University of Victoria

ECE 572 - Security, Privacy, and Data Analytics

TCP SYN DoS Attack Data Analysis and Prediction

Instructor:	Imen Bourguiba	
Student:	Guilin Xie	V00104104
	Lok In Liang	V00104104
Department:	ECE MTIS	

Contents

Figures of Contents

Tables of Contents

Abstract

Denial of Service (DoS) attack becomes a widespread problem with the rapid development of Internet technology. The detection of DoS attack is essential for those providing online services. This project simulates the DoS attack on the Mininet platform and collects DoS attack and regular traffic data. Then, it presents DoS attack detection based on network traffic behavior patterns with statistical methods and machine learning models. The experiments show that a DoS attack has different network behavior patterns compared to regular traffic, and it reaches an accuracy of 95% with a logistic regression model in detecting DoS attack.

Keywords: SYN Attack, DoS (Denial-of-Service), Virtual Network, Data Analysis, Machine Learning

1. Introduction

Internet technology develops rapidly, leading to an increase in cyber-attacks. One of the most frequent and harmful attacks is DoS (Denial-of-Service) attack. A DoS attack intends to make network resources unavailable to legitimate users by exhausting the resources on purpose. A typical DoS attack is accomplished by using continuously sending irrelevant requests to overload the target network and host, making them failed to respond to legitimate requests.

Generally, there are two types of DoS attack, network bandwidth attack and server resource attack [1]. Network Bandwidth DoS attack originated as early as 1996 [2]. It aims to consume available network resources like bandwidth and router resources within a target network or near a target host, resulting in failing to provide service to legitimate users. Another DoS attack is focusing on exhausting server resource. The attacker launches massive duty requests from the server in attempts to occupy the server resources so that the server can fail to serve other legitimate requests. One example is a SQL injection attack [3], in which attackers can inject SQL statement to select or filter the entire data from the database. In this way, the server puts all resource in finishing the attacking task and may refuse to follow legitimate requests.

A DDoS(Distributed Denial-of-Service) happens when hackers apply a large scale of DoS attacks at a single target using a botnet. Botnets are computers or machines controlled by hackers to execute attacking actions with the help of the command and control server method. In this attack, the incoming malicious traffic comes from different resources, which makes it more challenging to block the attack sources. Many countries are under the threat of DDoS attack. The Figure 1 shows the most active countries as attack source and destination from 1st Jan 2015 to 24th Jul 2019.

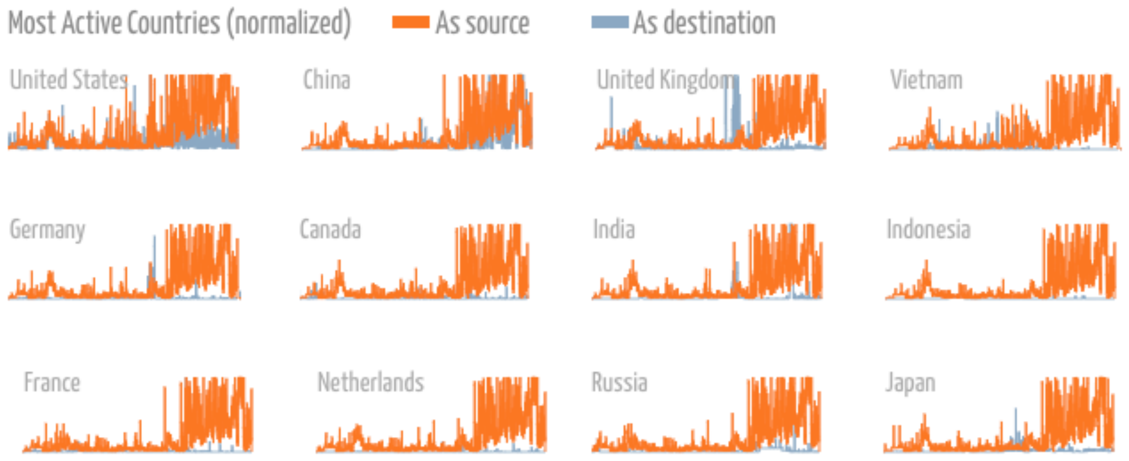


Figure 1: The most active countries as attack source and destination

In order to research, analyze, and prevent the DoS attack, this project simulates the DoS attack scenario with ***Mininet*** platform. We implement the DoS attack using SYN Flood approach in ***Mininet***. Then, collecting traffic packets data both in malicious DoS attacking and regular benign activities in the virtual network. Then we analyze the data details in Python with statistics and machine learning methods to discover hidden patterns behind the DoS attack and try to predict and prevent this attack in advance.

The limitation of our work is that we only simulate a small network system, while in a real-life DDoS attack scenario. There might be thousands of compromised computers controlled by hackers as bot machines. So the traffic from our simulation work might be different from a real case. To analyze the log data collected from a real DoS attack can get more accurate results.

The rest of this article is arranged as follows. Section 2 introduces the simulation environment, virtual network configuration, normal TCP traffic, TCP SYN Flood implementation, and normal and abnormal data traffic capturing by network sniffer tool. In Section 3, presenting network traffic data collection, preprocess, analysis, and prediction result. We conclude the paper in Section 4.

2. Simulation Method

2.1 Set up the Environment

The reason why we choose to do a simulation experiment is that we could implement the DoS attack without harming any real computers or networks. Another reason is that it is straightforward and simple to set up the virtual simulating environment instead of worrying about actual hardware. Therefore, we could focus more on the attack task and data analysis steps.

The simulation environment is based on ***Mininet***. It is a network emulation system [4], which can implement different network models with a collection of virtual end-hosts, switches, routers, and links on a single Linux kernel or machine. The models behave just like a real system, by sending packets through what simulates the real Ethernet interfaces, with a presetting link speed and delay. Packets can be processed by what simulates a real Ethernet switch, router, with a preset queue scheduling mechanism.

We configured the virtual network as follows. There are five virtual Ethernet switches, and each of the edges switches connected with a virtual host — the topology, as shown in Figure 2.

