An Investigative Study on Modeling a DDoS Attack and Defense Measure in OPNET

Shahid Akhter, Christopher Bowen, Jack Myers, Dr. Vasil Hnatyshin

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. OPNET modeling is a powerful tool to model not only the DDoS attacks themselves, but also the preventative measures. OPNET offers heavy-duty simulation capabilities to model a nearly unlimited number of network permutations, such as variations in network protocols, addressing and topology. Introducing OPNET into the arsenal of tools to harden modern networks is a valuable contribution to network analysis. This paper leverages OPNET to model one particular attack and defense.

1. Introduction

In the middle of our examination on 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 works from filmmakers, musicians, artists and writers. Over ten thousand 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 so far as the Internet is concerned, 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 more resiliency in today's Internet. .

2. Attack Types (SYN Flood)

DDoS attacks can be categorized based on the target of the attack, specifically:

- an application vulnerability,
- network nodes,
- the resources of the target, or
- the link which connects the target to the Internet.

2.1 Exploiting application vulnerabilities

An example of a DDoS attack specifically designed to exploit an application vulnerability is an attack targeted at Apache web servers.

This DDoS attack works by flooding an Apache web server with so much data that it locks up and can no longer serve web pages. 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 Exploiting network nodes

DDoS attacks that cause network nodes to fail are becoming more prevalent due to the rise in the number and importance of wireless networks.

Wireless communications are always vulnerable to interference. One well known example is the interference created between Microsoft's Xbox and the 802.11n wireless network. The Xbox emits wireless signals using the 2.4 GHz band, which is also employed by 802.11n (although 802.11n can also use a 5 GHz band).

This type of wireless interference can also be created using frequency jammers. Frequency jammers are legally used

outside the United States. In France, they are used 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 them to preserve the sanctity of religious services. Miniature jammers can be used in a distributed network for the purpose of intentional and malicious disruption of wireless networks.

Tiny, low power jammers – microelectromechanical systems (MEMS) using nanotechnology – can be built so small that they can be distributed in the air as "dust," forming a distributed jammer network. As jammers have a smaller function compared to sensors, (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 Exploiting the resources of the target

A SYN flood DDoS attack occurs when several zombies send multiple, rapidly repeated TCP SYN requests to the targeted server but do not acknowledge the SYN transmission. Such attacks will force the attacked node to maintain an excessive number of half-open TCP connections, which will eventually consume all of the available server resources and crash the node. This was the form of attack that was used to take Pirate Bay out of service in November 2012. If the zombies never acknowledge the SYN request, the server will keep open the connection until a time-out occurs. However, this form of attack has the distinct disadvantage of identifying the zombie via its source IP address.

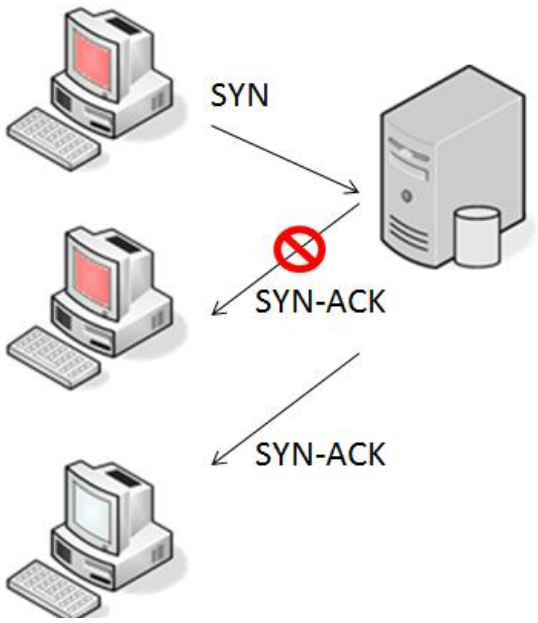

Figure 2-1. A TCP SYN attack

A better form of attack would be to spoof the IP address of an "innocent" end node, causing the targeted server to send a SYN-ACK to these spoofed addresses. The nodes that now receive the SYN-ACK from the server are either not in the SYN_SENT state or in that state with a different destination IP address. In either case, the innocent nodes will disregard the SYN-ACK and will not respond to the targeted server. Thus the targeted server will keep the connection open until the timeout expires – waiting for an ACK that will never come. These half-open connections consume resources on the targeted server. If enough of these connections are established, the server's resources could be exhausted, thereby blocking additional connections from occurring.

The attack can be further enhanced by generating random IP addresses, avoiding the suspicious behavior of an innocent node repeatedly requesting a large number of connections.

2.4 Exploiting the link connecting the target to the Internet.

Another form of DDoS attack focuses neither on the target end-node, the applications running on the node, nor the network devices the target relies on, but rather on the target's connection itself. This can be accomplished by flooding the network channels with so much traffic that the target effectively has no available bandwidth for its own legitimate Internet requests.

This type of attack can essentially be considered a hybrid attack with multiple victims. The first victim is the client whose bandwidth is being targeted as described earlier in this section. The second victim is the server which would suffer from the degradation of its resources as it is commandeered to send illegitimate traffic.

The UDP transport protocol is well suited for such attacks because it transmits unthrottled traffic. UDP flooding can be accomplished particularly well with high definition video streaming.

2.5 Selection of a DDoS Attack Mode

DDoS attacks focusing on particular application vulnerabilities would be difficult if not impossible to model in OPNET. These types of attacks are best modeled in a lab scenario where actual servers are equipped with the vulnerable applications. Hence this type of attack was not selected for OPNET modeling.

