**Bold – Complete change to the part**

*Italics – Minor changes*

~~Strike out~~ – No changes needed

* Problem statement
  + ~~Title, team, URL~~
  + **Description of problem**

**Problem Description:**

Network Address Translator (NAT) while quite ubiquitous in today’s world has a single functionality – translating public IP addresses which connect to it into private IP addresses recognisable only to the organization which uses the NAT. Although this functionality has quite a few advantages, including sharing of public IP addresses, greater control and increased security for organizations, however there is no way for a NAT to balance and distribute the packet incoming packets into the finite servers which exist to serve the clients. Hence the servers run the risk of being over-utilized or underutilized and in general not optimized for the network.

A solution to deal with the problem of load balancing in a NAT is to introduce this additional functionality in the NAT. This will greatly optimize the performance of the servers and hence the network.

**Proposed Solution:**

This project aims to implement a NAT with additional functionality of load balancing. Clients connecting to NAT will have public IP addresses which will be changed by the NAT into private IP addresses. Each of the client will then be connected to video servers. Load balancing will be done while this connection is being made. The project will implement two main techniques of load balancing – round robin scheduling and least rate scheduling. For the connection to the servers, the protocol used will be UDP, and each of the servers will play the same video for testing purposes of the load balancing techniques.

**Platform:**

The platform used in the project will be NetLabs. Hardware platform will be Ubuntu 14.04 machine whereas software platform will be Linux kernel 3.2.

**Revision from Proposal:**

**System Description:**

The development of the NAT will primarily consist of developing two modules – source NAT and destination NAT. A very important tool for manipulating IP packets inside the NAT is called Netfilter. Netfilter is a set of hooks inside Linux kernel, which allows kernel modules to register callback functions with the network stack in order to intercept and manipulate the network packet.

A Packet Traversing the Netfilter System:

--->[1]--->[ROUTE]--->[3]--->[4]--->

| ↑

| |

| [ROUTE]

↓ |

[2] [5]

| ↑

| |

↓ |

When a network packet comes in, it is passed to the netfilter’s first hook NF\_INET\_PRE\_ROUTING [1]. After that, the packet goes through the routing code [ROUTE], which decides where the packet is destined to, either another port in same network interface or another interface. It also might drop the packet if it’s unrouteable.

If the packet goes to another port in the same interface, the second hook NF\_INET\_LOCAL\_IN [2] is triggered. This happens before the packet reaches to the destination port.

If the packet goes to another network interface, the third hook NF\_INET\_FORWARD [3] is called, followed by the forth hook NF\_INET\_POST\_ROUTING [4] before the packet reaches wire again.

There is one hook left, NF\_INET\_LOCAL\_OUT [5]. It’s called for local outgoing packets. Routing code is called before this hook to figure out the IP address and after this hook to decide the route.

Kernel modules can register to any of the 5 hooks.

This project uses netfilter in conjunction with a data structure called sock\_buff. This is a large struct containing all the control information required for the packet (datagram, cell, whatever). The sk\_buff elements are organized as a doubly linked list, in such a way that it is very efficient to move a sk\_buff element from the beginning/end of a list to the beginning/end of another list. A queue is defined by struct sk\_buff\_head, which includes a head and a tail pointer to sk\_buff elements.astructure

The description of these modules are as follows:

**Destination NAT:** This module is chiefly used for incoming packets to the NAT. If the destination IP address in these incoming packets matches the public IP address of the NAT, then they are accepted, otherwise these packets are dropped. Destination NAT is responsible for translating the public source IP address of the incoming packets accepted by it into private IP addresses understandable only to the system. This module is also responsible for assigning video servers to the clients which have been accepted. Due to this, load balancing implementation is also done on destination NAT and hence both the round robin as well as least rate scheduling will be done on destination NAT. The load balancing NAT handles client requests by selecting the video server for a particular session. The selection of the server is based on the traffic handled by all the servers.

