Implementation, Simulation of Linux Virtual Server in ns-2

CMPT 885 Special Topics:
High Performance Networks

Project Final Report

Yuzhuang Hu yhu1@cs.sfu.ca

ABSTRACT

LVS(Linux Virtual Server) provides a basic framework to build highly available and highly scalable web services using a large cluster of commodity servers. It adopts three techniques--NAT(network address translation), IP Tunneling, and Direct Routing to achieve this goal.

This project is to investigate performance issues of LVS(Linux virtual server) under ns-2. In this project, the author implements a framework of the virtual server in ns-2 with an modified NAT, IP Tunneling and Direct Routing. After the implementation and simulations of the virtual server adopting different techniques under ns-2, the author analyses the results and further uses a hierarchical structure to improve the virtual server’s performance. Simulation results show that this structure is effective.

The simulations in this project mainly focuses on comparing the performance of virtual server adopting the three techniques, and investigating what factor is the whole system's bottleneck. Several different scheduling algorithms are also compared in this project.

Table of Contents

List of Tables and Figures

Introduction

With the explosive growth of the internet, internet servers must cope with greater demands than ever. A single sever usually is not sufficient to handle the increasing load. Clusters of commodity work stations emerged as a viable solution to build highly scalable and highly available web services. The goal of a cluster is to make it possible to share a computing load over several systems without either the users or system administrators needing to know that more than one system is involved.

A Linux Virtual Server (LVS) cluster is a collection of servers that have been specially configured to provide highly-available services. The diagram below illustrates how an LVS cluster works.

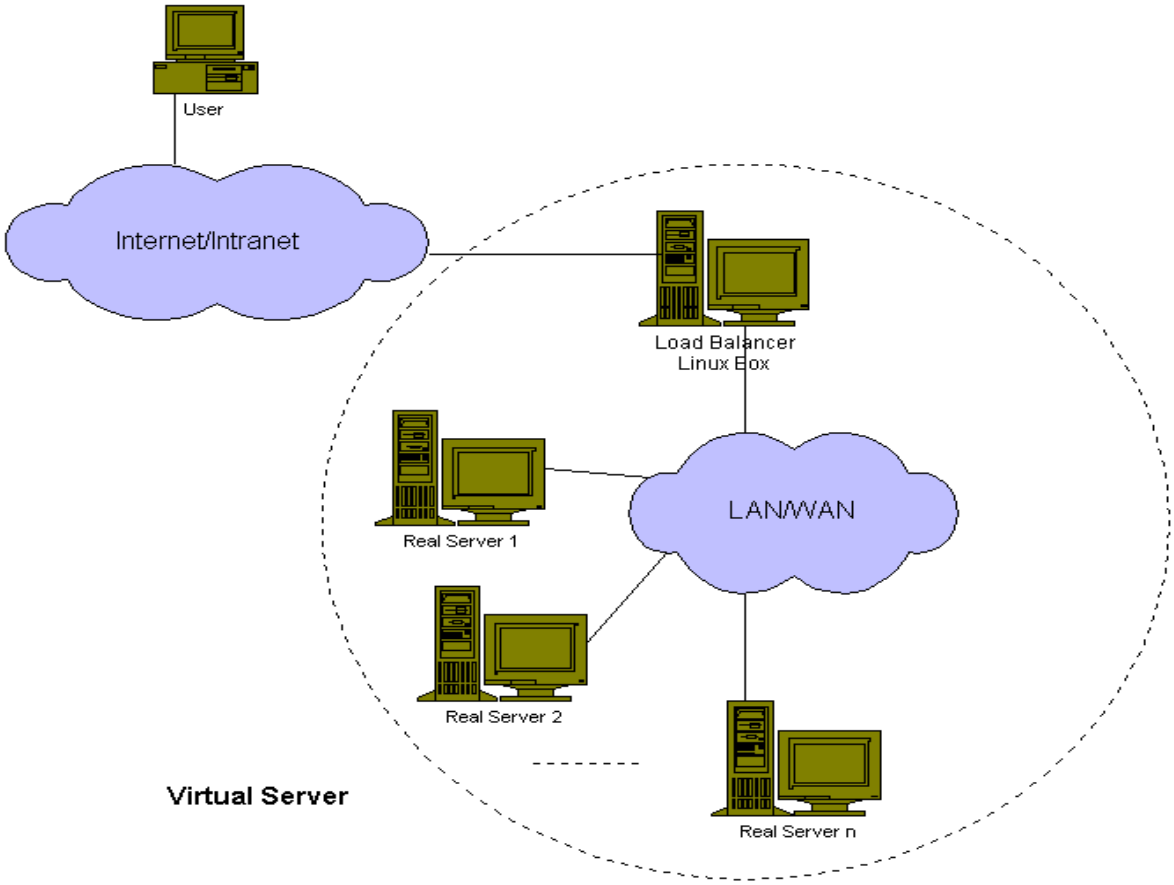

Figure 1 Structure of a Virtual Server

An LVS cluster consists of a load balancer and a variable number of application

servers. We will refer to the pool of application servers as *real servers*. Note that these terms designate *roles* rather than machines.

Service requests arriving at an LVS cluster are addressed to a *virtual server* IP address (sometimes referred to as a VIP address). This is a publicly-advertised address that an administrator at the site associates with a fully-qualified domain name (for example, www.sfu.ca). The role of the *active router* is to redirect service requests from a virtual server address to the real servers.

In fact, there are many ways to dispatch client requests to real severs. LVS, which uses three load balancing techniques---NAT(network address translation), IP Tunneling and Direct Routing is an approach in the IP layer. In LVS, these techniques are used by the load balancer to redirect IP packets to different real servers.

LVS via NAT

Figure 2 illustrates how LVS via NAT works:

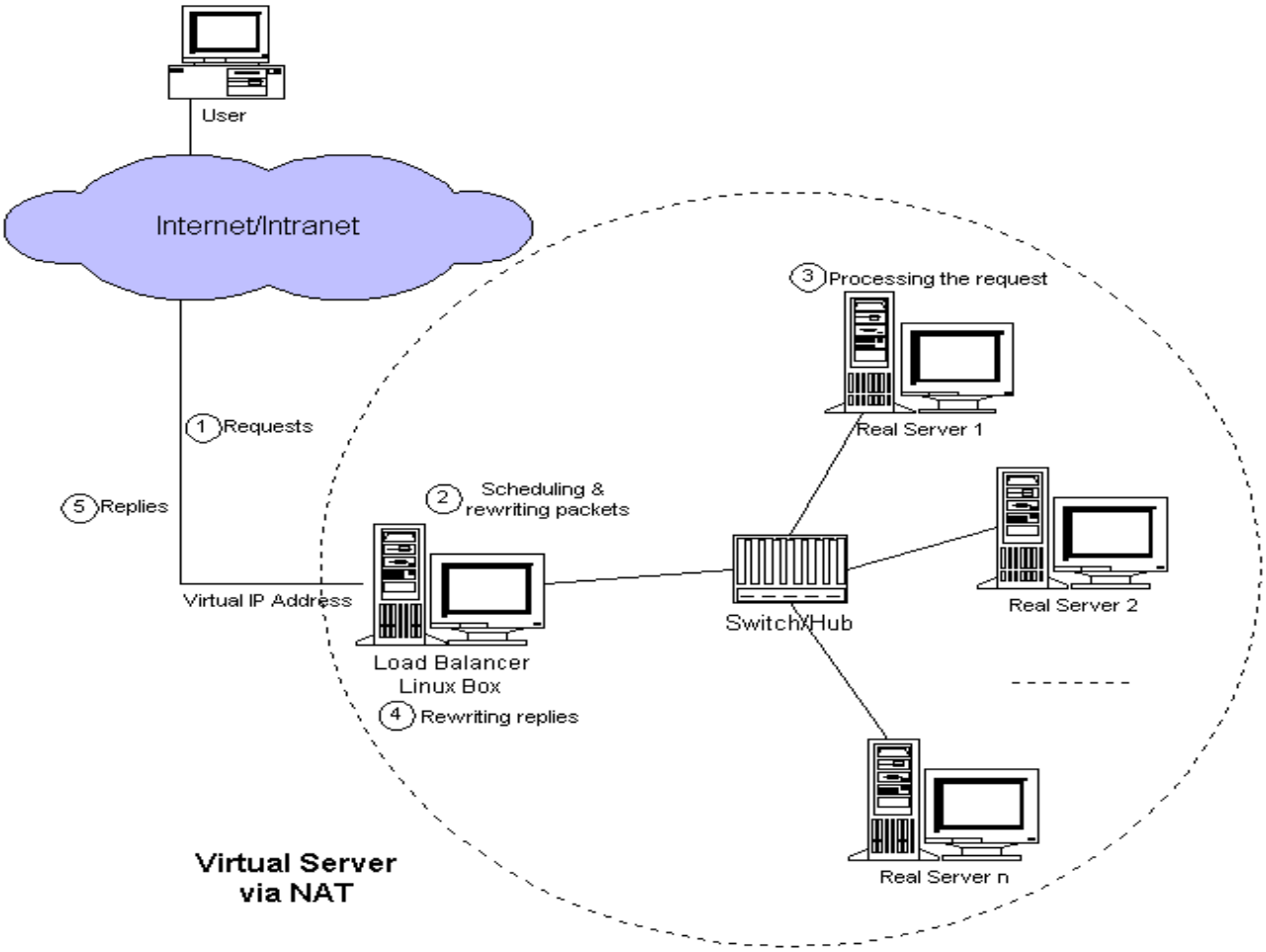

Figure 2 LVS via NAT

In LVS NAT, only the load balancer has the virtual server's IP address, the real severs can have private IP addresses. When a client sends a packet and the packet arrives at the load balancer(step 1), the load balancer chooses a real server, and rewrites the destination address of the packet to the real server's IP address(step 2), then forward the packet to the correct server. When the corresponding server processes the request and replies(step 3), the load balancer rewrite the destination address of the reply packet to the IP address of the virtual server, and then returns the packet to the client(step 4).