Attacks on network nodes, especially ones similar to distributed jamming of wireless nodes, would be

challenging to model in OPNET. Such models would require simulation of routers being disrupted by powerful rogue signals. Such attacks were also dismissed as not being good candidates for OPNET simulations.

The TCP-based attacks leaving the targeted server with too many half-open connections was appealing to model, especially given the recent successful attack on Pirate Bay. In fact, we initially attempted to simulate this model. We quickly discovered that while the complexities of TCP made this a most interesting study, it also presented a more complex model to implement. As our goal was to *introduce* DDoS modeling techniques in OPNET, we opted for the UDP-based flooding attacks as described above.

3. Defending Against Attacks Using Hop Count

To defend against spoofed traffic, the targeted server must discriminately respond to requests as shown in the figure below.

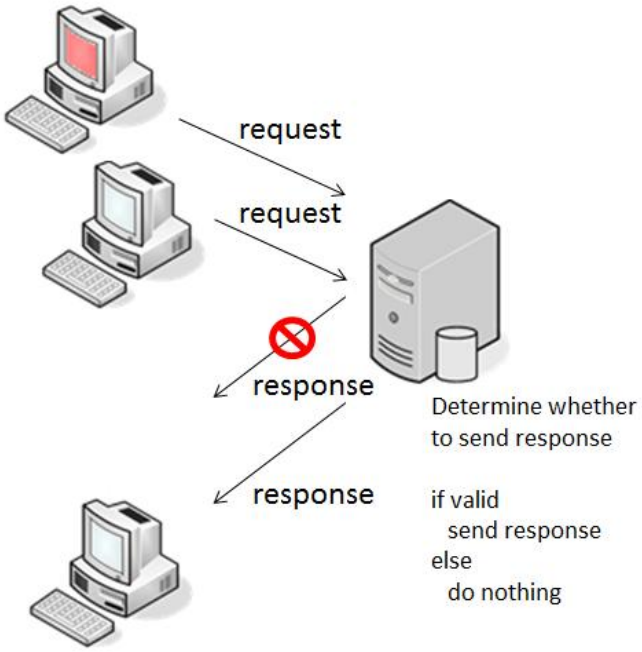

Figure 3-1. Desired behavior of targeted server

The server must be able to determine whether to send a response, and only send it when it ascertains the source is legitimate. The increasingly popular field of research that delves into detecting and defending against such attacks has spawned a plethora of acceptable solutions.

Proposed by Jin *et. al.*, Hop-Count filtering is a technique which utilizes hop counts derived from the IP header of a packet [6]. Upon creation of an IP packet, the time-to-live (TTL) field will hold the maximum lifetime of the packet which gets decremented by each node that forwards it. The difference between the initial TTL and the TTL at the destination is computed to be the hop count.

The packet filtering method requires the defending node to build a clustered address map that correlates the address of a node to the expected hop count. When receiving traffic from a zombie with spoofed packets, the server can compute the packets' actual hop counts and compare the values with the *expected hop count*. If these values differ, the packet is associated with an attack and is therefore discarded.

4. Simulation Description

4.1 Node descriptions

In order to successfully mirror a DDoS attack in OPNET, the following nodes have been created.

The ***Innocent Recipient*** is the end-node that will be the victim in our simulation. It will not be requesting any video from the *Video Server Farm*.

There are three **zombies** (*Zombie 1*, *Zombie 2*, *and Zombie 3*) in the simulation. These zombie nodes have been pressed into service of the mastermind behind the DDoS attack – likely through some deficiency in their security configurations. The Zombie nodes will request a service from the server farm with a spoofed source address so that the traffic would instead be redirected to the victim, the *Innocent Recipient*.

The ***Video Server Farm*** houses nodes *2*, *3*, and *4*. Each node provides a video conferencing service. These nodes will be manipulated by the zombies into attacking the *Innocent Recipient.* Originally, our intent was to compromise a single node which would host a high resolution video conference application. However, in order to create enough traffic to overwhelm the *Innocent Recipient*'s link, we needed to create additional nodes in the farm.

The ***Legitimate Requester*** is the end-node which will legitimately request low resolution video provided by node *2* in the *Video Server Farm*. The existence of a legitimate end-system is important to ensure well-behaving nodes will not be filtered out when running defensive measures against DDoS attacks.

Standard network **routing devices** have been added to our model, retaining their OPNET

default values. These routers simply forward packets along to their destination.

A ***subnet*** was created as a collection of *router* nodes *A-Z* with different entry points. The *Legitimate Requester* will enter the subnet at entry point *J*; zombies will enter at entry point *A*, and the *Innocent Recipient* will enter at entry point *U*.

The depiction of all nodes is shown in Figures 4-1 and 4-2.

4.2 OPNET Applications

In order to completely consume the link between the *Innocent Recipient* and its edge router, we needed to create applications that would send a large amount of traffic. In order to simplify the OPNET model, we selected the Video Conferencing application, as it would send a large amount of UDP traffic which would be unaffected by TCP flow and congestion control.

Source Node	Application	Destination Video Server
Zombie 1	SpoofIP_Video3	node 4
Zombie 2	SpoofIP_Video	node 2
Zombie 3	SpoofIP_Video2	node 3
Legitimate Requester	LegitIP_Video	node 2

Table 4-1. Deployed applications in the simulation

The three variations of SpoofIP_Video are configured to conduct a high resolution video conference; the LegitIP_Video application will initiate a low resolution video conference.

4.3 SpoofIP parameter

Within OPNET, it is possible to create an additional parameter for each node. We created a *spoofIP* parameter to differentiate between a normal mode of operations which sends UDP traffic legitimately (IP header contains the correct IP address for the sender), and a "zombie" mode of operation in which the source address in the header is changed to that of the victim. The new parameter, *spoofIP*, can be toggled between ON (spoofing enabled) and OFF (spoofing disabled).