We maintain two tables Table 1 and Table 2. Table 1 consists of client IP, client port, server IP, server port and timestamp which is looked up to determine the server IP & server port for each packet based on its client IP and client port. If an entry is found, the packet’s destination address and port are changed to the corresponding (server IP, server port) at the PRE\_ROUTING hook. The UDP and IP checksums are re- calculated. The timestamp is updated for the particular (client IP, client port) entry in Table 1. The selection of the server in least rate approach is based on the traffic handled by all the servers. A server with the least traffic rate is selected for a new connection.

Table 2 consists of load rates of the servers. The rate at which each server processes a packet is updated every time an incoming/ outgoing packet arrives at NAT. For a new connection, the server with the least load is selected.

A chosen set of entries

**Source NAT:** All outgoing packets from the video server are forwarded to the particular client which initiated the request. The source address of the packet from the video server is mapped to the public address of the NAT and the source port is mapped to the public port of NAT at the POST-ROUTING hook. The UDP and IP checksums are updated. The packet rate corresponding to the particular server is updated.

* Components
  + *Platform(s) for project – Kernel version*

**Platform:** Linux kernel will be used to implement NAT functionality and Load Balancing. The testing will be done in NetLabs.

**Project Area:** Forwarding IP packets incoming to a server based on round robin and least rate approach. This will be done using Linux kernel.

* + ~~Area(s) for project~~
* **Project Area:** Forwarding IP packets incoming to a server based on round robin and least rate approach. This will be done using Linux kernel.
  + **Major components decomposition**