The advantage of this method is that the real server can run any operating system supporting TCP/IP protocol, and only one IP address is needed by the whole system. The disadvantage is that since the destination address of packets must be rewritten

twice in the load balancer, the scalability of this method is limited.

LVS via IP Tunneling

Figure 3 illustrates how LVS via IP Tunneling works:

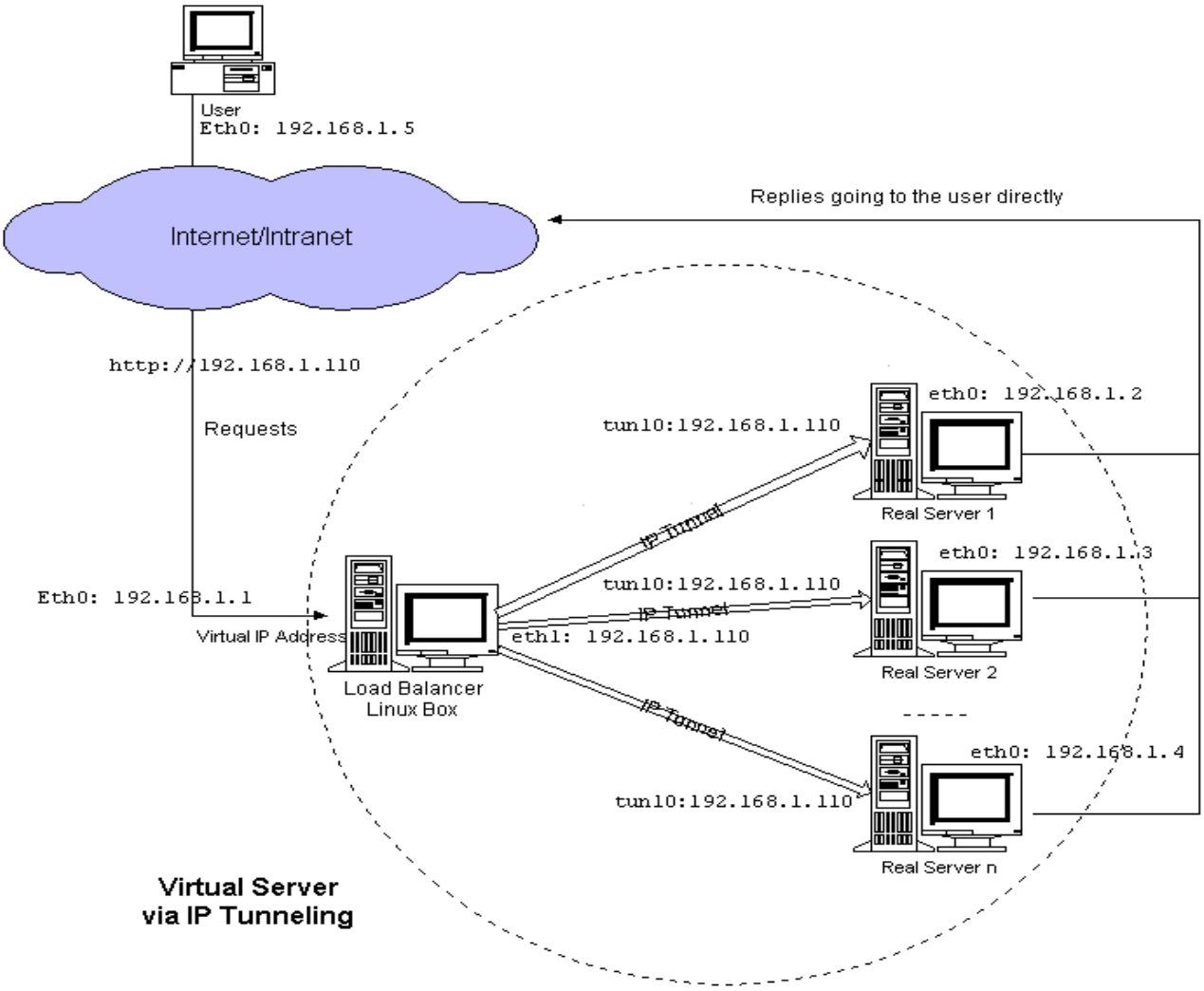

Figure 3 LVS via IP Tunneling

In figure 3, the virtual IP address is 192.163.1.1, and this address is configured on the tunnel interfaces of the corresponding real servers. The scalability of LVS via IP Tunneling is better than LVS via NAT, for when using IP Tunneling, packets only need to be rewritten once. When a packet arrives at the load balancer, the load balancer encapsulates the packet, then tunnels the packet to a chosen real server.

After receiving the packet, the real server decapsulates it and gets the original packet. Then the real server will reply directly to the client, needs not send the reply packet back to the load balancer as in LVS via NAT.

The disadvantage of this method is that it needs IP Tunneling support both in the load balancer and the real servers. Since the IP tunneling protocol is becoming a standard for all operating systems, LVS/TUN should be applicable to other operating systems.