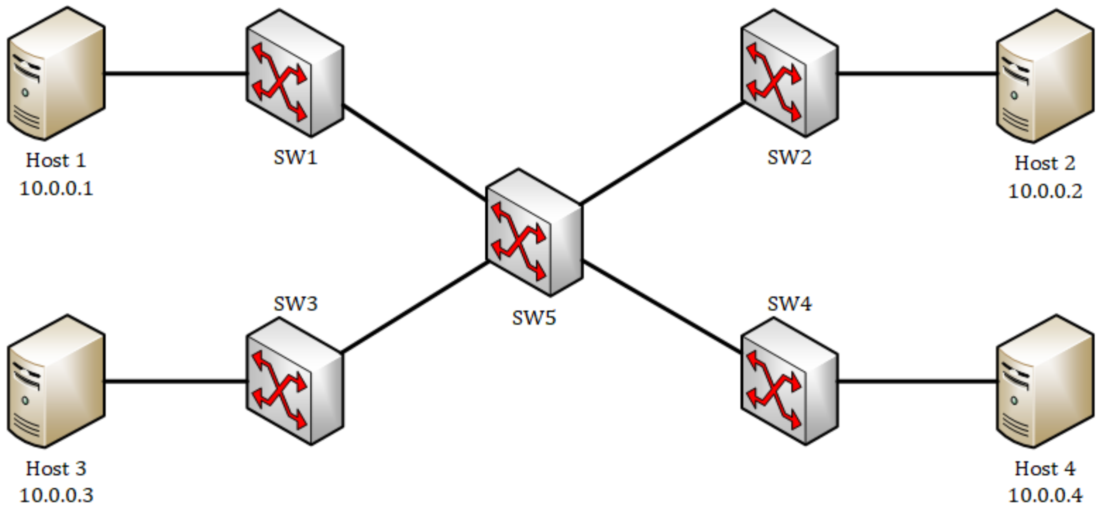
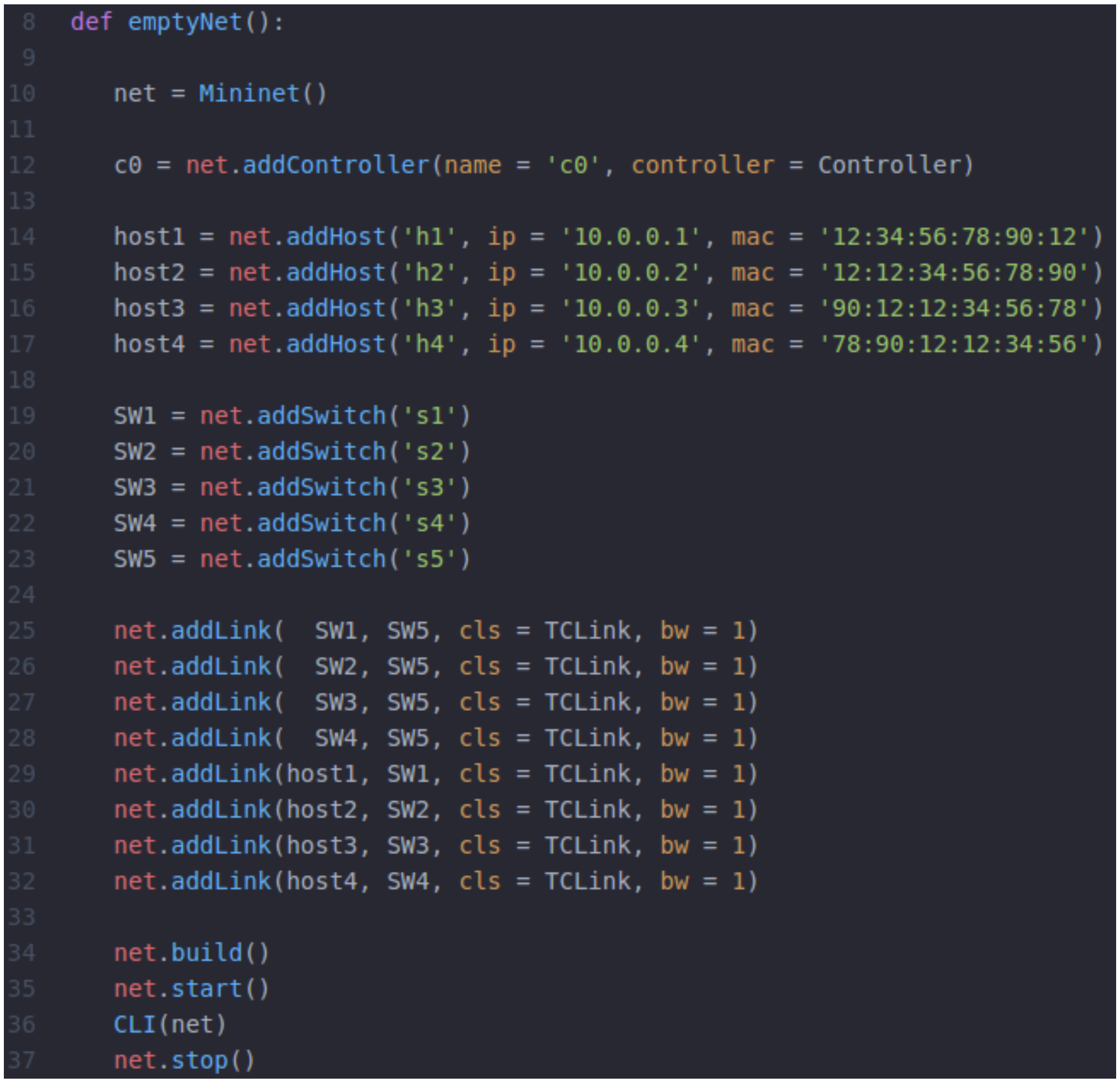


Figure 2: Simulation Virtual Network in Mininet

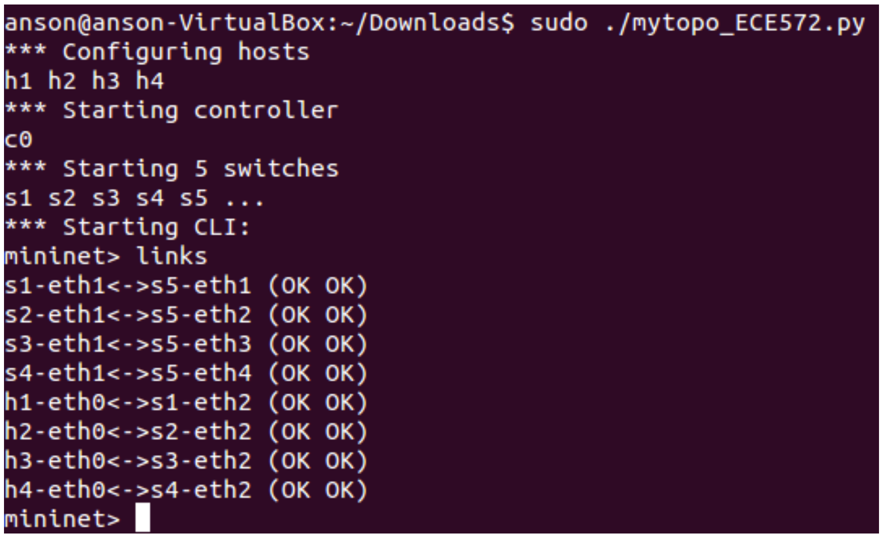
Mininet provides us with an effortless way to build a virtual network environment by using ***Python***. The configuration code illustrated in Figure 3.



```
 8  def emptyNet():
 9
10      net = Mininet()
11
12      c0 = net.addController(name = 'c0', controller = Controller)
13
14      host1 = net.addHost('h1', ip = '10.0.0.1', mac = '12:34:56:78:90:12')
15      host2 = net.addHost('h2', ip = '10.0.0.2', mac = '12:12:34:56:78:90')
16      host3 = net.addHost('h3', ip = '10.0.0.3', mac = '90:12:12:34:56:78')
17      host4 = net.addHost('h4', ip = '10.0.0.4', mac = '78:90:12:12:34:56')
18
19      SW1 = net.addSwitch('s1')
20      SW2 = net.addSwitch('s2')
21      SW3 = net.addSwitch('s3')
22      SW4 = net.addSwitch('s4')
23      SW5 = net.addSwitch('s5')
24
25      net.addLink(  SW1, SW5, cls = TCLink, bw = 1)
26      net.addLink(  SW2, SW5, cls = TCLink, bw = 1)
27      net.addLink(  SW3, SW5, cls = TCLink, bw = 1)
28      net.addLink(  SW4, SW5, cls = TCLink, bw = 1)
29      net.addLink(host1, SW1, cls = TCLink, bw = 1)
30      net.addLink(host2, SW2, cls = TCLink, bw = 1)
31      net.addLink(host3, SW3, cls = TCLink, bw = 1)
32      net.addLink(host4, SW4, cls = TCLink, bw = 1)
33
34      net.build()
35      net.start()
36      CLI(net)
37      net.stop()
```

Figure 3: Building the virtual network in Mininet by Python

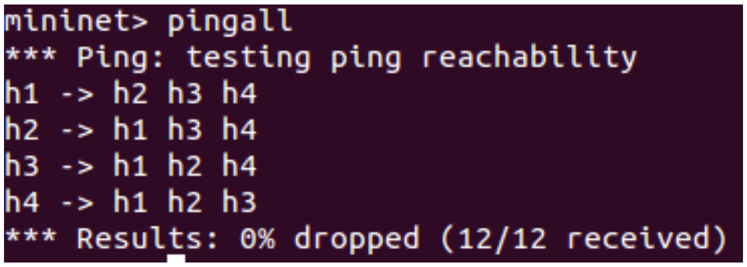
There are four parts in this configuration. First of all, it needs to set up a controller since ***Mininet*** support us to build a Software-Defined Network (SDN) [5] environment, it is required to have a controller in SDN. However, we do not need controller; therefore, we set up a default controller which is under Ethernet protocol in our environment. Secondly, adding hosts in our virtual network by using a simple Python code which is addHost. We can set up the name, IP and MAC address of a host. For example, the name of host1 is h1, the IP address is 10.0.0.1, and the MAC address is 12:34:56:78:90:12 (line 14). Then, it is the same idea as other hosts. Thirdly, adding switches in the environment by using addSwitch instruction. Currently, we have hosts and switches, but they are not connected. Lastly, connecting the devices and finalized our environment by using addLink code. For instance, we added a link between Switch 1 and Switch 5. Then, we would like to have the link with 1 Mpbs bandwidth (line 25); therefore, we set up this link as a TCLink, which is a traffic control link. Since it is not easy to simulate an actual DoS situation, we constrained the bandwidth of the environment for reducing the difficulty of our simulation. Other links are in the same setting.



```
anson@anson-VirtualBox:~/Downloads$ sudo ./mytopo_ECE572.py
*** Configuring hosts
h1 h2 h3 h4
*** Starting controller
c0
*** Starting 5 switches
s1 s2 s3 s4 s5 ...
*** Starting CLI:
mininet> links
s1-eth1<->s5-eth1 (OK OK)
s2-eth1<->s5-eth2 (OK OK)
s3-eth1<->s5-eth3 (OK OK)
s4-eth1<->s5-eth4 (OK OK)
h1-eth0<->s1-eth2 (OK OK)
h2-eth0<->s2-eth2 (OK OK)
h3-eth0<->s3-eth2 (OK OK)
h4-eth0<->s4-eth2 (OK OK)
mininet>
```

Figure 4: Welcome page when started the Mininet

When we started the ***Mininet***, it will show us the detail about the environment, which includes configuring hosts, started controller, started switches, and links in this environment, as shown in Figure 4. For testing our environment is worked, it can use ***pingall*** command in the terminal of ***Mininet***. The result shown in Figure 5, all links are connected and worked (0% dropped) in our environment.



```
mininet> pingall
*** Ping: testing ping reachability
h1 -> h2 h3 h4
h2 -> h1 h3 h4
h3 -> h1 h2 h4
h4 -> h1 h2 h3
*** Results: 0% dropped (12/12 received)
```

Figure 5: *pingall* in the Mininet for testing our environment

2.2 Normal TCP traffic

Simulating a more real-life situation of DoS, it has to generate some background / regular traffic in the virtual environment. We want to use a Linux tool ***iPerf*** [6] for creating regular TCP traffic. ***iPerf*** is a widely used tool for network performance measurement and tuning. It is significant as a cross-platform tool that can produce standardized performance measurements for any network. ***iPerf*** has client and server functionality and can create data streams to measure the throughput between the two ends in one or both directions. Typical ***iPerf*** output contains a time-stamped report of the amount of data transferred and the performance measured.