To simulate what a zombie would do in the real world, we adjusted the behavior of IP encapsulation in OPNET. By adding the following code to the **ip_encap_v4** process model within OPNET, we are able manually change the source address to that of the *Innocent Recipient*.

```
if (spoofIP) {
   ip_dgram_fd_ptr->src_addr = inet_address_create
                ("192.0.66.2", InetC_Addr_Family_v4);
}
```

This code checks to see if the IP parameter *spoofIP* is true. If the initiating node has *spoofIP* ON, the IP address of the victim (*Innocent Recipient*) will be stored in the "src_addr" header field in the IP datagram. Note that we leveraged the OPNET function inet_address_create which will build an IPv4 address (paramenter two) for the string containing the IPv4 address of the victim (parameter one).

4.4 Simulating knowledge of hop counts

Our selected defense against a DDoS attack with a spoofed address relies on knowledge of the actual hop counts one would expect from the source to the destination. As mentioned previously, the IP parameter time-to-live (TTL) is directly related to the hop count. In reality, a mechanism would exist to store and/or dynamically obtain the hop counts for each requester.

To simulate this knowledge, we ran our simulation where each zombie issued a request to the server farm with *spoofIP* set to OFF. The *Legitimate Requester* also issued a request for low resolution video traffic. The simulation was run again, replacing the *Innocent Recipient* for the *Legitimate Requester*.

Note that it was critical that each server in our video server farm was connected to the subnet through the exact same number of routers. In this way, the TTLs would not vary across *node2*, *node3* and *node4*.

While running all this legitimate traffic in the network, we activated additional code in the decapsulation process of **ip_encap_v4**.

```
/* Declare vars to capture information on IP datagrams */
char my_src_addr_str[INETC_ADDR_STR_LEN];
char my_dest_addr_str[INETC_ADDR_STR_LEN];
FILE *fp;

/* Open a file to record data in CSV format */
fp = fopen("C:\\Users\\Jack\\DDoSResults.csv", "a");
if(fp == NULL)
   fp = fopen("C:\\Users\\Jack\\DDoSResults.csv", "w");

fprintf(fp, "%s, %d, %s\n",
  inet_address_print (my_src_addr_str,
                      ip_dgram_fd_ptr->src_addr),
  ip_dgram_fd_ptr->ttl,
  inet_address_print (my_dest_addr_str,
                      ip_dgram_fd_ptr->dest_addr));
fclose(fp);
```

Whenever IP decapsulates, we record three values from the IP datagram header: the source IP address, the TTL, and the destination IP address.