LVS via Direct Routing

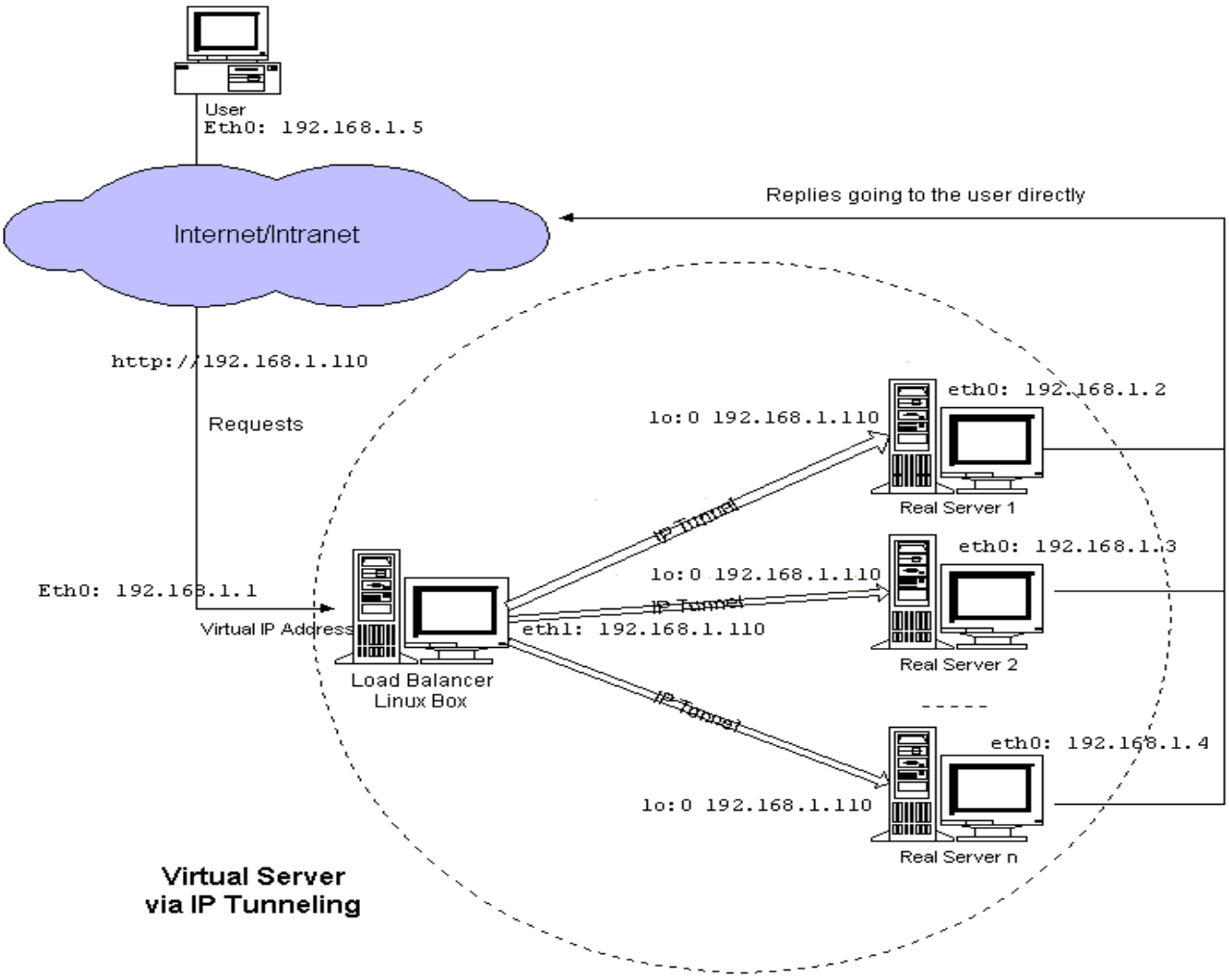

Figure 4 LVS via Direct Routing

Under this method, the load balancer and the real servers must be in the same

physical segment of a local-area network. The load balancer also processes only the client to load balancer side traffic. When the load balancer receives a packet from the client, it chooses a real server and changes the MAC address of the packet to that of the real server, then forwards the packet to the real sever. The real server has a virtual IP address configured in its loop back alias interface, and this interface must not do ARP response. In this way, the real server can generate a packet whose source address is the virtual IP address, and send the packet directly to the client.

The advantage of this method is that it does not have the IP tunneling overhead. The disadvantage is that the real severs and the load balancer must be in the same uninterruptible segment of a local-area network, and the real servers must have a loop back alias interface which does not respond to ARP.

Below is a summary of the advantages and disadvantages of these three techniques:

LVS type	NAT	Tunneling	Direct Routing
OS	any	must support IP tunneling	any
Server config	none	tunnel interface, no ARP	Loopback interface, no ARP
Server network	private	internet	local
Scalability	not so good	better than NAT	better than NAT

Table 1 Comparison of LVS via NAT, IP Tunneling and Direct Routing

Related work

LVS is a server side IP level approach of dispatching client requests to different servers. Besides this approach, existing dispatching techniques can be classified into following categories:

Client side approach

Berkley's Smart Client [] is an example of such an approach. It modifies client side applications, and provides an applet to make requests to real severs in the cluster, and collect load balancing information to choose the right server. The disadvantage of this approach is that it needs to modify client side applications, and increases network traffic by extra probing.

DNS redirection approach

Under this approach, the DNS server maps a single name to different IP addresses(which are possessed by real servers) in a round-robin manner. So each time when different clients make requests, they will get different IP addresses for the same name and load will be distributed among the servers.

However, due to the caching nature of clients and hierarchical DNS system, this approach can easily leads to dynamic load imbalancing among the severs. For the name mapping, there's a corresponding TTL(time to live) value. During this period, all the requests of the same client will be sent to the same server. The scheduling granularity is per host. Moreover, if the chosen server fails, during TTL, the result of client requests will remain “sever is down”, in spite of there exists other severs can serve these requests in the cluster.

Server side application level approach

The system's structure is similar to that of IP level server side approach. The difference is that in application level approach, the load balancer forwards http requests to real severs, and after getting the results from the real servers, it returns them to the clients. The disadvantage of this approach is that it needs two connections,

one exists between the clients and the load balancer, the other one exists between the load balancer and the real severs. In LVS, only one connection exists between the real servers and the clients. The cost of maintaining two connections is high, and the load balancer will become the whole system's bottleneck.

Implementation

Modifications to LVS

As mentioned above, LVS via NAT is not scalable well. In this project, a modified NAT has been implemented. The modified NAT works the same on the client to load balancer side processing. What's different is that, packets will not go to the load balancer again when the real server replies. Instead, the real server will set the correct destination IP address and port(which are the client's) itself, and forward the packets to the client directly.

The advantage of making this change is that, since a packet will only arrive at the load balancer once, the scalability will be better. And since the modified NAT only changes the destination IP address of the packet, we can think that its overhead is low and its performance is comparable to the other two techniques. In the later simulation, we will give the corresponding results.

The modified NAT can also use private IP addresses. By making modifications to ns-2, now agents can set a packet's source address to the virtual IP address. Thus in spite of the kind of address the real server uses, the packet can have the correct source IP address.

The disadvantage of the modified NAT is that, we need to change the codes on the real server side. Under LVS via NAT, the real server can run any operating system.

But since the modified NAT can improve the virtual server's performance and scalability(as our simulation results indicate), and it can also use private IP addresses, in some circumstances, it's worthwhile to make such a change. Let us suppose that when we try to construct a web server cluster, the real servers will all use Linux, and it lacks public IP addresses, to provide powerful web services, a virtual server using the modified NAT can be a very good choice.

Modifications in ns-2

Accept a packet destined to the VIP

In order to implement virtual server via tunneling and virtual server via direct routing, real severs should have two IP address. One is its real IP address, and the other one is the virtual server IP address. The real sever should accept packets whose destination address is the IP address. In Linux, this is achieved by adding an alias interface which is bound with the virtual IP address. How can we do this in ns? Below is a node's structure in ns-2:

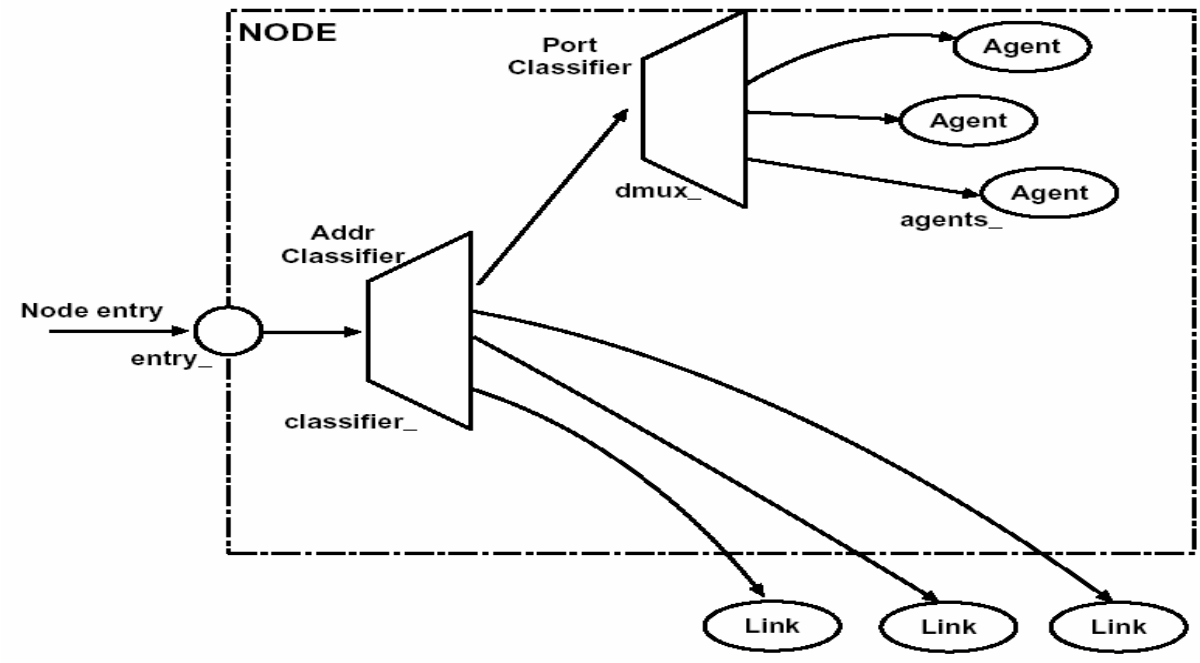

Figure 5 Node structure in ns-2

We can see a packet will arrive at the correct agent only after it can reach the classifier DeMux which is used to choose the correct agent according to the packet's destination port. And the node will forward the packet to other links or to the classifier Demux by looking its destination address in the node entry(which is a destination hash classifier). So this problem can be addressed by simply add a route whose destination address is the virtual server ip address, and the target is the node's classifier DeMux. In this way, packets destined for the VIP will be forwarded to the node. That is to say, the node will accept packets destined for the VIP.

Send a packet destined to the client

Virtual sever via tunneling and direct routing need to directly send packets whose source address is the VIP(different from its own IP address) to clients. In Linux, a SKB buffer is transferred up each layer and down each layer. In relevant procedures, SKB is one of their parameters. So when sending packets back to clients, Linux knows the correct source and destination IP address. Unfortunately, procedures in ns2 do not have such kind of a parameter. In ns2, a packet's source address and destination address are initialized in agents. There's a structure here_ in the class agent records the node's IP address. Whenever a new packet is initialized, the packet's source IP address is set to here_.addr_. That is to say, a node in ns2 can only have one IP address.

