Modeling DDoS Attacks with IP Spoofing and Hop-Count Defense Measure Using OPNET Modeler

Shahid Akhter, Jack Myers[1], Chris Bowen, Stephen Ferzetti, Patrick Belko, and Vasil Hnatyshin[2]

Department of Computer Science
Rowan University
Glassboro, NJ 08028

E-mail: {akhter88, bowenc65, ferzet31, belkop43}@students.rowan.edu,
[1]jackfmyers@gmail.com, [2]hnatyshin@rowan.edu

Abstract. Distributed Denial of Service (DDoS) attacks continue to plague today's Internet. The variety and ingenuity of such attacks requires network security analysts to perpetually develop more robust forms of attack identification and prevention. UDP flood is one of the simplest to deploy DDoS attacks. It is based on the idea of overwhelming the receiver with a huge amount of traffic causing congestion and preventing legitimate services. Such attacks are often launched together with IP spoofing which makes it difficult to identify the malicious traffic and distinguish it from legitimate connections. The hop count defense mechanism identifies malicious traffic and helps to thwart the DDoS attacks by comparing the TTL field value carried in the IP header of the arriving packet with the actual number of hops to the source node. This paper focuses on the methodology for modeling a DDoS UDP flood with an IP spoofing attack and hop count defense using OPNET Modeler network simulation software.

Keywords: Distributed Denial of Service, DDoS, hop-count countermeasure, OPNET Modeler

Tracks: Network Security, Security Management

1. Introduction

During our study of Distributed Denial of Service (DDoS) attacks, the world's 76th most visited website was brought down by one such attack. The Swedish website Pirate Bay, also the 74th most visited in the United States, and the 15th most visited in Sweden [1], is one of the world's most popular BitTorrent sites. Often criticized and reviled for promoting copyright infringements, the site also provides a peer-to-peer outlet for legal distribution of work by filmmakers, musicians, artists and writers. Over 10,000 independent artists have signed up with Pirate Bay in hopes of breaking through to a global audience [2]. Pirate Bay is associated with several private BitTorrent trackers which are designed to be selective in order to make sure members have a community-friendly upload to download ratio. The site was taken out of service on 14 November 2012 via a DDoS attack, when a disgruntled user was not admitted to a private tracker [3].

Even though DDoS attacks are nothing new, they are still causing site outages to this day. This paper is focused on using OPNET as a tool to model DDoS attacks, their detection and prevention, in the hopes that establishing such models will further facilitate resiliency in today's Internet. The rest of the paper is organized as follows. Section 2 provides a brief survey of existing DDoS attacks, followed by Section 3 which describes the hop-count countermeasure mechanism. Careful review of the OPNET Modeler simulation model of the UDP Flood and hop-count mechanism is provided in Section 4. We perform an analysis of collected results in Section 5, followed by conclusions and future work in Section 6.

2. Overview of DDoS attacks

The United States Computer Emergency Readiness Team [9] defines Denial of Service (DoS) attacks as "attempts to prevent legitimate users from accessing information or services," which may lead to inability to visit certain web sites, unusually slow network response times, etc. Typically, DoS attacks attempt to consume available network and computing resources such as bandwidth, CPU time, or a computer's main memory. As a result, the targeted machines and networks can no longer support the services they provide (e.g., e-mail, websites, online banking, online gaming) making them unavailable to legitimate users [8]. A Distributed Denial of Service (DDoS) attack is a variation of Denial of Service where multiple attackers (often consisting of compromised systems) located all over the Internet (i.e., distributed) participate in an attempt to bring down a targeted system. Generally, DDoS attacks can be categorized based on the target of the attack such as the vulnerability within the application's design or implementation, the network infrastructure, the resources on the victim's device, or the resources on the network that connect the victim to the Internet.

2.1 Exploiting application vulnerabilities

An example of a DDoS attack specifically designed to exploit application's vulnerability is an attack targeted at Apache web servers. This DDoS attack works by flooding an Apache server with so much data that it locks up and can no longer respond to the web page requests. Apache web servers, like most other web servers, enable a feature that allows a user to pause and resume HTTP downloads of large files. The Apache web server application is particularly vulnerable when hundreds of very large overlapping parts of a file are requested in a single request [4]. Attacks that exploit this vulnerability will crash an application running on the server, rather than the server itself.

