Trust Based DDoS Defense System

Project Report submitted in partial fulfillment of the requirement
for the degree of

Bachelor of Technology.

In

Computer Science & Engineering

Under the Supervision of

Ms. Ramanpreet Kaur

By

Shirshayan Brahmachary (111249)

to

JAYPEE UNIVERSITY OF INFORMATION TECHNOLOGY

WAKNAGHAT, SOLAN

HIMACHAL PRADESH

Certificate

This is to certify that project report entitled "***Trust Based DDoS Defense System***", submitted by ***Shirshayan Brahmachary*** in partial fulfillment for the award of degree of Bachelor of Technology in Computer Science & Engineering to Jaypee University of Information Technology, Waknaghat, Solan has been carried out under my supervision.

This work has not been submitted partially or fully to any other University or Institute for the award of this or any other degree or diploma.

Date:

Signature in full of supervisor

Name: Ms. Ramanpreet Kaur

Designation: Asst. Professor – CS/IT Dept

Acknowledgement

I would like to express my special thanks of gratitude to my guide, **Ms. Ramanpreet Kaur** as well as **Prof. Dr. S.P.Ghrera,** Head of Department - Computer Science, who gave me the golden opportunity to do this wonderful project on the topic **Trust Based DDoS Defense System** , which also helped me in doing a lot of research and I came to know about so many new things. I am extremely thankful to them.

Date: Name of the student:

Shirshayan Brahmachary (111249)

Table of contents

S.No. Topic Page No.

1. Introduction……………………………………………………………...…. 1
   - 1.1 Denial of Service attack…….……….…………………………...…. 1
   - 1.2 Distributed Denial of Service attack …..…………….…………..…….. 2
   - 1.3 What makes DDoS attacks easy…..……………………………..……... 4
   - 1.4 Why DDoS attacks carried out…………………………………..….. 5
   - 1.5 Recent DDoS attacks…………………………………………...… 6
2. Literature review……...…………………………………………………... 7
   - 2.1 Agent Handler model………………………………………………... 7
   - 2.2 Classification of DDoS attacks……………………………………… 8
   - 2.3 Bandwidth depletion attacks………………………………………… 9
     - 2.3.1 Flood attacks……………………………..……………. 9
     - 2.3.2 Amplification attacks…………………………………... 11
   - 2.4 Resource depletion attacks……………………………………………13
     - 2.4.1 Protocol exploit attacks………………………………….. 13
     - 2.4.2 Malformed packet attack……………………………….... 15
     - 2.4.3 Application layer attacks…..…………………………….. 16
3. DDoS defense……………...……………...…………………………………… 17
   - 3.1.1 Intrusion prevention……………………………………… 17
   - 3.1.2 Intrusion detection.……………………………...….. 17
   - 3.1.3 Intrusion tolerance and mitigation…………………….…. 17
4. Implementation……….…………………………………………………….. 18
   - 4.1 Operating environment……………...……………………………… 18
   - 4.2 Softwares and tools…………………………………………………. 18
   - 4.3 Proposed scheme…………………………………………………….. 19
   - 4.4 Performance evaluation parameters…………………………………..... 19
5. Observation……………………………………………………………….... 20
6. Conclusion and future scope ………………………………………………… 24
7. References…………………………………………………………….……. 25

Abbreviations

- DoS – Denial of Service
- DDoS – Distributed Denial of Service
- TCP – Transmission Control Protocol
- IP – Internet Protocol
- UDP – User Datagram Protocol
- ICMP - Internet Control Message Protocol
- ACK – Acknowledge
- SYN – Synchronize
- OMNeT – Objective Modular Network Testbed
- ReaSE – Realistic Simulation Environment

List of figures

S.No.	Title	Page No.
Fig 1	Schematic diagram representing a DoS attack process	3
Fig 2	Agent-Handler Model of DDoS Attack	**7**
Fig 3	Classification of DDoS attacks	9
Fig 4	Amplification flood attack	13
Fig 5	Demonstration of a SYN attack	15
Fig 6	Showing the process of initiation of packet transfer	21
Fig 7	Showing flow of packet between routers and host computers	21
Fig 8	Showing number of packets sent and received by a module	22
Fig 9	Showing queue length	22
Fig 10	Showing packet details	23
Fig 11	Showing event logs being created	23

Abstract

Distributed Denial of Service is an attack that aims to prevent legitimate clients to access a service by making it unavailable. This is done by sending multiple packets from different sources in order to disable the processing power of the victim system.

My project aims at devising a procedure to prevent DDoS attack from taking place.

Initially I have to simulate a DDoS attack using OMNeT++. I will be creating a network having multiple computers which act as the DDoS packet source. Also, there will be a server connected to a router. The router on the other side will be attached to these computer nodes.
After assigning trust rating to each node, I will increase or decrease their rating based on the current behavior of different nodes.

This way, my system attempts to block high packet traffic to the server so as to thwart the server to crash due to DDoS attack.

In phase 1 I have simulated a network topology to carry out DDoS attack on victim server.

In phase 2, which is in the future, I will be implementing the trust based scenario on the network.

CHAPTER-1

INTRODUCTION

Internet is an integral part of our life. Thousands of people across the world access the internet every second. Many transactions are carried out every day alongwith knowledge sharing. However, a major drawback of the openness of the internet is the lack of proper internet security guarantees. Through internet, any person can send any packet to anyone without being authenticated, including unsolicited packets that deliver useless or even malicious payloads. Lack of authentication also shows that it is easier for an attacker to hide his identity, making it difficult for the victims or the law enforcement agencies to identify the real source of the attack.

Thus, if a site comes under a cyber-attack, it affects many users who access the site. DDoS attack is a type of such cyber-attack which affects the traffic on the attacked site.

1.1 DENIAL OF SERVICE ATTACK

It is an attack that aims to prevent legitimate clients to access a service by making it unavailable. It is achieved by sending to a victim resource a stream of packets that swamps his network or processing capacity by denying its access to regular clients. A DoS attack can also be directed to an operating system.

1.2 DISTRIBUTED DENIAL OF SERVICE ATTACK

DDoS attack, that is the Distributed Denial of Service attack, is a kind of Denial of Service attack. It actually adds many-to-one dimension to the DoS problem making its prevention and mitigation more difficult and the impact proportionally severe.

Here the traffic of the packets comes from multiple sources, which may be geographically distributed across the internet. These attacks engage the power of the vast number of coordinated Internet hosts to consume some critical resource at the target and deny the service to legitimate clients. Since the traffic is highly aggregated, it is very difficult to distinguish between legitimate and malicious packets. Also, the attack volume can be larger than the system can handle. A DDoS victim can suffer from huge damages ranging from system shutdown and file corruption, to partial or total loss of services. This can also affect third party software or sites that are dependent on the attacked resource.

These days carrying out a DDoS attack is like a child's place. Sophisticated, user-friendly toolkits are available over the internet. Without a click of a few buttons, an attack can be directed to someone's network. Some examples of such tools are "Hyenae" tool for windows or the "Low Orbit Ion Cannon" tool for iOS or Android systems. These programs have very simple logic structure and use small memory sizes making them easier to implement and handle.

The DDoS is evolving quickly, thus making it harder to devise ways to prevent it. Although there are not much effective strategies, there are many countermeasures that focus on either making the attack even more difficult or on making the attacker accountable

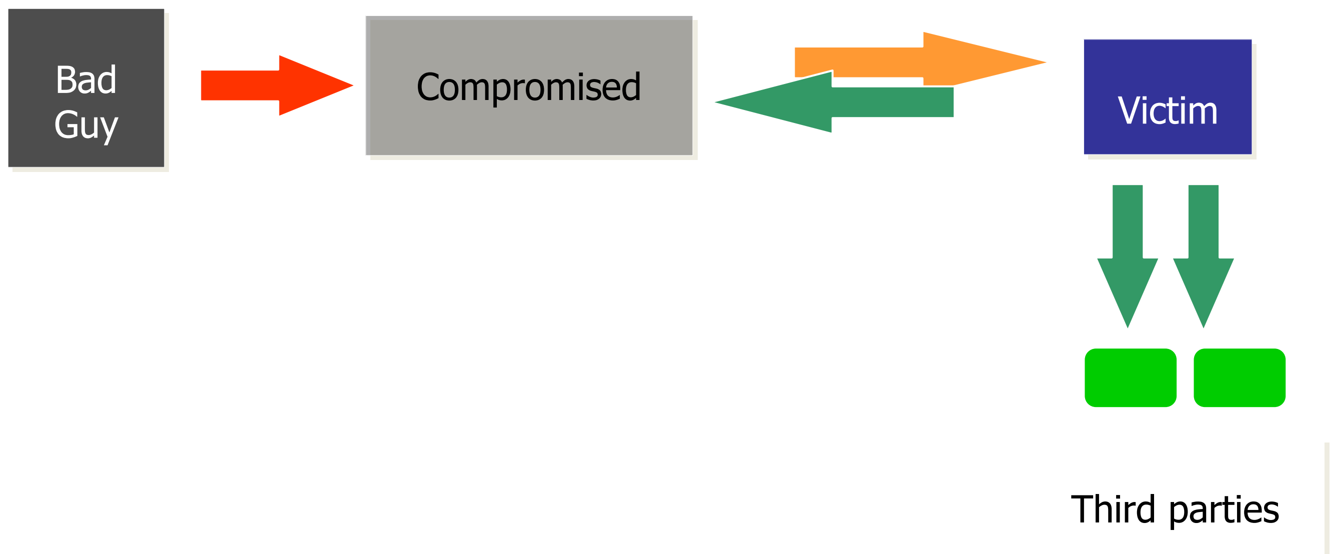

Fig 1. Schematic diagram representing a DoS attack process

1.3 WHAT MAKES DDoS ATTACKS EASY?

The TCP/IP Protocol suite, which is the most widely used protocol for data communication assumes that all systems participating in the communication have no malicious contents. Thus, there is almost no security built into the internet infrastructure that can protect the systems from other hosts not regulating their own behavior. Take for example the fact that TCP protocol assumes that the hosts will reduce the rate of packet transmission on detecting packet losses on account of congestion. If a particular host instead doesn't respond to the congestion condition, it can easily overwhelm the intermediate links to the destination.
Such architecture makes the internet vulnerable to many opportunities for denial of service attacks

Below are some features of the internet that make DDoS attack possible:

- **Internet Security is highly dependent:** DDoS attacks are launched from hosts whose security has been breached. Even if a host is highly secure, it opens itself to the possibility of a DDoS attack if there are other insecure hosts in the internet which can be used to carry out such attacks.