To address this problem, a virtual IP address is added to the class agent. When we create an agent, we can set the virtual IP address, and a route for this VIP will be added automatically to the classifier DeMux. Now when sending a packet, the source address will be initialized to the virtual IP address. By default, the VIP is the same as the agent's real IP address.

Modify a packet's MAC address

Normally MAC address is set by the link layer, but in order to implement LVS via Direct Routing, we need to set a packet's MAC address in the agent IPVS, which is in the IP layer. Linux uses a separate data structure passing routing information to the lower layer to set the MAC address, but in ns-2, agent can only pass the packet as the parameter to the lower layer. To simulate direct routing, we need to modify the link layer and the hash classifier class in ns-2, add extra functions using the destination address as one of its parameter. With these functions, now we can find the LL object by looking up the hash classifier, and then use the LL object to set the packet's MAC address.

Agent IPVS

Changes of the ns code in the real server side are only those mentioned above. The implementation of this project focuses on the load balancer. In the implementation, there's an agent(IPVS) in the load balancer, which is responsible to receive clients' requests and redirect them to the real servers. In the IPVS agent, we implement a hash table holding information about connections on the real servers, and a timer related with each connection entry in the connection hash table. We also implement three load balancing techniques----a modified NAT, IP Tunneling and Direct Routing, together with four scheduling algorithms----round robin, weighted round robin, least connection and weighted least connection. Below is the detailed description.

Connection hash table

The IPVS agent maintains a connection hash table for all the connections established on the real severs. We need a connection hash table, because we can get information useful for the purpose of load balancing, and more directly, for TCP connections, we need forward following packets of the same connection to the correct real server. Although this version of IPVS agent only supports UDP, the whole framework has been implemented. In fact, this version can be also used to redirect TCP packets. But

for finer control, we need to add different timeout values to different TCP states. And for ftp protocol, since it uses two connections at the same time, the pattern of one connection controls another connection needs to be included.

The connection hash table is an array of list heads pointing to connections with the same hash value. The hash is calculated using parameters of IP address, port and protocol. We use the JHASH function which is in the Linux kernel to calculate the hash key.

The connections maintained in IPVS agent are very different from TCP connections. The main purpose of this connection is to maintain a timeout value, after timeout, the connection expires and when the same client send a packet to the same port to the load balancer, the load balancer will create a new connection entry and may choose a different real server.

It is weird that the timer in ns-2 does not work when trying to use it to implement the timer for the connections. Instead we implement our own timer. First the connection class is defined as a subclass of class Event, then a timer class is defined by using the scheduler in ns-2. The timer class includes methods such as sched and resched which is implemented by using the scheduler. A private variable of event_ exists in the timer class. Each time when a new connection is created, event_ of its timer is set to the connection itself. In this way, when the timer expires, the scheduler can find the correct connection and execute the expire operation of the connection class.

Below is the main structure of the connection class:

```
class ip_vs_conn : public Event {
public:
        ip_vs_conn();
        struct list_head        *c_list;            /* hashed list heads */
```

```
    __u32                   caddr;          /* client address */
    __u32                   vaddr;          /* virtual address */
    __u32                   daddr;          /* destination address */
    __u16                   cport;
    __u16                   vport;
    __u16                   protocol;       /* Which protocol*/
    unsigned long           timeout;            /* timeout */
    struct ip_vs_timeout_table      *timeout_table;
    __u16                   flags;          /* status flags */
    __u16                   state;          /* state info */
    ip_vs_dest              *dest;          /* real server */
    int                     in_pkts;        /* incoming packet counter */
    virtual void expire();
    ip_vs_timer             *conn_timer;        /* expiration timer */
};
```

Real servers

For our virtual service, there's a class ip_vs_service describes it. This class mainly has the information of the virtual IP address, the port, statistics for the service, data needed by the scheduling algorithm, and a list of real servers. The real server class in our implementation is ip_vs_dest, as we have seen in the connection class. All of the real servers are linked together, a real server is added to the real server list when it is first registered.

The real server data structure mainly holds a list of agents, the server's IP address, the server's port, a weight, and connection numbers. The weight and connection numbers are used in the scheduling algorithms.

The list of agents is for the virtual server via the modified NAT, IP Tunneling and

Direct Routing. The reason we need a list of agents is that under these methods, the packet is transmitted directly to the client. It needs to set the destination IP address and port of the packet to those of the client. In ns-2, these fields are initialized by the agent. Each agent has a dst structure which records its destination IP address and port. In this version of n2-2, normally the destination address and port are statically set before performing simulation. Thus on each real server, we need a list of agents, each agent corresponds to one client and records the client's IP address and agent port in its dst structure.

Packet transmitting methods

As mentioned above, the modified NAT, IP Tunneling and Direct Routing have been implemented in this project. All methods need to search the agent list and get correct destination port. The modified NAT rewrites the packet's destination IP address and agent port, and recalculates both the IP header checksum and TCP/UDP pseudo header checksum(using a fast incremental update method, since only the IP address and port are modified). Virtual server via IP Tunneling needs to allocate a new packet, set the port of this packet to the Decapsulator agent on the corresponding real server, calculate the IP header checksum, and then encapsulate the original packet. Virtual server via direct routing needs to get the correct link layer object through lookup the node's destination hash. After getting the MAC address of the chosen server, this method sets the packet's MAC address and send the packet out. It also updates the IP header checksum.

Notice that the implementation of the HDR_ENCAP is not correct in ns-2. It may cause segmentation fault under some circumstances. For tunneling, it's better to use the Encapsulator and Decapsulator implemented in the mobile IP in ns-2.

Scheduling algorithms

We have implemented four scheduling algorithms for selecting a server from the cluster: Round-Robin, Weighted Round-Robin, Least Connection, and weighted Least Connection. The first two are self-explanatory, they don't use any load information to make the decision. For least connection scheduling algorithms, it needs to record connection number on each real server.

The Round-Robin scheduling algorithm directs the network connections to different servers in a Round-Robin manner. It treats all real servers equally, regardless of their different response time. The scheduling granularity of virtual server is connection-based, and is superior to the Round-Robin DNS approach.

The Weighted Round-Robin scheduling algorithm, however, gives each real server a weight. The weight is an integer, indicates the server's capability. During scheduling, a scheduling sequence is generated according to the server weights.

The least connection scheduling algorithm is a dynamic load balancing algorithm, it keeps the number of connections on each server, and chooses a server which has the least number of connections. The weighted least connection scheduling algorithm further assigns a weight to each server, and chooses a server according to both its weight and connection number.

Simulation

Topology of simulation scenarios

Below is the basic topology of all simulation scenarios in this project:

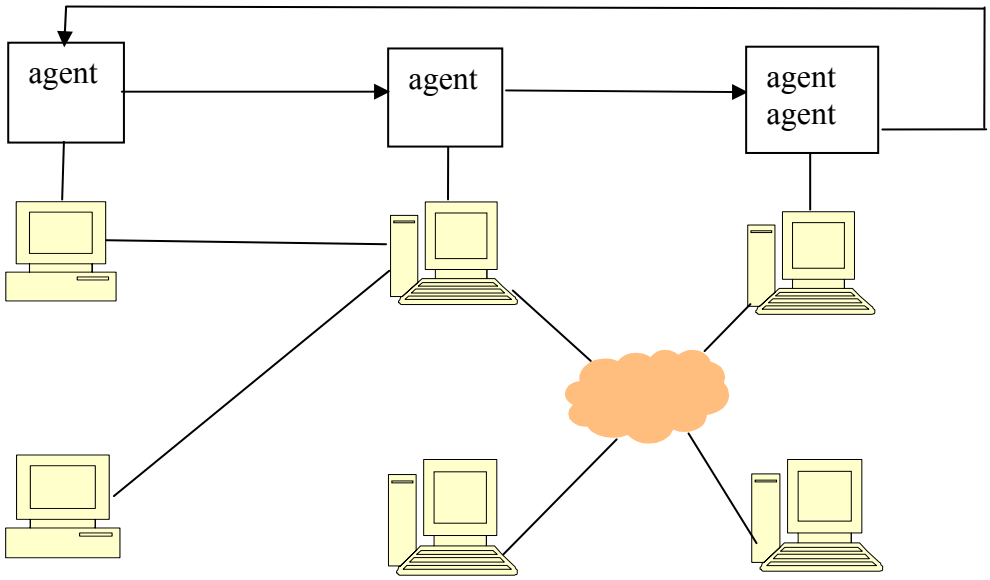

Figure 6 Basic topology of simulation scenarios

The left side represents client machines, the right four machines represents load balancer and servers connected by a local area network. Each client node has one UDP agent and a CBR traffic generator, the load balancer has one IPVS agent, and the real server has a list of agents which point to the clients. In virtual server via IP Tunneling, there's a Decapsulator agent on each server node, which is responsible for Decapsulating the packet and re-sending the Decapsulated packet to the correct agent on the same node. Notice that above is the basic structure of the simulation topology, the number of client and real server nodes, and other parameters vary in different simulations.