2.2 Targeting the network infrastructure

DDoS attacks against network infrastructure are becoming more prevalent due to the rise in the number and importance of wireless networks. Wireless communications have always been vulnerable to interference. For example, Microsoft's Xbox can interfere with the 802.11n networks since they both allow transmitting on the same 2.4 GHz band. A similar wireless interference can also be created using frequency jammers. Frequency jammers can be legally used outside the United States; e.g., France allows using jammers to block cell phone transmissions in theatres or restaurants; in Italy they are used in examination rooms to reduce the likelihood of academic dishonesty. Mexico uses wireless jammers to preserve the sanctity of religious services. Miniature jammers can be used in distributed networks for intentional or malicious disruption of wireless communication.

Nowadays, micro-electro-mechanical systems (MEMS) and nano-electro-mechanical (NEMS) systems can be used to build tiny, low-power jammers that can be distributed in the air as "dust," forming a distributed jammer network. As jammers have a simpler functionality as compared to sensors, (i.e., emitting noise signals versus performing complex modulation, filtering, and other signal processing functions) they can be miniaturized more easily. The United States employed such techniques in the second Gulf War in Iraq [5].

2.3 Targeting the victim's computing resources

A TCP SYN flood DDoS attack targets computing resources on the victim's computer by exploiting a vulnerability within TCP's connection establishment protocol. The idea of this attack is to establish a large number of half-opened TCP connections which eventually consume all of the resources at the victim's machine. This attack is implemented by sending a large number of TCP SYN packets – each of which requests the opening of a connection. The target machine acknowledges every request to open a new TCP connection by sending a TCP SYN + ACK packet. The attacker, instead of completing the three-way connection establishment handshake, sends additional requests to open new TCP connections at the target machine. Since the victim's machine allocates resources for each half-opened TCP connection, it will eventually run out of buffer space and crash.

The TCP SYN attacks are often used together with IP spoofing, where the attacker replaces the source IP address in each packet with some fake address. This prevents the victim from identifying from where the attack was launched. There are numerous defense mechanisms for dealing with TCP SYN attacks. For example, the SYN Cookie countermeasure does not allocate resources for half-open TCP connection until the sources complete the three-way handshake [10]. Unfortunately, not all computers update their software with the latest security patches, and such attacks still may happen. A variation of the TCP SYN attack was used to take Pirate Bay out of service in November 2012.

2.4 Targeting network connectivity

Another form of DDoS attack targets connectivity to the Internet by flooding the network connection with so much traffic that the victim has no available bandwidth for its own legitimate Internet requests. UDP flood or UDP packet storm is the form of DDoS attacks that targets connectivity by sending a large number of UDP packets to the network interface on the victim's computer. For example, a UDP flood attack can be implemented by sending spoofed UDP packets to the *chargen* or *echo* service of a target computer. Let us call such computer *victim A*. Spoofed UDP packets sent to victim A will have the source IP address set to IP address of another victim, let us call it *victim B,* and port number set to that of the *chargen* or *echo* service. Recall that *chargen* and *echo* services send a response message each time they receive a datagram. As a result, such an attack will create an infinite loop of useless UDP traffic between *victim A* and *victim B*, consuming available network resources, creating congestion, and denying service to the victim's legitimate traffic [11].

Comment [jfm1]: There are three complete sentences here…must either be separated by periods or semi-colons.

3. Defending Against Attacks Using Hop-Count

Typical defenses against distributed denial of service attacks rely on identifying the malicious traffic and preventing it from entering the network. Typically, the edge routers in the victim's network or the victims themselves identify the malicious traffic flows and relay the identity to the attacker upstream to the edge routers or Internet Service Providers (ISP) that the attacker uses to access the network. The ISPs and the edge routers then use the identity of the attacker to filter out malicious traffic, preventing it from entering the Internet altogether. That is why DDoS attacks are often launched together with IP spoofing – a masquerading technique in which the source address field in the IP header of the malicious packet is set to a fake value. The IP spoofing technique makes it harder for the victim to identify the source of the attack.

