ENSC 427: COMMUNICATION NETWORKS
SPRING 2023

FINAL PROJECT

Investigation into Wireless Device Botnets

https://www.sfu.ca/~jaforres/

Team Number: 1

Team Members:
Name: Jacob Forrest
Student Number: 301360304
Email: jaforres@sfu.ca

Name: Boris Perdija
Student Number: 301339378
Email: Bperdija@sfu.ca

Table of Contents

Abstract

Since the late 1900's, botnets have emerged as a serious threat to the security of networks and devices across the world. In particular, the growing number of Internet of Things (IoT) devices, such as smart home appliances, security cameras, and medical devices, has provided attackers with a large pool of vulnerable devices to exploit. The objective of this paper is to introduce botnets, how they operate, and to examine what they can be used to do. This paper will briefly discuss the Mirai Botnet as a major motivator for studying these threats, and an ns-3 simulation will be demonstrated to showcase the impact that botnets have when executing a Distributed Denial of Service (DDoS) attack using IoT devices. Ultimately, the results of these experiments showcased that network performance degraded significantly as the number of bots increased, highlighting the potential threat posed by Wireless Device Botnets. Continued research will be needed to address the growing threat of botnets and the development of effective strategies to combat them.

Introduction

Project Overview and Scope

Understanding the impact that IoT botnets can have on network performance is fundamental for recognizing the consequences of their malicious activities and the need for preventative and mitigative measures. As a result, the goal of this project was to gain a better understanding of what botnets are, the damage that they can cause, and to survey current literature regarding botnet traffic mitigation and detection. To gain insight on the effectiveness of these botnets, we utilized ns-3 to simulate and measure the impact of a DDoS attack against a single target node.

With regards to the scope of the simulation, we limited the ns-3 experimentations to simulations of a UDP Flood DDoS attack executed by botnets of wirelessly connected devices. Specifically, we modelled our simulation around a botnet of IoT devices connected to various home Wi-Fi networks. Due to time constraints, we did not implement a botnet traffic detection algorithm into our ns-3 simulations. Through these simulations, we were able to measure the network performance degradation caused by the botnet traffic.

Related Works & Literature Review

There are numerous studies, papers, and surveys regarding botnets that have helped us with this topic.

1. "*Botnet detection: countering the largest security threat*" by Lee, W., Wang, C., and Dagon, D. [1] has been a valuable resource for understanding important botnet concepts and how botnet traffic can be analyzed and detected. The book focuses on methods of detection based on botnet behaviours, as well as using honeynet's to trap or gain information on the attacking botnet.

2. "*An empirical comparison of botnet detection methods*" by S. García, M. Grill, J. Stiborek, and A. Zunino [2] was a paper that provided us a better understanding of botnet detection methods and how they can help prevent events such as the Mirai Botnet. Ultimately, the paper demonstrated that botnet detection methods are extremely difficult to rank and compare, largely due to poor documentation between detection methods, a lack of a common botnet dataset, and a lack of any uniform botnet comparison methodology. After looking at CAMNEP, BClus and BotHunter detection methods, the conclusion made was that comparing methods can be used to obtain a better understanding of detection effectiveness, improve current algorithms, and receive better datasets.

3. "*An Effective Conversation-Based Botnet Detection Method*" was a paper that explored a machine learning approach to detect botnets. Their method considered current flow-based methods (looking at transmission bytes and data packets), as well as a conversation-based detection method. A random forest algorithm was used to detect botnets and the detection rate

was the highest compared to all other methods, at up to 93.6% successful detections [3]. This has illustrated that new detection methods are constantly being released and that machine learning based solutions may be the future of effective detection.

4. "*A survey of DDoS attacking techniques and defence mechanisms in the IoT network*" [4] was a paper more in line with our focus of IoT devices and was useful for discussing DDoS attacks using IoT devices. The paper discusses the Mirai, Wirex, Reaper, and Torii botnets, as well as numerous DDoS defence mechanisms in IoT networks such as learned automation, honeypots, and software defined networking defences. The paper concludes by discussing research issues and challenges regarding DDoS defences in IoT and highlights that IoT environments are constantly changing, there is a lack of a standardized framework for IoT, and costs need to be factored for research.

5. "*Quantifying the reflective DDoS attack capability of household IoT devices*" [5] provided us more information on DDoS attacks using IoT devices, and also discussed a practical experiment that showcased the intensity rate of SNMP and SSDP based attacks on IoT devices such as lightbulbs, webcams, and printers. This paper explored the use of multiple devices acting as reflectors and showcased that eight, consumer level, IoT devices were able to amplify an attack by increasing unwanted traffic on a victim by a factor of 20. No solutions were provided in this paper, but it did demonstrate the danger of multiple compromised IoT devices.

6. "*DDoS in the IoT: Mirai and Other Botnets*" [6] became a valuable resource for understanding the Mirai botnet and how similar botnets can attack IoT devices. The Mirai malware and the resulting devastating botnet attacks ultimately served as a primary motivation for our experiments. More about the Mirai Botnet will be covered in a subsequent section.

Further information

A definition of Botnets

Botnets are networks of compromised devices, known as bots, controlled by a single attacking party to execute synchronized malicious activities. Ultimately, these bots "are computers infected with malicious program(s) that cause them to operate against the owners' intentions and without their knowledge" [1]. Any malware associated with botnets is often aimed at infecting as many devices as possible without their owner's awareness.

There are many methods of spreading botnet malware. Namely, they can be acquired from email attachments, malicious links, false software updates, infected downloads, pop-ups, and software engineering attacks.

Botnets are predominantly used for phishing attacks, financial breachers (crypto jacking and credential harvesting), and can cause serious disruptions to businesses, lost financial income, and compromised user information. Botnets are also largely responsible for DDoS attacks which will serve the basis of this paper.

Motivation for studying Botnets: Mirai Botnet

In 2016, the Mirai Botnet caused significant damage to numerous websites and internet infrastructure and ultimately led to the largest DDoS attack in History. This botnet was designed to infect vulnerable IoT devices and successfully used devices such as routers, webcams, and digital video recorders (DVR's) to launch several DDoS attacks [6].

The Mirai botnet was largely successful due to many IoT devices using default login credentials and lacking any meaningful security updates, making brute force logins relatively easy.

Understanding the techniques and tactics that botnets use with IoT devices will allow researchers and security professionals to develop more effective ways to prevent, detect, and mitigate similar attacks in the future. In addition, the Mirai Botnet provided us motivation to examine the impact of a DDoS attack through ns-3 simulations.

Ns-3 Experiments

Scenario A: IoT Botnet DDoS Simulation

Overview

The primary goal of our simulation was to emulate the infamous Mirai IoT botnet and investigate the impact of a DDoS attack that is launched from a similar network of IoT-based smart home consumer devices.

We used the ns-3 network simulator to simulate the DDoS attack. In our simulations, the botnet nodes were connected to Wi-Fi networks and generated UDP traffic to at a constant bitrate to flood and overwhelm the victim node. A single legitimate user competed with the botnet for network bandwidth for a TCP connection with the victim node. We ran the simulation with a varying number of infected Wi-Fi networks and botnet nodes. Specifically, we ran the simulation with 0, 4, 8, 10, 12, and 16 bots.

Our experiment aimed to assess the impact to performance of the legitimate user's TCP connection with the victim node during the DDoS attack. Specifically, we measured the congestion window size over time, the mean packet delay, the packet loss ratio, and the transmitting & receiving bitrates.

Ultimately, our simulations show that the botnet is very effective at denying the legitimate user from service with targeted victim server.

Topology

A diagram of the simulated network topology is showcased in Figure 1. The red boxes signify infected IoT devices. The green box indicates the single legitimate user. Each Wi-Fi network consisted of two bot nodes. In the simulation, the server node is the recipient of both the botnet's generated traffic as well as the legitimate user's TCP connection traffic. All point-to-point links in the network were configured for a bandwidth of 100 Mbps, a delay of 2ms, and implemented a Drop Tail Queue of 100 packets. Additionally, a channel error rate model was implemented with a probability of 0.0001.

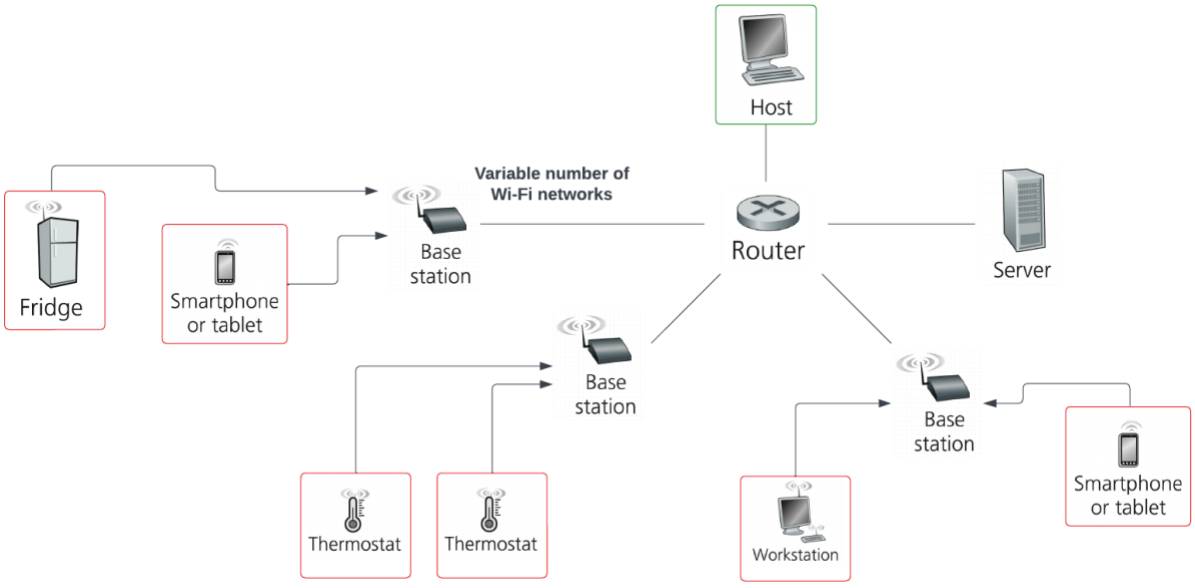

Figure 1: Simulation Topology

Applications

For the legitimate user, we simulated a TCP connection with the victim server. The TCP connection transmitted 1024-byte packets with a data rate of 100 Mbps. For the bot nodes, we simulated a UDP flood application, which transmitted 1024-byte packets at a constant bit rate of 10 Mbps.