Stressing the load balancer

Since each client request will first be processed by the load balancer, the design of the load balancer is crucial to the virtual server's performance. How about the LVS load balancer? Simulations below will stress the load balancer, and investigate whether the load balancer via the modified NAT, IP Tunneling and Direct Routing will become the bottleneck of the whole system. Meanwhile performances of virtual

server via different techniques will be compared and analyzed.

First Topology

In this topology, the load balancer also is the default gateway to connect clients and servers. Figure shows the structure of the topology, node 0 is the load balancer and also the default gateway. Node 1, 2 are real servers, and node 3 is the VLAN node in ns-2. The IPVS sink agent on the servers accepts packets and send back the replying packets immediately. Node 4 to 13 are client nodes, these clients send request to the virtual server at an interval of 1ms, it is to say 1000 packets per client arrives at the load balancer per second. The load balancer uses Round-Robin scheduling algorithm. Other parameters are: the bandwidth of link between the clients and load balancer is 10M, delay is 1ms. The local area network bandwidth is 100M, and delay is 1ms. The queue size of all links is 100. And in all simulations, the packet only includes necessary packet headers to reduce the memory consumption. These packet headers are IP, UDP, RTP, IPVS, LL, MAC, Flags, ARP and IPinIP(for tunneling).

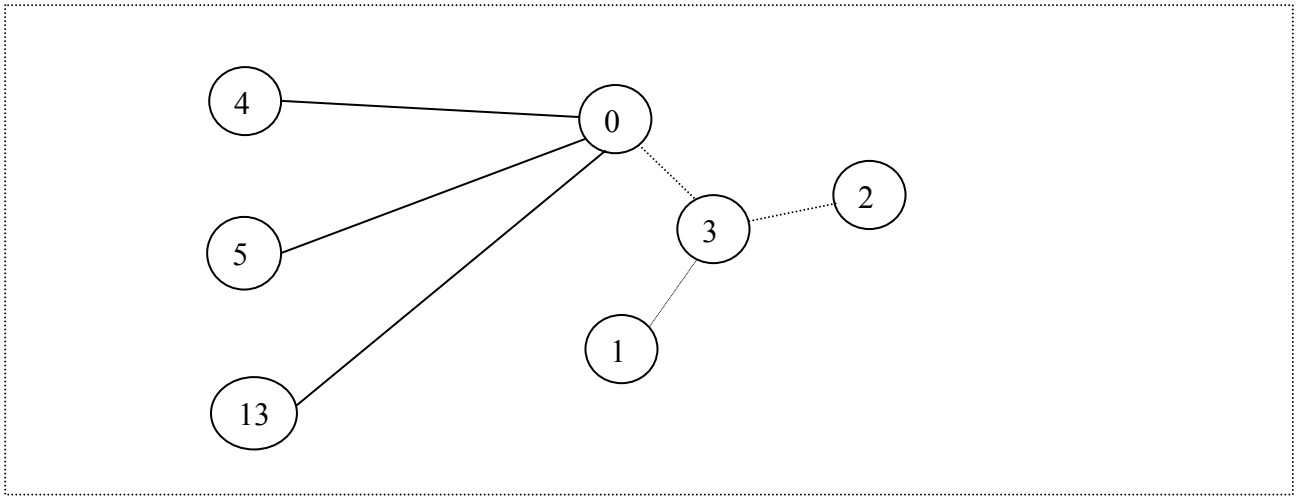

Figure 7 structure of topology 1

We stresses the load balancer by setting the sending interval of CBR to 1ms, and gradually increasing the number of requests by adding more client nodes to the topology. The reason to do so is that, since each client node is connected to the load balancer by a separate link, the load balancer can have enough capability to

communicate with the clients. If instead we use a smaller CBR sending interval and smaller number of client nodes, the packets may be dropped before they are processed by the IPVS agent. Simulations below just ensure that each packet sending by the client nodes will reach the IPVS agent, thus stresses the processing ability of the load balancer.

Metrics used in the simulations are throughput(replying rate), receiving rate and loss rate. Throughput here means the number of packets replied by the load balancer per second to the clients. Receiving rate means the number of packets received by the real severs per second, and loss rate means the number of packets dropped by the load balancer and the real servers every second. The round trip time is not an accurate reflex of the virtual server's processing capability, for the number of requests fully processed by the virtual server with different load balancing techniques may vary.

First we choose only one real sever. Below is the result:

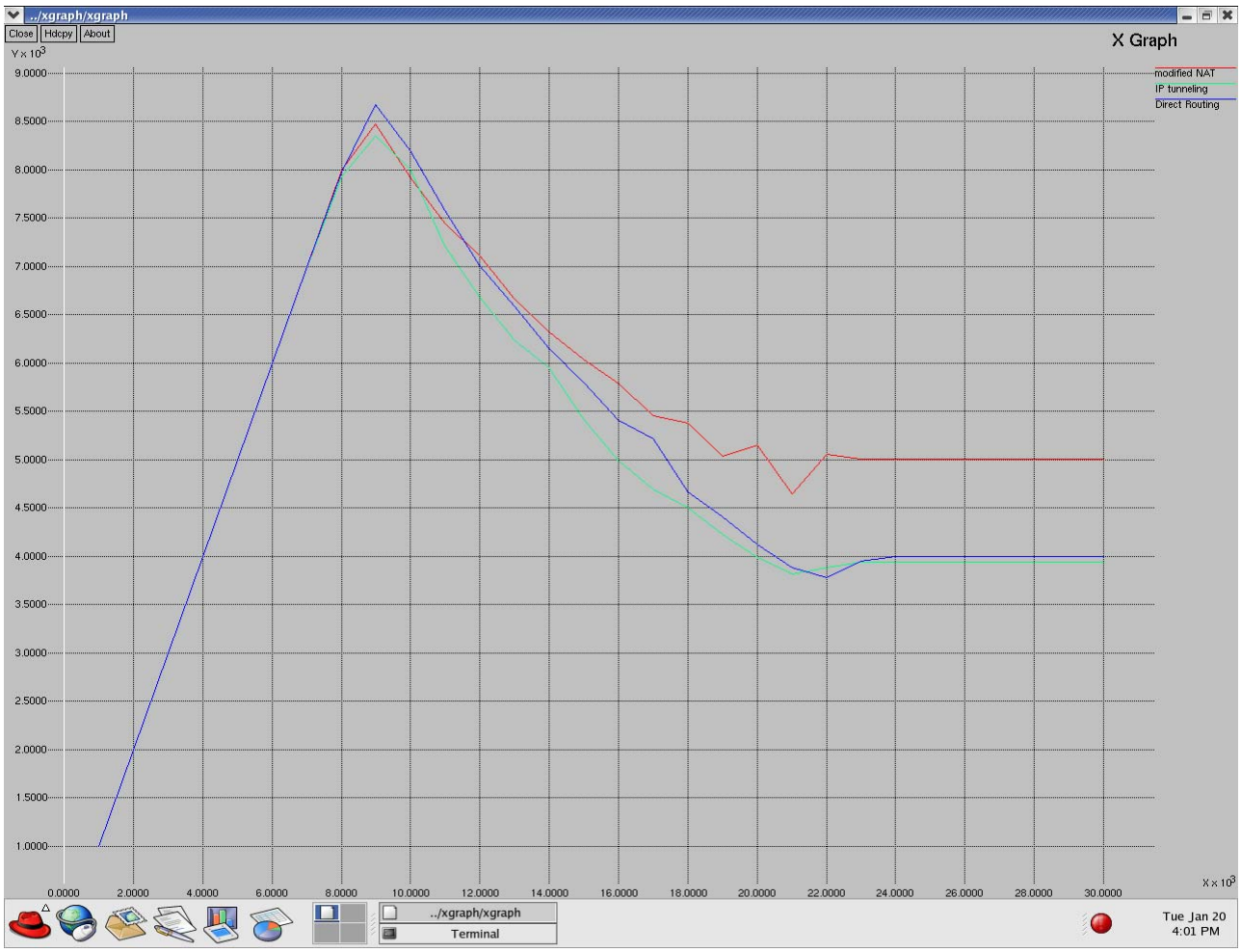

Figure 8 throughput, topology 1, 100M LAN, 1 server

Figure 8 is the throughput of the virtual server via three techniques. The horizontal axis is the connection rate on the load balancer, the vertical axis is the throughput per second. We can see from the figure above that when the connection rate is 9000/s, the throughput of the virtual server under all three techniques reaches its peak, and gradually goes down with the increasing of the connection rate. When the connection rate is larger than 22,000/s, the throughput is rather steady, for the modified NAT, it's only 5,000/s, for the other two techniques, it's only 4,000/s.