Hop-count filtering is a technique which utilizes hop counts derived from the IP header of a packet to identify malicious traffic flows [6]. The IP packet header contains the time-to-live (TTL) field which stores the maximum lifetime of the packet. Each time a packet travels through an intermediate node, the TTL field value is decremented by 1. If the length of the path from source to destination is known, then by examining the TTL field value, the victim can determine if the IP address carried in the packet is valid or not. Specifically, the length of the path from source to destination can be computed as the difference between the initial and the final values of TTL field in the arriving packet. Typically, the initial value of the TTL field is set to some default value such as 255. Once the length of the path is computed from the IP header of the arriving packet, it is then compared to the actual value of the path length. If the values match, then the packet is considered legitimate and will be accepted by the host. Otherwise the packet is deemed to be malicious, likely with the spoofed source IP address value, and is discarded.

The hop-count filtering method requires the host to build and maintain a table that contains the length of the path (i.e. the hop count) to known sources, identified by their IP address. Such a table can be built by pinging (i.e., sending an ICMP request message) any nodes with an unknown hop count.

4. Modeling Hop-Count Defense Mechanism

In this section we describe our endeavors creating a simulation model of the DDoS UDP flood attack and a hop-count countermeasure in OPNET Modeler.

4.1 IP Spoofing

The first step in creating a model of a UDP flood attack was to implement IP spoofing. For that purpose, we modified the **ip_dispatch** process model and added a new model attribute called *Spoofed IP Address*. This attribute specifies the IP address value set in the source address field of the spoofed packet's IP header. If the *Spoofed IP Address* value is set to "*None*", then the current node does not perform IP spoofing. Otherwise, all outgoing packets will have the value of the source address in the IP header set to the value of the *Spoofed IP Address* attribute.

To set the value of the source address field in the IP header, we modified the **ip_encap_v4** process model (Figure 1). Specifically, first we added a statement in the **INIT** state to read the value of the *Spoofed IP Address* attribute:

```
op_ima_obj_attr_get (own_node_objid, "ip.Spoofed IP", &spoofIP);
```

Next we modified the **ENCAP** state of the **ip_encap_v4** process model to set the source IP address of the outgoing spoofed packet to specified value:

```
if (strcmp(spoofIP, "None") != 0){
   ip_dgram_fd_ptr->src_addr =
           inet_address_create(spoofIP, InetC_Addr_Family_v4);
}
```

The above code ensures that when configured to spoof IP traffic, all the packets leaving this node will have the source IP address set to spoofIP, the value of the *Spoofed IP Address* attribute.

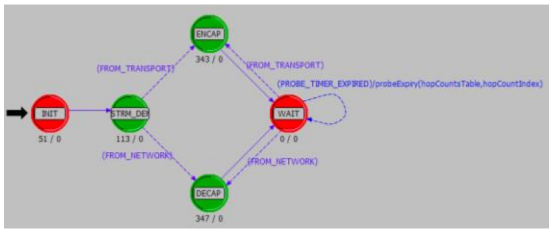

Figure 1: Updated ip_encap_v4 process model

4.2 Modeling Hop Count Countermeasure

Similarly, the first step in implementing the hop count defense was to add a configuration parameter for differentiating between the nodes that support this mechanism and those that do not. We added a model attribute called *Hop Count Defense* in the **ip_dispatch** process model and parsed its value in the **INIT** state of the **ip_encap_v4** process model:

```
op_ima_obj_attr_get(own_node_objid,"ip.Hop Count", &isDefender);
```

Next we added functionality to keep track of known hop counts to various nodes in the network. For that purpose, we created a hop count table. Each entry in that table consists of three values: source IP address, hop distance to the node with that IP address, and an event handle for the self-interrupt that is scheduled when the probe message is generated. The **ip_encap_v4** process model denotes the packet arrival from the network layer into the IP layer via a transition from the **STRM_DEMUX** state into the **DECAP** state. This is why we implemented the hop count defense functionality in the **DECAP** state, i.e., when an IP packet arrives from the network layer.

The hop count countermeasure was implemented as follows: upon the packet arrival from the network layer, the node consults the hop count table to check if this is a spoofed packet. If the packet is not spoofed, then the packet is processed normally. Otherwise the spoofed packets are discarded. The packet is considered spoofed if the length of the path to the packet's source node does not match the value stored in the hop count table. The hop distance to the packet's source node is computed as being equal to 255 minus the value of TTL field. By default, IP sets the TTL field value to 255, the maximum possible value.

It is also possible that the hop count table does not contain an entry for the packet's source IP address. In this case the node, hereafter referred to as the *originator*, performs the following three steps:

1. The packet is processed normally and is forwarded to the upper layer.
2. A new entry for the packet's source IP address is added into the hop count table. The path length in the new hop count table entry is set to an invalid value.
3. A probe message is sent to the packet's source node.