With this information, we were able to definitively know

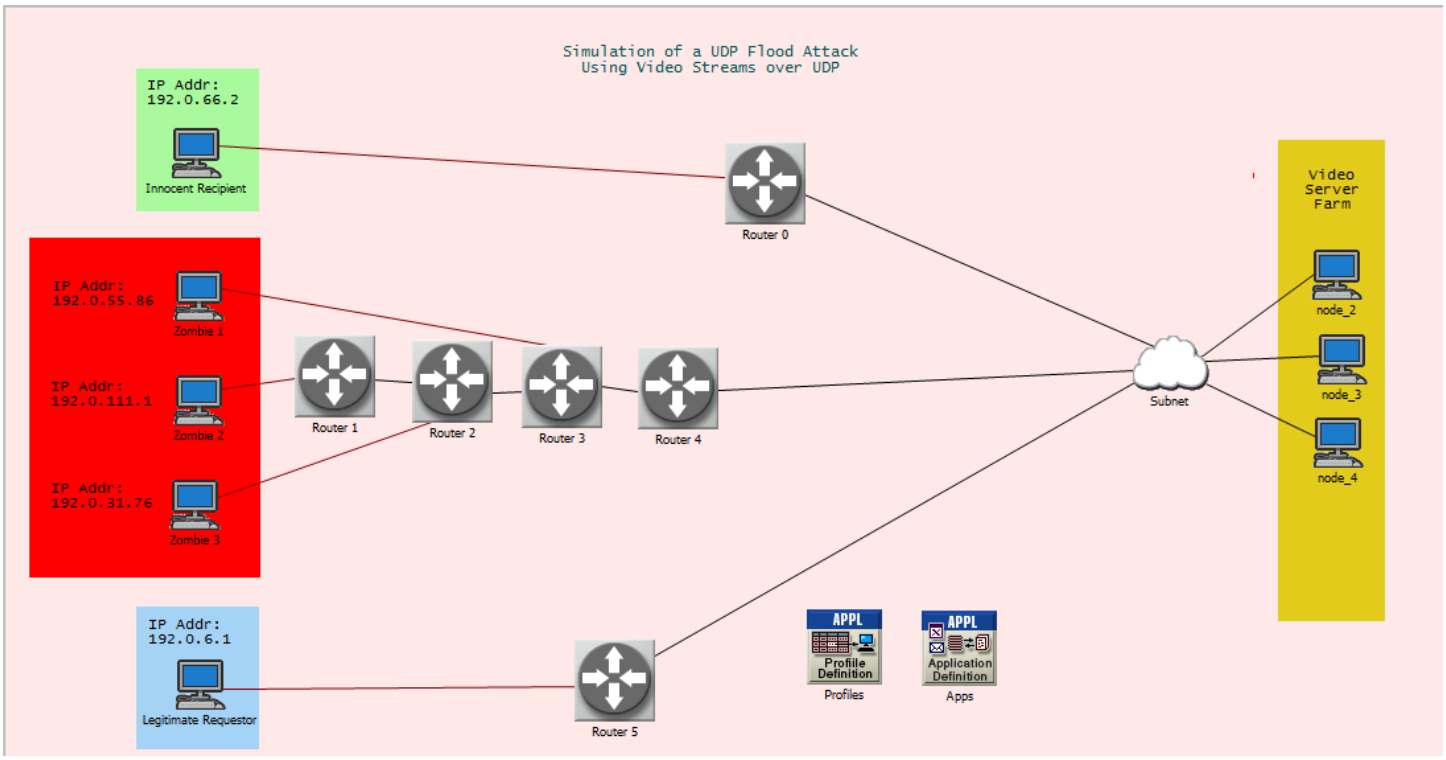

Figure 4-1. Network Topology

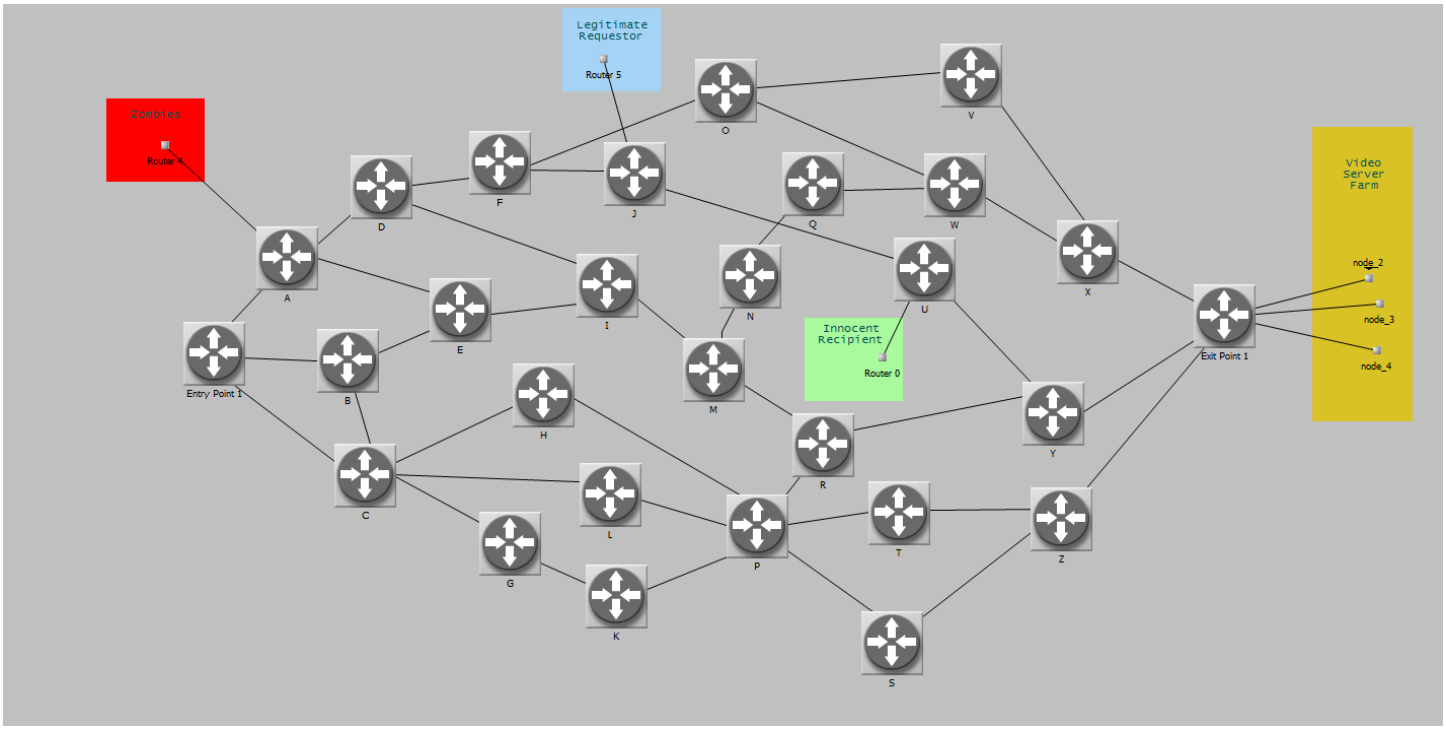

Figure 4-2. Subnet Topology

the actual TTLs for the requesting nodes, as seen in the table below.

Source Node	TTL
Zombie 1	23
Zombie 2	21
Zombie 3	22
Legitimate Requester	27
Innocent Recipient	28

Table 4-2. Actual TTLs for nodes in the simulation

4.5 Modifying the IP protocol for DDoS defense

The next addition to the simulation is to code the defensive measure. We again utilized the IP decapsulation process of **ip_encap_v4** to model our defense.

Firstly, we created two new Boolean variables to record various states of the simulation. The variable *packet_spoofed* will be set to TRUE (1) when the defensive measure detects traffic with a spoofed IP address. The variable *defense_on* will be toggled between zero and one depending on whether the simulation is running with the hop count defense. (If time had permitted, we would have promoted this as an OPNET parameter.)

```
Boolean packet_spoofed=0;
Boolean defense_on=0;
```

The code below would be activated in defensive mode.

```
if (defense_on) {

   if (strcmp(inet_address_print (my_src_addr_str,
         ip_dgram_fd_ptr->src_addr),"192.0.66.2")==0) {
      // Coming from Innocent Recipient, may be spoofed
      if (ip_dgram_fd_ptr->ttl != 28) {
         // Discard the IP packet.
         packet_spoofed=1;
         }
      }
   if (strcmp(inet_address_print (my_src_addr_str,
          ip_dgram_fd_ptr->src_addr),"192.0.6.1")==0) {
      // Coming from Legitimate Recipient, may be spoofed
      if (ip_dgram_fd_ptr->ttl != 27) {
         // Discard the IP packet.
         packet_spoofed=1;
         }
      }
   }
```

Observe that the data obtained in section 4.4 is hard coded into the if statements. If the TTL is not the expected value, the packet needs to be dropped, so no UDP response will be generated.