Results

The resulting measurements from our simulations are plotted in figures 2 through 11. The results show that as the volume of botnet traffic increases, the network's performance degrades rapidly. These results indicate that effective DDoS mitigation strategies, such as traffic filtering, firewalls, and intrusion detection systems, should be implemented to protect against such attacks as they can cause serious outages for legitimate customers.

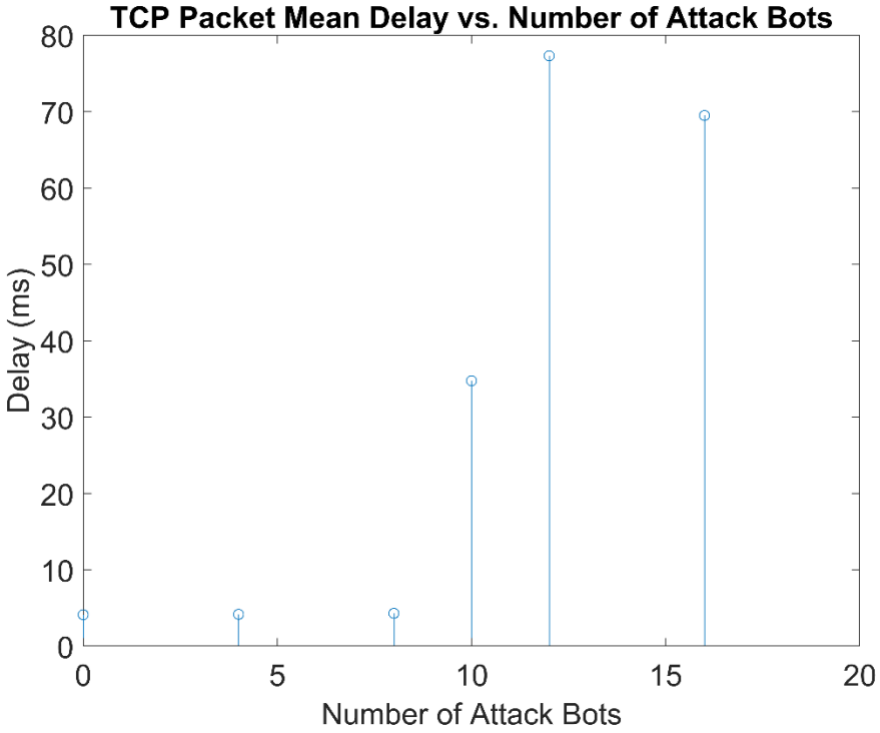

Figure 2: Scenario A – TCP Mean Delay Results

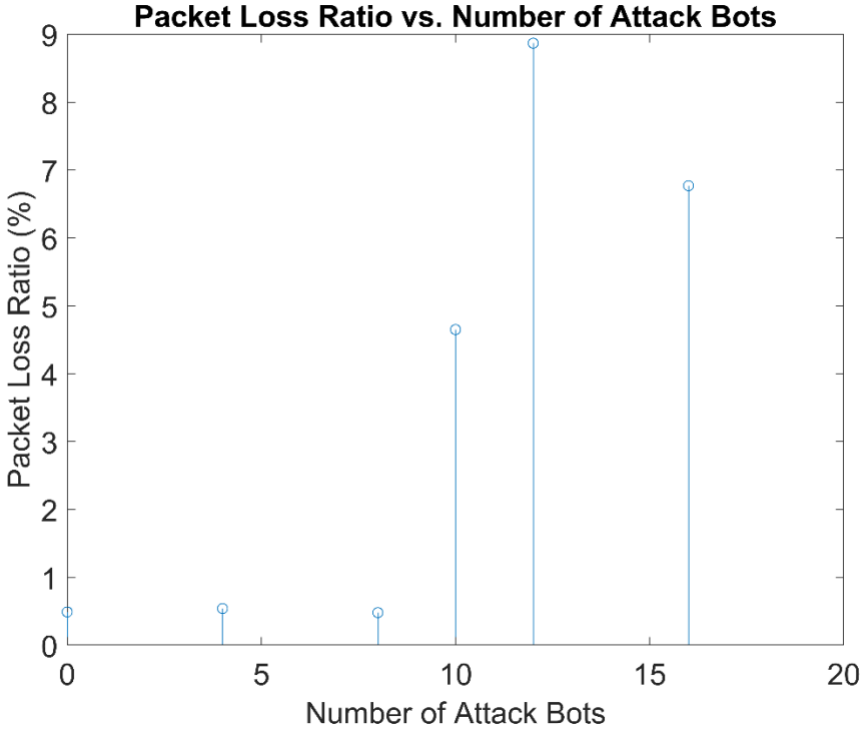

Figure 3: Scenario A – TCP Packet Loss Ratio Results

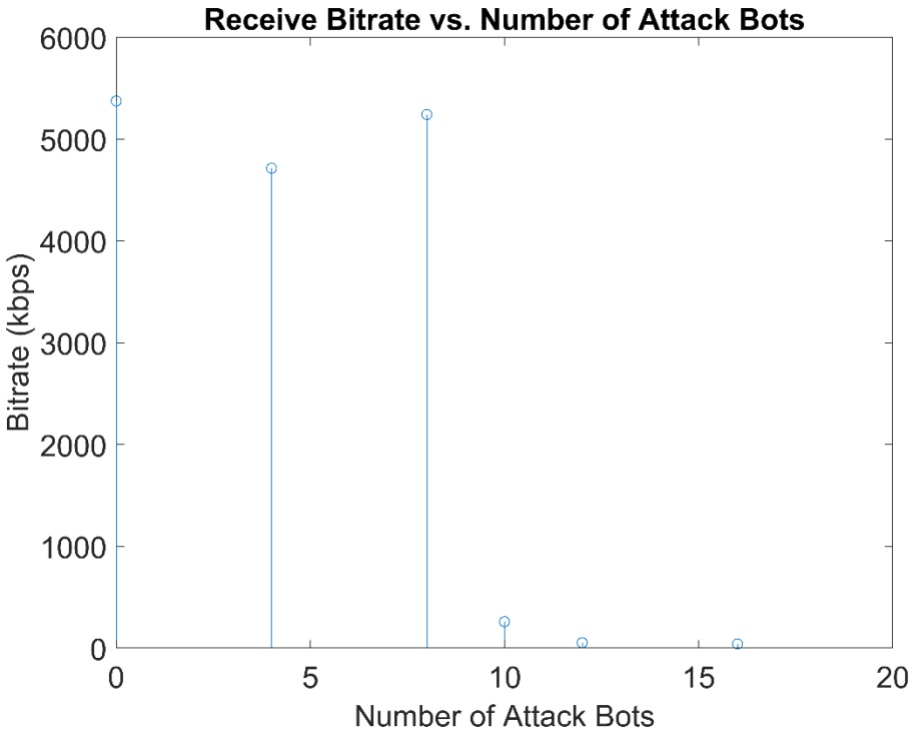

Figure 4: Scenario A – TCP RX Bitrate Results

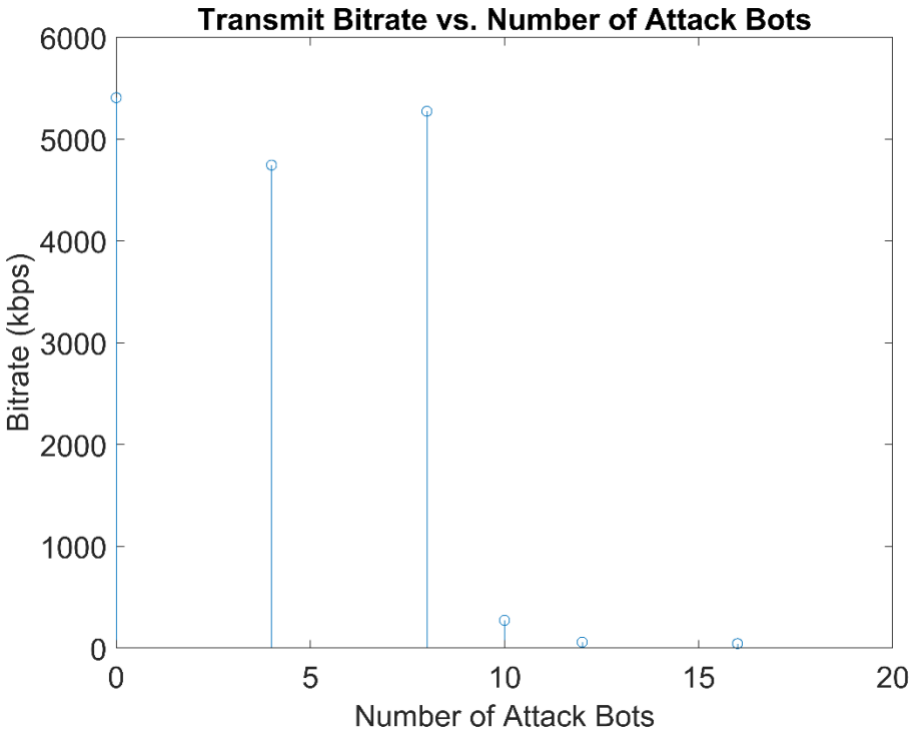

Figure 5: Scenario A – TCP TX Bitrate Results

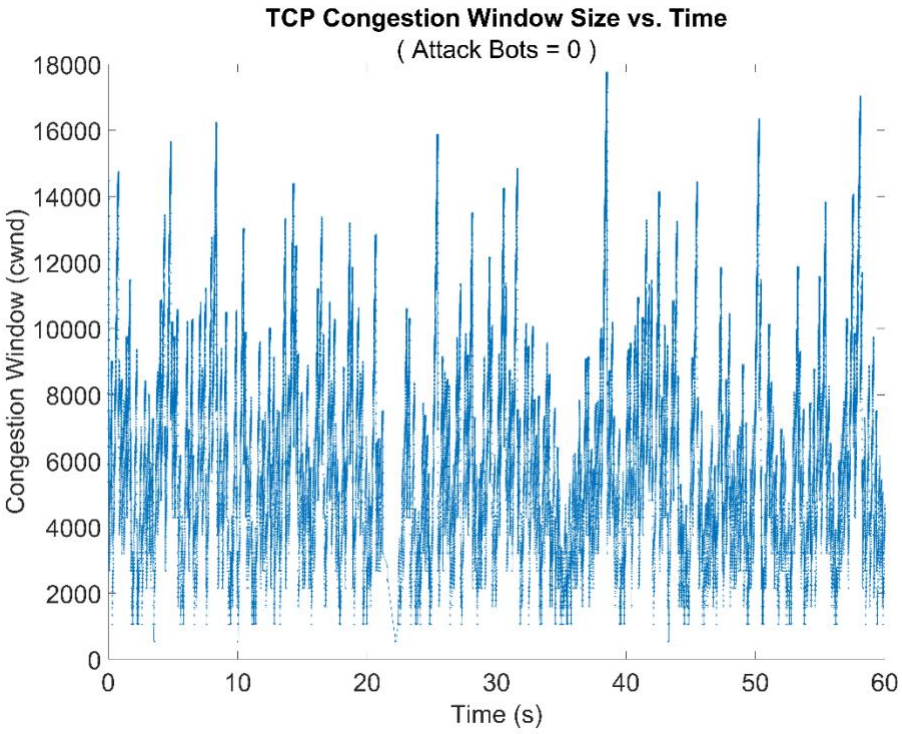

Figure 6: Scenario A – TCP Congestion Window Results – 0 Bots

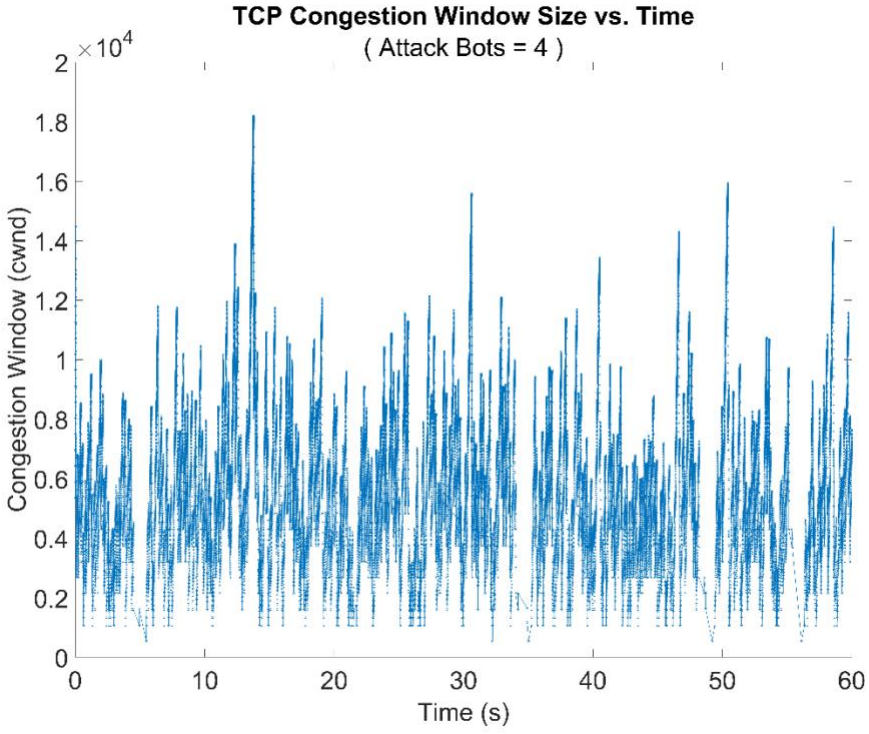

Figure 7: Scenario A – TCP Congestion Window Results – 4 Bots

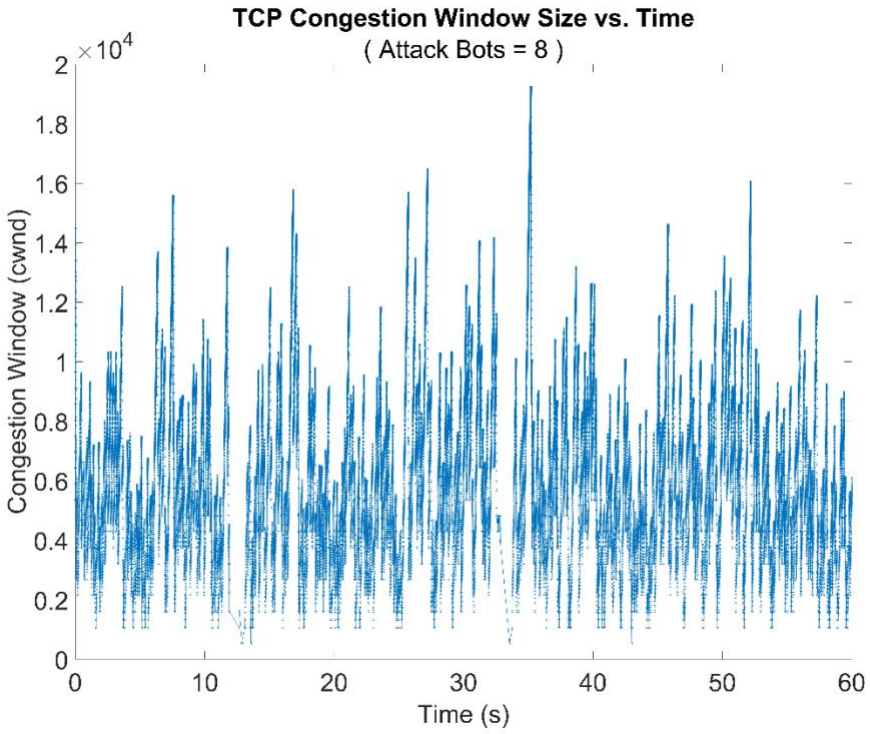

Figure 8: Scenario A – TCP Congestion Window Results – 8 Bots

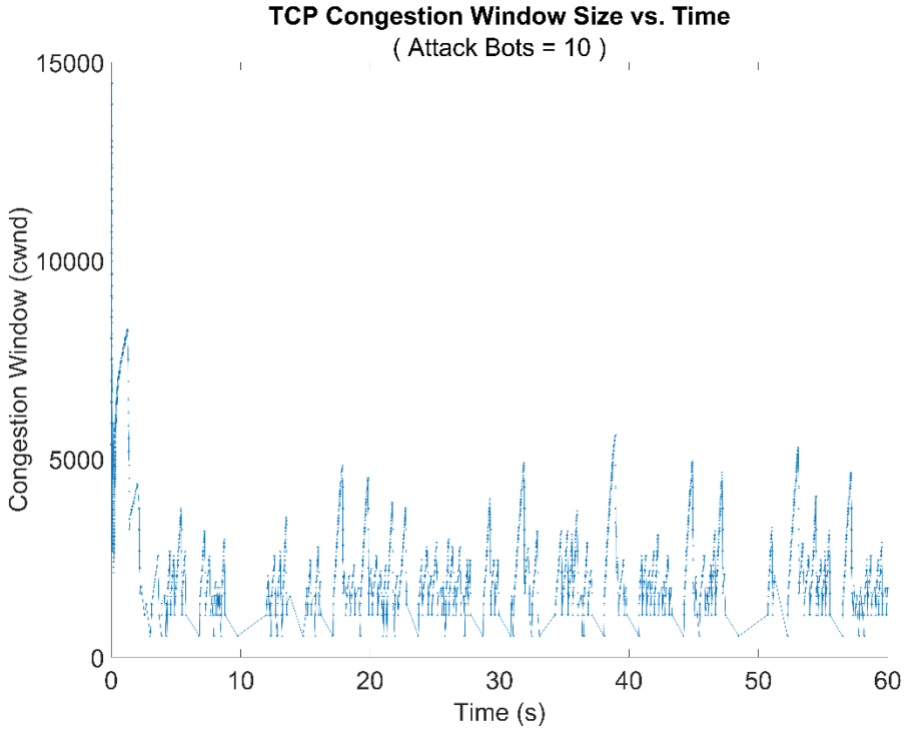

Figure 9: Scenario A – TCP Congestion Window Results – 10 Bots

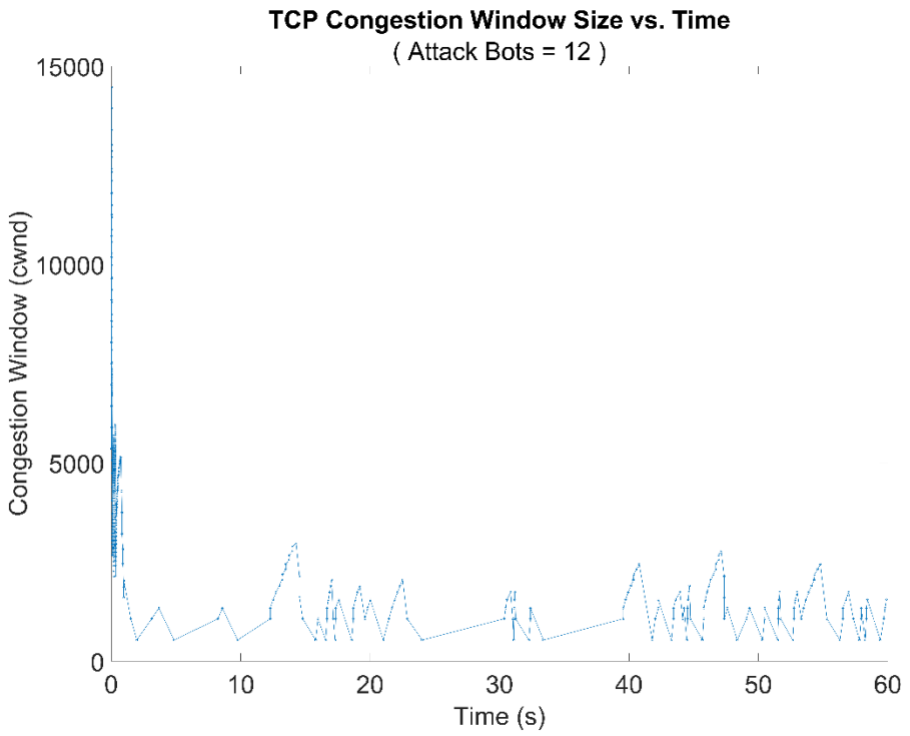

Figure 10: Scenario A – TCP Congestion Window Results – 12 Bots

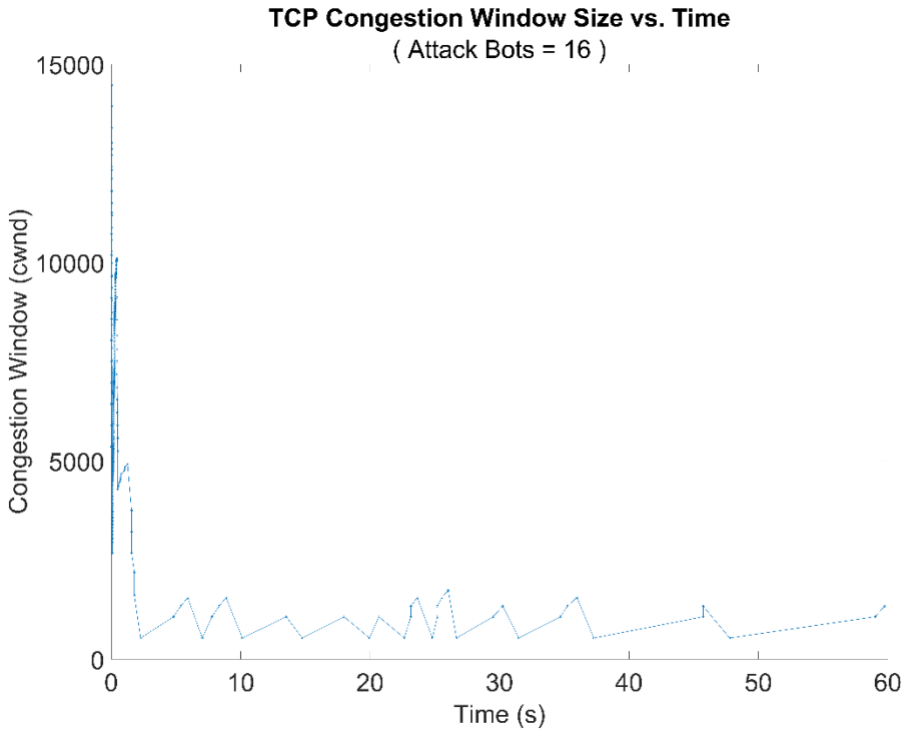

Figure 11: Scenario A – TCP Congestion Window Results – 16 Bots

Scenario B: IoT Botnet DDoS Simulation with Background Wi-Fi Traffic

Overview

