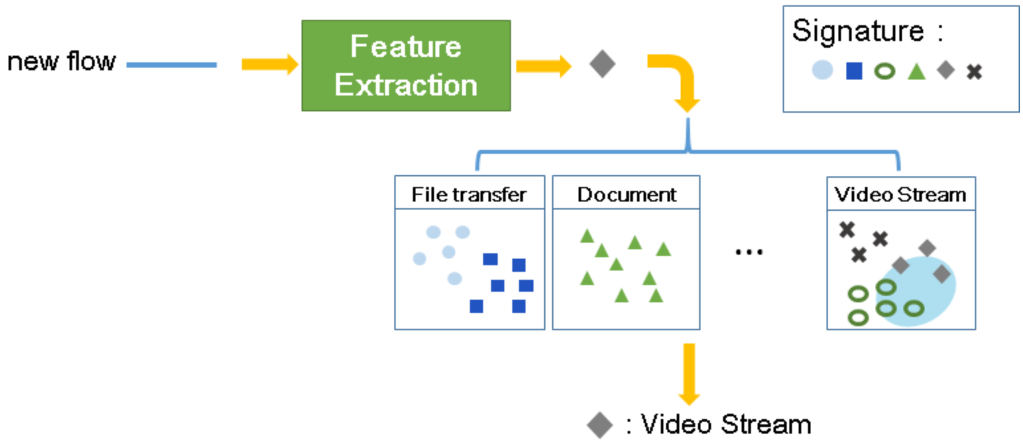


Figure 4.2: Categorization of web applications.

4.3 Classification results

After we collect sufficient data from the browsers, we employ Weka to evaluate whether the feature is effective or not. Weka supports a collection of machine learning algorithms for data mining, among which we choose NBtree, Random Forest, J48graft and Naive Bayes to compare the classification accuracy by using different algorithms. We use 10-fold cross-validation to ensure the reliability. When testing the performance, we divide the web applications into two groups, i.e., interaction applications (document, map, game) and file transfer applications (video streaming, file transfer), since we want to compare the result with the previous method, which tests the accuracy of interaction applications and file transfer applications separately. In addition, the possibility of early classification is also evaluated and it will be described in detail on Section 4.4.4.

We choose the previous study to compare with our mechanism. Lin et al. presented the method based on fine-grained classification and used the average and the standard deviation of the request/response lengths as the features for classification. The difference between our targets is that we only want to classify the kinds of web applications; however, they want to distinguish what action a user does. In the work, they subdivide the flows into search keystrokes, editing, action and download. To dislodge the unnecessary message, they artificially input the first request time or the first request bytes. Nevertheless, that method is more suitable for offline analysis. The reason we choose this study as our comparison is that the way of features extraction ,which is independent of the network condition under the application layer, is similar to our mechanism.

4.3.1 Classification with Similar Interactions in Each Run

Table 4.2 shows the classification accuracy of each web application. The accuracy of all classifiers reach 95% or above with our method, but the values still are slightly lower than previous work. There are some reasons may affect the result. For instance, our targets are different and we do not divide search keystroke messages from the flows we extract, since search packets and interaction packets may be transferred in the same connection. The true-positive rate in Google Map does not achieve 1 is because a few instances of Google Map are misidentified as those of Google document.

Table 4.2: True/false positive rates of classifying the interactive connections.

	Correctly classified (%)		Document TP/FP		Map TP/FP		Game TP/FP	
	Prior	New	Prior	New	Prior	New	Prior	New
NBtree	99.80	97.50	0.993/ 0.001	1/ 0.033	0.999/ 0.005	0.90/0	1/0	1/0
RandomForest	99.89	98.75	1/ 0.001	1/ 0.017	0.999/ 0	0.95/0	1/0	1/0
J48graft	99.28	97.50	0.993/ 0.004	1/ 0.033	0.996/ 0.019	0.90/0	0.95/0	1/0
NaiveBayes	99.49	95.00	0.987/ 0	1/ 0.067	0.996/ 0.009	0.80/0	1/0	1/0

Prior: Previous work / New : Our method

We also chose three additional web applications to disturb classification:

Google sheets (type: document), Bing map (type: map) and Dungeon Blitz (type: game). We followed the scenarios described in Table 3.1, and operated each web applications ten times on either Chrome or Firefox. We gathered the features generated from these additional applications and the original training set to verify the accuracy after the disturbance. The result is showed in Table 4.3 and the accuracy after 10-fold cross-validation can be still up to 97.14% for NBtree. In this work, we add additional applications only to the group of interaction applications to verify the accuracy because the variation of operations for file transfer applications are usually low.