Elsewhere in the code, we react to the Boolean value packet_spoofed.

```
if (packet_spoofed) {
   printf("\nPacket Dropped.\n");
   packet_spoofed=0;  // Reset to default.
   }
else {
   /* Install the ICI and send the packet to the
      higher layer.                                  */
   op_ici_install (transp_iciptr);
   op_pk_send (pkptr, output_strm);
   }
```

The original code (marked in red font) was placed inside of a conditional, such that it only executed when a spoofed packet was not detected. Otherwise, we printed a message to OPNET's simulation console and reset the Boolean back to the default.

5. Analysis/Results

5.1 Spoofing with Vulnerable Server Farm

Using the *spoofIP* parameter, it is now possible to enable spoofing for the zombie nodes and analyze the data collected in the scenario.

In the simulation of the network with vulnerability, several outcomes will be expected if the model was configured correctly:

- the *Legitimate Requester* should receive only the data it is requesting;
- the amount of data that is received by the Innocent Recipient and Legitimate Requester should closely resemble the amount of data sent by the server farm;
- the zombies should not receive any traffic despite requesting traffic.

It is important to first verify these expectations by inspecting different graphs that help to solidify the expected results.

The Legitimate Requester issues a request for a video conference with the server farm.

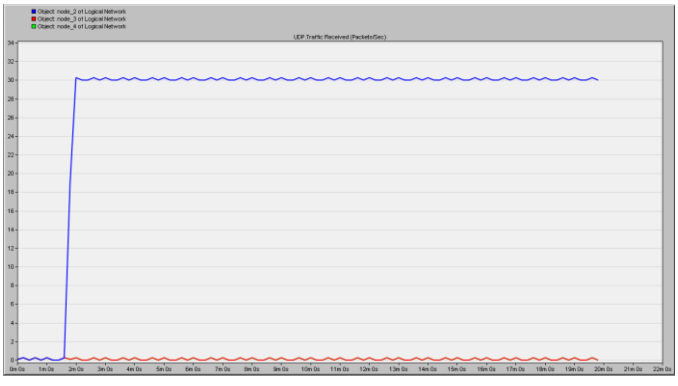

Figure 5-1. Traffic received by the server farm.

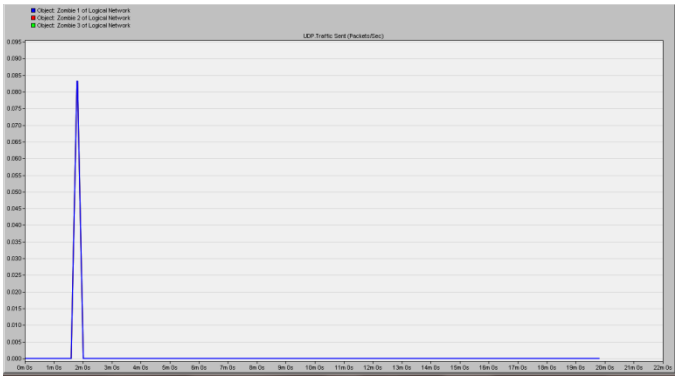

Figure 5-2. Traffic sent by zombie nodes

Figure 5-1 above verifies that only the traffic sent by the *Legitimate Requester* is being received – in this case, by *node_2*. When *node_2* receives the request for a video conference, it responds as expected and a video conference is initiated between the two nodes. It can also be noted that the other two server nodes receive no traffic during this simulation. Since the *Innocent Recipient* did not initiate any requests, when the server nodes receive the request from the spoofed zombies (the initial request can be seen in Figure 5-2) and they try to begin video conferencing, the *Innocent Recipient* will not send any traffic back.

To further validate the setup of this scenario, it is helpful to look at traffic that is received by the *Innocent Recipient* and the *Legitimate Requestor* in comparison to the traffic that is sent by the server farms, as can be seen in the following graphs. The zombies are spoofing their source IP address to be that of the *Innocent Recipient*'s; therefore all traffic initiated by the zombies should be sent to the *Innocent Recipient*.

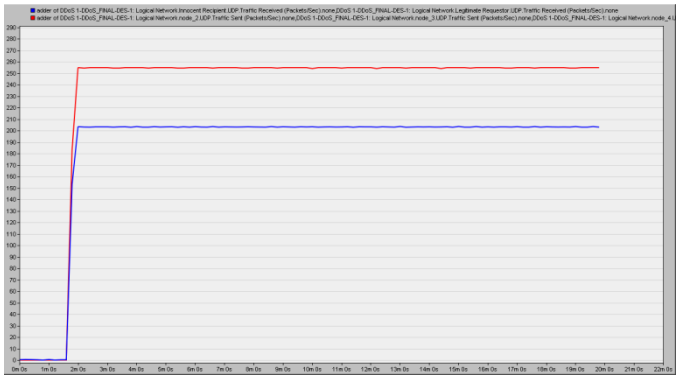

Figure 5-3. Total traffic received by *Innocent Recipient* and *Legitimate Requester* **combined** compared to traffic sent by **all** servers in the Server Farm.