In scenario A, all devices in the Wi-Fi networks were active bots. However, in a real home Wi-Fi network, there are typically many connected devices, and it is unlikely that all of them are infected with botnet malware. Therefore, the goal of scenario B was to add background traffic to each Wi-Fi network to create a more realistic network simulation.

Scenario B was also implemented with the ns-3 network simulator. The traffic generating applications installed on the original botnet nodes and the original legitimate user node were unchanged. In addition to the two bot nodes in each Wi-Fi network, four legitimate Wi-Fi nodes were added to generate background UDP traffic. In this simulation, the bot nodes must compete for bandwidth in the Wi-Fi network with the legitimate Wi-Fi nodes.

For scenario B, we conducted the simulation with varying numbers of infected Wi-Fi networks and botnet nodes. Specifically, we ran the simulation with 0, 20, 40, 80, 160, and 240 bots. We measured the same performance indicators of the legitimate user's TCP connection with the victim node as we did in scenario A.

The results of scenario B will show that the performance of the botnet's DDoS attack is significantly poorer with the addition of background traffic in each Wi-Fi network.

Topology

A diagram of the simulated network topology for this experiment is showcased in Figure 12. The red boxes indicate the network's bot nodes, and the green boxes indicate that the node is generating legitimate user traffic. The purple box indicates the DDoS attack victim that is being targeted with UDP packets by the bot nodes. The TCP connection being observed and measured in the simulation is