- **Difficulty in tracing back the attack source:** TCP/IP protocol is connectionless service. At each intermediate step, starting from the source to the final destination, the decision about the next host to hop to forward the packet is made. All such decisions are made on the basis of the destination address. So, it is easier to generate packets with incorrect or forged IP address and use them to launch the Denial of Service attacks. Example, in Linux, raw sockets can be created which a super user can use to construct all the packet contents and headers for a given packet.

  This is just the case for one source of packet generation. In Distributed Denial of Service attack, the sources are multiple. Thus, more such forgery can be made. This makes the takes the task of determining the true source of attack very difficult

Nowdays attackers even randomly change all the headers in an IP datagram keeping just the destination address constant. This makes dropping of packets based on certain characteristics very difficult as distinguishing attack packets from genuine packets becomes almost impossible.

- **A target rich environment:** There are thousands of hosts and networks over the internet with vulnerabilities that can be exploited to get access to the machines there. It is thus easier to gain some control of a large number of hosts that can be then used as a zombie to launch DDoS attacks.

- **Limited resources:** The infrastructure of the interconnected hosts and network is comprised of very limited resources. Storage capacity and bandwidth processing ability are all targets of DDoS attacks. If these resources are increased by utilizing some investments, it raises the bar on the degree an attack must reach to be effective. Even if the attack is incapable of shutting down the victim completely, it may waste it resources, reducing the level of quality as seen by end user, and making the service provider incur heavy financial losses.

- **Easy to break systems than to make them:** It takes less work to break any system than to make it. Similarly, it is easier to break the networking infrastructure than to develop them. All the hosts in the internet as well as the intermediate routers expect certain packet formats and traffic behavior. Since, at the time of the design of these hardware and software, no one wondered the system being used for malicious purposes, this can lead to unexpected behavior of the network systems as response to unexpected packets. Example, all routers are allocated buffers in memory while waiting for all the fragments of a datagram to arrive. If such a router is sent badly formed fragments, the router will keep on allocating huge amounts of memory till it runs out of memory and cannot process more datagrams.

1.4 WHY DDoS ATTACKS CARRIED OUT??

There are many reasons why DDoS attacks are carried out. Some of them being:

- To have an advantage over another political party or business venture by making its website inaccessible to general public.

- To extort money from fast-growing established companies by gaining control over its online systems.
- To gain respect in the cyber world.
- Even common internet users can use tools and carry out DDoS attack just for fun.

1.5 RECENT DDoS ATTACKS

- Israeli Government's website shut down in retaliation for killing 14-year old Orwah Hammad – October 24, 2014
- Anonymous attacked Chinese government website to leak the data and bring it down – October 14,2014
- A massive DNS DDoS attack was reported by US security firm Incapsula on one of its clients – May 16, 2014
- Chat system Campfire came under DDoS attack after hackers blackmailed the company for money which was refused by them. – March 24, 2014
- Viber's website brought down by Syrian Army – October 13, 2013

CHAPTER-2

LITERATURE REVIEW

Here I will be mentioning about types of DDoS atacks and its common model.

2.1 AGENT HANDLER MODEL

An Agent-Handler DDoS attack network consists of clients, handlers and agents.

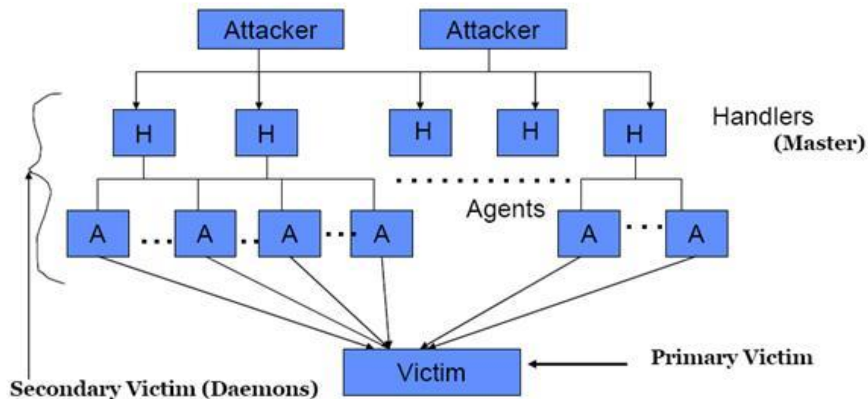

Fig 2. Agent-Handler Model of DDoS Attack.

The client platform is the one where the attacker communicates with the rest of the DDoS attack network. The handlers are software packages located on computing systems throughout the internet. The attacker communicates with the agents, indirectly, with the help of these handlers. The agent softwares are placed in the compromised systems that will eventually carry out the attack on the victim's system. The attacker routinely communicates with many handlers to identify which agents are on and running, when to schedule an attack or when to

upgrade agents. Usually, the attackers try to place the handler softwares on compromised network servers or router that handle large amount of traffic. Thus, this technique makes it harder to identify messages transferred between the client and handler and between the handler and agents. The communications between these entities are usually via TCP, UDP or ICMP protocol services. Another advantage of this technique for the attackers is that the owners and users of these agent systems have no idea that their systems are compromised and are used to carry out DDoS attacks. When an agent is being used for the attack, they are instructed to use only a small amount of resources in terms of both memory and bandwidth, so that the owners of these systems experience minimal change in performance.

Handlers are sometime also referred to as masters since they give instructions as guided by the attacker. Agents on the other hand are referred to as daemons or slave since they take the instructions and act accordingly. There are also called the secondary victim; the attacked system being the primary victim

.

2.2 CLASSIFICATION OF DDoS ATTACKS

There are many types of DDoS Attacks. But mainly there are two classes: **bandwidth depletion** (or volume-based) and **resource depletion** (or protocol-based) attack.

A bandwidth depletion attack is designed to flood the victim's network with huge, unwanted amount of traffic so as to prevent the legitimate traffic from reaching the primary victim system.

A resource depletion attack is designed to tie up the resources of a victim system. So, it actually targets a sever or process on the victim system by sending packets that misuse network protocol communication or send malformed packets to tie up legitimate resources and preventing them to reach legitimate users.

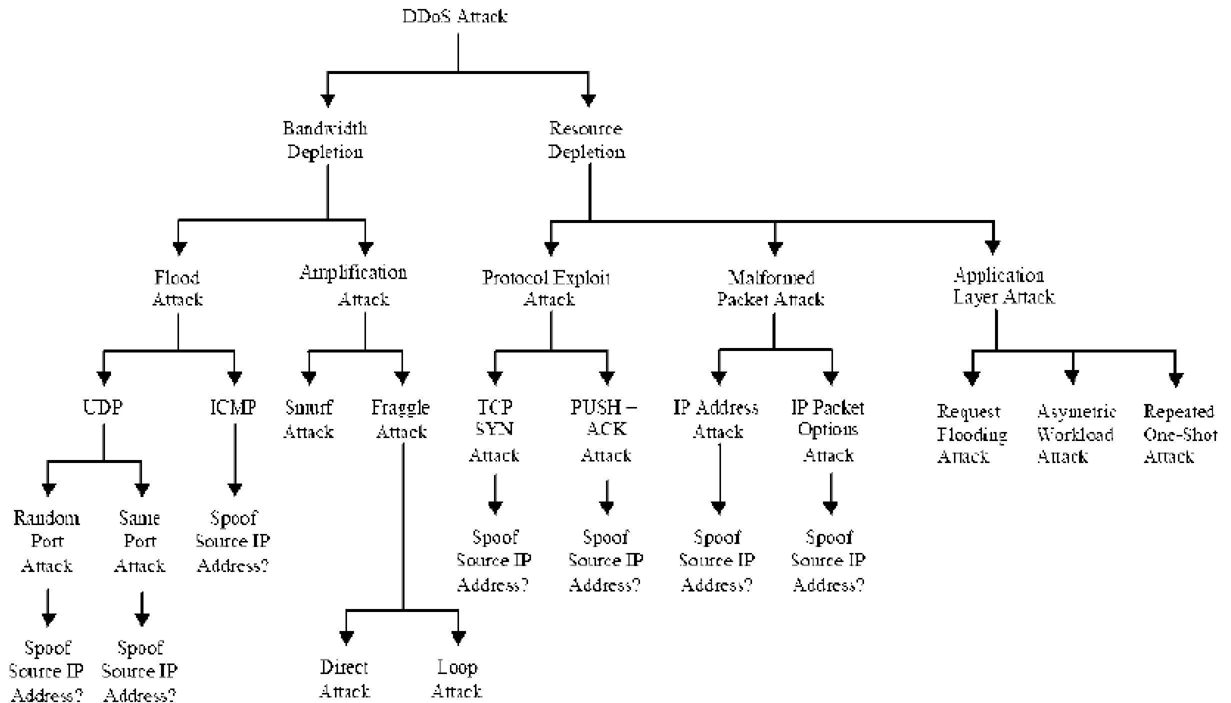

Fig 3. Classification of DDoS attacks

2.3 BANDWIDTH DEPLETION ATTACKS

There are two classes of DDoS bandwidth depletion attacks – **flood attack** and **amplification attack.**