Figures below is helpful to see what happens:

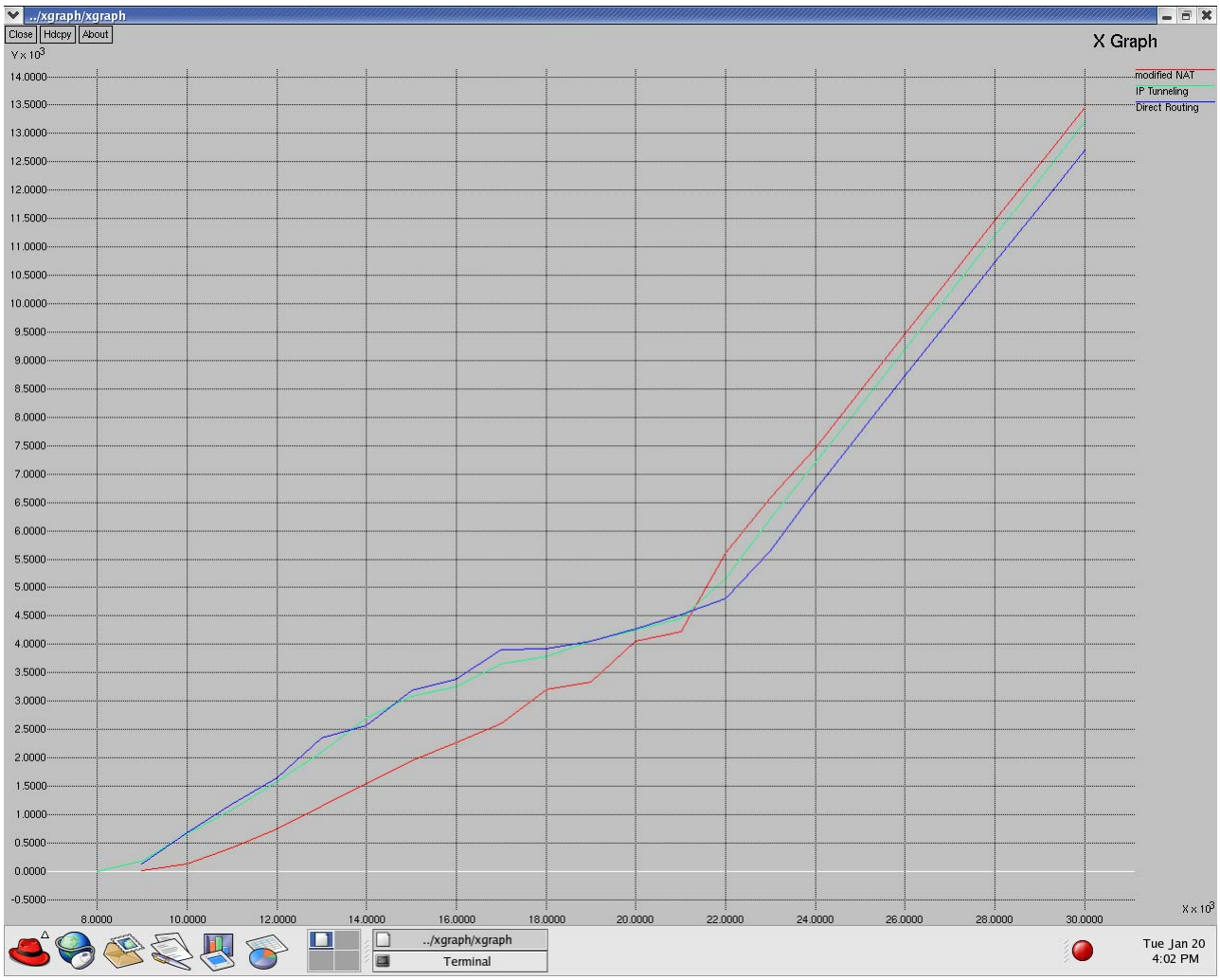

Figure 9 loss rate on the load balancer, topology 1, 100M LAN, 1 server

Figure 9 shows the loss rate on the load balancer, the horizontal axis represents the connection rate, the vertical axis represents the loss rate. From this figure, we know that when the connection rate is lower than 6,000/s, no packet loss occurs. The loss rate increases drastically, especially when the connection rate is higher than 22,000/s. This greatly affects the number of packets received on the real server. Figure 10 shows the receiving rate on the server:

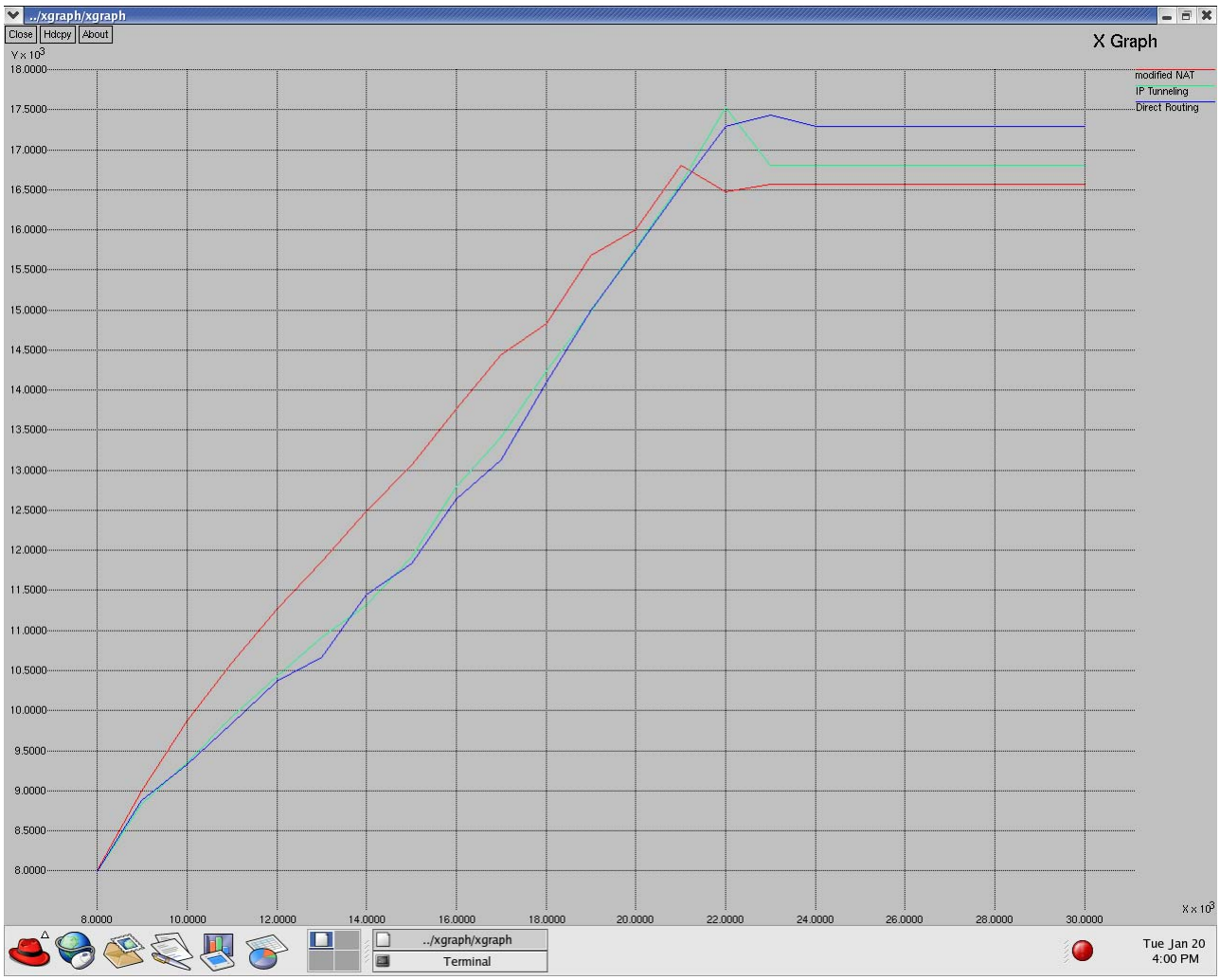

Figure 10 receiving rate on the real server, topology 1, 100M LAN, 1 server

In figure 10, the real server can not receive more packets when the connection rate is higher than 22,000/s, this is mainly caused by the sharply increasing loss rate on the load balancer.