To implement the Normal TCP traffic, we used ***iPerf*** command as follows:
iperf -s, and
iperf -c [destination IP address] -i 1 -t 100
where the parameters are in the following table.

Table 1: The parameters of iPerf for sending Normal TCP traffic

Parameters	**Meaning**
-s	Server side
-c	Client side
-i	Sets the interval time in seconds between periodic bandwidth
-t	The time in seconds to transmit

The normal TCP traffic sends from host 1 (terminal of left-hand-side) to host 3 (terminal of right-hand-side), as shown in Figure 7. And, the topology as shown in Figure 6.

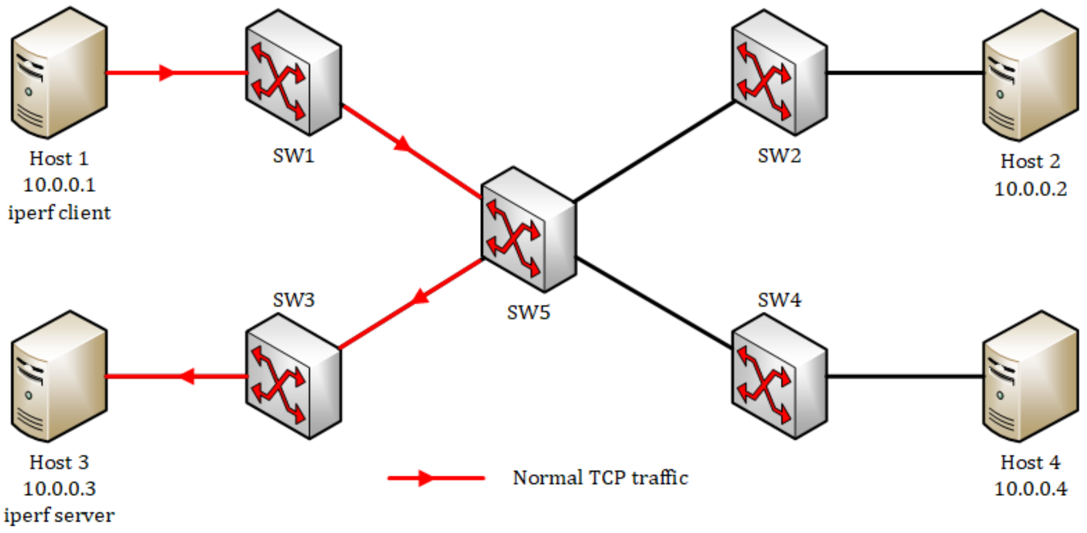
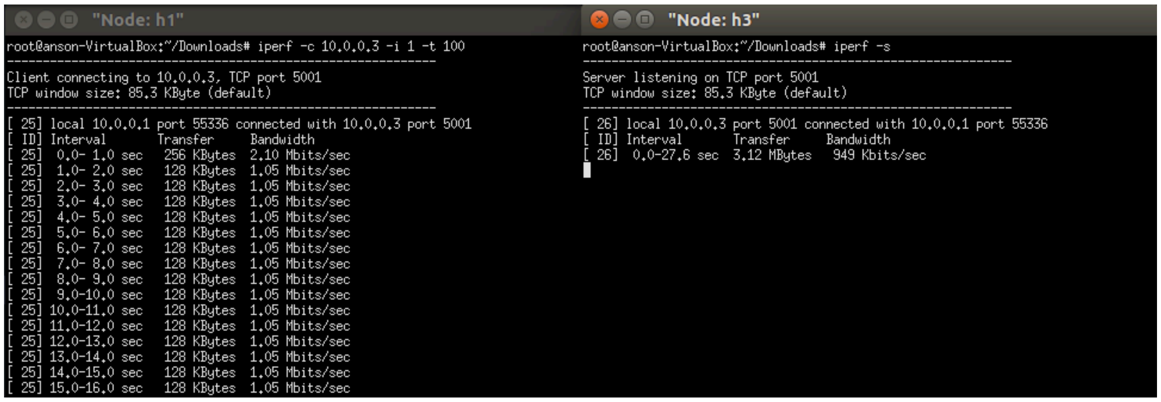


Figure 6: Topology of sending normal TCP traffic


"Node: h1"

```
root@anson-VirtualBox:~/Downloads# iperf -c 10.0.0.3 -i 1 -t 100
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 25] local 10.0.0.1 port 55336 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 25]  0.0- 1.0 sec   256 KBytes  2.10 Mbits/sec
[ 25]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  9.0-10.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 10.0-11.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 11.0-12.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 12.0-13.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 13.0-14.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 14.0-15.0 sec   128 KBytes  1.05 Mbits/sec
[ 25] 15.0-16.0 sec   128 KBytes  1.05 Mbits/sec
```

"Node: h3"

```
root@anson-VirtualBox:~/Downloads# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 26] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 55336
[ ID] Interval       Transfer     Bandwidth
[ 26]  0.0-27.6 sec  3.12 MBytes   949 Kbits/sec
```

Figure 7: Sending Normal TCP traffic by iPerf

To be illustrated in Figure 6 and Figure 7, host 3 is the server, and host 1 is the client. The command of the server in host 3 is ***iperf -s***, therefore, ***-s*** means that is server-side. On the other side, the command of the client in host 1 is ***iperf -c [destination IP address] -i 1 -t 100***, which means that ***-c*** is the client-side with destination IP address, ***-i 1*** means that showing the result every one second, lastly, ***-t 100*** means the total time 100 seconds for sending the traffic. If there is no specific setting for ***iperf***, it will be in default for sending TCP traffic. The result shows that the bandwidth between host 1 and host 2 is around 1 Mbps which fit our setting in our environment.

2.3 TCP SYN Flood Implementation

TCP SYN flood is a kind of DoS attack that exploits the regular TCP three-way handshake. Basically, in a TCP connection between two hosts, when a client and server needs to establish a standard TCP "three-way handshake", the three-way packets exchange in the following order and direction, as shown in Figure 8.

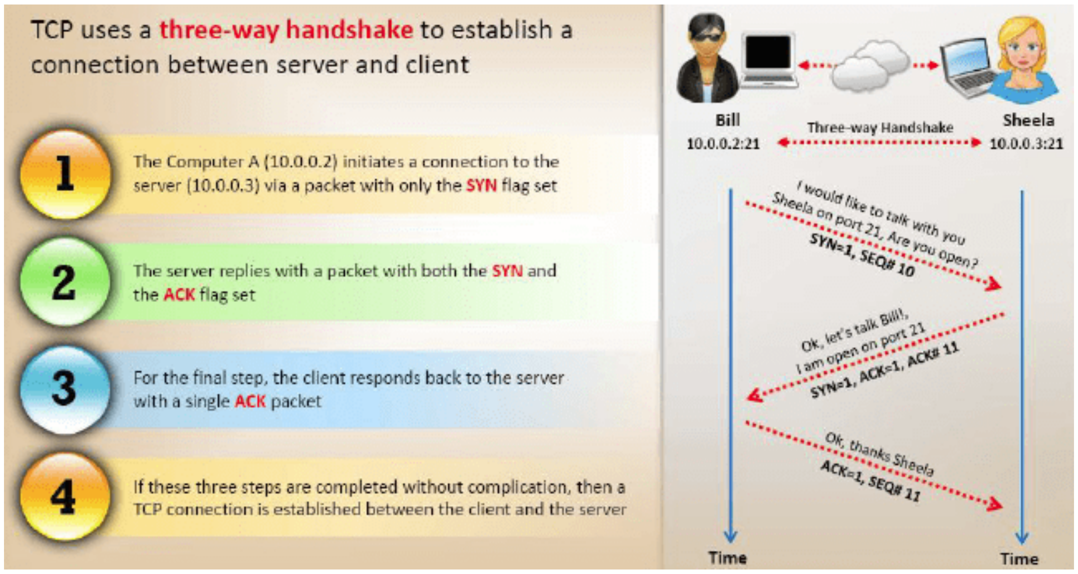

TCP uses a three-way handshake to establish a connection between server and client
1 The Computer A (10.0.0.2) initiates a connection to the server (10.0.0.3) via a packet with only the SYN flag set
2 The server replies with a packet with both the SYN and the ACK flag set
3 For the final step, the client responds back to the server with a single ACK packet
4 If these three steps are completed without complication, then a TCP connection is established between the client and the server
Bill
10.0.0.2:21
Three-way Handshake
Sheela
10.0.0.3:21
I would like to talk with you Sheela on port 21. Are you open?
SYN=1, SEQ# 10
Ok, let's talk Bill!, I am open on port 21
SYN=1, ACK=1, ACK# 11
Ok, thanks Sheela
ACK=1, SEQ# 11
Time
Time