2.3.1 FLOOD ATTACKS

In a DDoS flood attack, the zombies flood the victim system with IP traffic. The large volume of packets sent by the zombies to the victim's system slows it down, crashes the system or saturates the network bandwidth. This prevents the legitimate users from accessing the victim.

UDP FLOOD ATTACKS

User Datagram Protocol is connectionless protocol. When data packets are sent via UDP, there is no handshaking required between sender and receiver, and the receiver system will just receive packets it must process. A large number of UDP packets sent to a victim system can saturate the network, depleting the bandwidth available for legitimate service requests to the victim system. In a DDoS User Datagram Protocol Flood attack, the UDP packets are sent to either random or specified ports on the victim system.

Typically, User Datagram Protocol Flood attacks are designed to attack random victim ports. This causes the victim system to process the incoming data to try to determine which applications have requested data. If the victim system is not running any application on the targeted port, then the victim system will send out an ICMP packet to the sending system indicating a "destined port unreachable" message. Often, the attacking DDoS tool will also spoof the source IP address of the attacking packets. This helps hide the identity of the secondary victim and it insures that return packets from the victim system are not sent back to the zombies, but to another computer with the spoofed address.

User Datagram Protocol Flood attack may also fill the bandwidth of connections located around the victim system, depending on the network architecture and line-speed. This can sometime cause systems connected to a network near a victim system to experience problems with their connectivity.

ICMP FLOOD ATTACKS

Internet Control Message Protocol packets are designed for network management features such as location network equipment and determining the number of hops or round-trip-time to get from the source location to the destination. Like for example, ICMP_ECHO_REPLY packets, that is, “ping” allow the user to send a request to a destination system and receive a response with the round trip time. A DDoS Internet Control Message Protocol flood attack occurs when the zombies send large volumes ICMP_ECHO_REPLY packets to the victim system. These packets signal the victim system to reply and the combination of traffic saturates the bandwidth of the victim’s network connection. As for the User Datagram Protocol flood attack, the source IP address may be spoofed.

2.3.2 AMPLIFICATION ATTACKS

A DDoS amplification attack aims at using the broadcast IP address feature found on most of the routers to amplify and reflect the attack. This scenario is shown in fig.4

This technique allows a sending system to specify a broadcast IP address as the destination address rather than a specific address. This instructs the routers servicing the packets within the network to send them to all the IP addresses within the broadcast address range. For this type of DDoS attack, the attacker can send the broadcast message directly, or the attacker can use the agents to send the broadcast message to increase the volume of attacking traffic. If the attacker decides to send the broadcast message directly, this attack provides the attacker with the ability to use the systems within the broadcast network as zombies without requiring to infiltrate them or install any agent software.

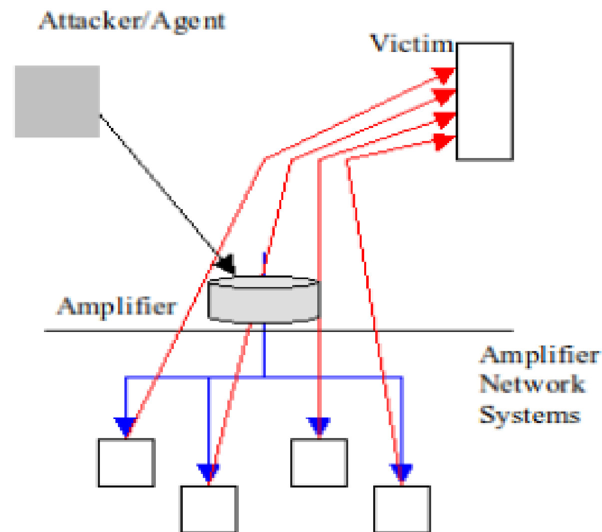

Fig 4. Amplification flood attack

There are further two types of amplification attacks – **Smurf attack** and **Fraggle attack**.

SMURF ATTACKS

This is a kind of amplification DDoS attack where the attacker sends packets to a network amplifier which is a system supporting broadcast addressing, with the return address spoofed to the victim's IP address. The attacking packets are mainly ICMP ECHO REQUESTs, which are packets that request the receiver to generate ICMP ECHO REPLY packet. The amplifier sends the ICMP ECHO REQUEST packets to all the systems within the broadcast address range, and each of these systems returns an ICMP ECHO REPLY to the target victim's IP address. This type of attack amplifies the original packet tens or hundreds of times.

FRAGGLE ATTACKS

A DDoS Fraggle attack is almost similar to a Smurf attack in the sense that the attacker sends packets to a network amplifier. Fraggle differs from Smurf in that Fraggle uses UDP ECHO packets instead of ICMP ECHO packets. There is a variation of the Fraggle attack where the UDP ECHO packets are sent to the port that supports character generation, such as the Chargen, port 19 of UNIX systems, with the return address spoofed to the victim's echo service, that is, echo port 7 of UNIX systems, creating an infinite loop. The UDP Fraggle packet will target the character generator in the systems reached by the broadcast address. These systems each generate a character to send to the echo service in the victim system, which will resend an echo packet back to the character generator, and the process repeats. This attack generates even more malicious traffic and can create even more damaging effects than just a Smurf attack.

2.4 RESOURCE DEPLETION ATTACKS

DDoS resource depletion attacks involve the attacker sending packets that misuse the network protocol communications or sending malformed packets that tie up network resources so that none are left for legitimate users.

2.4.1 PROTOCOL EXPLOIT ATTACKS

TCP SYN ATTACKS

In this type of Protocol exploit-resource depletion attack, the attacker sends a succession of SYN requests to a target's system. When a client attempts to start a TCP connection to

a server, the client and server exchange a series of messages which normally runs like follows:

1. The client requests a connection by sending a SYN (synchronize) message to the server.

2. The server acknowledges this request by sending SYN-ACK back to the client.

3. The client responds with an ACK, and the connection is established.

This is called the TCP three-way handshake, and is used for every connection establishment using the TCP protocol.

The SYN attack works if a server allocates resources after receiving a SYN, but before it has received the ACK. The attacker instructs the zombies to send bogus, random TCP SYN requests to a victim server in order to tip up the server's processor resources. Thus, prevent the server from responding to legitimate requests. The TCP SYN attack exploits the three-way handshake between sending system and the receiving system by sending large volumes of TCP SYN packets to the victim system with spoofed source IP address, so that victim system responds to a non-requesting system. Eventually, if volume of TCP SYN attack requests is large and they continue over time, the victim system will run out of resources and be unable to respond to any legitimate users.

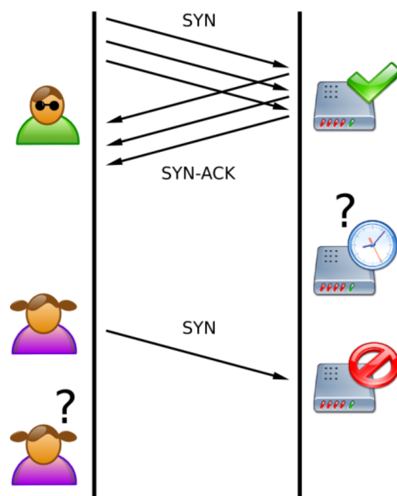

Fig 5. Demonstration of a SYN attack

PUSH ACK ATTACKS

In TCP protocol, packets that are sent to a destination are buffered within the TCP stack and when the stack is full, the packets get sent on to the receiving system. However, the sender can request the receiving system to unload the contents of the buffer before the buffer becomes full by sending a packet with the PUSH bit set to one.

PUSH is a one-bit flag within the TCP header. TCP stores incoming data in large blocks for passage on the receiving system in order to minimize the processing overhead required by the receiving system each time it must unload a non-empty buffer.

The PUSH ACK attack is similar to TCP SYN attack in that its goal is to deplete the resources of the victim system. The attacking agents send TCP packets with the PUSH and ACK bits set to one. These packets instruct the victim system to unload all data in the TCP buffer, regardless of whether or not the buffer is full, and send an acknowledgement when complete. If this process is repeated with multiple agents, the receiving system cannot process the large volume of incoming packets and it will crash.