between the DDoS victim and the point-to-point connected legitimate user node. The blue boxes indicate the server that will receive only the traffic generated by the legitimate nodes in each Wi-Fi network. Each Wi-Fi network consists of two bot nodes, and four legitimate user nodes. All point-to-point links in the network were configured for a bandwidth of 100 Mbps, a delay of 2ms, and implemented a Drop Tail Queue of 100 packets. Additionally, a channel error rate model was implemented with a probability of 0.0001.

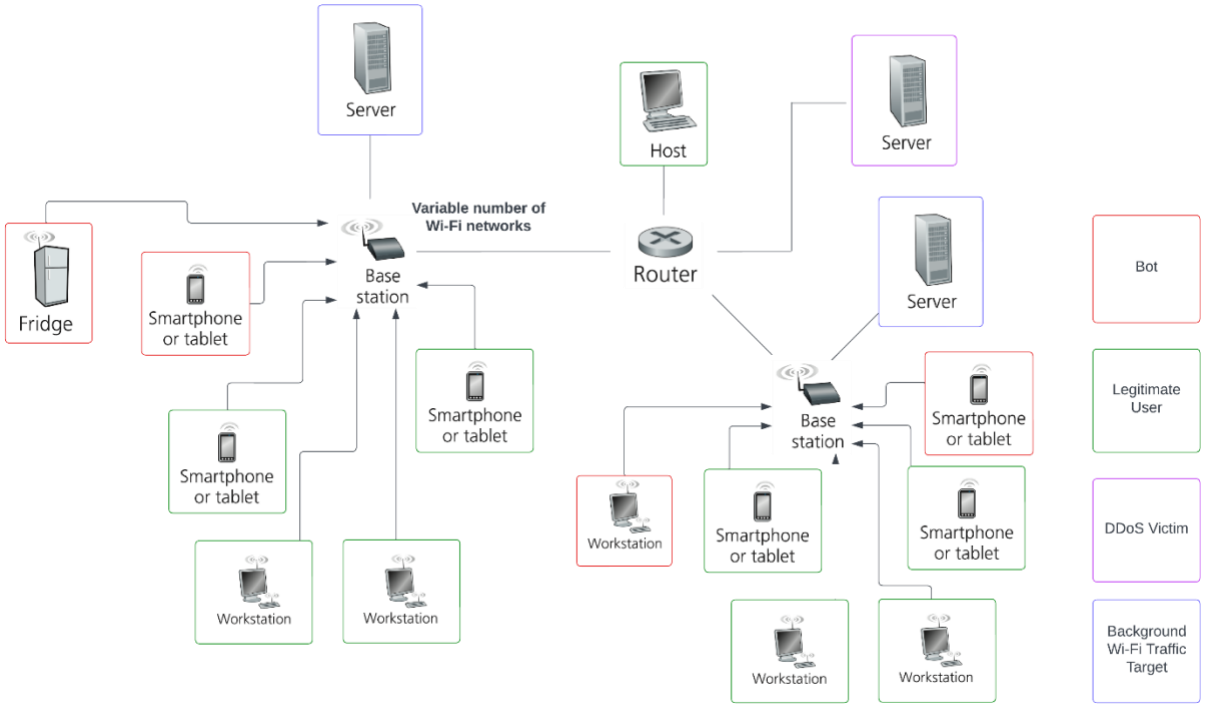

Figure 12: Simulation Topology (Background Wi-Fi Traffic)

Applications

For the point-to-point connected legitimate user node, we simulated a TCP connection with the victim server. The TCP connection transmitted 1024-byte packets with a data rate of 100 Mbps. For the bot Wi-Fi nodes, we simulated a UDP flood application, which transmitted 1024-byte packets at a constant bit rate of 10 Mbps.

For the real Wi-Fi user nodes, we simulated background traffic via an ns-3 onoff UDP application with a duty cycle of 50% (5 seconds on, 5 seconds off). The application was configured to send 1024-byte packets at a rate of 10 Mbps to a specific server associated with each Wi-Fi network.

Results

The resulting measurements from the simulations are plotted in figures 13 through 22. The results show that, compared to scenario A, the number of bots required to effectively deny bandwidth to the legitimate TCP connection was much higher. This is likely because the bots must compete for bandwidth with the legitimate users within their Wi-Fi networks.