Figure 8: Three-Way Handshake [7]

From the above three-way process, we know that in a normal TCP connection, the first step is that the client sends an SYN message to the server. The second step is that the server replies to the SYN message from the client with an SYN/ACK message to the client to acknowledge that it has already received the initial SYN message. After this step, the TCP server is half open and reserves the required resources for the clients. The third step is that the client must reply to the server again with another ACK message. Then the TCP connection is completely open, and packets can transfer between the client and the server.

In a TCP SYN flooding DoS attack, an attacker only sends a huge number of SYN messages with spoofing IP addresses to the server without acknowledging these messages after the server replies SYN/ACK. In this way, the server holds too many half-open TCP connections, and finally has no resource to respond to new legitimate requests.

To implement the TCP SYN flooding attack, we used ***hping3*** command as follows [8]:

- ***hping3 -d 120 -S -p 80 --flood --rand-source [destination IP address]***, or
- ***hping3 -S --flood -V [destination IP address]***

where the parameters are in the following table.

Table 2: The parameters of hping3 for sending TCP SYN flooding

Parameters	**Meaning**
-d	The data size of a packet
-S	The type of packet is SYN flag
-p	The destination port
--flood	Attacking mode in flood. sent packets as fast as possible
--rand-source	Random source address mode

The topology of sending TCP SYN flooding traffic, as shown in Figure 9. Then, the terminal capture of sending traffic from host 2 (terminal of left-hand-side) to host 4 (terminal of right-hand-side), as shown in Figure 10.

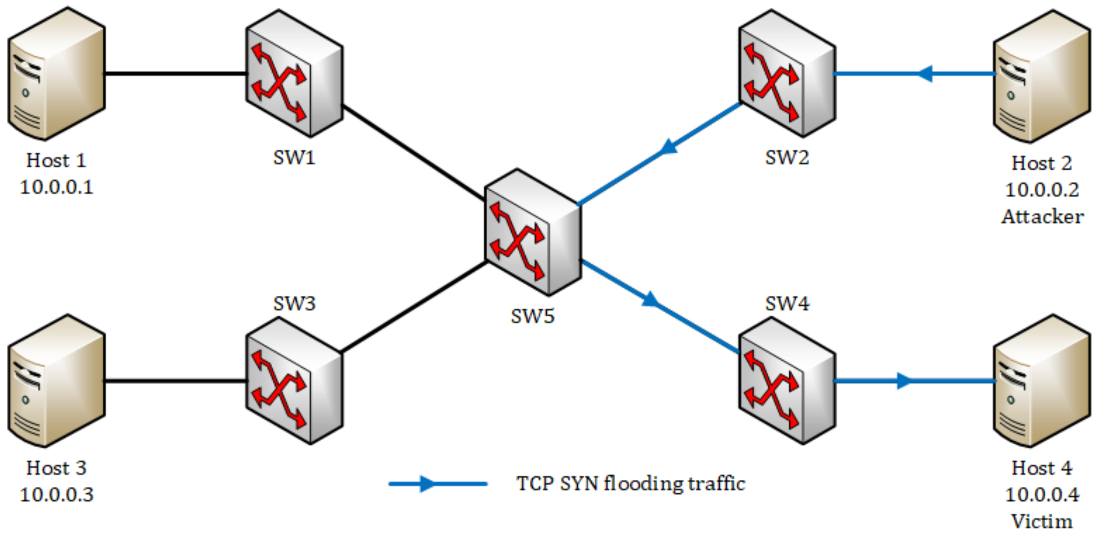
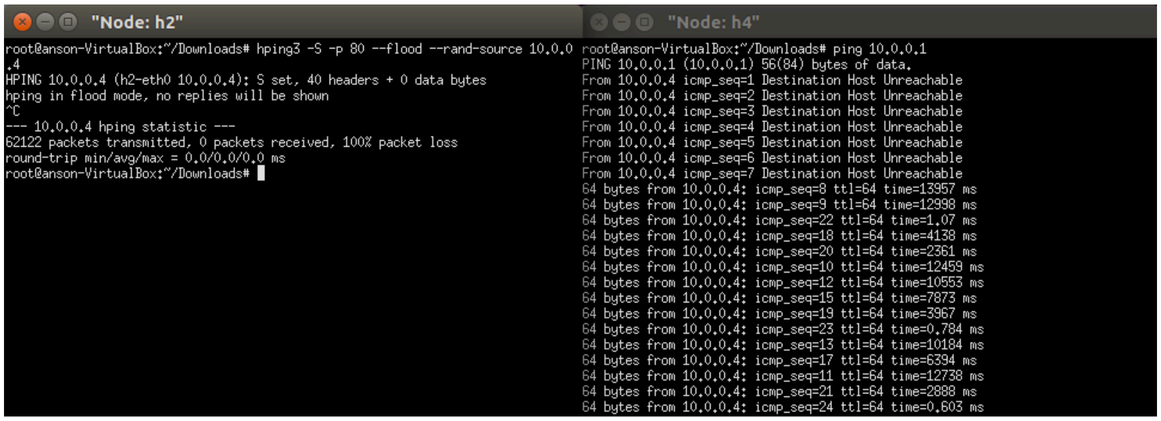


Figure 9: Topology of sending TCP SYN Flooding traffic



```
root@anson-VirtualBox:~/Downloads# hping3 -S -p 80 --flood --rand-source 10.0.0
.4
HPING 10.0.0.4 (h2-eth0 10.0.0.4): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.4 hping statistic ---
62122 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@anson-VirtualBox:~/Downloads# 
```

```
root@anson-VirtualBox:~/Downloads# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
From 10.0.0.4 icmp_seq=3 Destination Host Unreachable
From 10.0.0.4 icmp_seq=4 Destination Host Unreachable
From 10.0.0.4 icmp_seq=5 Destination Host Unreachable
From 10.0.0.4 icmp_seq=6 Destination Host Unreachable
From 10.0.0.4 icmp_seq=7 Destination Host Unreachable
64 bytes from 10.0.0.4: icmp_seq=8 ttl=64 time=13957 ms
64 bytes from 10.0.0.4: icmp_seq=9 ttl=64 time=12998 ms
64 bytes from 10.0.0.4: icmp_seq=22 ttl=64 time=1.07 ms
64 bytes from 10.0.0.4: icmp_seq=18 ttl=64 time=4138 ms
64 bytes from 10.0.0.4: icmp_seq=20 ttl=64 time=2361 ms
64 bytes from 10.0.0.4: icmp_seq=10 ttl=64 time=12459 ms
64 bytes from 10.0.0.4: icmp_seq=12 ttl=64 time=10553 ms
64 bytes from 10.0.0.4: icmp_seq=15 ttl=64 time=7873 ms
64 bytes from 10.0.0.4: icmp_seq=19 ttl=64 time=3967 ms
64 bytes from 10.0.0.4: icmp_seq=23 ttl=64 time=0.784 ms
64 bytes from 10.0.0.4: icmp_seq=13 ttl=64 time=10184 ms
64 bytes from 10.0.0.4: icmp_seq=17 ttl=64 time=6394 ms
64 bytes from 10.0.0.4: icmp_seq=11 ttl=64 time=12738 ms
64 bytes from 10.0.0.4: icmp_seq=21 ttl=64 time=2888 ms
64 bytes from 10.0.0.4: icmp_seq=24 ttl=64 time=0.603 ms
```

Figure 10: Sending TCP SYN Flooding traffic by hping3

To be illustrated in Figure 9 and Figure 10, host 2 is the attacker, and host 4 is the victim. The command of the attacker in host 2 is ***hping3 -S -p 80 –flood –rand-source [destination IP address]***, therefore, ***-S*** means that is sending TCP SYN packet; then, ***-p*** is the destination post that is attacked; and, ***--flood*** is sending the packets as fast as possible; lastly, ***--rand-source*** is random the source IP address which can let victim does not know the actual source. On the other side, the command of the victim in host 4 is ***ping [destination IP address]***, which is used to see what going on when the attacker is attacking host 4. As shown the result in the right-hand side of Figure 10, the first seven packets are unreachable; after that, other packets with a long duration time. Obviously, the attacker successfully attacked the victim of host 4 by using a TCP SYN flooding attack.