2.4.2 MALFORMED PACKET ATTACK

A malformed packet attack is an attack where the attacker instructs the zombies to send incorrectly formed IP packets to the victim system in order to crash the victim system. There are two types of malformed packets attacks.

In **IP address attack**, the packet contains the same source and destination IP address. This can confuse the operating system of the victim system and cause the victim system to crash. In an **IP packet options attack**, a malformed packet may randomize the optional fiends within an IP packet and set all quantity of service bits to one so that the victim system must use additional processing time to analyze the traffic. If this attack is

multiplied using enough agents, it can shut down the processing ability of the victim system.

2.4.3 APPLICATION LAYER ATTACKS

Nowadays, attackers are quickly tweaking their botnets to make attack traffic look highly similar to legitimate, routine traffic. It is said that instead of the huge burst of traffic that marks when an attack begins, traffic will begin to ramp up slowly as bots join the attack at random intervals with each bot varying its attack style, making it increasingly difficult to separate real users from bots. There are several tools that automatically launch the Layer 7 DDoS attacks. Example is the Apache L7DA.

CHAPTER-3

DDoS DEFENSE

Even though DDoS attacks are hard to solve, there are ways to defense against them. There are broadly three classes of DDoS defense mechanisms – **Intrusion Prevention**, **Intrusion Detection**, **Intrusion tolerance and mitigation**.

3.1 INTRUSION PREVENTION

The best mitigation strategy against any attack is to completely prevent the attack. In this stage, we try to stop DDoS attacks from being launch in the first place. There are many DDoS defense mechanisms that attempt to prevent systems from attacks like globally **coordinated filters, ingress filtering, egress filtering, route-based distributed packet filtering.**

3.2 INTRUSION DETECTION

This technique is a very active research area. By performing intrusion detection, a host computer and a network can guard themselves against being a source of network attacks or being a victim of a DDoS attack. Intrusion detection systems detect DDoS attacks either by using the database of know signature or by recognizing anomalies in system behavior.

3.3 INTRUSION TOLERANCE AND MITIGATION

Research on intrusion tolerance accepts that it is impossible to prevent or stop DDoS completely and focuses on minimizing the attack impact and on maximizing the quality of its services.

CHAPTER-4

IMPLEMENTATION

Here I will be discussing about my project implementations.

4.1 Operating Environment

- OS: Linux-Ubuntu Version 14.
- Processor: Intel core i5

4.2 Softwares & Tools

- OMNeT++ version 4.6
  OMNeT is an event simulation environment. Its primary application area is simulation of communication networks. But it can be used in other areas like the simulation of complex IT systems, queuing networks or hardware architectures as well.

  It provides architecture for models. Components or modules are programmed in C++, and then assembled into larger components and models using a high level language called NED.

- ReaSE
  It is an open-source network simulation framework for OMNeT++. It is able to create realistic simulation environments with respect to hierarchical network topologies, self-similar background traffic, and traffic based on real attack tools.

It provides a GUI for generation of NED files including a realistic topology and necessary traffic generation entities. In addition, OMNeT++ is extended in a way that hierarchical routing and traffic generation are enabled within a user's simulation model.

4.3 Proposed scheme

- Make a network and carry out flood attack on a router. [*Presently completed*]
- Each packet source component is given an initial '**Trust Rating**' of **0.5**
- Observe the normal traffic flow on a normal network and set a threshold value based on the observation.
- Based on threshold, incoming traffic from different components of the simulated network is examined periodically.
- If from a source, incoming traffic to the router is abnormal or above threshold, decrement its trust rating. If from a source, incoming traffic to the router is normal or below threshold, increment its trust rating.
- Based on different rating, priority is set for the sources and requests from them are served accordingly. For, source whose trust rating reaches 0, temporarily disable its connection

4.4 PERFORMANCE EVALUATION PARAMETERS

- **Throughput:** Rate of successful packet delivery over a channel. Measured in data packets per second.
- **Link utilization:** Actual amount of packets being delivered over the channel.
- **Packet drop:** Takes place when packets of data fail to reach their destination.

Observations

I have made a network of 4 host computers, 2 each connected to 2 different routers

I have observed that the network is working perfectly. The packet flow is being shown and an "Arrived" receipt is shown when a packet received by a component and a "Sending a Packet" message is shown when a host sends a packet.

I have provided flow of TCP packets

When the network cannot allow beyond a specific number of packets, the remaining packets are stored in the queue and then they are allowed to flow when the packet gets less congested.

On clicking on a node, a separate window appears which shows the number of packets received and number of packets sent by that module.