**Components:**

1. Load balancing NAT using least rate (Kernel modules)

a) Destination NAT

b) Source NAT

c) Loadbalancer

1. Load balancing NAT using round robin (Kernel modules)

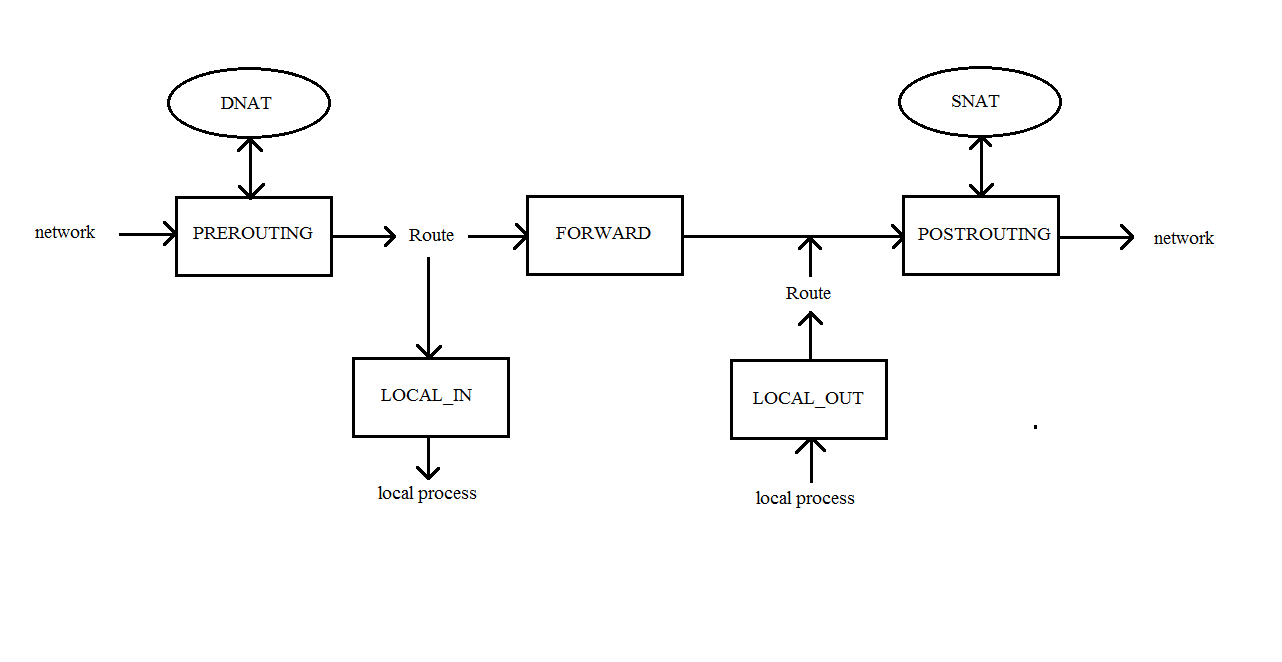
a) Destination NAT

b) Source NAT

c) Loadbalancer

1. Video servers
2. Clients

* Preliminary design and development plan
  + **High-level proposed design**

****

**Figure 2.**

* + **Low-level design – Same as per component**

1 Load balancing NAT using least rate:

1. Destination NAT:

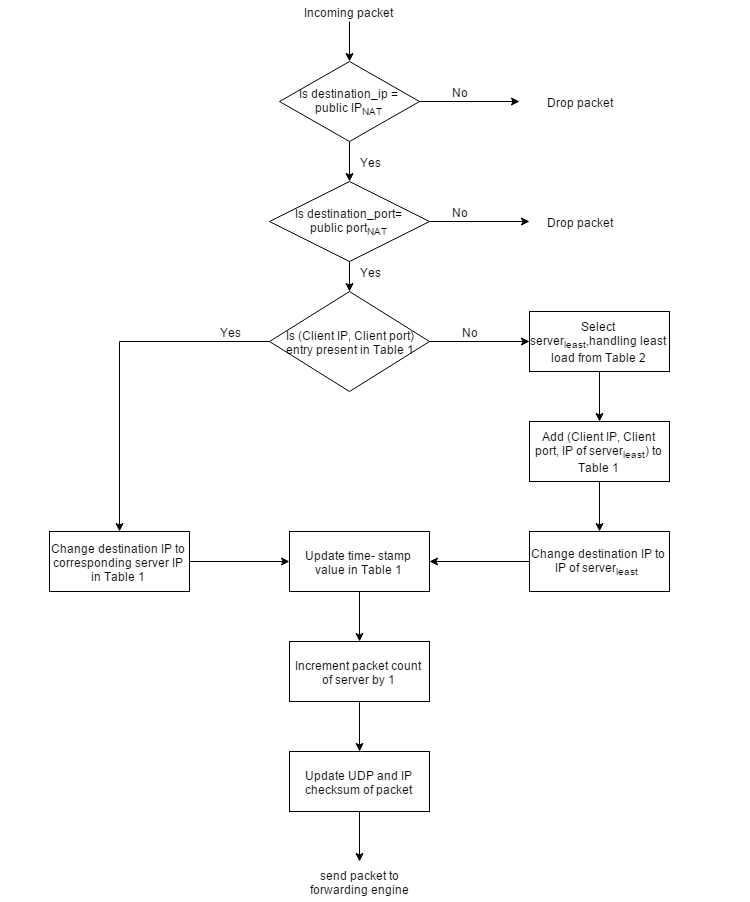


Figure 3

Destination NAT functionality is placed in the PREROUTING hook of the netfilter architecture as shown in Figure 2**(NF\_HOOKS.png)**. It modifies the destination IP address of a packet to one of the server IP addresses in the private network. The selection of server for a connection is based on least rate scheduling.

The steps taken by the DNAT for incoming packets (packets from the client) before routing is shown in Figure 3**(DNAT\_LR)**.

The destination IP address and port of packets is compared with the assigned public IP address and public port of the NAT and the packets are dropped if either or both of the comparison fail. If both match, the packet is destined to one of the servers.

The load balancing is accomplished by maintaining 2 tables. Table 1 consists of the mapping from {Client IP address, Client port} to {Server IP address}. {Client IP address, Client port} in Table 1 is searched by using {Source IP address, Source port} of the UDP packet as the key. If the key is found, the destination IP address of the packet is changed to the corresponding Server IP address. If the key is not found, it corresponds to a new connection and an entry {Client IP address, Client port, Server IP address} is added to Table 1. The destination IP address of the packet is changed to the corresponding server IP address. The selection of Server IP address is based on the packet rate entry for each server in Table 2.