The originator node sends the probe message in an attempt to discover the actual hop distance to the source node. Upon the arrival of the probe message at the source node, a probe reply message is sent back to originator. When the originator receives a prove reply message, it updates the hop count entry with the correct value of the hop distance to the source node. Both the probe and probe reply messages carry no data except for the IP header. We use the protocol field in the IP header to identify the probe and probe reply messages. The actual hop distance to the source node is computed as being equal to 255 minus the value of the TTL field in the probe reply message.

It is possible that the probe reply will never arrive back to the originator node because the attackers used an invalid IP address. To handle such a situation, we added a timer that is started when the originator sends a probe message. If the probe reply arrives before the timer expiration, then the interrupt event is canceled. Otherwise if the probe timer expires, then the corresponding entry in the hop count table has the length of the path set to the invalid value of 256.

We added the following code in the DECAP state of the **ip_encap_v4** process model to implement the hop-count defenses:

```
// Node receives a probe packet
if(packet_type == PROBE){
   Send_ProbeResponse(DestAddr);
}
// Node with Hop Count Defense ON, receives prove reply
else if (packet_type == RESPONSE &&  isDefender) {
   updateHopCount(SrcAddr, 255 - TTL);
   op_ev_cancel(hopCountsTable[addressIndex].probeTimer_evh);
}
// Node with Hop Count Defense ON, receives data packet
else  if(isDefender){
   // No entry in the Hop Count Table
```

```
    if (getHopCountEntry(SrcAddr) == null){
       Send_Probe(SrcAddr);
       HopCountsTable.add(SrcAddr, UKNOWN);
       hopCountsTable[i].probeTimer_evh =
          op_intrpt_schedule_self(op_sim_time () + PROBE_TIMER, i);
    }
    // Discard packet if hop distance incorrect
    else if (HopCountsTable.numHops != 255 - TTL){
       // The packet is considered spoofed and discarded
       discard(packet);
    }
    // If hop distance is correct then continue as before
 }
```

We added the following function to handle probe timer expiry:

```
 static void probeExpiry(HopCountEntry hopTable[], int numEntries){
    // Interrupt code carries an index into hop count table
    // Verify that we received a valid index into hop count table
    if (op_intrpt_code() <= numEntries) {
       // Probe Reply did not arrive in time
       hopTable[op_intrpt_code()].numHops = INVALID_DISTANCE;
    }
 }
```

4.3 Network Topology

To study the UDP flood attack and hop count countermeasure, we used topology depicted in Figure 2. In our study, the attacker nodes are configured to send UDP Flood traffic to the defender node. The attackers also spoof the source IP address of their outgoing packets. The reflector and the regular user nodes sent only legitimate traffic that should not be discarded. The defender node implements the hop count defense and, when identified, discards all packets with the spoofed source IP address.

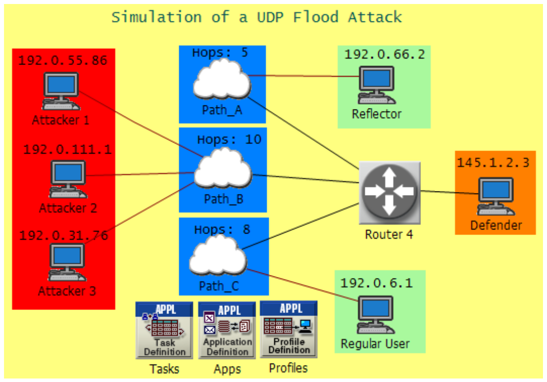

Figure 2: Network Topology

The attacker, regular user, and reflector nodes all travel via different paths to reach the defender node. For example, the reflector node needs to traverse *Path_A* sub-network which consists of 6 routers connected in a straight line. The length of the path from reflector to defender is 8 hops. Similarly, the attacker nodes travel through the *Path_B* sub-network to reach the reflector. The length of the path from the attackers to the defender is 13 hops. The regular user node sends its traffic though the *Path_C* sub-network, and the total length of the path from the regular user to the defender is 11 hops. Therefore, when the defender receives attacker packets containing a spoofed source IP address, it should be able to use the hop-count defense to identify malicious traffic and discard it.