Table 4.3: True/false positive rates of classifying the interactive connections with additional web applications.

	Correctly classified (%)		Document TP/FP		Map TP/FP		Game TP/FP	
	Orig.	Add.	Orig.	Add.	Orig.	Add.	Orig.	Add.
NBtree	97.50	97.14	1/0.033	0.975/0.03	0.90/0	0.925/0.01	1/0	1/0
RandomRorest	98.75	95.71	1/0.017	0.95/0.04	0.95/0	0.90/0.02	1/0	1/0
J48graft	97.50	92.14	1/0.033	0.90/0.06	0.90/0	0.85/0.04	1/0	0.983/0.013
NaiveBayes	95.00	95.71	1/0.067	0.975/0.05	0.80/0	0.875/0.01	1/0	1/0

Orig.: Original training set / Add.: With additional web applications

We also differentiate between downloading of video stream and general file transfer. We typed a video name and arbitrarily switched to multiple time positions when watching a video or just silently watched a video until the end on Youtube, Dailymotion and Tudou on the two browsers, but we up/downloaded files by Google drive and Dropbox. In summary, we watched 60 videos and transferred 40 files. After extracting the features, we used 10-fold cross-validation to evaluate the classification. It is presented in Table 4.4 that the classification accuracy can be up to 97% for all the classifiers, even better than previous work.

Table 4.4: True/false positive rates of classifying downloading connections.

	Correctly classified (%)		Video Streaming TP/FP		Files Transfer TP/FP	
	Prior	New	Prior	New	Prior	New
NBtree	92.72	97.00	0.907/ 0	0.967/ 0.025	1/ 0.093	0.975/ 0.033
RandomForest	92.33	97.00	0.902/ 0	0.967/ 0.025	1/ 0.098	0.975/ 0.033
J48graft	91.57	97.00	0.893/ 0	0.967/ 0.025	1/ 0.107	0.975/ 0.033
NaiveBayes	92.72	97.00	0.907/ 0	0.967/ 0.025	1/ 0.093	0.975/ 0.033

Prior : Previous work / New : Our method

4.3.2 Classification with Diverse Interactions in Each Run

In this subsection, instead of separating the web applications into interaction function and download function, we classify the traffic into five categories: document, map, game, video stream and file transfer. In the prior study of [32], the authors evaluated classification separately because the features for interaction function and download function are different in that study, so we also separate the evaluation as well in the last subsection to compare with the prior study fairly. However, we use the same feature, i.e., top-5 most frequent message sizes from the requests, for all the web applications, so we classify all the categories for evaluation in this subsection. We also operated each web application ten times on either Chrome or Firefox, but deliberately performed an extremely different action in the same scenario. For example, we may just continuously type or intermittently edit a document for each time we collected in document function. The classification accuracy after 10-fold cross-validation also can be up to 93.89% for Random Forest, meaning that the feature is stable and can be used for the classification with high accuracy.

Table 4.5: True/false positive rates of classifying all connections.

	Correctly classified (%)	Document TP/FP	Map TP/FP	Game TP/FP	Video Streaming TP/FP	Files Transfer TP/FP
NBtree	92.22	1/ 0.05	0.80/ 0.006	0.95/ 0.007	0.917/ 0	0.925/ 0.029
RandomForest	93.89	0.90/ 0.013	0.90/ 0.019	0.975/ 0.021	0.933/ 0.017	0.95/ 0.007
J48graft	92.22	0.95/ 0.031	0.80/ 0.013	0.975/ 0.021	0.917/ 0.008	0.925/ 0.021
NaiveBayes	92.22	0.95/ 0.025	0.95/ 0.013	0.95/ 0.029	0.917/ 0.017	0.875/ 0.014

4.3.3 Classification with Interactions from Multiple Users

To ensure the accuracy does not too depend on specific users, we invited eight users to generate the traffic for the classification. In this subsection, we compare only interactive functions because the their actions are more diverse than download functions. The users were asked to perform similar actions to those from a single user and repeated three times for each web applications on Chrome and Firefox. The total number of packet traces is 192: 48 for Google document, 48 for Google map, 48 for Dungeon Rampage and 48 for Tetris Battle.