Server IP address = IP address of the server with the minimum rate of packets flowing through it as per Table 2.

The timestamp for the corresponding row in Table 1 is updated for each packet. To compute the packet rate in Table 2 for the assigned server, the packet count of the server is incremented by 1.

Before the packet is sent to the routing function, the UDP and IP checksum values are updated. This is done as the destination IP address of the packet is changed by the DNAT function.

The timestamps in Table 1 serve the purpose of replacing table entries when the table is full. If the number of rows in the table has reached the maximum limit when a new connection request arrives, the entry with the earliest timestamp is replaced with the current value of {Client IP address, Client port, Server IP address}. This functionality helps to remove inactive entries in the table. The maximum size of the table is chosen so that active connections are not compromised by the deletion function.

1. Source NAT:

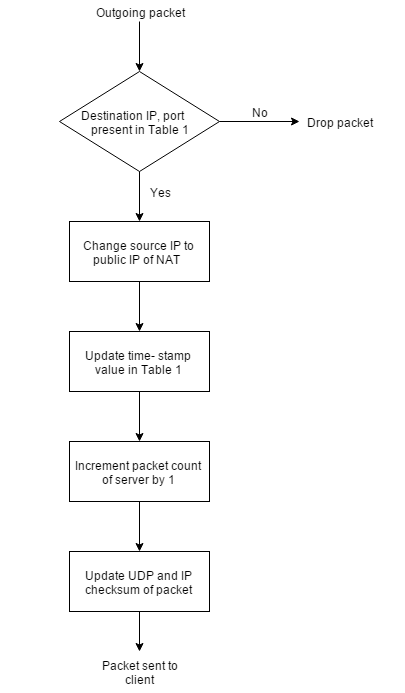


Figure 4

Source NAT functionality is placed in the POSTROUTING hook of the netfilter architecture as shown in Figure 2**(NF\_HOOKS)**. It modifies the source IP address of a packet from the server to the public IP address of the NAT in order to hide the IP addresses in the private network.

The steps taken by the SNAT for outgoing packets (packets from the servers to the clients) after routing is shown in Figure 4**(SNAT\_LR)**.

The source IP of the packet from the server is changed to the public IP address of the NAT. This hides the private network from the clients as the server IP addresses are not known to the clients.

Table 1 is searched with the key {Destination IP address, Destination port}. The packet is dropped if the key is not found. If the entry is found, the timestamp for the corresponding row in Table 1 is updated. The packet count of the server is incremented by 1.

Before the packet is sent out to the network, the UDP and IP checksum values are updated. This is required as the source IP address of the packet is changed by the SNAT function.

2 Load balancing NAT using round robin:

* Destination NAT:

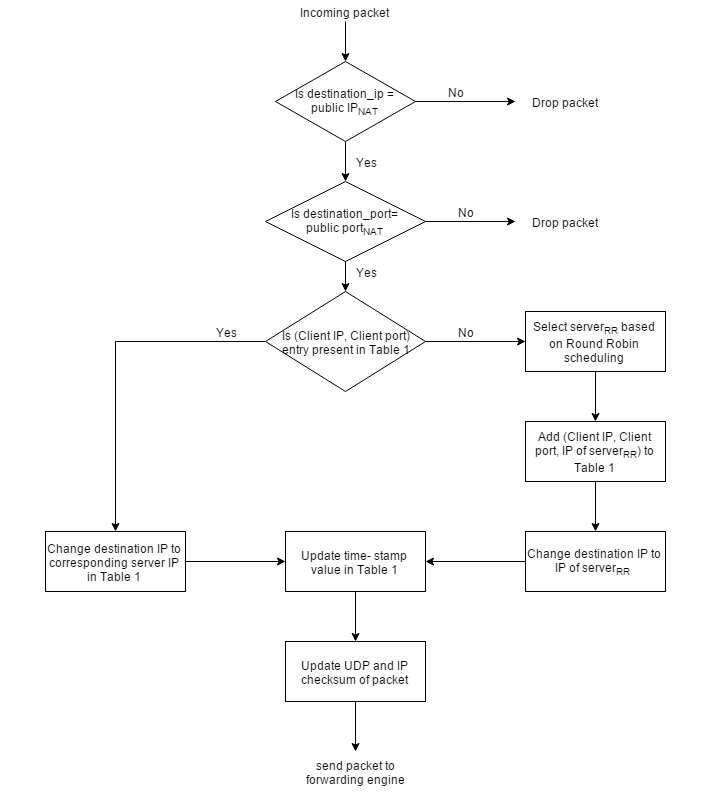


Figure 5

Destination NAT functionality is placed in the PREROUTING hook of the netfilter architecture as shown in Figure 2**(NF\_HOOKS)**. Destination NAT performs IP address translation similar to the DNAT in least rate as given in **section 1a** but the load balancing is based on round robin algorithm.

The steps taken by the DNAT for incoming packets (packets from the client) before routing is shown in Figure 5**(DNAT\_RR)**.

The destination IP address and port of packets is compared with the assigned public IP address and public port of the NAT and the packets are dropped if either or both of the comparison fail. If both match, the packet is destined to one of the servers.

The load balancing is accomplished by using Table 1. Table 1 consists of the mapping from {Client IP address, Client port} to {Server IP address}. {Client IP address, Client port} in Table 1 is searched by using {Source IP address, Source port} of the packet as the key. If the key is found, the destination IP address of the packet is changed to the corresponding Server IP address. If the key is not found, it corresponds to a new connection and an entry {Client IP address, Client port, Server IP address} is added to Table 1. The destination IP address of the packet is changed to the corresponding server IP address. The selection of Server IP address is based on round robin. For the first connection, Server 1 is chosen and for further connections, the chosen IP address is given by:

Server IP address = (Server IP address assigned to previous connection) modulo (Total number of servers).

The computation of timestamp, deletion of entries from Table 1 and the checksums in the packet headers are updated similar to DNAT for least rate.

1. Source NAT:

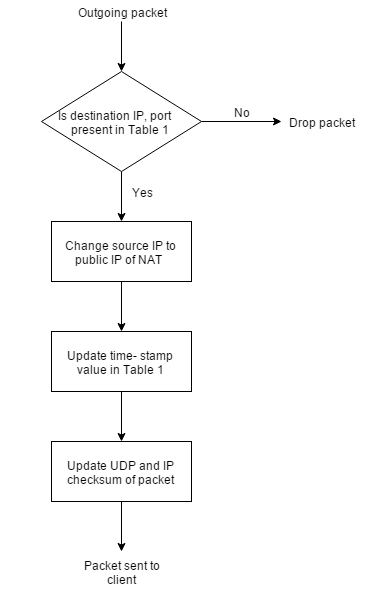


Figure 6

Source NAT functionality is placed in the POSTROUTING hook of the netfilter architecture as shown in Figure 2**(NF\_HOOKS)**. Source NAT performs IP address translation similar to the SNAT in least rate as given in section 1b.

The steps taken by the SNAT for outgoing packets (packets from the servers to the clients) after routing is shown in Figure 6**(SNAT\_RR)**.

Table 1 is searched with the key {Destination IP address, Destination port}. The packet is dropped if the key is not found. If the entry is found, the source IP of the packet from the server is changed to the public IP address of the NAT. The timestamp for the corresponding row in Table 1 is updated and the checksums are computed as given in section 1b.

* + (Other sections as before)
  + ~~Team composition~~
  + *Per-member responsibilities*
  + **Per-member learning objectives**
  + **Timeline**
  + **Descriptive Diagram of the system**
* Verification and validation plan(*Updates to V&V plan if any*
  + *Test application*
  + *Test plan*
  + **Demo playbook**
* **Self-study Plan**
  + **Description of base case**
  + **Characteristics to observe**
  + **Range of scenarios to investigate**