In this study, we used the **ethernet_wkstn_adv** node model to represent end nodes and the **ethernet2_slip8_gtwy_adv** node model to represent routers. All the end nodes were connected to routers via **1000BaseX** duplex links; routers were connected to one another via **PPP_DS3** links.

4.4 Modeling UDP Flood Attack

To model UDP Flood traffic we created a custom application, called **Direct Flood**, where the attackers send a stream of traffic with the spoofed source IP address directly to the defender. The **Direct Flood** custom application consists of a single task where the source sends 1000 requests. Each request consists of a single packet of size 10KB. The request inter-arrival time is distributed exponentially with the mean outcome of 0.1 second. The destination node does not generate a response upon the request packet arrival.

We also created an application called **Reflection Flood**. This application models a UDP flood attack where attackers send requests to the reflector node which in turn responds by sending a stream of data to the target machine identified by the source IP address in the spoofed request packets. We used a standard video conferencing application to generate traffic sent by the reflector node. Overall, this UDP Flood attack works as follows: attackers send request packets with the source IP address set to address of the defender node. Reflector responds to arriving requests by sending a stream of video traffic to the defender.

5. Simulation Study Results

5.1 Defending Against Direct UDP Flood Attack

To illustrate effectiveness of the hop count defense, we set up a simulation study where the attacker, reflector, and regular user nodes (Figure 2) send the **Direct Flood** application traffic to the defender node. The attacker nodes spoof the source IP address of their outgoing packets and set them to 192.0.66.2, the IP address of the reflector node. We also tested a scenario where the attackers set the spoofed IP address to an invalid value, i.e., 192.012.13, an IP address that is unused in our scenario. In both cases we received almost identical results.

The distribution of the generated traffic in the network is shown in Figure 3. The red line with the square symbols denotes all the traffic generated in the network, while the green line with triangle and blue line with diamond denote the attacker and legitimate user traffic respectively. Legitimate user traffic consists of the traffic flows generated by the reflector and the regular user nodes.

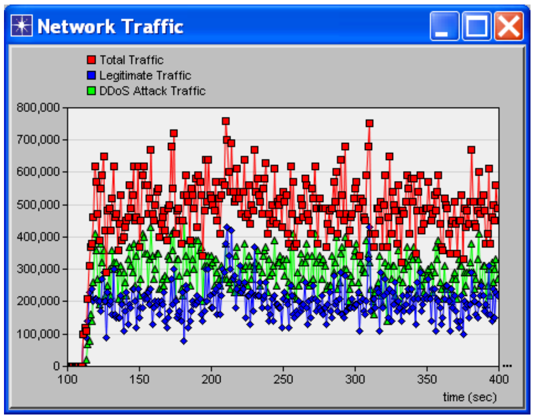

Figure 3: Direct UDP Flood Attack Traffic Distribution

Figure 4 illustrates how the defender node handles the Direct UDP Flood attack. When the hop count defense is on, the defender node is able to identify and filter out all of the traffic generated by the attacker nodes, as depicted in Figure 4 by a blue line with the square symbol. However when the hop count defense is disabled, the defender node accepts all of the traffic. The defender is able to identify and discard the packets with the spoofed IP address because the length of the path from the attacker nodes to the defender is different than the length of the path from the reflector node to defender.

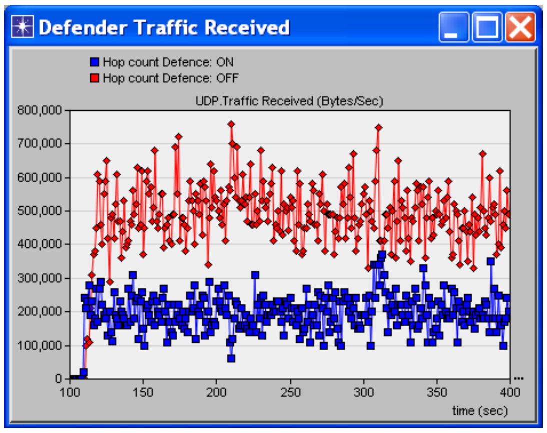

Figure 4: Traffic Received by the Defender Node with the Hop Count Countermeasure enabled and disabled

The attackers set the source address field of the spoofed packets to the IP address of the reflector node. When the defender received the first packet from the attacker, it consulted the hop-count table. Since at this point the length of the path from reflector to defender is unknown, the defender sent a probe message to reflector and accepted all the spoofed packets with the source address set to the reflector's until it received a probe reply. Upon the probe reply arrival, the defender recorded the actual length of the path to reflector. All subsequent packets with the source IP address set to that of the reflector were verified that they traversed the number of hops equal to the recorded length of the path to the reflector node. Since the spoofed packets originated from the attacker nodes, they traverse a different number of hops to reach the defender and thus were identified as malicious and are discarded.