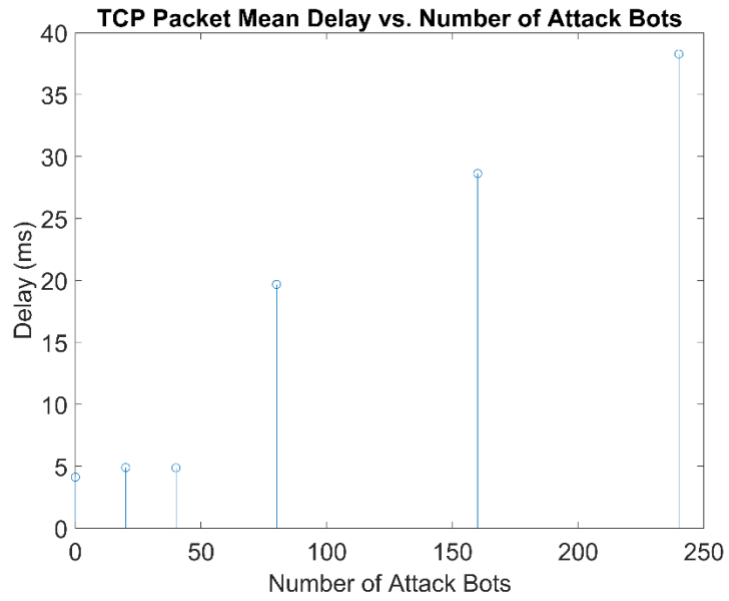

Figure 13: Scenario B – TCP Mean Delay Results

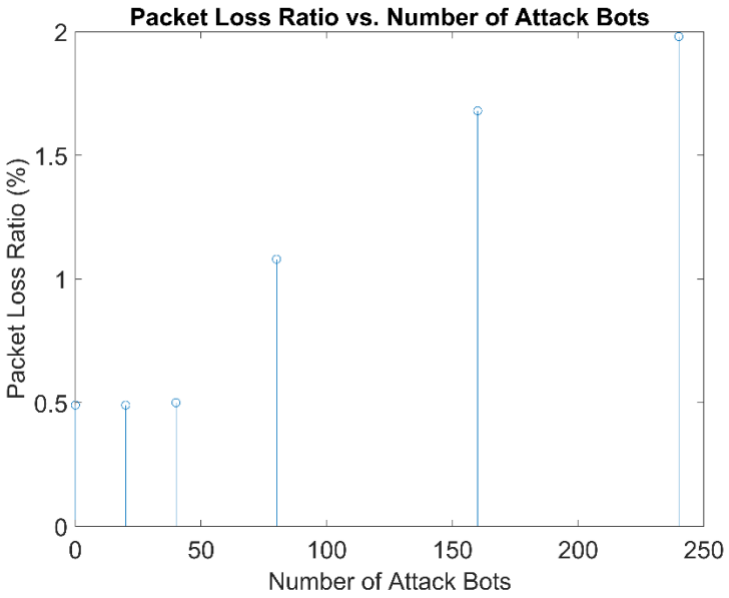

Figure 14: Scenario B – TCP Packet Loss Ratio Results

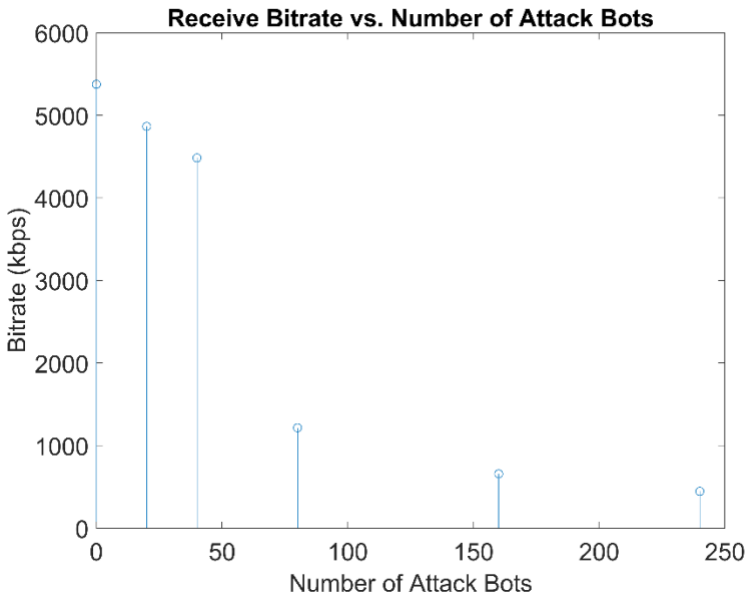

Figure 15: Scenario B – TCP RX Bitrate Results

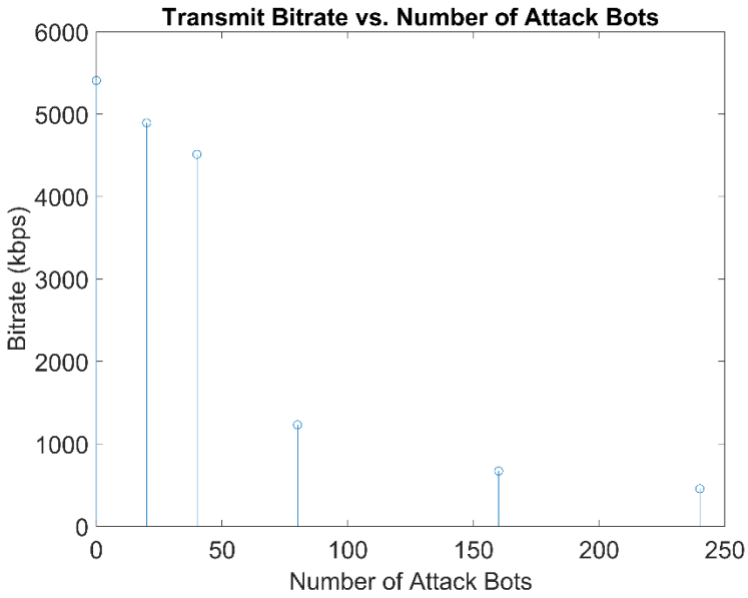

Figure 16: Scenario B – TCP TX Bitrate Results

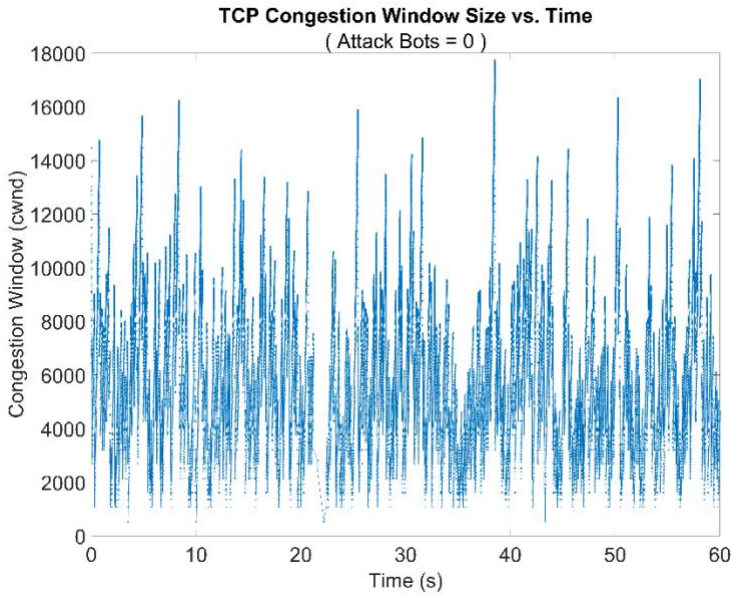

Figure 17: Scenario B – TCP Congestion Window Results – 0 Bots

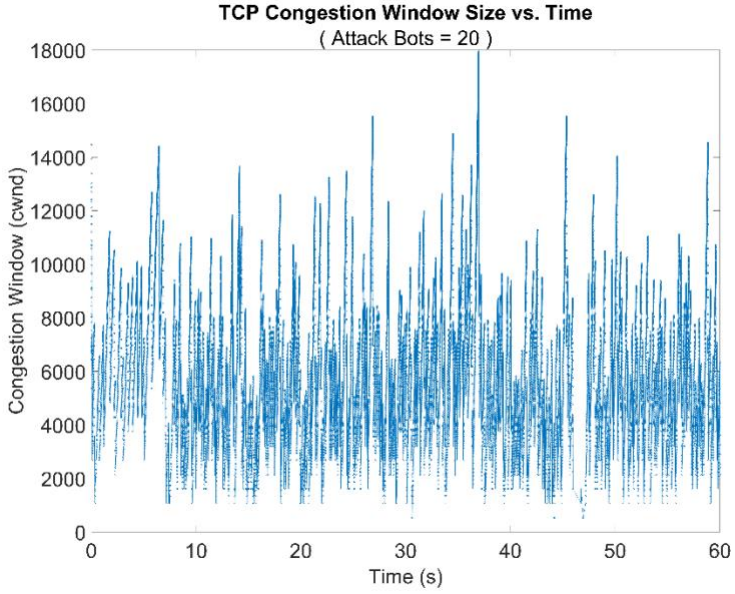

Figure 18: Scenario B – TCP Congestion Window Results – 20 Bots

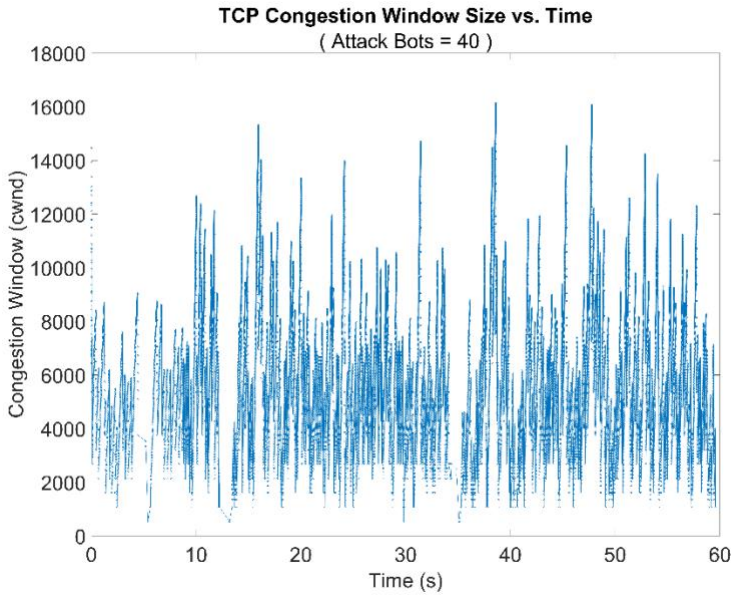

Figure 19: Scenario B – TCP Congestion Window Results – 40 Bots

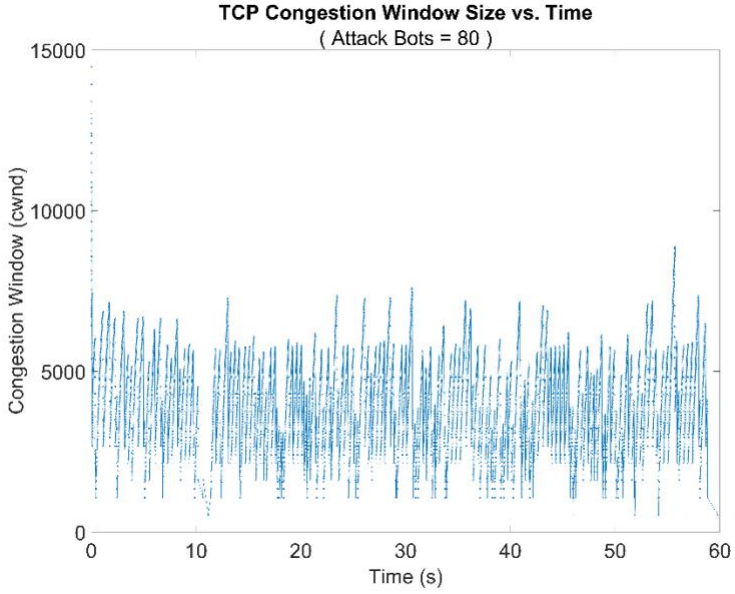

Figure 20: Scenario B – TCP Congestion Window Results – 80 Bots

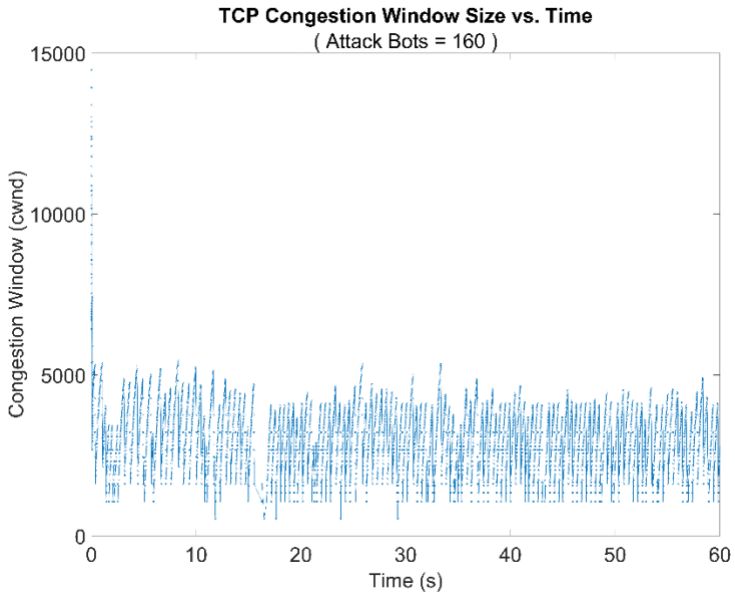

Figure 21: Scenario B – TCP Congestion Window Results – 160 Bots

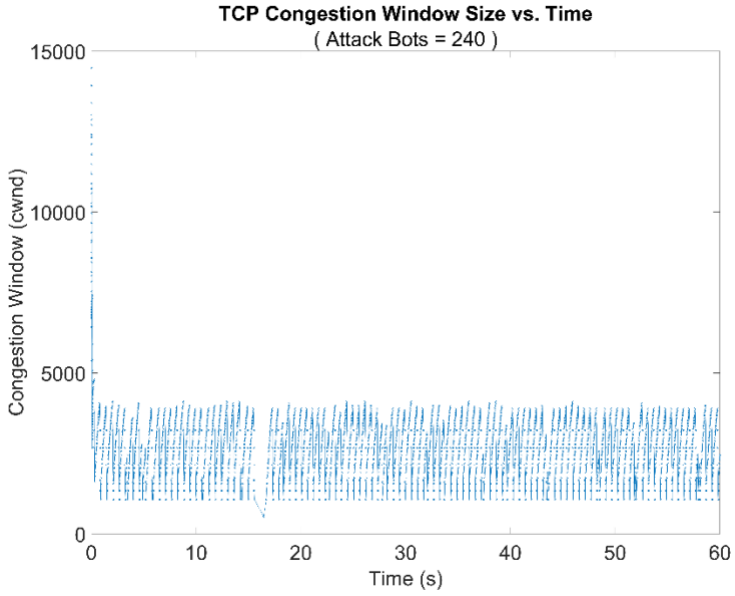

Figure 22: Scenario B – TCP Congestion Window Results – 240 Bots

Discussion

Summary

This paper introduces the concept of botnets, how they operate, and their potential for causing harm, especially in the context of IoT devices. We have discussed the Mirai Botnet and provided a brief showcase of current literary works that look at and study botnets in today's modern age. Additionally, we have demonstrated the impact of botnets through ns-3 simulations of a UDP Flood DDoS attack on a single target node.

Tracing

PCAP Tracing

PCAP tracing is a feature in ns-3 that captures and saves network packets generated during simulations. In our project, we used it to trace packets transmitted over the point-to-point net devices and AP Wi-Fi net devices.

Flow Monitor

The ns-3 FlowMonitor is a tool used to monitor traffic flows in a simulated network. For our simulations, we used the FlowMonitor ns3 module to record statistics of the TCP connection's traffic flow such as the TX bitrate, the RX bitrate, the Mean Delay, and the Packet Loss Ratio.

The following table contains the raw data for these statistics from our two simulation scenarios:

Scenario A		**Scenario B**	
Number of Bots	**Statistics**	**Number of Bots**	**Statistics**