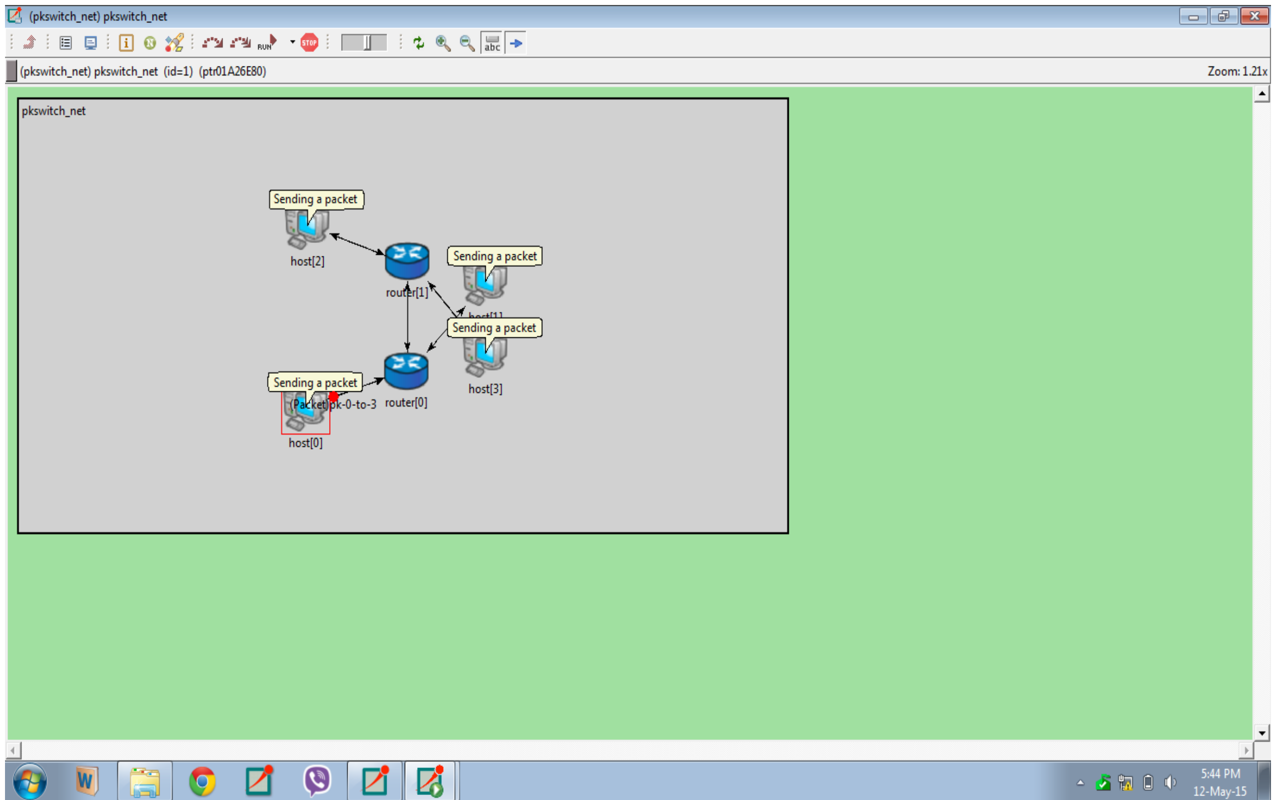

Fig 6. Showing the process of initiation of packet transfer

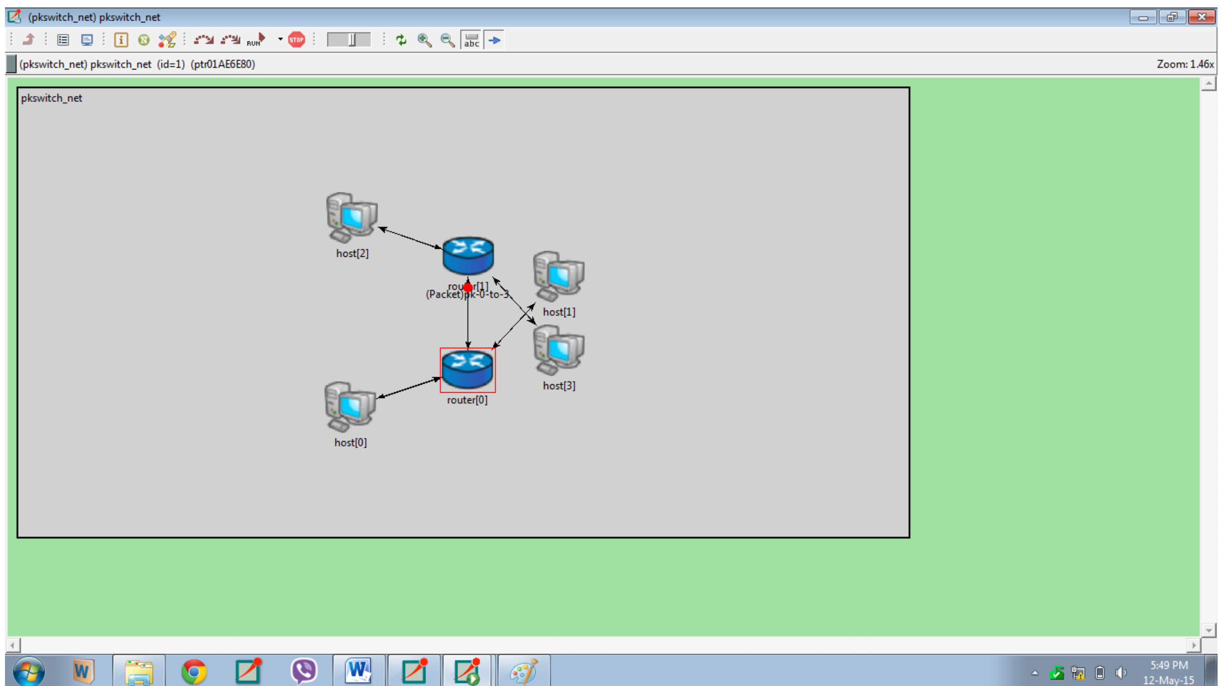

Fig 7. Showing flow of packet between routers and host computers

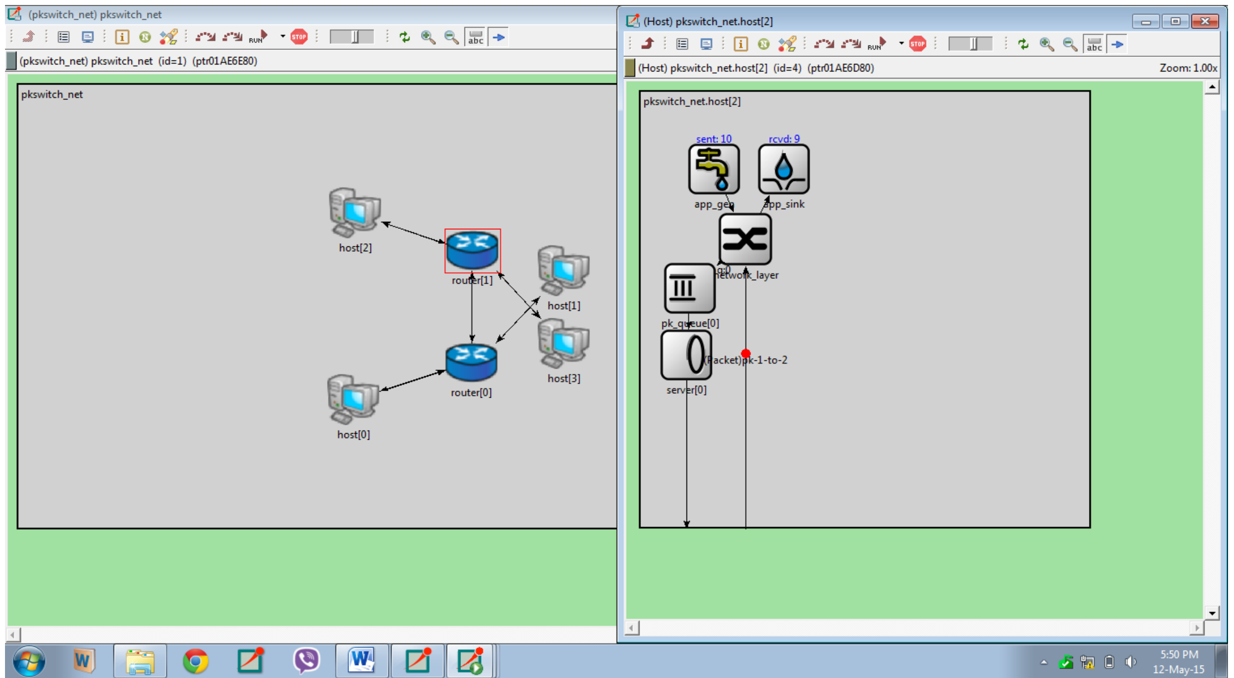

Fig 8. Showing number of packets sent and received by a particular module

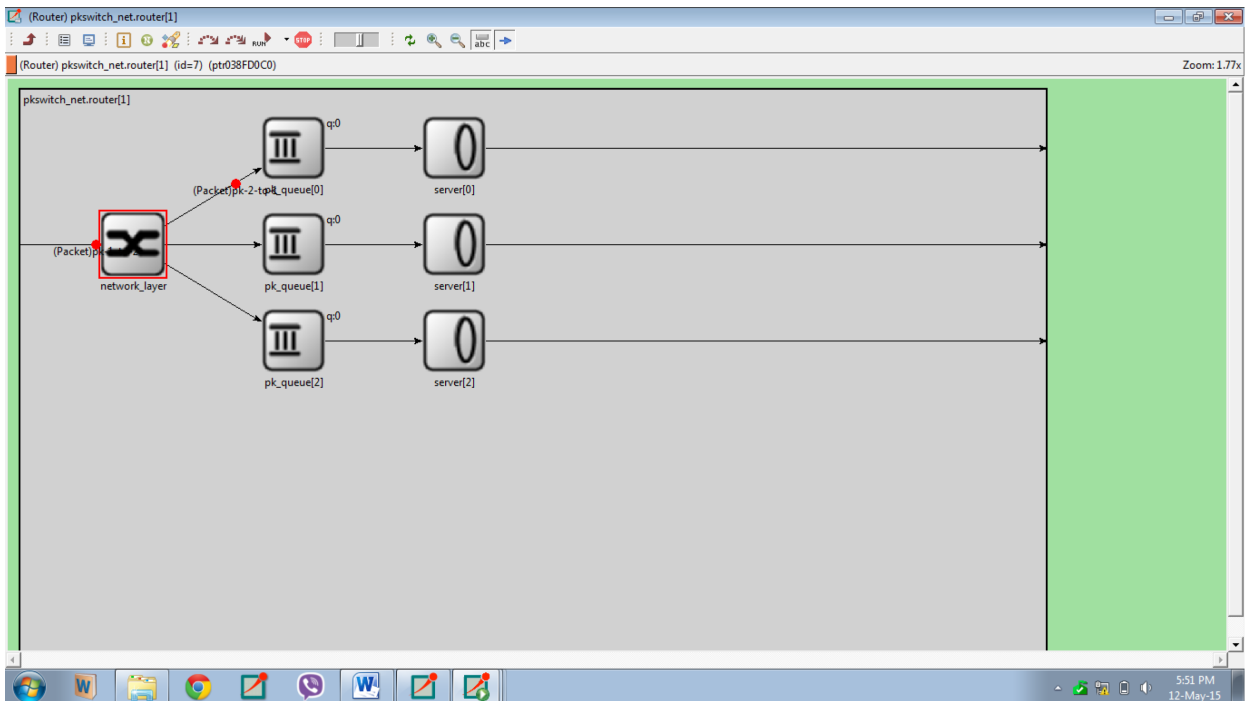

Fig 9. Showing queue length

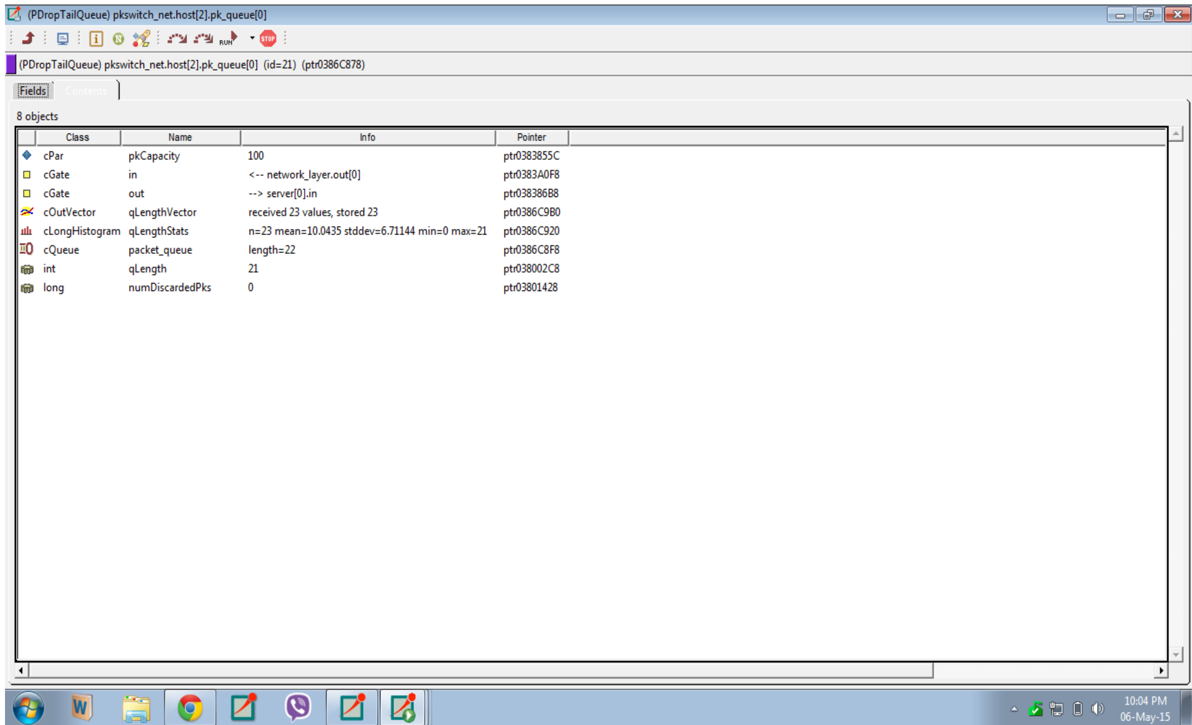

Fig 10. Showing packet details

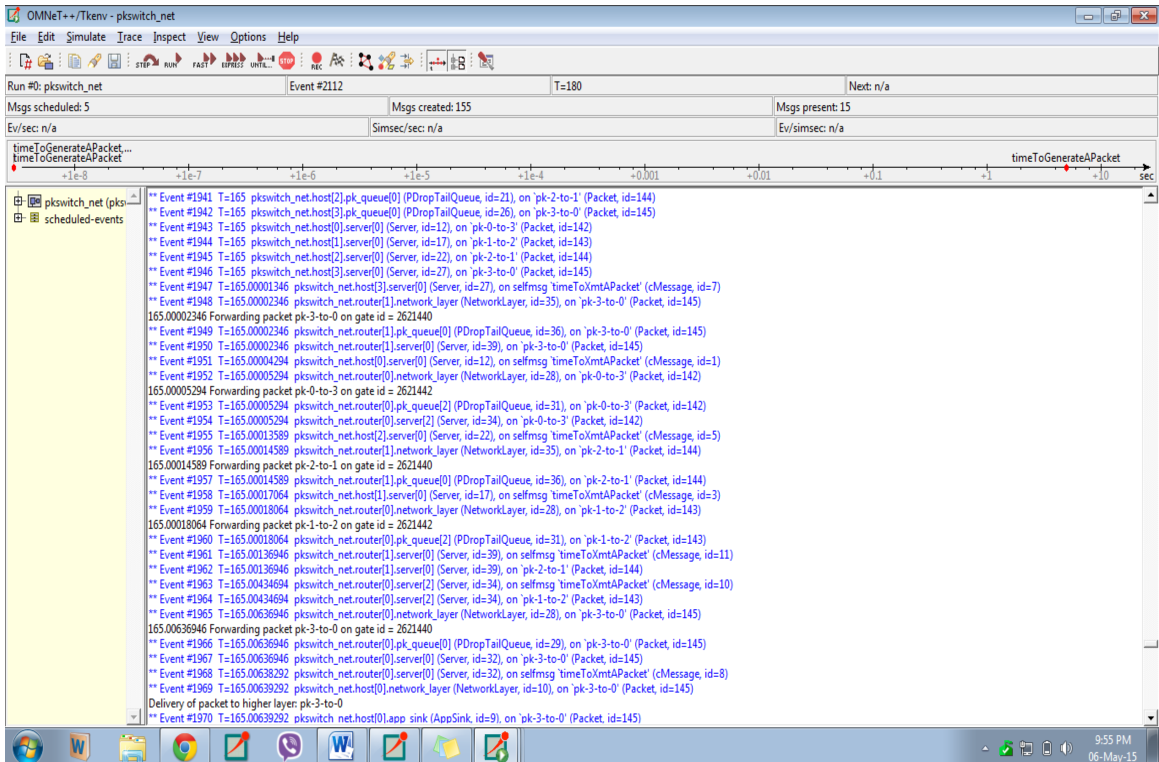

Fig 11. Showing event logs being created.

Conclusion

In this paper, I provided a review on Distributed Denial of Service attack, its common attacking techniques, the client handler model, which is the most commonly used architecture to carry out a DDoS attack, and defenses strategies.

I have also provided my implementation procedure on setting up a network of systems and showed packet flow.

Future scope

I have implemented only the network and showed flow of packets. The project can be carried further to simulate a DDoS attack and then a procedure can be implemented to stop the attacks by providing trust rating to each host and then checking the flow of packets from the hosts and varying the trust ratings based on the same. Thus those hosts having below average trust rating can be blocked as they could be sending malicious packets.

References

Websites:

- www.omnetpp.org
- www.scolar.google.co.in
- www.wikipedia.org
- https://projekte.tm.uka.de

Papers:

- **David Paltrinieri**,*"DDoS mitigation through a collaborative trust-based request prioritization",* Year 2009, pp.156
- **Christos Douligeris**, *"DDoS attacks and defense mechanisms: classification and state-of-the-art",* Year 2003, pp.24
- **Puneet Zaroo,** *"A survey of DDoS attacks and some DDoS defense mechanisms",* pp.16
- **Jelena Mirkovic,** *"A taxomony of DDoS attacks and DDoS defense mechanisms",* pp.12

Appendix

1. Code for app_gen.cc

```
#include <vector>

#include <omnetpp.h>

#include "packet_m.h"

#include "app_gen.h"

// Generates traffic for the network.

Define_Module(AppGen);

void AppGen::initialize()

{

  numSent = 0;

  WATCH(pkSize);

  WATCH(iaTime);

  WATCH(numSent);

  const char *destAddressesPar = par("destAddresses");

  cStringTokenizer tokenizer(destAddressesPar);

  const char *token;

  while ((token = tokenizer.nextToken()) != NULL) {

    destAddresses.push_back(atoi(token));

  }

  // Schedule an event for sending the first packet.

  timeToGenerateAPacket = new cMessage("timeToGenerateAPacket");

  iaTime = par("iaTime");

  scheduleAt(iaTime, timeToGenerateAPacket);

}

void AppGen::handleMessage(cMessage *msg)

{

  // Time to create and send a packet.

  if (ev.isGUI()) getParentModule()->bubble("Sending a packet");

  numSent++;

  pkSize = par("pkSize");

  // Choose a destination.

  int destAddress = destAddresses[intuniform(0, destAddresses.size()-1)];
```

```
  // Display the number of packets sent.
  char buf[40];
  sprintf(buf, "sent: %ld", numSent);
  getDisplayString().setTagArg("t", 0, buf);
  // Prepare a packet.
  // Application layer does not know the node address yet (hence -1)
  char pkname[40];
  sprintf(pkname,"pk-(%d)-to-%d-#%ld", -1, destAddress, numSent);
  ev << simTime() << " generating a packet " << pkname << endl;