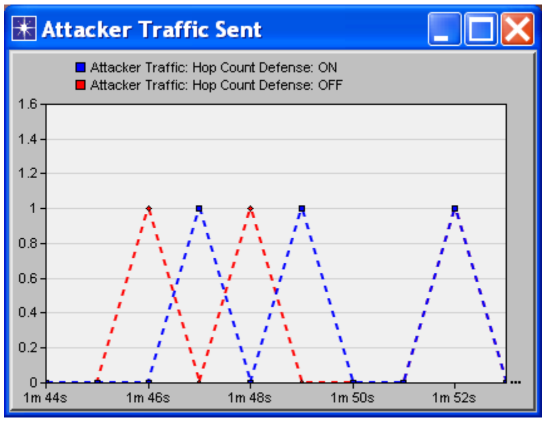

Figure 5: Traffic Generated by the Attacker Nodes in the Reflection UDP Flood Attack scenario

A similar scenario plays out when the attacker sets the source IP address of spoofed traffic to an invalid value. In this case, the defender's probe timer will expire, at which point it will conclude that the packet arrived from an unknown source. It will then set the path length to an invalid value and discard all subsequent packets that arrive from the same IP address.

Please note that the legitimate traffic is not filtered out because the path length determined by the probe reply will match the path length computed based on the TTL field of arriving data packets. Unfortunately, the hop count defense does not work well in certain situations as we describe in the following section.

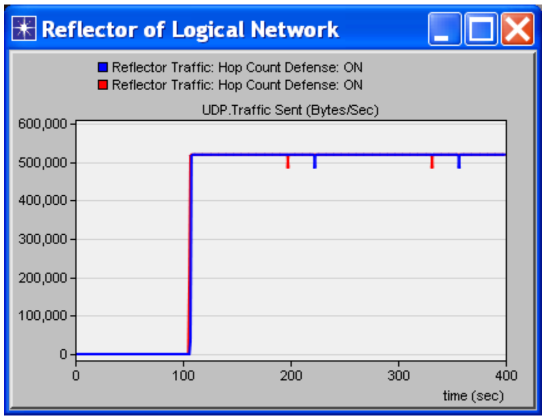

Figure 6: Traffic Generated by the Reflector Node in the Reflection UDP Flood Attack scenario

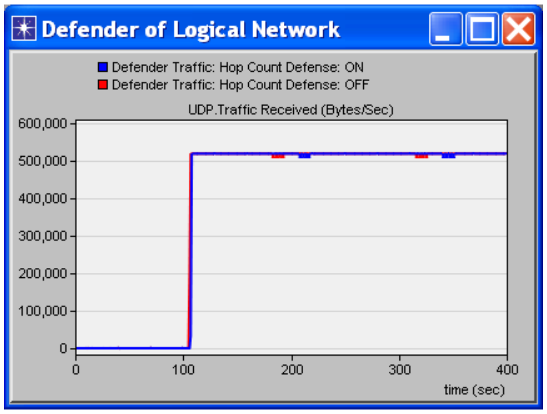

Figure 7: Traffic Generated by the Reflector Node in the Reflection UDP Flood Attack scenario

5.2 Defending Against Reflection UDP Flood Attack

To illustrate situations where the hop count attack fails to identify and discard malicious traffic, we created a simulation study where the attacker nodes send the **Reflection Flood** application traffic to the reflector node. Attackers spoofed the source IP address in the outgoing packet and set it to 145.1.2.3, the IP address of the defender node. Figure 5 depicts the total amount of traffic generated by the attacker nodes when the hop-count defense is enabled and disabled. In both cases each of the attackers generated a single request to the reflector node. Please note that in both scenarios the third request was generated at the same time of 1 minute and 52 seconds which results in the two data points overlapping. In response to each of these requests, the reflector node sends a stream of video traffic to the defender node as shown in Figure 6 which depicts the total amount of traffic generated by the reflector node.