Table 4.6 shows the accuracy for each algorithm. We also used 10-fold cross-validation for this classification. The accuracy of our method for Naive Bayes is even higher than the results in Section 4.4, and can be up to 97.92%

with random forest. Compared the result with the training set shown in last subsection, the accuracy is degraded just slightly, meaning that the feature is stable.

Table 4.6: True/false positive rates of classifying the interactive connections for multiple users.

	Correctly classified (%)		Document TP/FP		Map TP/FP		Game TP/FP	
	Prior	New	Prior	New	Prior	New	Prior	New
NBtree	96.63	96.35	0.939/ 0.025	0.938/ 0.028	0.971/ 0.044	0.938/ 0.021	0.993/ 0	0.99/0
RandomForest	98.28	97.92	0.954/ 0.009	1/ 0.007	0.99/ 0.032	0.979/ 0.014	1/0	0.969/ 0.01
J48graft	97.65	93.75	0.939/ 0.011	0.917/ 0.035	0.986/ 0.044	0.917/ 0.035	0.993/ 0.001	0.958/ 0.021
NaiveBayes	95.74	95.83	0.87/ 0.017	0.938/ 0.028	0.98/ 0.091	0.938/ 0.028	1/ 0.001	0.979/ 0

Pre: Previous work / New : Our method

4.3.4 Early Classification

The classification method in this work can be applied to manage various web applications, even though the web traffic is encrypted. The specifics of classification may vary in practice, depending on the purpose of management. If the purpose is accounting the usage of web applications in a network, it would suffice to classify the web applications offline based on the features from *the entire packet traces* that have been observed. However, if the purpose is access control, classification after extracting the features from the entire packet traces will be too late to block the traffic in time. Thus, we also evaluate the effectiveness of early classification with the features from the partial messages in the beginning of the connections. In early classification, a connection can be blocked as soon as the web application is recognized based on the features from the partial messages.

The packet traces for evaluating early classification are same as those described in the last subsection. We extracted the first k messages from each main connection ($k = 15, 20, 25, 30, 50, 100$) and used the size distribution of only these messages for the evaluation. Like the features from the entire packet traces, we also extracted the top five frequent message sizes in terms of occurrence frequency.

The previous analysis show that the true-positive rate of almost all kinds of functions are at least up to 0.90; however, the true-positive rate for the map application is decreased to 0.80 because its features are not so stable as others. We choose Dungeon Rampage on Chrome as example for stable one shown as Figure 4.3 and Google map on Chrome as example for unstable one shown as Figure 4.4. The line chart is created by entire messages within a main connection and the scatter chart is created by the first k messages in each figure.

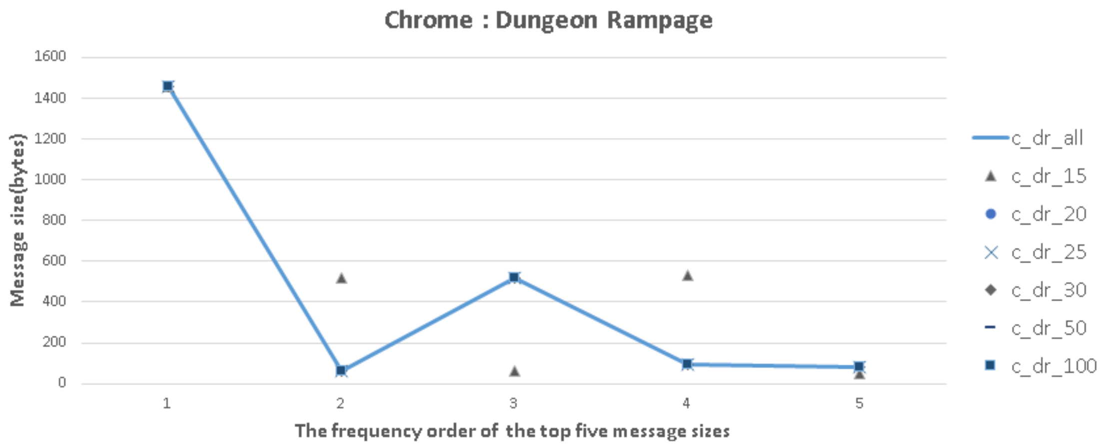


Figure 4.3: The message size distribution for Dungeon Rampage.