  Packet *pk = new Packet(pkname);
  pk->setSrcAddr(-1); // Application does not know the source address.
  pk->setDestAddr(destAddress);
  pk->setBitLength(pkSize);
  pk->setTimestamp();
  send(pk,"out");
  // Schedule an event for sending the next packet.
  iaTime = par("iaTime");
  scheduleAt(simTime()+iaTime, timeToGenerateAPacket);
}
void AppGen::finish()
{
  ev << getParentModule()->getFullName() << "." << getFullName() << ":" << endl;
  ev << "  pks. sent:     " << numSent << endl;
  ev << " " << endl;
  recordScalar("#sent", numSent);
}
```

2. Code for app_gen.h

```
#ifndef __APPGEN_H
#define __APPGEN_H
class AppGen : public cSimpleModule
{
  private:   int pkSize;
```

```
    double iaTime;
    // Statistics
    long numSent;
    std::vector<int> destAddresses;
    cMessage *timeToGenerateAPacket;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};
#endif
```

3. Code for app_sink.cc

```
#include <vector>
#include <omnetpp.h>
#include "packet_m.h"
#include "app_sink.h"
Define_Module(AppSink);
void AppSink::initialize()
{
  numReceived = 0;
  WATCH(numReceived);
  hopCountVector.setName("hopCountVector");
  hopCountStats.setName("hopCountStats");
  hopCountStats.setRangeAutoUpper(0,20,1.5);
  pkDelayVector.setName("pkDelayVector");
  pkDelayStats.setName("pkDelayStats");
  // pkDelayStats.setRangeAutoUpper(0,20,1.5);
  pkSizeStats.setName("pkSizeStats");
  // pkSizeStats.setRangeAutoUpper(0,20,1.5);
}
```

```
void AppSink::handleMessage(cMessage *msg)
{
  // Handle incoming packet
  Packet *pk = check_and_cast<Packet *>(msg);
  if (ev.isGUI()) getParentModule()->bubble("Arrived");
  numReceived++;
  // Display the number of packets received.
  char buf[40];
  sprintf(buf, "rcvd: %ld", numReceived);
  getDisplayString().setTagArg("t", 0, buf);
  ev << simTime() << " received packet " << pk->getName() << " after " << pk->getHopCount() << "hops" << endl;
  hopCountVector.record(pk->getHopCount());
  hopCountStats.collect(pk->getHopCount());
  pkDelayVector.record(simTime() - pk->getTimestamp());
  pkDelayStats.collect(simTime() - pk->getTimestamp());
  pkSizeStats.collect(pk->getBitLength());
  delete pk; pk = NULL;
}
void AppSink::finish()
{
  ev << getParentModule()->getFullName() << "." << getFullName() << ":" << endl;
  ev << "  pks. received: " << numReceived << endl;
  ev << "  hop count, min:    " << hopCountStats.getMin() << endl;
  ev << "  hop count, max:    " << hopCountStats.getMax() << endl;
  ev << "  hop count, mean:   " << hopCountStats.getMean() << endl;
  ev << "  hop count, stddev: " << hopCountStats.getStddev() << endl;
  ev << "  pk. delay, min:    " << pkDelayStats.getMin() << endl;
  ev << "  pk. delay, max:    " << pkDelayStats.getMax() << endl;
  ev << "  pk. delay, mean:   " << pkDelayStats.getMean() << endl;
  ev << "  pk. delay, stddev: " << pkDelayStats.getStddev() << endl;
  ev << "  pk. size, min:    " << pkSizeStats.getMin() << endl;
  ev << "  pk. size, max:    " << pkSizeStats.getMax() << endl;
  ev << "  pk. size, mean:   " << pkSizeStats.getMean() << endl;
  ev << "  pk. size, stddev: " << pkSizeStats.getStddev() << endl;
```

```
  ev << " " << endl;
  recordScalar("#received", numReceived);
  hopCountStats.recordAs("hop count");
  pkDelayStats.recordAs("pk delay");
  pkSizeStats.recordAs("pk size");
}
```