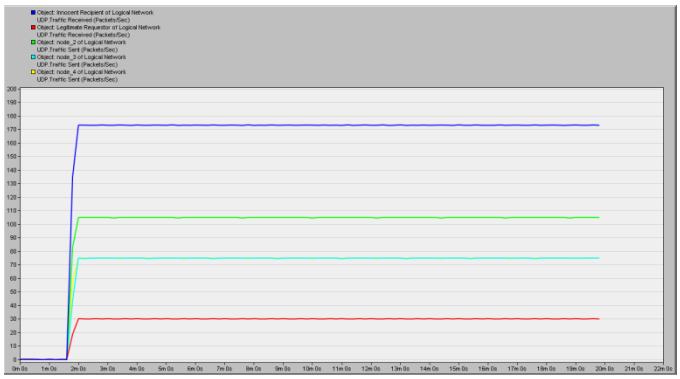

Figure 5-4. Traffic received by the *Innocent Recipient* Vs. *Legitimate Requester* Vs. the Server Farm.

As it can be seen in Figure 5-3, the total number of packets sent by the server farm is ***greater*** than the traffic actually received by the *Innocent Recipient* and the *Legitimate Requester* combined. It is our hypothesis that this is caused by the simulation ending before the packets actually get received (which will be further clarified when queuing delay is discussed). However, it can be observed that the actual amount of data received by the *Innocent Recipient* far exceeds what was sent by any individual server in the farm (Figure 5-4).

Arguably, the most important piece of data in this simulation is the link utilization to the *Innocent Recipient* from its edge router. This is the crucial element of the DDoS simulation. If the simulated attack was successful, the link utilization should be much higher than normal (optimally 100%) due to the traffic flow from the zombie requests.

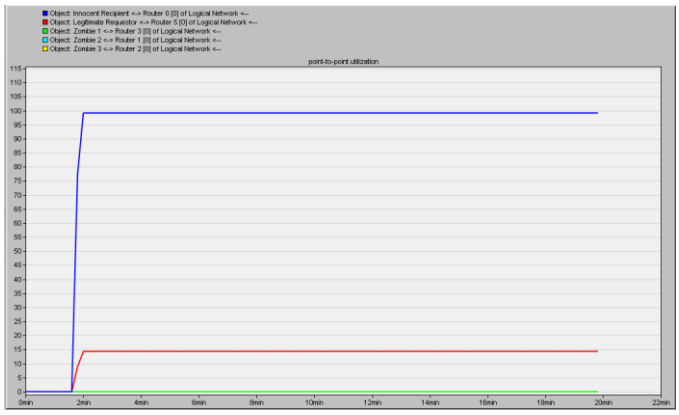

Figure 5-5. Link utilization connecting to the client and zombie nodes from their respective edge routers.

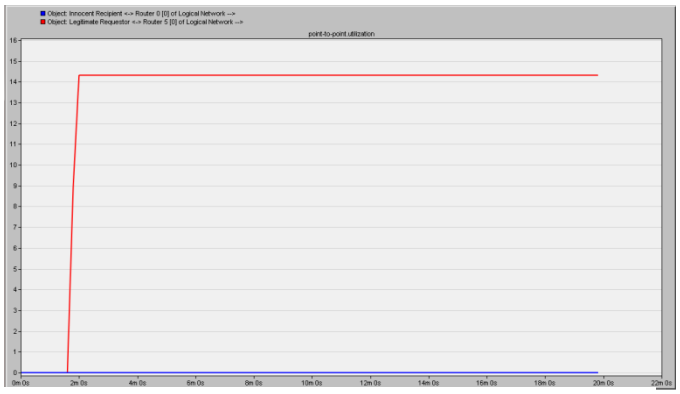

Figure 5-6. Link utilization from client nodes to their respective edge routers.

In Figure 5-5, the zombie nodes all have a link utilization of zero. This is expected because no data will be transmitted back to them after they send their initial request with a spoofed IP address. The *Legitimate Requester* (the red line in Figure 5-5) does have some link utilization due to the video conference occurring between it and the server farm. The link from *router_0* to the *Innocent Recipient,* however, is highly utilized as was expected. Even though the *Innocent Recipient* is never sending any traffic, it has the most utilized link. This is caused by the amount of traffic that is being transmitted from the server nodes as requested by the three zombies.

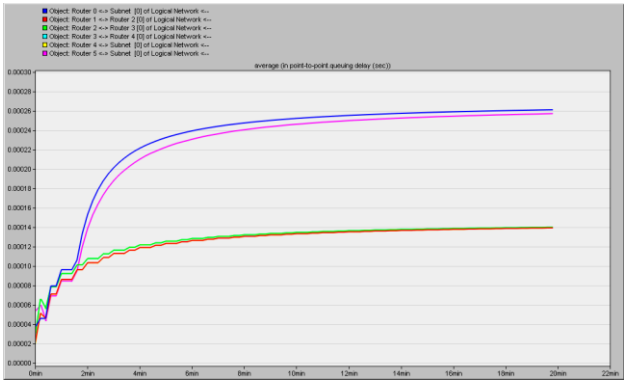

Figure 5-7. Average queuing delay to the edge routers from the subnets.

Since the utilization for the *Innocent Recipient* is so high (Figure 5-5), it actually has a negative effect on the ***entire network***. As the link utilization for the *Innocent Recipient* rises, the queuing delay for its respective edge router will also grow. This results in a domino effect that will be manifested as increased queuing delay across all network paths that carry the UDP traffic to the *Innocent Recipient*. This effect can be confirmed by examining the queuing delay experienced by the *Legitimate Requester*. The delay for both the *Innocent Recipient* and the *Legitimate Requester* is almost equivalent.

So as we have seen, the zombies are now able to spoof source IP addresses and catastrophically effect the network. Not only is the link utilization for the *Innocent Recipient* directly affected, but the entire network begins to see the increase of queuing delay. All of the initial expectations of this experiment were verified in this simulation.