2.4 Simulation Normal and Abnormal data traffic capturing

Capturing data for detecting or predicting the TCP SYN flooding attack in this environment, we used a network sniffer tool – ***Wireshark***. We would like to have two traffic sending at the same time, which is normal TCP traffic and TCP SYN flooding. Firstly, we set up the normal TCP traffic which sending from host 1 to host 3 by using ***iPerf***. Secondly, we set up the abnormal TCP SYN flooding traffic which sending from host 2 to host 4 by using ***hping3***. Finally, we open the Wireshark in host 3 and host 4 for capturing normal and abnormal data traffic, respectively. The topology as shown in Figure 11.

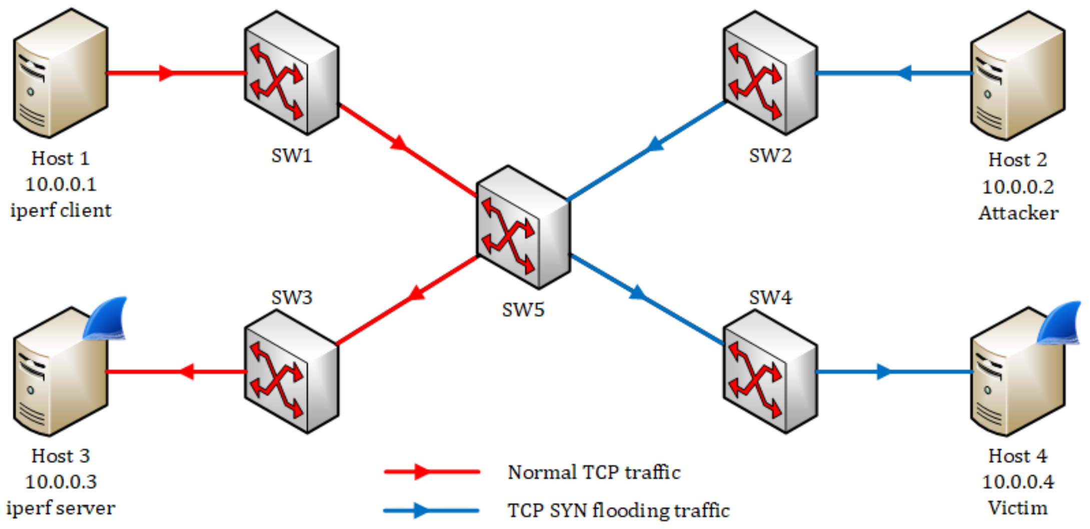
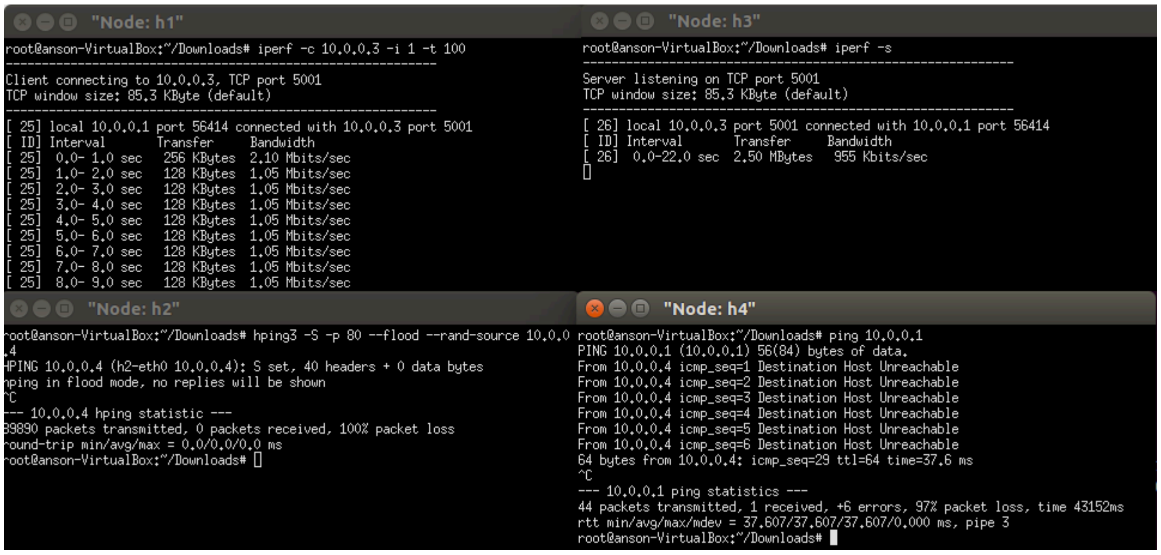


Figure 11: Topology of sending normal and abnormal traffic

To be illustrated in Figure 12, the upper part of the terminal is responsible for the normal TCP traffic sending by ***iPerf***. Then, the lower part of the terminal is sending the abnormal SYN TCP flooding traffic by ***hping3***. As a result, showing that the normal TCP traffic is good, there is not any error and packet loss. However, the victim computer sends the ping packet to host 1 almost got 100% packet loss due to the attacker attack host 4 by DoS.



```
root@anson-VirtualBox:~/Downloads# iperf -c 10.0.0.3 -i 1 -t 100
------------------------------------------------------------
Client connecting to 10.0.0.3, TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 25] local 10.0.0.1 port 56414 connected with 10.0.0.3 port 5001
[ ID] Interval       Transfer     Bandwidth
[ 25]  0.0- 1.0 sec   256 KBytes  2.10 Mbits/sec
[ 25]  1.0- 2.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  2.0- 3.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  3.0- 4.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  4.0- 5.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  5.0- 6.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  6.0- 7.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  7.0- 8.0 sec   128 KBytes  1.05 Mbits/sec
[ 25]  8.0- 9.0 sec   128 KBytes  1.05 Mbits/sec
```

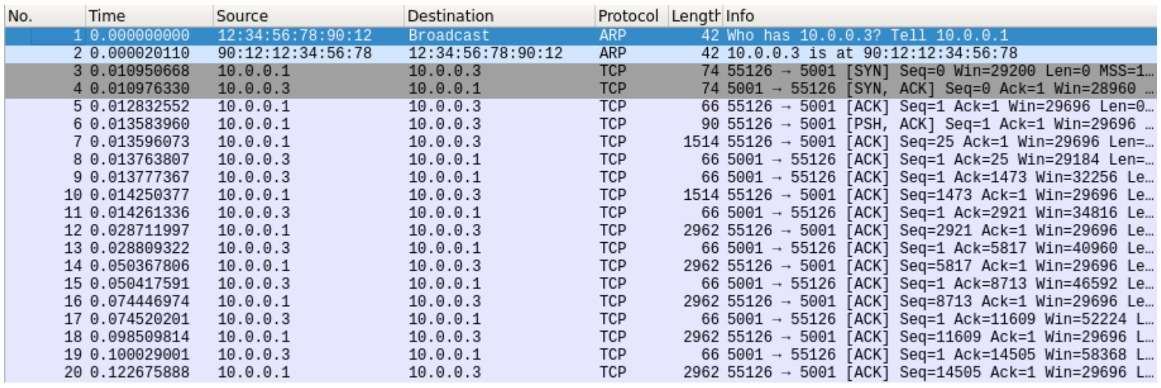
```
root@anson-VirtualBox:~/Downloads# iperf -s
------------------------------------------------------------
Server listening on TCP port 5001
TCP window size: 85.3 KByte (default)
------------------------------------------------------------
[ 26] local 10.0.0.3 port 5001 connected with 10.0.0.1 port 56414
[ ID] Interval       Transfer     Bandwidth
[ 26]  0.0-22.0 sec  2.50 MBytes   955 Kbits/sec
```

```
root@anson-VirtualBox:~/Downloads# hping3 -S -p 80 --flood --rand-source 10.0.0
.4
HPING 10.0.0.4 (h2-eth0 10.0.0.4): S set, 40 headers + 0 data bytes
hping in flood mode, no replies will be shown
^C
--- 10.0.0.4 hping statistic ---
39890 packets transmitted, 0 packets received, 100% packet loss
round-trip min/avg/max = 0.0/0.0/0.0 ms
root@anson-VirtualBox:~/Downloads#
```

```
root@anson-VirtualBox:~/Downloads# ping 10.0.0.1
PING 10.0.0.1 (10.0.0.1) 56(84) bytes of data.
From 10.0.0.4 icmp_seq=1 Destination Host Unreachable
From 10.0.0.4 icmp_seq=2 Destination Host Unreachable
From 10.0.0.4 icmp_seq=3 Destination Host Unreachable
From 10.0.0.4 icmp_seq=4 Destination Host Unreachable
From 10.0.0.4 icmp_seq=5 Destination Host Unreachable
From 10.0.0.4 icmp_seq=6 Destination Host Unreachable
64 bytes from 10.0.0.4: icmp_seq=29 ttl=64 time=37.6 ms
^C
--- 10.0.0.1 ping statistics ---
44 packets transmitted, 1 received, +6 errors, 97% packet loss, time 43152ms
rtt min/avg/max/mdev = 37.607/37.607/37.607/0.000 ms, pipe 3
root@anson-VirtualBox:~/Downloads#
```