0	TX bitrate: 5407.28 kbit/s RX bitrate: 5374.76 kbit/s Mean Delay: 4.12 ms Packet Loss Ratio: 0.49 %	0	TX bitrate: 5407.28 kbit/s RX bitrate: 5374.76 kbit/s Mean Delay: 4.12 ms Packet Loss Ratio: 0.49 %
4	TX bitrate: 4745.33 kbit/s RX bitrate: 4714.12 kbit/s Mean Delay: 4.17 ms Packet Loss Ratio: 0.54 %	20	TX bitrate: 4893.71 kbit/s RX bitrate: 4864.25 kbit/s Mean Delay: 4.89 ms Packet Loss Ratio: 0.49 %
8	TX bitrate: 5273.58 kbit/s RX bitrate: 5242.14 kbit/s Mean Delay: 4.30 ms Packet Loss Ratio: 0.48 %	40	TX bitrate: 4511.90 kbit/s RX bitrate: 4484.12 kbit/s Mean Delay: 4.87 ms Packet Loss Ratio: 0.50 %

10	TX bitrate: 273.72 kbit/s RX bitrate: 260.58 kbit/s Mean Delay: 34.76 ms Packet Loss Ratio: 4.65 %	80	TX bitrate: 1230.11 kbit/s RX bitrate: 1215.23 kbit/s Mean Delay: 19.68 ms Packet Loss Ratio: 1.08 %
12	TX bitrate: 59.38 kbit/s RX bitrate: 53.89 kbit/s Mean Delay: 77.32 ms Packet Loss Ratio: 8.87 %	160	TX bitrate: 669.46 kbit/s RX bitrate: 657.31 kbit/s Mean Delay: 28.62 ms Packet Loss Ratio: 1.68 %
16	TX bitrate: 45.28 kbit/s RX bitrate: 42.28 kbit/s Mean Delay: 69.50 ms Packet Loss Ratio: 6.77 %	240	TX bitrate: 455.70 kbit/s RX bitrate: 446.19 kbit/s Mean Delay: 38.28 ms Packet Loss Ratio: 1.98 %

Application - Trace Callbacks

The ns-3 simulator provides the 'trace callback' mechanism which enables applications to call specific callback functions whenever a specified event occurs or a variable changes value.

In our project we made use of trace callback mechanism to record updates to the congestion window size and the round-trip time of the TCP traffic flow throughout the duration of the simulations. We implemented a custom application based on the code provided in the fifth.cc script of the ns-3 tutorial. For our custom application, we created trace sink functions that output changes to the 'cwnd' and 'rtt' variables of the TCP flow to separate output files and configured them to be called whenever the corresponding variables are updated during the simulation. By using these trace callbacks, we were able record the behavior of the TCP flow during the simulation and then plot this data using a Matlab script.