5.2 Spoofing with a Defended Server Farm

By implementing Hop-Count filtering, it is possible to prevent the calamitous consumption of the link as discussed in section 5.1. This defense mechanism, implemented as described in section 4.5, would prevent packets from being transmitted by the server farm to the *Innocent Recipient*. This would eliminate, not only the collapse of the *Innocent Recipient's* link, but also the large queuing delay throughout the network.

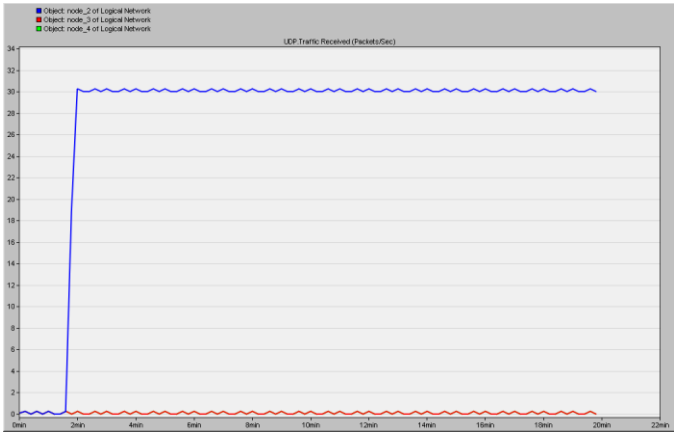

Figure 5-8. Traffic received by the server farm in a defended scenario.

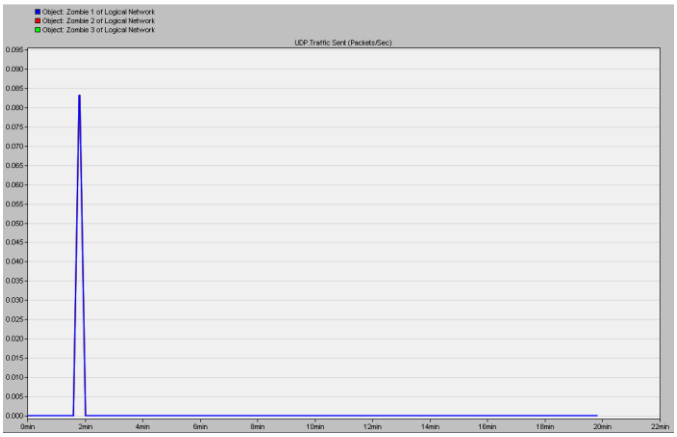

Figure 5-9. Traffic sent by the zombies in a defended scenario.

As seen in section 5.1, when the defensive measure was applied, there was no deviation in the UDP traffic received by the server farm and the UDP traffic sent by the zombies (Figure 5-8 and 5-9 respectively). However, the amount of

traffic sent by the server has significantly decreased (Figure 5-10). The only traffic actually being sent by the server farm is the traffic that was requested by the *Legitimate Requester*. The server farm detects a difference in the expected hop count and never sends traffic to the *Innocent Recipient*.

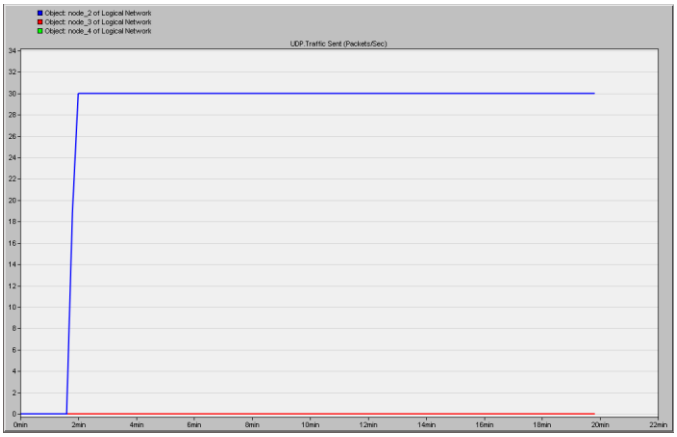

Figure 5-10. Traffic sent by the server farm in a defended scenario.

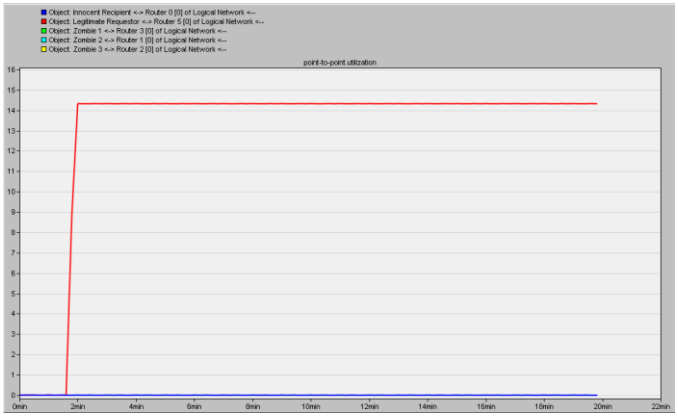

Figure 5-11. Link utilization for the clients and zombies from their respective edge routers.

This can be further verified by inspecting the link utilization to the *Innocent Recipient* from its edge router. In the previous simulation, we saw that link was highly utilized when the defense was disabled. When enabled, the utilization of this link is now close to zero, indicating the success of the defense mechanism. The utilization for the *Legitimate Requester* (the red line in Figure 5-11) now becomes the highest value in the simulation.

As expected, the noticeable decrease in link utilization for the *Innocent Recipient* also has a major impact on the queuing delay between the *Innocent Recipient*'s edge router and the subnet in both directions.