In this scenario, the attackers tricked the reflector into thinking that the defender made a request. This resulted in the reflector sending a huge amount of data to the defender. Since the source address in the data traffic from the reflector to the defender was not spoofed, the defender is unable to identify and discard malicious traffic. As a result, all the traffic sent by the reflector is accepted by the defender node. This phenomenon is illustrated in Figures 6 and 7, which shows the total amount of traffic generated by the reflector and accepted by the defender node. In both cases, when the hop count defense is enabled and disabled, the defender fails to filter out malicious flows and accepts all incoming traffic.

6. Conclusion

This paper presents a practical methodology for modeling UDP Flood distributed denial of service attacks and the hop count countermeasure. This project acts as a springboard into further study, refinements, and development of new models for simulation of distributed denial of service attacks and defenses.

The results of the simulation study suggest that the hop count countermeasure is ineffective against the attacks where the legitimate users are tricked into sending huge amounts of traffic to the victim's machine by means other than the IP spoofing. The hop count defense is only effective against the attacks where the IP address spoofing is involved. To prevent the reflection UDP flood attack, the reflector node must also deploy the hop-count defense and should not forward the packets to the application layer until it verifies the length of the path to the source node of the packet. However, delaying delivery of the data packets to the upper layers could be undesirable for some applications. Furthermore, discarding malicious traffic at the destination is not a good approach since the node has to allocate both the bandwidth in its local network and computing resources to deal with these malicious packets. A better approach is to notify the nodes upstream about identified malicious flows and have the upstream edge routers filter out these flows thereby preventing them from entering the protected network domain altogether.

We plan to continue our investigation of various DDoS attacks and defenses and examine the possibility of their implementation in OPNET Modeler. In particular, we would like to develop a signaling protocol which will allow the end nodes to notify the edge routers about identified malicious traffic that enters their network domain.

We also would like to further refine the current implementation of the hop count defense by adding statistics for recording the number of identified malicious flows, the number of false-positive and false-negative classifications, the number of queued, discarded, and forwarded packet that were classified as malicious and as legitimate. In addition, we are studying machine learning and statistic-based techniques for identifying malicious traffic flows and looking into possible ways to implement and test these techniques in the OPNET Modeler environment.

References

[1] http://www.alexa.com/siteinfo/thepiratebay.se, last accessed 4/18/13

[2] "10,000 Artists Sign Up for Pirate Bay Promotion," http://torrentfreak.com/10000-artists-signed-up-for-pirate-bay-promotion-12110/, last accessed 4/18/13

[3] "Piratebay Servers Down Due to DDoS," http://zerosecurity.org/piracy/piratebay-servers-down-due-to-ddos/, last accessed 4/18/13

[4] M. Stockley, "Apache Exploit Leaves P to 65% of All Websites Vulnerable," http://nakedsecurity.sophos.com/2011/08/26/apache-exploit-leaves-up-to-65-of-all-websites-vulnerable/, last accessed 4/18/13

[5] H. Huang, N. Ahmed, P. Karthik, "On a New Type of Denial of Service Attack in Wireless Networks: The Distributed Jammer Network," IEEE Trans. on Wireless Communications, Vol. 10, No.7, pp. 2316- 2324, 2011

[6] C. Jin, H. Wang, K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in Proc. of the 10th ACM conference on Computer and communications security, pp. 30–41, Oct. 2003.

[7] A. Yaar, A. Perrig, D. Song, "Pi: A path identification mechanism to defend against DDoS attacks," In Proc. of IEEE Symposium on Security and Privacy, May 2003.

[8] J. Mirkovic, S. Dietrich, D. Dittrich, P. Reiher, "Internet Denial of Service: Attack and Defense Mechanisms," Prentice Hall, 2005, ISBN-10: 0131475738, ISBN-13: 978-0131475731

[9] Understanding Denial-of-Service Attacks, http://www.us-cert.gov/ncas/tips/ST04-015, last accessed 3/21/13

[10] RFC 4987: TCP SYN Flooding Attacks and Common Mitigations, http://tools.ietf.org/html/rfc4987, last accessed 4/18/2013

[11] CERT® Advisory CA-1996-01 UDP Port Denial-of-Service Attack, http://www.cert.org/advisories/CA-1996-01.html, last accessed 4/18/2013

Formatted: Font: 10 pt