Figure 4.3 depicts that the scatter charts are similar with the trend of line chart when we extract more than the first 20 messages. Figure 4.4 depicts that the scatter charts are almost similar with the trend of line chart when we extract more than the first 30 messages. So we finally extracted the first 30 messages from each web applications as our feature for early classification. The classification accuracy after 10-fold cross-validation can be at least 86.67% and even can be up to 93.89% for Random Forest.

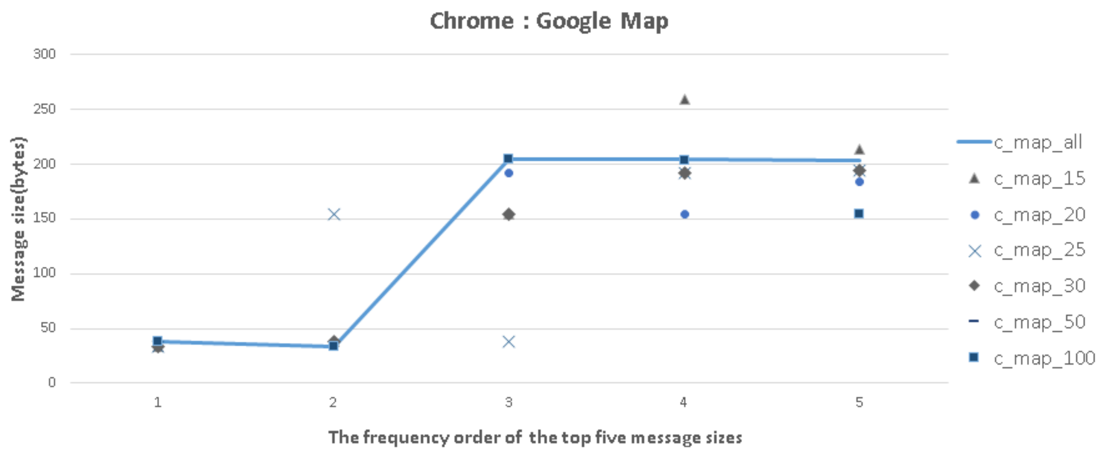


Figure 4.4: The message size distribution for Google map.

Table 4.7: Early classification true/false positive rate in all connections.

	Correctly classified (%)	Document TP/FP	Map TP/FP	Game TP/FP	Video Stream TP/FP	File Transfer TP/FP
NBtree	86.67	0.70/ 0.038	0.95/ 0.019	0.925/ 0.007	0.933/ 0.025	0.75/ 0.079
RandomForest	93.89	0.85/ 0.013	1/0	1/ 0.014	0.933/ 0.017	0.90/ 0.036
J48graft	88.33	0.85/ 0.019	1/ 0.019	0.95/ 0.029	0.85/ 0.042	0.825/ 0.043
NaveBayes	87.22	0.85/ 0.063	0.95 /0.013	0.925/ 0.021	0.95/ 0.017	0.675/ 0.043

4.4 Practice and Limitation

Even though the classification accuracy is high for extracted packet traces from real user interactions with web applications, there are still some issues that should be addressed to deploy this classification in practice.

First, although one IP address may be associated with more than one web application, as we demonstrated earlier in this work, we can still record the mapping between the IP addresses and their associated applications from earlier classification in a list to speed up classification. Since the mapping may be one-to-many, if a remote IP address can be found in the list and it is mapped to only one application, we can leverage the result of earlier classification to label the traffic to that application directly; otherwise, we can follow the procedure described in Chapter 3 to classify the web application. The result from earlier classification can at least help reduce the scope of possible labels.