4. Code for app_sink.h

```
#ifndef __APPSINK_H
#define __APPSINK_H
class AppSink : public cSimpleModule
{
  private:
    // Statistics
    long numReceived;
    cLongHistogram hopCountStats;
    cOutVector hopCountVector;
    cDoubleHistogram pkDelayStats;
    cOutVector pkDelayVector;
    cLongHistogram pkSizeStats;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
};
#endif
```

5. Code for hode.ned

```
module Host
{
    parameters:
      int address;
```

```
    string destAddressList; // list of destination host addresses,
                            // separated by space. E.g. "1 3 9"
    @display("bgb=309,399");
gates:
    input in[];
    output out[];
submodules:
    app_gen: AppGen {
        parameters:
            destAddresses = destAddressList;
            @display("p=82,88;i=block/source");
    }
    app_sink: AppSink {
        parameters:
            @display("p=160,88;i=block/sink");
    }
    network_layer: NetworkLayer {
        parameters:
            @display("p=117,168;i=block/switch");
        gates:
            in[sizeof(in)];
            out[sizeof(out)];
    }
    pk_queue[sizeof(out)]: PDropTailQueue {
        parameters:
            @display("p=55,224;i=block/passiveq;q=packet_queue");
    }
    server[sizeof(out)]: Server {
        parameters:
            @display("p=51,300,c,60;i=block/server");
    }
connections:
    network_layer.localOut --> app_sink.in;
    network_layer.localIn <-- app_gen.out;
```

```
    for i=0..sizeof(in)-1 {
        network_layer.out[i] --> pk_queue[i].in;
        pk_queue[i].out --> server[i].in;
        server[i].out --> { @display("m=s"); } --> out[i];
        network_layer.in[i] <-- { @display("m=s"); } <-- in[i];
    } }
```

6. Code for module.ned

```
simple AppGen
{
    parameters:
        string destAddresses;
        volatile int pkSize;
        volatile double iaTime;
    gates:
        output out;
}
simple AppSink
{
    gates:
    input in;
}
simple NetworkLayer
{
    gates:
        input in[];
        output out[];
        input localIn;
        output localOut;
}
simple PDropTailQueue
{
    parameters:
        int pkCapacity;
```

```
    gates:
        input in;
        output out;
}
simple Server
{
    gates:
        input in;
        output out;
}
```

7. Code for networklayer.cc

```
#include <map>
#include <omnetpp.h>
#include "packet_m.h"
class NetworkLayer : public cSimpleModule
{
  private:
    int myAddress;
    bool router;
    typedef std::map<int,int> RoutingTable; // destaddr --> port
    RoutingTable rtable;
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
    virtual void finish();
    void initializeRoutingTable();
};
Define_Module(NetworkLayer);
void NetworkLayer::initialize()
{
  myAddress = getParentModule()->par("address");
```

```
  WATCH(myAddress);
  if (strcmp("router", getParentModule()->getName()) == 0) {
    router = true;
  } else {
    router = false;
  }
  initializeRoutingTable();
}
void NetworkLayer::handleMessage(cMessage *msg)
{
  Packet *pk = check_and_cast<Packet *>(msg);
  int destAddr = pk->getDestAddr();
  if (destAddr == myAddress) {
    if (router) { // Routers don't have an application layer.
      ev << "Packet discarded: " << pk->getName() << endl;
      delete pk; pk = NULL;
    } else {
      ev << "Delivery of packet to higher layer: " << pk->getName() << endl;
      send(pk,"localOut");
    }
    return;
  }
  RoutingTable::iterator it = rtable.find(destAddr);
  if (it == rtable.end()) {
    ev << "Address " << destAddr << " unreachable, discarding packet " << pk->getName() << endl;
    delete pk; pk = NULL;
    return;
  }
  if (pk->getSrcAddr() == -1) {
    char pkname[40];
    sprintf(pkname,"pk-%d-to-%d", myAddress, pk->getDestAddr());
    pk->setSrcAddr(myAddress);
    pk->setName(pkname);
  }
```

```
  int outGateId = (*it).second;

  ev << simTime() << " Forwarding packet " << pk->getName() << " on gate id = " << outGateId << endl;

  pk->setHopCount(pk->getHopCount()+1);

  send(pk, outGateId);

}

void NetworkLayer::initializeRoutingTable()

{

  cTopology *topo = new cTopology("topo");

  topo->extractByNedTypeName(cStringTokenizer("Host Router").asVector());

  ev << getParentModule()->getFullName() << " (addr = " << myAddress << "): cTopology found " << topo->getNumNodes() << " nodes\n";

  cTopology::Node *thisNode = topo->getNodeFor(getParentModule());

  for (int i = 0; i < topo->getNumNodes(); i++) {

    if (topo->getNode(i) == thisNode) continue; // skip ourselves

    topo->calculateUnweightedSingleShortestPathsTo(topo->getNode(i));

    if (thisNode->getNumPaths() == 0) continue; // This node is not connected to

                                      // any other node.

    cGate *parentModuleGate = thisNode->getPath(0)->getLocalGate();

    cModule *serverModule = parentModuleGate->getPreviousGate()->getOwnerModule();

    cModule *outputQueueModule = serverModule->gate("in")->getPreviousGate()->getOwnerModule();

    int gateId = outputQueueModule->gate("in")->getPreviousGate()->getId();

    int address = topo->getNode(i)->getModule()->par("address");

    rtable[address] = gateId;

    ev << "  towards address " << address << " gateId is " << gateId << endl;

  }

  ev << " " << endl;

  delete topo; topo = NULL;

}

void NetworkLayer::finish()

{

  RoutingTable::iterator it;

  int dest_address;

  int outGateId;

  ev << getParentModule()->getFullName() << " final routing table:" << endl;
```

```
  ev << "  destination\tnext hop" << endl;
  ev << "    address" << endl;
  ev << "  -----------\t--------" << endl;
  for(it = rtable.begin(); it != rtable.end(); it++) {
    dest_address = (*it).first;
    outGateId = (*it).second;
      cGate *outGate = gate(outGateId);
    cModule *outputQueueModule = outGate->getNextGate()->getOwnerModule();
    cModule *serverModule = outputQueueModule->gate("out")->getNextGate()->getOwnerModule();
    cGate *parentModuleGate = serverModule->gate("out")->getNextGate();
    cModule *nextHopModule = parentModuleGate->getNextGate()->getOwnerModule();
    ev << "        " << dest_address << "\t" << nextHopModule->getFullName() << endl;
  }
  ev << " " << endl;

}
```