Future Work & Improvements

There are several potential areas for future work and improvements to our current network simulation. One future improvement that we would like to make would be to implement a botnet traffic detection algorithm, which could help to detect and mitigate the effects of the simulated DDoS attack. This proved to be a non-trivial task, as we would have needed more time to study the latest botnet attack methods, and have needed additional expertise in machine learning, packet inspection, and behavioral analysis techniques. Additionally, we could consider allowing for a variable number of bot devices and legitimate traffic devices on each Wi-Fi network, which would help to make the simulation more realistic and allow us to study the impact of different network configurations on the effectiveness of botnet attacks. Finally, we could expand our simulation by implementing botnet devices on other wireless technologies, such as LoRaWAN and ZigBee, which would allow us to study the impact of botnets on a wider range of networks and devices.

Conclusion

Botnets are a serious threat in the world of cybersecurity. They consist of networks of computer devices that are compromised and under the control of a master attacker. As IoT devices continue to become a commonality, botnets have the potential of becoming a much larger and more dangerous threat in modern day society. The expanse of the Internet of Things has provided a vast number of vulnerable devices for botnet networks to target and control, ranging from smart home devices to industrial control systems. Once a botnet is established, it can be used for a variety of malicious purposes, including DDoS attacks, financial breaches, targeted intrusions, and more. Our research using ns-3 simulation has demonstrated how a botnet consisting of wireless devices can be used to flood a network with UDP packets, significantly reducing the bandwidth available to legitimate users. Our simulations in scenario B represented a more realistic network with the addition of legitimate user traffic in each Wi-Fi network. In this more realistic scenario, we observed that the DDoS performance was significantly worse as the bot nodes needed to compete for bandwidth in the Wi-Fi networks. Therefore, a real botnet needs many nodes to effectively execute a DDoS attack. Ultimately, these studies and experiments are vital for understanding how botnets operate and, in the future, we would like to research additional detection and prevention methods.

References

Bibliography

[1] W. Lee, C. Wang and D. Dagon, Botnet Detection: Countering the Largest Security Threat, Springer New York, NY, 2008.

[2] S. Garcia, M. Grill, J. Stiborek and A. Zunino, "An empirical comparison of botnet detection methods," *Computers & Security 45,* pp. 100-123, 2014.

[3] R. Chen, W. Niu, X. Zhang, Z. Zhuo and F. Lv, "An Effective Conversation-Based Botnet Detection Method," 2017.

[4] R. Vishwakarma and A. Jain, "A survey of DDoS attacking techniques and defence mechanisms in the IoT network," *Telecommun Syst,* vol. 73, pp. 3-25, 2020.

[5] M. Lyu, D. Sherratt, A. Sivanathan, H. H. Gharakheili, A. Radford and V. Sivaraman, "Quantifying the reflective DDoS attack capability of household IoT devices," *WiSec '17: Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks,* pp. 46-51, 2018.

[6] C. Kolias, G. Kambourakis, A. Stavrou and J. Voas, "DDoS in the IoT: Mirai and Other Botnet," *Computer (Long Beach, Calif.),* vol. 50, no. 7, pp. 80-84, 2017.

Contributions

Task	**Jacob Forrest**	**Boris Perdija**
Literature Review (References)	30%	70%
Project Website	70%	30%
Simulation Scenarios	80%	20%
Implementation	70%	30%
Analysis	70%	30%
Discussion of Simulation Results	60%	40%
Project Presentation	35%	65%
Written Final Report	40%	60%
Total	57%	43%

Appendix A: Code Listing

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
#include "ns3/yans-wifi-helper.h"
#include "ns3/ssid.h"
#include "ns3/mobility-module.h"
#include "ns3/trace-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/flow-monitor-helper.h"


#define BACKGROUND_TRAFFIC

#define NUM_BOT_NETWORKS 120
#define SIM_TIME 60
#define NUM_PACKETS 5000000
#define DATA_RATE "100Mbps"
#define TCP_RATE "100Mbps"
#define DDOS_RATE "10Mbps"
#define BACKGROUND_RATE "10Mbps"
#define CHANNEL_DELAY "2ms"
#define PORT 25565
#define PORT2 25566

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("BotNet");

class MyApp : public Application
{
public:

  MyApp ();
  virtual ~MyApp();

  void Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate);

private:
  virtual void StartApplication (void);
  virtual void StopApplication (void);
```

```
  void ScheduleTx (void);
  void SendPacket (void);

  Ptr<Socket>     m_socket;
  Address         m_peer;
  uint32_t        m_packetSize;
  uint32_t        m_nPackets;
  DataRate        m_dataRate;
  EventId         m_sendEvent;
  bool            m_running;
  uint32_t        m_packetsSent;
};

MyApp::MyApp ()
  : m_socket (0),
    m_peer (),
    m_packetSize (0),
    m_nPackets (0),
    m_dataRate (0),
    m_sendEvent (),
    m_running (false),
    m_packetsSent (0)
{
}

MyApp::~MyApp()
{
  m_socket = 0;
}

void
MyApp::Setup (Ptr<Socket> socket, Address address, uint32_t packetSize, uint32_t nPackets, DataRate dataRate)
{
  m_socket = socket;
  m_peer = address;
  m_packetSize = packetSize;
  m_nPackets = nPackets;
  m_dataRate = dataRate;
}

void
MyApp::StartApplication (void)
{
  m_running = true;
  m_packetsSent = 0;
```

```
  m_socket->Bind ();
  m_socket->Connect (m_peer);
  SendPacket ();
}

void
MyApp::StopApplication (void)
{
  m_running = false;

  if (m_sendEvent.IsRunning ())
    {
      Simulator::Cancel (m_sendEvent);
    }

  if (m_socket)
    {
      m_socket->Close ();
    }
}

void
MyApp::SendPacket (void)
{
  Ptr<Packet> packet = Create<Packet> (m_packetSize);
  m_socket->Send (packet);

  if (++m_packetsSent < m_nPackets)
    {
      ScheduleTx ();
    }
}

void
MyApp::ScheduleTx (void)
{
  if (m_running)
    {
      Time tNext (Seconds (m_packetSize * 8 / static_cast<double> (m_dataRate.GetBitRate ())));
      m_sendEvent = Simulator::Schedule (tNext, &MyApp::SendPacket, this);
    }
}

static void
CwndChange (Ptr<OutputStreamWrapper> stream, uint32_t oldCwnd, uint32_t newCwnd)
```

```
{
  //NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << newCwnd);
  *stream->GetStream() << Simulator::Now().GetSeconds() << "\t" << newCwnd << std::endl;
}

static void
RttChange (Ptr<OutputStreamWrapper> stream, Time oldRtt, Time newRtt)
{
  //NS_LOG_UNCOND (Simulator::Now ().GetSeconds () << "\t" << rtt);
  *stream->GetStream() << Simulator::Now().GetSeconds() << "\t" << newRtt << std::endl;
}

/*
static void
RxDrop (Ptr<const Packet> p)
{
  NS_LOG_UNCOND ("RxDrop at " << Simulator::Now ().GetSeconds ());
}
*/


int
main (int argc, char *argv[])
{
  CommandLine cmd (__FILE__);
  cmd.Parse (argc, argv);

  Time::SetResolution (Time::NS);

  AsciiTraceHelper asciiTraceHelper;
  Ptr<OutputStreamWrapper> stream1 = asciiTraceHelper.CreateFileStream ("tcp_cwnd.dat");
  Ptr<OutputStreamWrapper> stream2 = asciiTraceHelper.CreateFileStream ("tcp_rtt.dat");

  NodeContainer Nodes;
  Nodes.Create (3);


  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate", StringValue (DATA_RATE));
  pointToPoint.SetChannelAttribute ("Delay", StringValue (CHANNEL_DELAY));
  pointToPoint.SetQueue ("ns3::DropTailQueue", "MaxSize", StringValue ("100p"));

  NetDeviceContainer devices_n1_n0;
  devices_n1_n0 = pointToPoint.Install (Nodes.Get(1), Nodes.Get(0));
```

```
//   n0
//    |
//    |
//   n1
//
//   point-to-point


NetDeviceContainer devices_n1_n2;
devices_n1_n2 = pointToPoint.Install (Nodes.Get(1), Nodes.Get(2));

Ptr<RateErrorModel> em = CreateObject<RateErrorModel> ();
em->SetAttribute ("ErrorRate", DoubleValue (0.00001));
devices_n1_n2.Get (1)->SetAttribute ("ReceiveErrorModel", PointerValue (em));


//   n0
//    |
//    |
//   n1 ---------------- n2
//
//   point-to-point



NodeContainer Bot_AP_Nodes;
Bot_AP_Nodes.Create (NUM_BOT_NETWORKS);
NetDeviceContainer devices_Bot_AP[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  devices_Bot_AP[i] = pointToPoint.Install (Nodes.Get(1), Bot_AP_Nodes.Get(i));
}

//                                n0
//                                 |
//                                 |
//   b0_0 ---------------- n1 ---------------- n2
//                               / |
//                              /  |
//                             /   |
//                           ...   |
//                               bn_0
//
//                point-to-point
```

```
NodeContainer J_Nodes;
J_Nodes.Create(NUM_BOT_NETWORKS);
NetDeviceContainer devices_J_Nodes[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  devices_J_Nodes[i] = pointToPoint.Install (Bot_AP_Nodes.Get(i), J_Nodes.Get(i));
}

//                              n0
//                               |
//                               |
//   b0_0 ---------------- n1 ---------------- n2
//    |                      /    |
//    |                     /     |
//    |                    /      |
//    j0               ...        |
//                            bn_0 --------------- jn
//
//                      point-to-point



NodeContainer wifi_AP_Nodes[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  wifi_AP_Nodes[i] = Bot_AP_Nodes.Get(i);
}


NodeContainer bot_wifiStaNodes[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  bot_wifiStaNodes[i].Create (2);
}
```