Second, the traffic for each packet trace is just generated from a specific web application in this work, but in reality, we will need to analyze multiple applications at the same time and extracting the main connection is a problem. To solve this issue, we will observe the traffic density and quantity of each IP address during a period time and if the values both reach the thresholds, we will extract it as the main connection for classification. After classification, if the device that employs the design determines to block the main connection, the connections having the same pair of source/destination IP addresses as the main connection and happening around it will be blocked as well. Furthermore, users may use a web application not in the training set. Thus, after traffic classification, we will have to compute the distance between the feature vectors of the analyzed traffic and the application(s) that the analyzed traffic is supposed to be. If the distance is too long, than the analyzed traffic will be labeled as unknown.

Third, if the main connection is identified after the web application has been executed for a period of time, it may be too late to block an application for access control since the function is likely to have already completed its execution when the flows are collected and analyzed. The early classification described in Section 4.3.4 can address this issue. Moreover, the accuracy may be decreased if the execution time of an web application function is too short to extract meaningful feature.

Chapter 5

Conclusion and future work

We can only try to anticipate what the attackers may try to target with SDNs. The deployments are new, the protocols are new, the controller software is new, and the history of past SDN attacks is unknown. Based on the SDN architecture, we can predict where an attacker may be likely to strike. If we put ourselves in the attackers shoes, we might be able to spot a weakness to exploit. Then we can harden that weakness ahead of time.

Before an organization embarks on an SDN deployment project, they should consider how they will secure the system during the early design stage. Dont leave security until the final clean-up phase. If an organization waits until it is working, then hardening the northbound and southbound control messages may cause service-affecting problems. Like most things, setting it up right from the start will save organizations many problems down the road.

Bibliography

[1] Adnan Nadeem and Michael P. Howarth, "A Survey of MANET Intrusion Detection & Prevention Approaches for Network Layer Attacks,", In Proc. of IEEE COMMUNICATIONS SURVEYS & TUTORIALS, VOL. 15, NO. 4, FOURTH QUARTER 2013.

[2] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martn Casado, Nick McKeown and Scott Shenker "NOX: Towards an Operating System for Networks", ACM SIGCOMM Computer Communication Review 38.3 (2008): 105-110.

[3] EugeneNg, ZhengCai AlanL Cox TS., "Maestro: Balancing fairness, latency and throughput in the openflow control plane", Tech. rep., Rice University, 2011.

[4] Anurag Khandelwal, Navendu Jain and Seny Kamara "Attacking Data Center Networks from the Inside",

[5] Markku Antikainen, Tuomas Aura, and Mikko Srel, "Spook in Your Network: Attacking an SDN with a Compromised OpenFlow Switch", In proc. of Springer International Publishing Switzerland 2014.

[6] T.V. Lakshman, T. Nandagopal, R. Ramjee, K. Sabnani and T. Woo "The softrouter architecture", In Proc. of ACM SIGCOMM Workshop on Hot Topics in Networking. Vol. 2004. 2004.

[7] TIEN THANH BUI, "Analysis of Topology Poisoning Attacks in Software-Defined Networking", Degree project in security and mobile computing, Second Level Stockholm, Sweden 2015.

[8] Farzaneh Pakzad, Marius Portmann, Wee Lum Tan and Jadwiga Indulska, "Efficient Topology Discovery in Software Defined Networks", In Signal Processing and Communication Systems (ICSPCS), 2014 8th International Conference on, pp. 1-8. IEEE, 2014.

[9] Po-Wen Chi, Chien-Ting Kuo, Jing-Wei Guo and Chin-Laung Lei, "How to Detect a Compromised SDN Switch", In Network Softwarization (NetSoft), 2015 1st IEEE Conference on (pp. 1-6). IEEE.

[10] Sungmin Hong, Lei Xu, Haopei Wang and Guofei Gu, "Poisoning Network Visibility in Software-Defined Networks: New Attacks and Countermeasures", In NDSS. 2015.

[11] Alharbi, Talal, Marius Portmann, and Farzaneh Pakzad, "The (In) Security of Topology Discovery in Software Defined Networks.", In Proc. of Local Computer Networks (LCN), 2015 IEEE 40th Conference on. IEEE, 2015.

[12] Kreutz, Diego, Fernando Ramos, and Paulo Verissimo, "Towards secure and dependable software-defined networks", In Proc. of the second ACM SIGCOMM workshop on Hot topics in software defined networking. ACM, 2013.