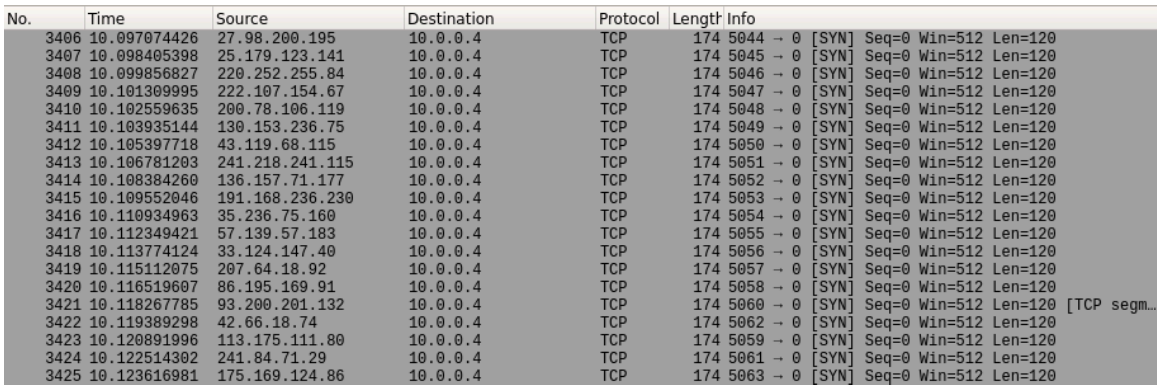
Figure 12: Sending normal and abnormal traffic by iperf and hping3, respectively

As shown in Figure 13, it is evident that TCP three-way handshake is happened by sending the normal TCP traffic, which includes SYN, SYN/ACK, and ACK packets. On the other hand, there are only SYN packets when we send the abnormal traffic of TCP SYN flooding, as shown in Figure 14.



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000000	12:34:56:78:90:12	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
2	0.000020110	90:12:12:34:56:78	12:34:56:78:90:12	ARP	42	10.0.0.3 is at 90:12:12:34:56:78
3	0.010950668	10.0.0.1	10.0.0.3	TCP	74	55126 → 5001 [SYN] Seq=0 Win=29200 Len=0 MSS=1…
4	0.010976330	10.0.0.3	10.0.0.1	TCP	74	5001 → 55126 [SYN, ACK] Seq=0 Ack=1 Win=28960 …
5	0.012832552	10.0.0.1	10.0.0.3	TCP	66	55126 → 5001 [ACK] Seq=1 Ack=1 Win=29696 Len=0…
6	0.013583960	10.0.0.1	10.0.0.3	TCP	90	55126 → 5001 [PSH, ACK] Seq=1 Ack=1 Win=29696 …
7	0.013596073	10.0.0.1	10.0.0.3	TCP	1514	55126 → 5001 [ACK] Seq=25 Ack=1 Win=29696 Len=…
8	0.013763807	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=25 Win=29184 Len=…
9	0.013777367	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=1473 Win=32256 Le…
10	0.014250377	10.0.0.1	10.0.0.3	TCP	1514	55126 → 5001 [ACK] Seq=1473 Ack=1 Win=29696 Le…
11	0.014261336	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=2921 Win=34816 Le…
12	0.028711997	10.0.0.1	10.0.0.3	TCP	2962	55126 → 5001 [ACK] Seq=2921 Ack=1 Win=29696 Le…
13	0.028809322	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=5817 Win=40960 Le…
14	0.050367806	10.0.0.1	10.0.0.3	TCP	2962	55126 → 5001 [ACK] Seq=5817 Ack=1 Win=29696 Le…
15	0.050417591	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=8713 Win=46592 Le…
16	0.074446974	10.0.0.1	10.0.0.3	TCP	2962	55126 → 5001 [ACK] Seq=8713 Ack=1 Win=29696 Le…
17	0.074520201	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=11609 Win=52224 L…
18	0.098509814	10.0.0.1	10.0.0.3	TCP	2962	55126 → 5001 [ACK] Seq=11609 Ack=1 Win=29696 L…
19	0.100029001	10.0.0.3	10.0.0.1	TCP	66	5001 → 55126 [ACK] Seq=1 Ack=14505 Win=58368 L…
20	0.122675888	10.0.0.1	10.0.0.3	TCP	2962	55126 → 5001 [ACK] Seq=14505 Ack=1 Win=29696 L…

Figure 13: Normal TCP traffic by iperf in Wireshark



No.	Time	Source	Destination	Protocol	Length	Info
3406	10.097074426	27.98.200.195	10.0.0.4	TCP	174	5044 → 0 [SYN] Seq=0 Win=512 Len=120
3407	10.098405398	25.179.123.141	10.0.0.4	TCP	174	5045 → 0 [SYN] Seq=0 Win=512 Len=120
3408	10.099856827	220.252.255.84	10.0.0.4	TCP	174	5046 → 0 [SYN] Seq=0 Win=512 Len=120
3409	10.101309995	222.107.154.67	10.0.0.4	TCP	174	5047 → 0 [SYN] Seq=0 Win=512 Len=120
3410	10.102559635	200.78.106.119	10.0.0.4	TCP	174	5048 → 0 [SYN] Seq=0 Win=512 Len=120
3411	10.103935144	130.153.236.75	10.0.0.4	TCP	174	5049 → 0 [SYN] Seq=0 Win=512 Len=120
3412	10.105397718	43.119.68.115	10.0.0.4	TCP	174	5050 → 0 [SYN] Seq=0 Win=512 Len=120
3413	10.106781203	241.218.241.115	10.0.0.4	TCP	174	5051 → 0 [SYN] Seq=0 Win=512 Len=120
3414	10.108384260	136.157.71.177	10.0.0.4	TCP	174	5052 → 0 [SYN] Seq=0 Win=512 Len=120
3415	10.109552046	191.168.236.230	10.0.0.4	TCP	174	5053 → 0 [SYN] Seq=0 Win=512 Len=120
3416	10.110934963	35.236.75.160	10.0.0.4	TCP	174	5054 → 0 [SYN] Seq=0 Win=512 Len=120
3417	10.112349421	57.139.57.183	10.0.0.4	TCP	174	5055 → 0 [SYN] Seq=0 Win=512 Len=120
3418	10.113774124	33.124.147.40	10.0.0.4	TCP	174	5056 → 0 [SYN] Seq=0 Win=512 Len=120
3419	10.115112075	207.64.18.92	10.0.0.4	TCP	174	5057 → 0 [SYN] Seq=0 Win=512 Len=120
3420	10.116519607	86.195.169.91	10.0.0.4	TCP	174	5058 → 0 [SYN] Seq=0 Win=512 Len=120
3421	10.118267785	93.200.201.132	10.0.0.4	TCP	174	5060 → 0 [SYN] Seq=0 Win=512 Len=120 [TCP segm…
3422	10.119389298	42.66.18.74	10.0.0.4	TCP	174	5062 → 0 [SYN] Seq=0 Win=512 Len=120
3423	10.120891996	113.175.111.80	10.0.0.4	TCP	174	5059 → 0 [SYN] Seq=0 Win=512 Len=120
3424	10.122514302	241.84.71.29	10.0.0.4	TCP	174	5061 → 0 [SYN] Seq=0 Win=512 Len=120
3425	10.123616981	175.169.124.86	10.0.0.4	TCP	174	5063 → 0 [SYN] Seq=0 Win=512 Len=120

Figure 14: Abnormal TCP SYN flooding by hping3 in Wireshark

After we capture the data from ***Wireshark***, we save as the Wireshark file to ***CSV*** file, which can let us insert to our Python code for finding hidden patterns by using our knowledge in data analysis.