```
//    WiFi                        point-to-point
 //   *          *            *                      n0
 //   |          |            |                       |
 //   |          |            |                       |
 // b0_2,  b0_1,       b0_0 ---------------- n1 ---------------- n2
 //                           |                   /   |
 //                           |                  /    |
 //                           |                 /     |
 //                          j0              /        |
 //                                         ...       |
 // bn_2,  bn_1,                                bn_0 --------------- jn
 //     |        |                                    |
 //     |        |                                    |
 //     *        *                                    *
 //    WiFi


NodeContainer real_wifiStaNodes[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  real_wifiStaNodes[i].Create (4);
}

//                  WiFi                      Point-To-Point
//    real users                    bots               AP
//    *            *              *          *            *                    n0
//    |            |              |          |            |                     |
//    |            |              |          |            |                     |
//  r0_0, ..., r0_7,    ...,     b0_2,  b0_1,      b0_0 ---------------- n1 ---------------- n2
//                                          |                              /    |
//                                          |                             /     |
//                                          |                            /      |
//                                         j0                         /       |
//                                                                   ...        |
//  rn_0, ..., rn_7,    ...,     bn_2,  bn_1,                          bn_0 --------------- jn
//    |            |                |        |                                 |
//    |            |                |        |                                 |
//    *            *                *        *                                 *
//    real users                    bots                                      AP
//                  WiFi
```

```
//   set up Wi-Fi network
YansWifiChannelHelper channels[NUM_BOT_NETWORKS];
YansWifiPhyHelper phy[NUM_BOT_NETWORKS];
WifiHelper wifi[NUM_BOT_NETWORKS];
WifiMacHelper mac[NUM_BOT_NETWORKS];
Ssid ssid[NUM_BOT_NETWORKS];
NetDeviceContainer bot_staDevices[NUM_BOT_NETWORKS];
NetDeviceContainer real_staDevices[NUM_BOT_NETWORKS];
NetDeviceContainer bot_apDevices[NUM_BOT_NETWORKS];

MobilityHelper mobility;
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");

for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
 channels[i] = YansWifiChannelHelper::Default ();
 phy[i].SetChannel (channels[i].Create ());
 wifi[i].SetRemoteStationManager ("ns3::AarfWifiManager");
 ssid[i] = Ssid ("ns-3-ssid");
 mac[i].SetType ("ns3::StaWifiMac",
          "Ssid", SsidValue (ssid[i]),
          "ActiveProbing", BooleanValue (false));
 bot_staDevices[i] = wifi[i].Install (phy[i], mac[i], bot_wifiStaNodes[i]);
 real_staDevices[i] = wifi[i].Install (phy[i], mac[i], real_wifiStaNodes[i]);
 mac[i].SetType("ns3::ApWifiMac",
           "Ssid", SsidValue (ssid[i]));
 bot_apDevices[i] = wifi[i].Install (phy[i], mac[i], wifi_AP_Nodes[i]);
 mobility.Install (bot_wifiStaNodes[i]);
 mobility.Install (real_wifiStaNodes[i]);
 mobility.Install (wifi_AP_Nodes[i]);

}
InternetStackHelper stack;
stack.Install (Nodes);
stack.Install (Bot_AP_Nodes);
stack.Install (J_Nodes);

for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
 stack.Install (bot_wifiStaNodes[i]);
 stack.Install (real_wifiStaNodes[i]);
}

Ipv4AddressHelper address;
```

```
// Assign IP to nodes
address.SetBase ("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces_n1_n0;
interfaces_n1_n0 = address.Assign (devices_n1_n0);

address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces_n1_n2;
interfaces_n1_n2 = address.Assign (devices_n1_n2);

// Assign IP to AP nodes
address.SetBase ("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces_n1_b[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  interfaces_n1_b[i] = address.Assign (devices_Bot_AP[i]);
}

// Assign IP to J nodes
address.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer interfaces_b_j[NUM_BOT_NETWORKS];
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  interfaces_b_j[i] = address.Assign (devices_J_Nodes[i]);
}

// Assign IP to WiFi nodes
address.SetBase ("10.2.0.0", "255.255.0.0");
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  address.Assign(bot_apDevices[i]);
  address.Assign(bot_staDevices[i]);
  address.Assign(real_staDevices[i]);
}

// UDPSink
PacketSinkHelper UDPsink("ns3::UdpSocketFactory", Address(InetSocketAddress(Ipv4Address::GetAny(), PORT)));
ApplicationContainer UDPSinkApp = UDPsink.Install(Nodes.Get(2));
UDPSinkApp.Start(Seconds(0.0));
UDPSinkApp.Stop(Seconds(SIM_TIME));

for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  ApplicationContainer UDPSinkApp = UDPsink.Install(J_Nodes.Get(i));
  UDPSinkApp.Start(Seconds(0.0));
  UDPSinkApp.Stop(Seconds(SIM_TIME));
```

```
}

#ifdef BACKGROUND_TRAFFIC
  // Real user onoff Application Behaviour
  OnOffHelper background_onoff("ns3::UdpSocketFactory", Address());
  background_onoff.SetConstantRate(DataRate(BACKGROUND_RATE));
  background_onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=5]"));
  background_onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=5]"));
  background_onoff.SetAttribute("PacketSize", UintegerValue(1024));

  // Install application in real wifi users
  ApplicationContainer background_onOffApp[NUM_BOT_NETWORKS * 4];
  int t = 0;
  for (int i = 0; i < NUM_BOT_NETWORKS * 4; i = i + 4)
  {
    background_onoff.SetAttribute("Remote", AddressValue(InetSocketAddress(interfaces_b_j[t].GetAddress(1), PORT)));

    background_onOffApp[i] = background_onoff.Install(real_wifiStaNodes[t].Get(0));
    background_onOffApp[i+1] = background_onoff.Install(real_wifiStaNodes[t].Get(1));
    background_onOffApp[i+2] = background_onoff.Install(real_wifiStaNodes[t].Get(2));
    background_onOffApp[i+3] = background_onoff.Install(real_wifiStaNodes[t].Get(3));


    background_onOffApp[i].Start(Seconds(0.0));
    background_onOffApp[i].Stop(Seconds(SIM_TIME));

    background_onOffApp[i+1].Start(Seconds(2.5));
    background_onOffApp[i+1].Stop(Seconds(SIM_TIME));

    background_onOffApp[i+2].Start(Seconds(5.0));
    background_onOffApp[i+2].Stop(Seconds(SIM_TIME));

    background_onOffApp[i+3].Start(Seconds(7.5));
    background_onOffApp[i+3].Stop(Seconds(SIM_TIME));

    t++;

  }
#endif


// Bot onoff Application Behaviour
OnOffHelper onoff("ns3::UdpSocketFactory", Address(InetSocketAddress(interfaces_n1_n2.GetAddress(1), PORT)));
onoff.SetConstantRate(DataRate(DDOS_RATE));
onoff.SetAttribute("OnTime", StringValue("ns3::ConstantRandomVariable[Constant=30]"));
```

```
onoff.SetAttribute("OffTime", StringValue("ns3::ConstantRandomVariable[Constant=0]"));
onoff.SetAttribute("PacketSize", UintegerValue(1024));

// Install application in all bots
ApplicationContainer onOffApp[NUM_BOT_NETWORKS * 2];
int j = 0;
for (int i = 0; i < NUM_BOT_NETWORKS * 2; i = i + 2)
{

   onOffApp[i] = onoff.Install(bot_wifiStaNodes[j].Get(0));
   onOffApp[i+1] = onoff.Install(bot_wifiStaNodes[j].Get(1));


   onOffApp[i].Start(Seconds(0.0));
   onOffApp[i].Stop(Seconds(SIM_TIME));

   onOffApp[i+1].Start(Seconds(0.0));
   onOffApp[i+1].Stop(Seconds(SIM_TIME));

   j++;

}


// TCP connection between node n0 and node n2
Address sinkAddress (InetSocketAddress(interfaces_n1_n2.GetAddress (1), PORT2));
PacketSinkHelper packetSinkHelper ("ns3::TcpSocketFactory",
  InetSocketAddress (Ipv4Address::GetAny (), PORT2));
ApplicationContainer sinkApps = packetSinkHelper.Install (Nodes.Get (2));
sinkApps.Start (Seconds (0.0));
sinkApps.Stop (Seconds (SIM_TIME));

Ptr<Socket> ns3TcpSocket = Socket::CreateSocket (Nodes.Get (0), TcpSocketFactory::GetTypeId ());
ns3TcpSocket->TraceConnectWithoutContext ("CongestionWindow", MakeBoundCallback (&CwndChange, stream1));
ns3TcpSocket->TraceConnectWithoutContext ("RTT", MakeBoundCallback (&RttChange, stream2));

Ptr<MyApp> app = CreateObject<MyApp> ();
app->Setup (ns3TcpSocket, sinkAddress, 1040, NUM_PACKETS, DataRate (TCP_RATE));
Nodes.Get (0)->AddApplication (app);
app->SetStartTime (Seconds (0.0));
app->SetStopTime (Seconds (SIM_TIME));

//devices_n1_n2.Get (1)->TraceConnectWithoutContext ("PhyRxDrop", MakeCallback (&RxDrop));

Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```

```
//AsciiTraceHelper ascii;
//pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("BotNet.tr"));

/*
pointToPoint.EnablePcapAll("BotNet");
for (int i = 0; i < NUM_BOT_NETWORKS; i++)
{
  phy[i].EnablePcap ("BotNet", bot_apDevices[i].Get (0));
}
*/


// Flow monitor
Ptr<FlowMonitor> flowMonitor;
FlowMonitorHelper flowHelper;
flowMonitor = flowHelper.InstallAll();


Simulator::Stop (Seconds (SIM_TIME));
Simulator::Run ();
Simulator::Destroy ();

flowMonitor->SerializeToXmlFile("flowBotNet.xml", true, true);

return 0;
}
```