[13] Ferraiolo, David F., and D. Richard Kuhn., “Role-based access controls”, arXiv preprint arXiv:0903.2171 (2009).

[14] Hu, Ningning, and Peter Steenkiste, “Estimating available bandwidth using packet pair probing”, No. CMU-CS-02-166. CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.

[15] Ralph Holz, Thomas Riedmaier, Nils Kammenhuber, Georg Carle, “X. 509 Forensics: Detecting and Localising the SSL/TLS Men-in-the-middle” , In Computer securityesorics 2012 (pp. 217-234). Springer Berlin Heidelberg.

[16] Yan, Zheng, and Christian Prehofer, “Autonomic trust management for a component-based software system”, Dependable and Secure Computing, IEEE Transactions on 8.6 (2011)

[17] Shin, S., Porras, P. A., Yegneswaran, V., Fong, M. W., Gu, G., and Tyson, M., “FRESCO: Modular Composable Security Services for Software-Defined Networks”, in NDSS 2013.

[18] Phillip Porras, Seungwon Shin, Vinod Yegneswaran, Martin Fong, Mabry Tyson and Guofei Gu, “A Security Enforcement Kernel for OpenFLow Networks”, In Proc. of the first workshop on Hot topics in software defined networks. ACM, 2012.

[19] `https://www.opennetworking.org/images/stories/downloads/sdn-resources/onfspecifications/openflow/openflow-switch-v1.5.1.pdf`

[20] `http://groups.geni.net/geni/wiki/OpenFlowDiscoveryProtocol`

[21] `https://wiki.wireshark.org/LinkLayerDiscoveryProtocol`

[22] http://www.networkworld.com/article/2840273/sdn/sdn-security-attack-vectors-and-sdn-hardening.html

[23] http://www.wipro.com/documents/insights/the-thinking-network.pdf

[24] http://h10032.www1.hp.com/ctg/Manual/c04495114

[25] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-6271

[26] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-3566

[27] https://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-0160

[28] http://usenix.org/legacy/publications/library/proceedings/usits01/full_papers/lai/lai_html/node2.html ———————-

[29] T. T. Nguyen and G. Armitage, "A Survey Of Techniques for Internet Traffic Classification Using Machine Learning," In Proc. of IEEE Comm. Surveys Tutorials, vol. 10, no. 4, pp. 56-76, Oct.-Dec.2008.

[30] J. S. Park, S. H. Yoon and M. S. Kim, "Performance Improvement of Payload Signature-based Traffic Classification System Using Application Traffic Temporal Locality," In Proc. of the 15th Network Operations and Management Symposium (APNOMS), Sept. 2013.

[31] M. Roughan, S. Sen, O. Spatscheck and N. Duffield, "Class-of-servicemapping for QoS: a statistical signature-based approach to IP traffic classification," In Proc. of ACM SIGCOMM Conference on Internet Measurement, Oct. 2004.

[32] P. C. Lin, S. Y. Chen and C. H. Lin, "Towards Fine-grained Traffic Classification for Web Applications," In Proc. of Australasian Telecommunication Networks and Applications Conference (ATNAC), Nov. 2014.

[33] P. Schneider, "TCP/IP traffic Classification Based on port numbers," Diploma Thesis, Division of Applied Science, Harvard University, 29 Oxford Street, Cambridge, MA 02138, USA, 1-6

[34] Y. Xue, D. Wang and L. Zhang, "Traffic Classification: Issues and Challenges," In Proc. of the International Conference on Computing, Networking and Communications (ICNC), Jan. 2013.

[35] http://www.ntop.org/products/deep-packet-inspection/ndpi/

[36] http://l7-filter.clearos.com/

[37] B. Hullar, S. Laki and A. Gyorgy, "Efficient Methods for Early Protocol Identification," IEEE Journal on Selected Areas in Communications, vol. 32, issue 10, Oct. 2014.

[38] L. Bernaille, R. Teixeira and K. Salamatian, "Early application identification," In Proc. of the Conference on Future Networking Technologies, 2006.

[39] H. M. An, M. S. Kim and J. H. Ham, "Application traffic classification using statistic signature," In Proc. of the 15th Asia-Pacific Network Operations and Management Symposium (APNOMS), Sept. 2013.