3. Data Analysis and Results

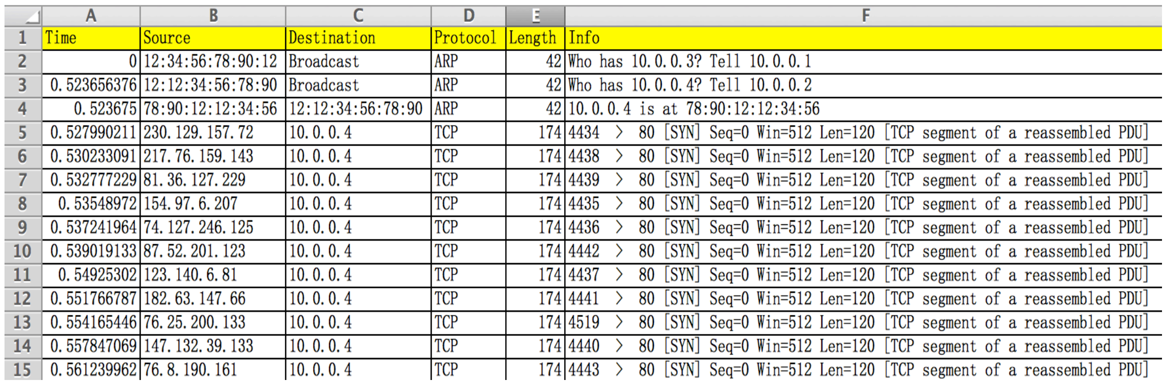
To analyze the data, we use the Python platform. Python is popular in data mining area because it has thousands of free libraries using cutting-edge algorithms. The experiment is based on Python 3.7 with the libraries of ***numpy***, ***pandas***, and ***scikit-learn***.

In cybersecurity, it is essential to have security protection systems, such as Intrusion Detection System(IDS) and Intrusion Prevention System(IPS). IDS and IPS systems are typically based on analyzing the incoming traffic to figure out the behavior patterns of attacks. Then, we can detect all incoming traffic to check if their behavior matches known malicious behavior patterns. This process relies much on the analysis of traffic data. So in this step, we need to further our study in data collecting, preprocess, and analyzing to the previous TCP SYN flooding attack.

3.1 Data Collection

In the TCP SYN flooding DoS attack, we use a network packet sniff tool named ***Wireshark*** to collect traffic data. It can catch traffic packets from and to the hosts, and we collected the following data from both DoS attacking and regular activities.

In total, we collected ***9438*** data log from the machine that is under DoS attack, and ***1319*** data log from the machine with normal activities. The original sample data is in ***CSV*** format as follows.



	A	B	C	D	E	F
1	Time	Source	Destination	Protocol	Length	Info
2	0	12:34:56:78:90:12	Broadcast	ARP	42	Who has 10.0.0.3? Tell 10.0.0.1
3	0.523656376	12:12:34:56:78:90	Broadcast	ARP	42	Who has 10.0.0.4? Tell 10.0.0.2
4	0.523675	78:90:12:12:34:56	12:12:34:56:78:90	ARP	42	10.0.0.4 is at 78:90:12:12:34:56
5	0.527990211	230.129.157.72	10.0.0.4	TCP	174	4434 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
6	0.530233091	217.76.159.143	10.0.0.4	TCP	174	4438 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
7	0.532777229	81.36.127.229	10.0.0.4	TCP	174	4439 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
8	0.53548972	154.97.6.207	10.0.0.4	TCP	174	4435 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
9	0.537241964	74.127.246.125	10.0.0.4	TCP	174	4436 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
10	0.539019133	87.52.201.123	10.0.0.4	TCP	174	4442 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
11	0.54925302	123.140.6.81	10.0.0.4	TCP	174	4437 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
12	0.551766787	182.63.147.66	10.0.0.4	TCP	174	4441 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
13	0.554165446	76.25.200.133	10.0.0.4	TCP	174	4519 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
14	0.557847069	147.132.39.133	10.0.0.4	TCP	174	4440 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]
15	0.561239962	76.8.190.161	10.0.0.4	TCP	174	4443 > 80 [SYN] Seq=0 Win=512 Len=120 [TCP segment of a reassembled PDU]

Figure 15: Original Data Captured Samples

3.2 Data Cleansing and Feature Extraction

- The original data has six columns, namely *Time, Source, Destination, Protocol, Length, and Info*. Then we used excel to parse to generate new columns, filter the missing value (like null or space), compute and delete data samples containing null data for cleaning purpose, and then extract the interesting information.
- After filtering the data that contains null and space, there remain ***1186*** regular packets and ***9187*** DoS attack packets data.
- In this step, we extracted the following ***four features*** from the dataset, and get them ready for following statistics and machine learning.

Table 3: Data Elements and Meaning

Data Elements	**Meaning**
TimeInterval	Time Interval when Receiving the Packet from Previous Packet.
TotalPacketsOfEachSource	Total Packets Sent by the Source IP during the last 15 seconds
PacketLength	Length of the Packet
Packet Type	The type of the packet

3.3 Data Normalization and Transformation

The real data shows the following table. We need to normalize it before applying any machine learning method because the features have different numerical ranges. After normalization, it can be easier to handle in machine learning models, and also can be more accurate in results.

Table 4: Original Data (samples)

TimeInterval	TotalPacketsEachSource	Length	Traffic Type
0.0000256620	605	74	SYN
0.0018562220	581	74	SYN
0.0007514080	605	66	ACK
0.0000121130	605	90	PSH

- For the normalizing step, we used the following normalization method to normalize the numerical features.

$$x' = \frac{x - average(x)}{max(x) - min(x)}$$

After normalization, the new data of the features became as follows.

Table 5: Data Normalization Result (samples)

TimeInterval	TotalPacketsEachSource	Length	Traffic Type
-0.035995706	0.489881956	-0.493350135	SYN
-0.030641104	-0.510118044	-0.493350135	SYN
-0.033872814	0.489881956	-0.496112566	ACK
-0.036035339	0.489881956	-0.487825273	PSH

- Another step is to transform data. The ***Traffic Type*** is a categorical feature with ACK, FIN, PSH, and SYN. We need to transform the categorical feature to one hot encoder features. It can help machine learning models to predict more accurate. After the one-hot encoding, the data now became as follows.

Table 6: Data Transformation Result (samples)

TimeInterval	TotalPacketsEachSource	Length	Traffic Type			
			Type1	Type2	Type3	Type4
-0.035776	0.489882	0.503887	1	0	0	0
0.034133	-0.510118	-0.496113	1	0	0	0
-0.035665	0.489882	0.379578	0	1	0	0
-0.005104	-0.510118	-0.496113	1	0	0	0

3.4 Statistical Analysis

In this stage, we use statistical theory to research on the details of the packets to understand the data, explore hidden patterns, and detect potential DoS attack. We did this by analyzing the relations in activities, comparing data difference between attack and normal traffic, and summarize possible behavior patterns behind DoS attack.

- First, we check the difference in the number of packets sent between regular traffic and DoS attack traffic. From the following statistic, we can see that there were many communication packets between the client and the server in the regular traffic. However, from the DoS attacking traffic, each of the source IPs only sent one packet to the server, and never respond a second packet to further communicate with the server, which can be a behavior pattern of DoS attack with spoofed source IP technique.

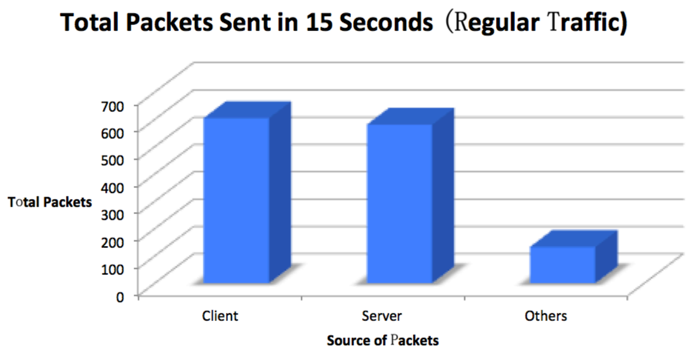


Figure 16: Packets Sent from different Source in Regular Traffic

Table 7: Packets Sent from different Source in DoS Attack Traffic (Sample)

Source IP	Total Packets Sent
68.171.1.72	1
223.140.5.218	1
8.185.69.93	1
54.159.207.12	1

- Then, we compared the difference between the packet length from both normal traffic and DoS attack traffic. From the following statistic data, we can infer that in the normal traffic, packet length varies according to actual transferring needs, but the length of packets for DoS attack almost remains the same. It is understandable as the attacker only sends numerous nearly the same SYN packets to the server. So if all packets length is almost the same, then there might be a DoS attack, which can be another behavior pattern of DoS attack.