8. Code for packet.msg

```
packet Packet
{
    int srcAddr;
    int destAddr;
    int hopCount;
}
```

9. Code for packet_m.cc

```
#ifdef _MSC_VER
#  pragma warning(disable:4101)
#  pragma warning(disable:4065)
#endif
#include <iostream>
#include <sstream>
#include "packet_m.h"
template<typename T>
std::ostream& operator<<(std::ostream& out,const T&) {return out;}
template<typename T>
```

```
void doPacking(cCommBuffer *, T& t) {

    throw cRuntimeError("Parsim error: no doPacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)));

}

template<typename T>

void doUnpacking(cCommBuffer *, T& t) {

 throw cRuntimeError("Parsim error: no doUnpacking() function for type %s or its base class (check .msg and _m.cc/h files!)",opp_typename(typeid(t)));

}

Register_Class(Packet);

Packet::Packet(const char *name, int kind) : cPacket(name,kind)

{

    this->srcAddr_var = 0;

    this->destAddr_var = 0;

    this->hopCount_var = 0;

}

Packet::Packet(const Packet& other) : cPacket(other)

{

    copy(other);

}

Packet::~Packet()

{

}

Packet& Packet::operator=(const Packet& other)

{

    if (this==&other) return *this;

    cPacket::operator=(other);

    copy(other);

    return *this;

}

void Packet::copy(const Packet& other)

{

    this->srcAddr_var = other.srcAddr_var;

    this->destAddr_var = other.destAddr_var;

    this->hopCount_var = other.hopCount_var;

}

void Packet::parsimPack(cCommBuffer *b)
```

```
{
    cPacket::parsimPack(b);
    doPacking(b,this->srcAddr_var);
    doPacking(b,this->destAddr_var);
    doPacking(b,this->hopCount_var);
}
void Packet::parsimUnpack(cCommBuffer *b)
{
    cPacket::parsimUnpack(b);
    doUnpacking(b,this->srcAddr_var);
    doUnpacking(b,this->destAddr_var);
    doUnpacking(b,this->hopCount_var);
}
int Packet::getSrcAddr() const
{
    return srcAddr_var;
}
void Packet::setSrcAddr(int srcAddr)
{
    this->srcAddr_var = srcAddr;
}
int Packet::getDestAddr() const
{
    return destAddr_var;
}
void Packet::setDestAddr(int destAddr)
{
    this->destAddr_var = destAddr;
}
int Packet::getHopCount() const
{
    return hopCount_var;
}
void Packet::setHopCount(int hopCount)
{
    this->hopCount_var = hopCount;}
```

```
class PacketDescriptor : public cClassDescriptor
{
  public:
    PacketDescriptor();
    virtual ~PacketDescriptor();
    virtual bool doesSupport(cObject *obj) const;
    virtual const char *getProperty(const char *propertyname) const;
    virtual int getFieldCount(void *object) const;
    virtual const char *getFieldName(void *object, int field) const;
    virtual int findField(void *object, const char *fieldName) const;
    virtual unsigned int getFieldTypeFlags(void *object, int field) const;
    virtual const char *getFieldTypeString(void *object, int field) const;
    virtual const char *getFieldProperty(void *object, int field, const char *propertyname) const;
    virtual int getArraySize(void *object, int field) const;
    virtual std::string getFieldAsString(void *object, int field, int i) const;
    virtual bool setFieldAsString(void *object, int field, int i, const char *value) const;
    virtual const char *getFieldStructName(void *object, int field) const;
    virtual void *getFieldStructPointer(void *object, int field, int i) const;
};
Register_ClassDescriptor(PacketDescriptor);
PacketDescriptor::PacketDescriptor() : cClassDescriptor("Packet", "cPacket")
{
}
PacketDescriptor::~PacketDescriptor()
{
}
bool PacketDescriptor::doesSupport(cObject *obj) const
{
    return dynamic_cast<Packet *>(obj)!=NULL;
}
const char *PacketDescriptor::getProperty(const char *propertyname) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? basedesc->getProperty(propertyname) : NULL;
}
int PacketDescriptor::getFieldCount(void *object) const
```

```
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    return basedesc ? 3+basedesc->getFieldCount(object) : 3;
}
unsigned int PacketDescriptor::getFieldTypeFlags(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeFlags(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static unsigned int fieldTypeFlags[] = {
        FD_ISEDITABLE,
        FD_ISEDITABLE,
        FD_ISEDITABLE,
    };
    return (field>=0 && field<3) ? fieldTypeFlags[field] : 0;
}
const char *PacketDescriptor::getFieldName(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldName(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldNames[] = {
        "srcAddr",
        "destAddr",
        "hopCount",
    };
    return (field>=0 && field<3) ? fieldNames[field] : NULL;
}
int PacketDescriptor::findField(void *object, const char *fieldName) const
{
```

```
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    int base = basedesc ? basedesc->getFieldCount(object) : 0;
    if (fieldName[0]=='s' && strcmp(fieldName, "srcAddr")==0) return base+0;
    if (fieldName[0]=='d' && strcmp(fieldName, "destAddr")==0) return base+1;
    if (fieldName[0]=='h' && strcmp(fieldName, "hopCount")==0) return base+2;
    return basedesc ? basedesc->findField(object, fieldName) : -1;
}
const char *PacketDescriptor::getFieldTypeString(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldTypeString(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldTypeStrings[] = {
        "int",        "int",  "int",
    };
    return (field>=0 && field<3) ? fieldTypeStrings[field] : NULL;
}
const char *PacketDescriptor::getFieldProperty(void *object, int field, const char *propertyname) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldProperty(object, field, propertyname);
        field -= basedesc->getFieldCount(object);
    }
    switch (field) {
        default: return NULL;
    }
}
int PacketDescriptor::getArraySize(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
```

```
        if (field < basedesc->getFieldCount(object))
            return basedesc->getArraySize(object, field);
        field -= basedesc->getFieldCount(object);
    }
    Packet *pp = (Packet *)object; (void)pp;
    switch (field) {
        default: return 0;
    }
}
std::string PacketDescriptor::getFieldAsString(void *object, int field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldAsString(object,field,i);
        field -= basedesc->getFieldCount(object);
    }
    Packet *pp = (Packet *)object; (void)pp;
    switch (field) {
        case 0: return long2string(pp->getSrcAddr());
        case 1: return long2string(pp->getDestAddr());
        case 2: return long2string(pp->getHopCount());
        default: return "";
    }
}
bool PacketDescriptor::setFieldAsString(void *object, int field, int i, const char *value) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->setFieldAsString(object,field,i,value);
        field -= basedesc->getFieldCount(object);
    }
    Packet *pp = (Packet *)object; (void)pp;
    switch (field) {
        case 0: pp->setSrcAddr(string2long(value)); return true;
```

```
        case 1: pp->setDestAddr(string2long(value)); return true;
        case 2: pp->setHopCount(string2long(value)); return true;
        default: return false;
    }
}
const char *PacketDescriptor::getFieldStructName(void *object, int field) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructName(object, field);
        field -= basedesc->getFieldCount(object);
    }
    static const char *fieldStructNames[] = {
        NULL,
        NULL,
        NULL,
    };
    return (field>=0 && field<3) ? fieldStructNames[field] : NULL;
}
void *PacketDescriptor::getFieldStructPointer(void *object, int field, int i) const
{
    cClassDescriptor *basedesc = getBaseClassDescriptor();
    if (basedesc) {
        if (field < basedesc->getFieldCount(object))
            return basedesc->getFieldStructPointer(object, field, i);
        field -= basedesc->getFieldCount(object);
    }
    Packet *pp = (Packet *)object; (void)pp;
    switch (field) {
        default: return NULL;
    }
}
```

10. Code for packet.h

```
#ifndef _PACKET_M_H_

#define _PACKET_M_H_

#include <omnetpp.h>

// opp_msgc version check

#define MSGC_VERSION 0x0402

#if (MSGC_VERSION!=OMNETPP_VERSION)

#    error Version mismatch! Probably this file was generated by an earlier version of opp_msgc: 'make clean' should help.

#endif

class Packet : public ::cPacket

{

  protected:

    int srcAddr_var;

    int destAddr_var;

    int hopCount_var;

  private:

    void copy(const Packet& other);

  protected:

    // protected and unimplemented operator==(), to prevent accidental usage

    bool operator==(const Packet&);

  public:

    Packet(const char *name=NULL, int kind=0);

    Packet(const Packet& other);

    virtual ~Packet();

    Packet& operator=(const Packet& other);

    virtual Packet *dup() const {return new Packet(*this);}

    virtual void parsimPack(cCommBuffer *b);

    virtual void parsimUnpack(cCommBuffer *b);

    virtual int getSrcAddr() const;

    virtual void setSrcAddr(int srcAddr);

    virtual int getDestAddr() const;

    virtual void setDestAddr(int destAddr);

    virtual int getHopCount() const;

    virtual void setHopCount(int hopCount);
```

```
inline void doPacking(cCommBuffer *b, Packet& obj) {obj.parsimPack(b);}

inline void doUnpacking(cCommBuffer *b, Packet& obj) {obj.parsimUnpack(b);}

#endif // _PACKET_M_H_
```

11. Code for prodtailqueue.h

```
#ifndef __PDROPTAILQUEUE_H

#define __PDROPTAILQUEUE_H

class PDropTailQueue : public cSimpleModule

{

  private:

    cQueue pkQueue;

    cLongHistogram qLengthStats;

    cOutVector qLengthVector;

    int pkCapacity;

    long numDiscardedPks;

    int qLength;

    bool serverIdle;

  protected:

    virtual void initialize();

    virtual void handleMessage(cMessage *msg);

    virtual void finish();

    ~PDropTailQueue();

  public:

    virtual void request(); // This method is called by the server to request a packet.

};

#endif
```

12. Code for packet_geometric.cc

```
#include <omnetpp.h>

double pk_geometric(double min_pk_size, double avg_pk_size)

{

  double p = 1.0/(avg_pk_size - min_pk_size);
```

```
  return (geometric(p) + min_pk_size);
}
Define_Function(pk_geometric, 2);
```

13. Code for router.ned

```
module Router
{
    parameters:
        int address;
        @display("bgb=328,186");
    gates:
        input in[];
        output out[];
    submodules:
        network_layer: NetworkLayer {
            parameters:
                @display("p=70,96;i=block/switch");
            gates:
                in[sizeof(in)];
                out[sizeof(out)];
        }
        pk_queue[sizeof(out)]: PDropTailQueue {
            parameters:
                @display("p=170,36,column,60;i=block/passiveq;q=packet_queue");
        }
        server[sizeof(out)]: Server {
            parameters:
                @display("p=270,36,column,60;i=block/server");
        }
    connections allowunconnected:
        for i=0..sizeof(in)-1 {
            network_layer.out[i] --> pk_queue[i].in;
            pk_queue[i].out --> server[i].in;
            server[i].out --> { @display("m=e"); } --> out[i];
            network_layer.in[i] <-- { @display("m=w,,,48,0"); } <-- in[i];
```