[40] B. Schmidt, D. Kountanis and A. Al-Fuqaha, "Artificial Immune System Inspired Algorithm for Flow-Based Internet Traffic Classification," In Proc. of the IEEE 6th International Conference on Cloud Computing Technology and Science (CloudCom), Dec. 2014.

[41] S. Tapaswi and A. S. Gupta, "Flow-Based P2P Network Traffic Classification Using Machine Learning," In Proc. of the International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), Oct. 2013.

[42] F .Dehghani, N .Movahhedinia, M. R. Khayyambashi and S. Kianian, "Real-Time Traffic Classification Based on Statistical and Payload Content Features," In Proc. of the 2nd International Workshop on Intelligent Systems and Applications (ISA), May 2010.

[43] S. Huang, K. Chen, C. Liu, A. Liang and H. Guan, "A statistical-feature-based approach to internet traffic classification using Machine Learning," In Proc. of the International Conference on Ultra Modern Telecommunications and Workshops (ICUMT), Oct. 2009.

[44] M. Kohara, T. Hori, K. Sakurai, H. Lee and J. C. Ryou, "Flow Traffic Classification with Support Vector Machine by Using Payload Length," In Proc. of the 2nd International Conference on Computer Science and its Applications (CSA), Dec. 2009.

[45] M. Roughan, S. Sen, O. Spatscheck and N. Duffield, "Class-of-Service Mapping for Qos: A Statistical Signature-based Approach to IP Traffic Classification," In Proc. of the Internet Measurement Conference (IMC), Oct. 2004.

[46] J. Zhang, C. Chen, Y. Xiang and W. Zhou, "Classification of Correlated Internet Traffic Flows," In Proc. of the 11th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), June 2012.

[47] S. Valenti and D. Rossi, "Fine-grained behavioral classification in the core: the issue of flow sampling," In Proc. of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC), July 2011.

[48] B. Park, J. W. Hong and Y. J. Won, "Toward Fine-Grained Traffic Classification," IEEE Communication Magazine, vol. 49, issue. 7, pp. 104-111, July 2011.

[49] B. Augustin and A. Mellouk, "On Traffic Patterns of HTTP Applications," In Proc. of the IEEE Global Communications Conference (GLOBECOM), Dec. 2011.

[50] R. Archibald, Y. Liu, C. Corbett and D. Ghosal, "Disambiguating HTTP: Classifying web Applications," In Proc. of the 7th International Wireless Communications and Mobile Computing Conference (IWCMC), July 2011.

[51] P. Casas and P. Fiadino, "Mini-IPC: A minimalist approach for HTTP traffic classification using IP addresses," In Proc. of the 9th International Wireless Communications and Mobile Computing Conference (IWCMC), July 2013.

[52] `http://libnids.sourceforge.net`.

[53] M. Lee, R. R. Kompella and S. Singh, "Ajaxtracker: active measurement system for high-fidelity characterization of Ajax applications," In Proc. of the 2010 USENIX conference on Web application development (WebApps '10), 2010.

[54] S. Veres and D. Ionescu, "Measurement-based traffic characterization for Web 2.0 applications," In Proc. of the IEEE Instrumentation and Measurement Technology Conference (I2MTC), May 2009.

[55] L. Shuai, G. Xie and J. Yang, "Characterization of HTTP behavior on access networks in Web 2.0," In Proc. of the International Conference on Telecommunications (ICT), June 2008.

[56] S. Lin, Z. Gao and K. Xu, "Web 2.0 traffic measurement: analysis on online map applications," In Proc. of the 18th International Workshop on Network and Operating Systems Support for digital audio and video, ser. NOSSDAV 09, 2009.

[57] M. Belshe, BitGo, R. Peon and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)," RFC 7540, May 2015.

[58] http://internet-browser-review.toptenreviews.com/.

[59] http://www.w3schools.com/browsers/browsers_stats.asp.

[60] F. Schneider, S. Agarwal, T. Alpcan and A. Feldmann, " The New Web: Characterizing AJAX Traffic," In Proc. of the Passive and Active Measurement Conference, 2008.

[61] "The Data Mining and Knowledge Discovery Handbook," In Oded Maimon and Lior Rokach Tel-Aviv (eds.), University Israel, pp. 40-41.