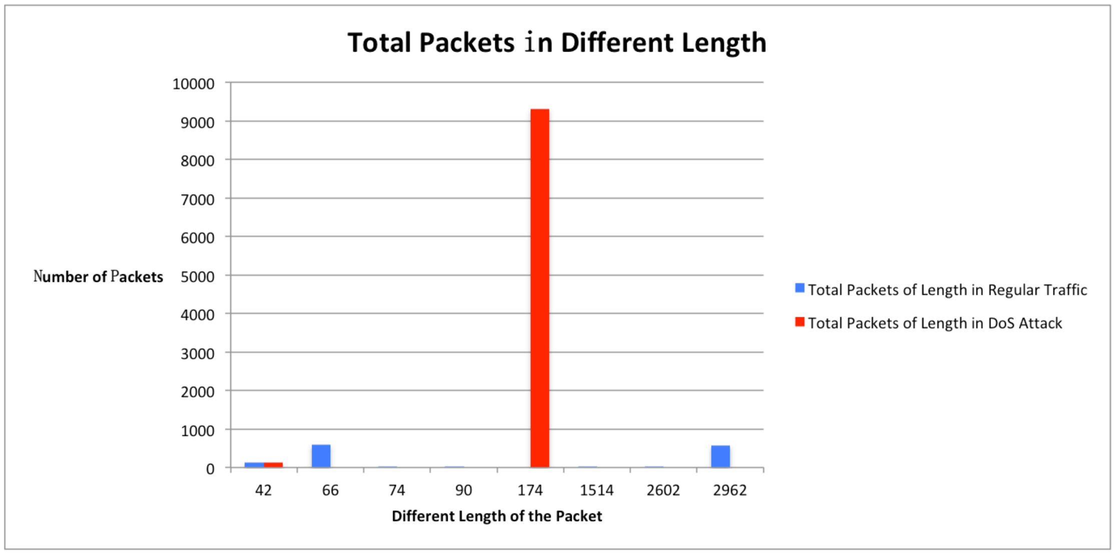


Figure 17: Total Packets in Different Length

- Then, we calculated the number of different packet types from both traffics. It shows that in a regular TCP traffic, most of the packets are ACK type, which associates with acknowledge from server about receiving the packets from the client, while in an SYN flooding DoS attack, almost all the packets are SYN type, in our case, the percentage of SYN packets even reaches as high as 100%. So the percentage of SYN type packets can be regarded as another behavior pattern of DoS attack.

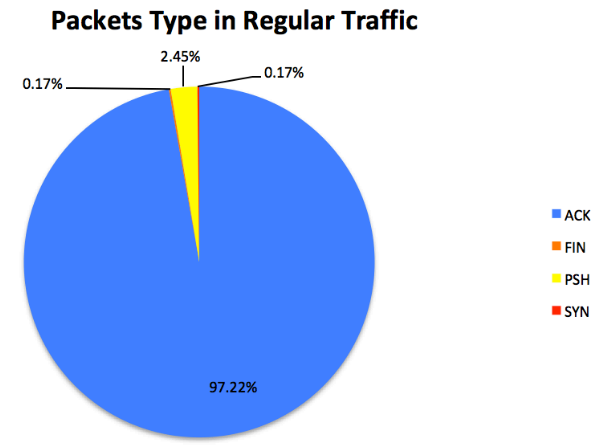


Figure 18: Percentage of Packets Type in Regular Traffic

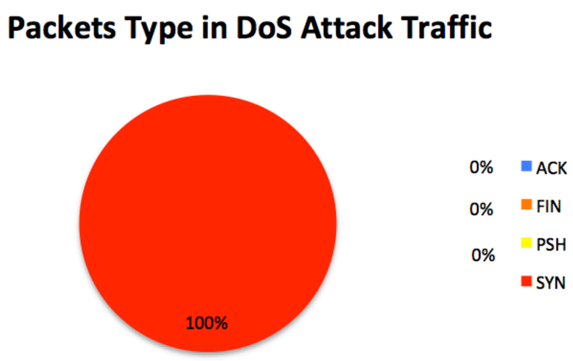


Figure 19: Percentage of Packets Type in DoS Attack Traffic

3.5 Machine Learning

In this stage, we tried to apply a machine learning algorithm to classify DoS attack packets. We chose the Logistic Regression model for this classification task, and implemented in ***Python*** with open libraries of ***pandas***, ***sklearn***, and ***numpy***.

3.5.1 Logistic Regression

The logistic regression model can be used to predict a conditional probability [9]. The data is then classified to the largest conditional probability.

- The equation of logistic regression is as follows.

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}$$

  where x is the input features, θ is the parameter. The job is to find the best θ to minimize a loss function, which can be done by optimization algorithms, such as gradient descent.
- In our DoS attack prediction task, the dataset is $D = \{(x_n, y_n), n = 1,2,\dots,N)\}$, where N=10373 is the sum of normal and attack data, and

$$y_n = \begin{cases} +1, & DoS\ Attack \\ -1, & Regular \end{cases}$$

- The process of training and testing the logistic regression model is as follows.

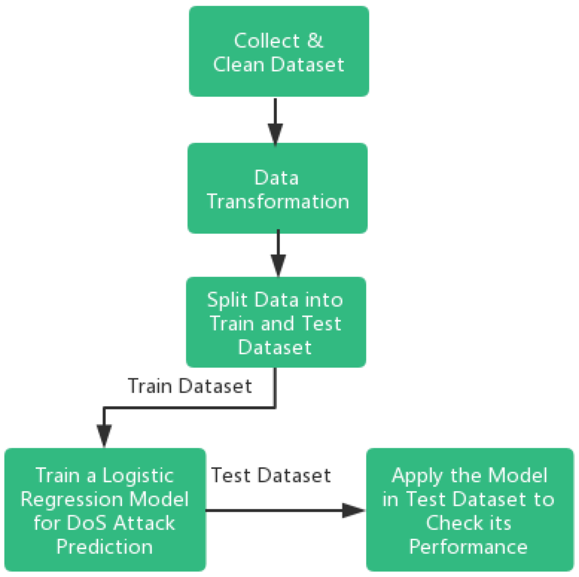

Collect & Clean Dataset
Data Transformation
Split Data into Train and Test Dataset
Train Dataset
Train a Logistic Regression Model for DoS Attack Prediction
Test Dataset
Apply the Model in Test Dataset to Check its Performance

Figure 20: Logistic Regression Model Training Process

3.5.2 Prediction Result

We split the dataset into training and testing dataset and allocated 30% of the data as testing data. We used the prediction accuracy to measure the performance of the model as follows.

$$\text{Accuracy} = \frac{Total\ Misclassification\ Data}{Total\ Data}$$

With the logistic regression model, we get the prediction accuracy of 95%, which is an amazing and promising result. However, 95% accuracy is too good to believe in real-world cases, which may be associated with problems like overfitting. However, still, it shows that it is highly possible that we can use a machine learning algorithm to do cybersecurity work, like the DoS attack prediction and detection in our project.

4. Conclusion

In this project, we conducted a DoS attack simulation and collected data from both the DoS attack and regular network traffic. Then we cleaned, extracted, and transformed data to make them more organized for analyzing. Next, we applied statistic thought to analyze the data, compared the difference between normal and attack packets, and discovered some behavior patterns of DoS attack action. Finally, we implemented a logistic regression model to classify the DoS attack packets automatically with a prediction accuracy of 95%.

Appendix

Python code for Logistic Regression

```python
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

# Reader data from local files
trainData = pd.read_table('trainData.txt', sep='\t')

# Data transformation
one_hot = pd.get_dummies(trainData['TrafficType'])
trainData = trainData.drop('TrafficType', axis = 1)
trainData = trainData.join(one_hot)
print (trainData)

# Shuffle data
trainData = shuffle(trainData)

# Split the data into train dataset and test dataset
train, test = train_test_split(trainData, test_size=0.3)

# generate features and labels
X = train.iloc[: ,[0,1,2,3]]
y = train.iloc[:, [4]]
X_test = test.iloc[: ,[0,1,2,3]]
y_test = test.iloc[:, [4]]

# train Logistic Regression model
clf = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomial').fit(X, y)

# test model results
print(clf.predict_proba(X_test))
print(clf.score(X_test, y_test))
```

Reference

[1] Anjali. M , “Detection of DDoS Attacks based on Network Traffic Prediction and Chaos Theory”, *International Journal of Computer Science and Information Technologies (IJCSIT),* Vol. 5 (5) , 2014, 6502-6505

[2] Keromytis A.D. (2011) In: van Tilborg H.C.A., Jajodia S. (eds), “Network Bandwidth Denial of Service (DoS)”, *Encyclopedia of Cryptography and Security, Springer, Boston, MA*

[3] “How Hackers Take Over Web Sites with SQL Injection and DDoS”, https://www.howtogeek.com/97971/htg-explains-how-hackers-take-over-web-sites-with-sql-injection-ddos/

[4] “Mininet”, https://github.com/mininet

[5] “What is SDN?”, https://www.ciena.com/insights/what-is/What-Is-SDN.html

[6] “iPerf”, https://en.wikipedia.org/wiki/Iperf

[7] “TCP Three way handshake”, https://rumyittips.com/tcp-three-way-handshake/

[8] “Implementation of DoS using hping3/nping”, http://topspeedsnail.com/user-nping-hping3-dos/

[9] Wusheng Lu, “Machine Learning Optimization, Chapter 2”, *University of Victoria*