The queuing delay experienced by the *Innocent Recipient's* edge router is now "normalized" to that of the edge routers for the zombie nodes. The only prevalent queuing delay is now experienced by the *Legitimate Requester's* edge router, since it is the only node sending and receiving data.

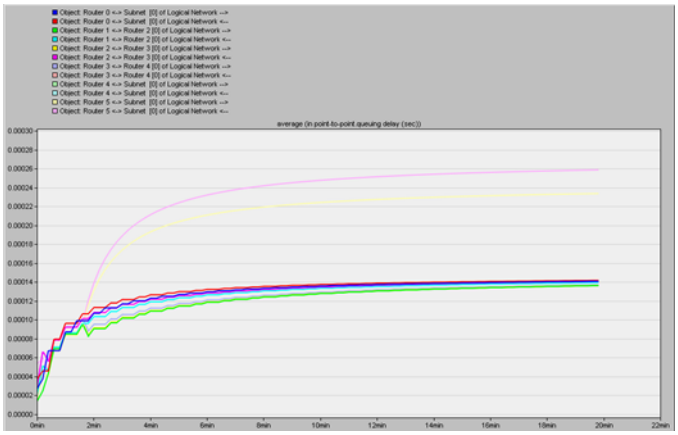

Figure 5-12. Average queuing delay between edge router and subnet (both directions).

By implementing a defensive mechanism, (in this case Hop-Count filtering), it is possible to prevent a DDoS attack over UDP. This defense not only protects the *Innocent Recipient* by reducing link utilization and queuing delay, but also reduces the strain on the resources of the server farm.

6. Conclusion

In conclusion, we have demonstrated the ease at which a DDoS attack and defense can be modeled in OPNET. This research acts as a springboard into further refinements of the selected model as well as paving the way for future expansions into other forms of attack and defense.

7. Future Work

Some improvements can be made to the OPNET DDoS defense/attack model used in this paper. It would be wise to promote the *defense_mode* variable to a parameter which can be toggled as was done for *spoofIP*.

In the actual Hop-Count filtering defense, a table which correlates an IP address to its hop count is utilized. As described in section 4, we hard coded these values for simplicity. The next step in improving the model would be to implement a hash table which would contain these values so that actual lookup of hop count can occur.

When a packet is detected as part of an attack, our current implementation ignores the packet but only reports this packet loss in the simulation console. A more elegant

alternative to this is to update OPNET's packet drop statistic to enable comparison of the capabilities of different defensive measures.

Although Hop-Count filtering provides a light-weight and effective method in defending against DDoS attacks, it could potentially be improved. As it stands now, this method computes the TTL for *every* packet and compares it with the expected TTL to determine if the packet is a threat. A significant improvement would be the incorporation of a packet marking approach as proposed by Yaar et. al. called Path Identification or "Pi" [7].

In Pi, a packet is embedded with an identifier based on the path that it traverses. This is accomplished by a router which marks the last *n* bits of the hash of its IP address in the IP identification field of the packet it forwards. To ensure similar paths do not share the same identification, the markings are adjusted based on the previous router's IP address.

The path information can be obtained from the IP identification field which will include the hash of every router from any source to the server. In this way, the server can immediately disregard any requests coming from a node that follows the same network path as a previous attack. Coupled with the detection measure in Hop-Count filtering, as soon as a packet is regarded as an attack, all subsequent packets which traverse the same path will automatically be discarded. This eliminates the need to compute the hop counts for every packet as those who bear the path markings of an attacker will be immediately ignored. Not only will this combined method potentially improve overall efficiency, but the algorithm will become more robust in the event the attacker/zombie has a varying spoofed source address. Since the path markings will not change based on the source address, these packets will be dropped without requiring further Hop-Count computations.

This combined technique can be represented in the following algorithm:

```
if ip_identification_field == any known attacker
   do nothing
else
   if actual_hop_count != expected_hop_count
      record ip_identification_field as attacker
      do nothing
   else
      send response
```

In order to simulate this in OPNET, the IP process model must be altered to allow for a packet identification field. To utilize this, a node model must be altered so that it marks its last *n* bits into the packet identification field. Once complete, a similar simulation can be created as mentioned in Section 4 to verify the correctness of the proposed combined defensive measure.

References

[1] "Thepiratebay.se." Thepiratebay.se Site Info. Alexa, n.d. Web. 14 Dec. 2012. <http://www.alexa.com/siteinfo/thepiratebay.se>.

[2] Ernesto. "10,000 Artists Sign Up for Pirate Bay Promotion." 10,000 Artists Sign Up for Pirate Bay Promotion. TorrentFreak, 5 Nov. 2012. Web. 14 Dec. 2012. <http://torrentfreak.com/10000-artists-signed-up-for-pirate-bay-promotion-12110/>.

[3] "Piratebay Servers Down Due to DDoS." ZeroSecurity Tech News and Tutorials. FastFlux, 14 Nov. 2012. Web. 14 Dec. 2012.

[4] M. Stockley. "Apache Exploit Leaves P to 65% of All Websites Vulnerable."Naked Security. N.p., 26 Aug. 2011. Web. 15 Dec. 2012.

[5] H. Huang, N. Ahmed, P. Karthik. On a New Type of Denial of Service Attack in Wireless Networks: The Distributed Jammer Network. IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, VOL. 10, NO. 7, JULY 2011

[6] C. Jin, H. Wang, K. G. Shin, "Hop-count filtering: An effective defense against spoofed DDoS traffic," in Proc. 10th ACMConf. Comput. Commun. Security, Oct. 2003, pp. 30–41.

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