Besides the packet loss on the load balancer, there's still a gap between the throughput and the connection rate. For example, when the connection rate is 22,000/s, the throughput of virtual server via the modified NAT is only about 5,000/s. And the load balancer only drops about 5,000 packets, the gap is about 17,000/s. In a 100M LAN, most of these packets are lost on the real server, very few of them are lost due to collision. Following figure describes the loss rate on the real server in this simulation.

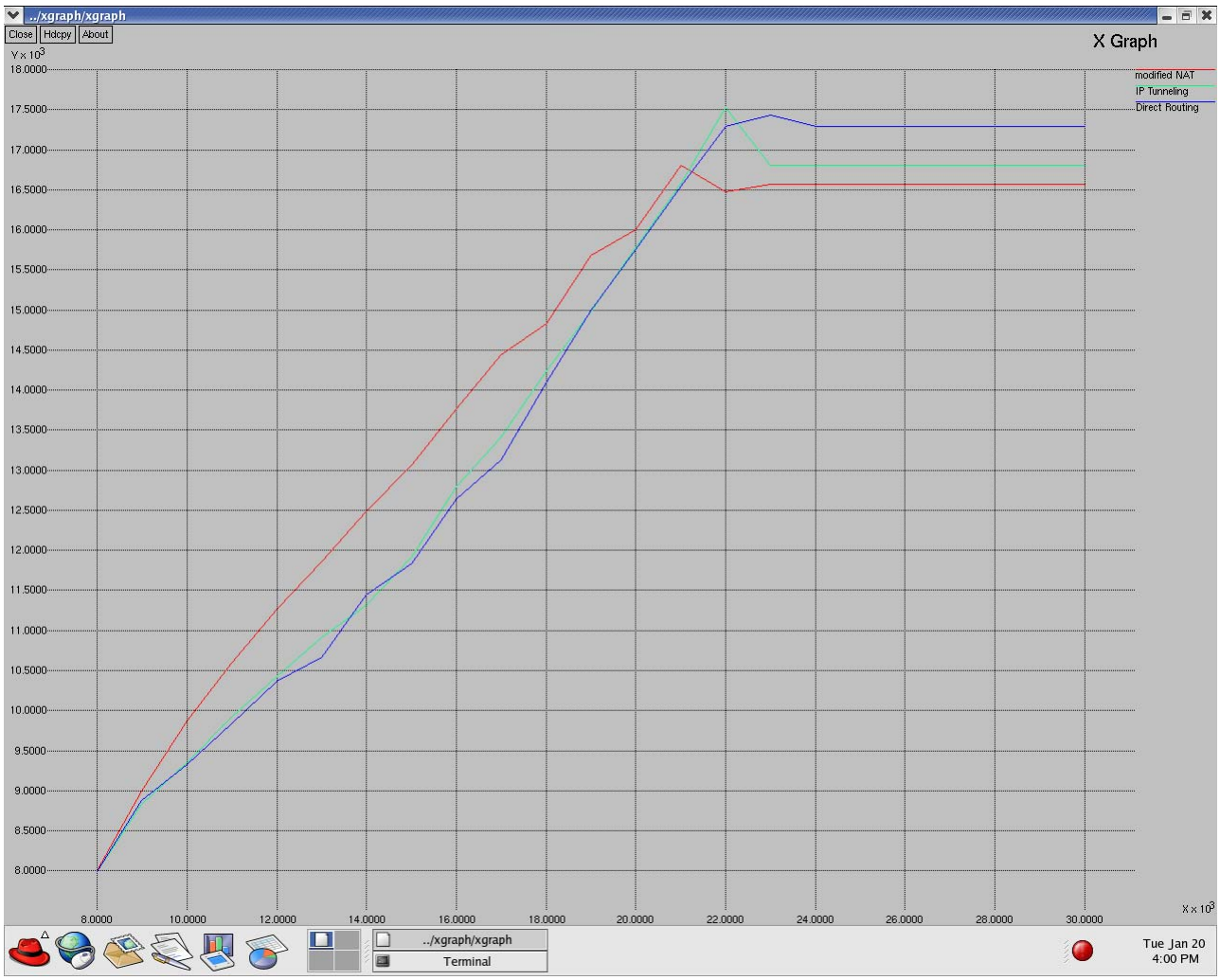

Figure 11 loss rate on the real server, topology 1, 100M LAN, 1 server

We can see from figure 11 that the packet loss on the real server also increases drastically, but after the connection rate exceeds 22,000/s, the loss rate is steady. This is because of the real sever can not receive more packets, the extra packets are dropped on the load balancer(in figure 9).

So the packet loss on the load balancer, the real server and caused by collision is the main factor affects the throughput. This illustrates that a 100M LAN is not capable of dealing with the increasing traffic. It's clearer when adding more real servers to the virtual server. Below are the throughputs of the virtual server with 1, 4, 8, and 12 real servers:

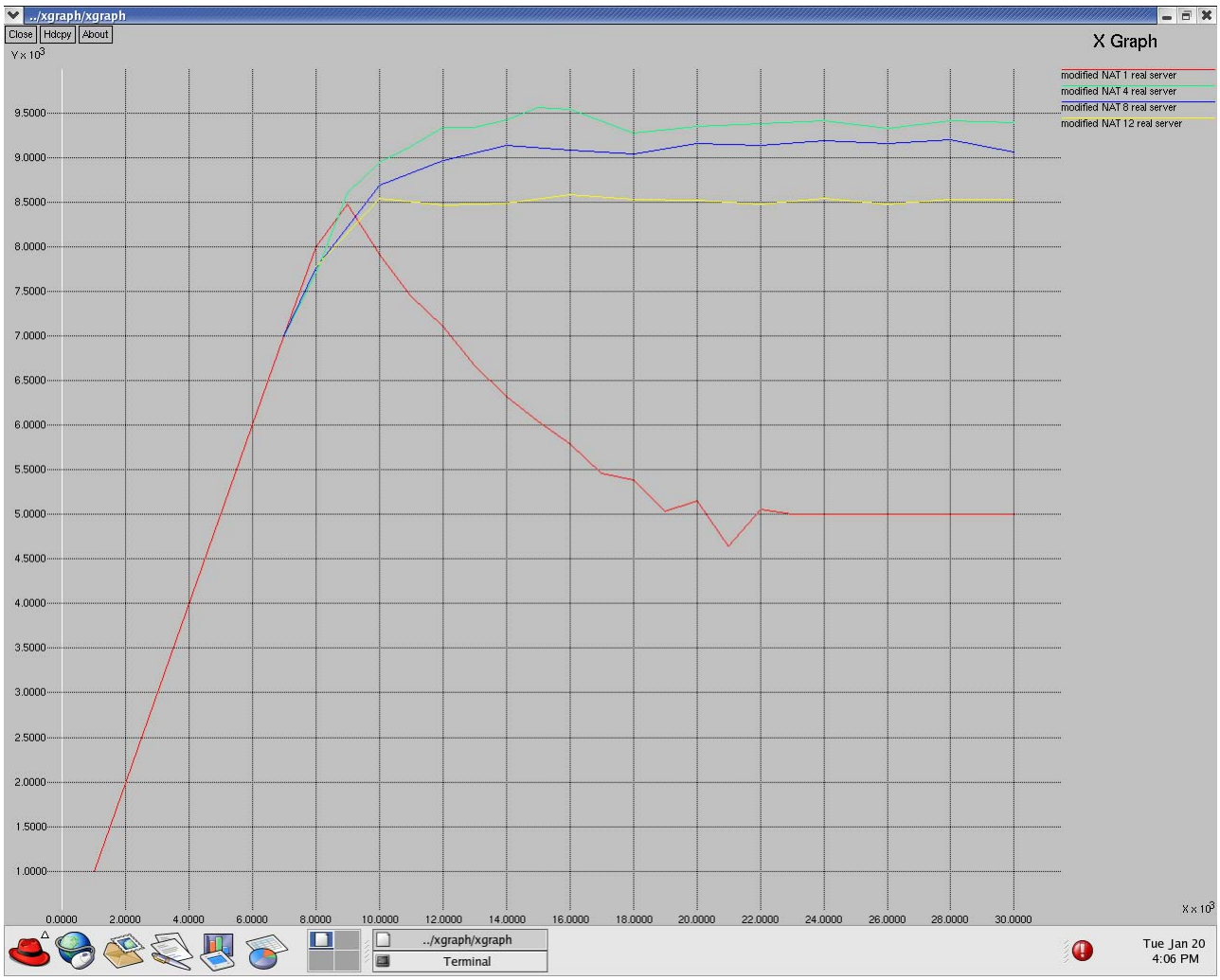

Figure 12 throughput of the VS via the modified NAT, topology 1, 100M LAN

We can see from the figure that the throughputs of the virtual server having 4, 8 and 12 real servers are higher than that of having 1 real server. But the throughput of having 8 servers is lower than that of having 4 servers, and the throughput of having 12 servers is further lower than 8 servers. Below gives the loss rate on the load balancer and real servers to have a further look on this phenomenon:

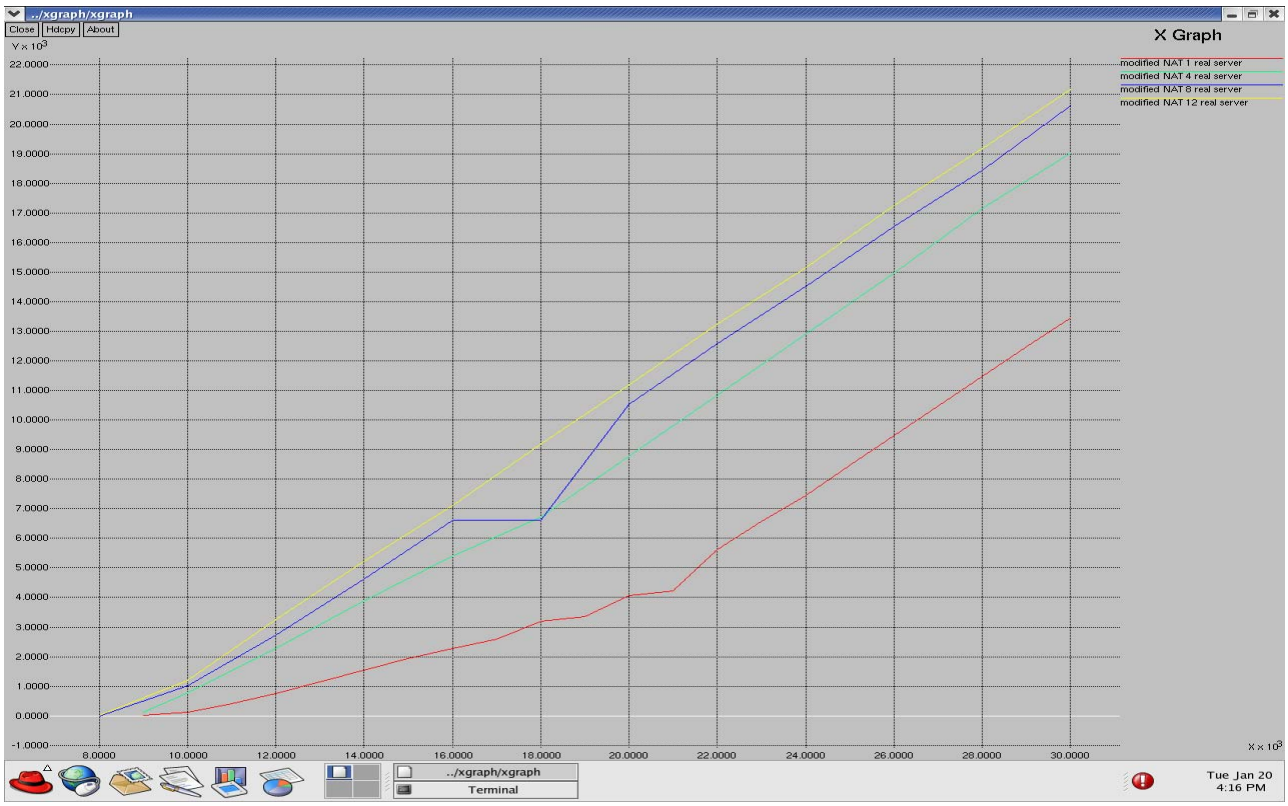

Figure 13 loss rate on the load balancer, the modified NAT, topology 1, 100M LAN

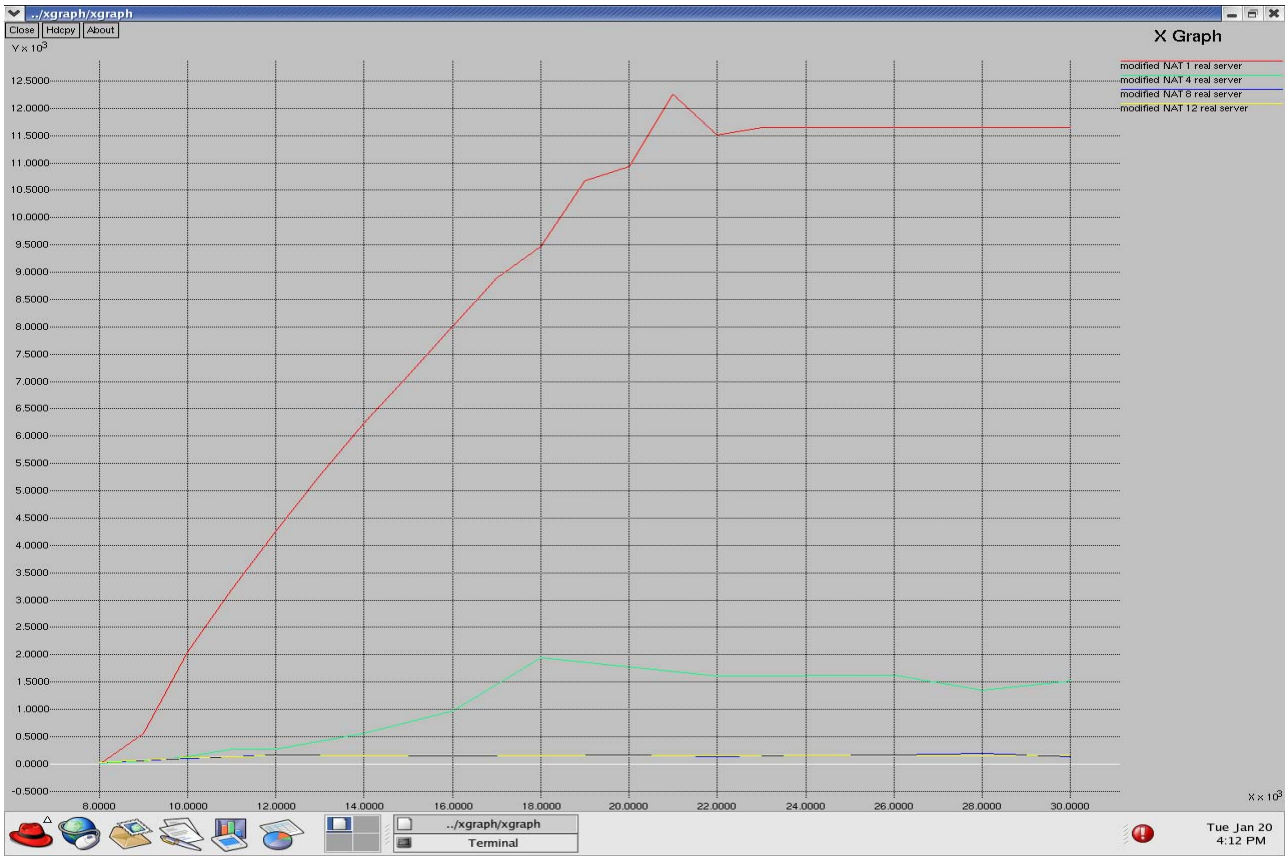

Figure 14 loss rate on the real servers, the modified NAT, topology 1, 100M LAN

We can see from figure 14 that packet loss on the real servers decreases very sharply when adding more real servers. Very few packets are dropped in the virtual server with 8 and 12 real servers. Most of the un-replied packets are lost on the load balancer. For example, when the connection rate is 30,000/s, the replying rate of the virtual server with 12 servers is about 8,500/s, there're 21,500 un-replied packets per second. Only about 160 packets are dropped on the real server per second, meanwhile the loss rate on the load balancer is about 21,100/s. The high packet loss rate on the load balancer decides the low throughput of the virtual server.

The load balancer seems to be the bottleneck of the virtual server. Is this caused by the implementation of the IPVS agent? Through further analysis, we can see that this bottleneck is due to the local area network transmitting capability, not the implementation of IPVS agent. Since it is guaranteed that no packet will be dropped on the links between the load balancer and the clients, if the transmitting speed of the local area network can catch up with that of the IPVS agent and therefore is capable of transmitting packets modified by the IPVS agent in time, no packet will be lost on the load balancer. Instead the throughput of the load balancer will be a rather constant number which is decided by the corresponding processing speed of the virtual server using different transmitting techniques. However, as above figure shows, packet loss rate on the load balancer is high, and this loss severely affects the throughput of the virtual server. The reason of the throughput of the virtual server via the modified NAT decreases when having more real servers is that, the local area network is busier due to its broadcasting nature, and this causes the higher loss rate on the load balancer. Under above simulation configurations, the local area network becomes the system's bottleneck.

The lower loss rate on the real servers when having more real servers is due to the traffic are distributed among the real servers, each real server receives less number of packets if having more real servers. And under the same connection rate, more packets are dropped on the load balancer as the number of real servers increases, the

real servers also receives less number of packets as a whole.

The situation when using the other two transmitting methods are almost the same. Below are their results:

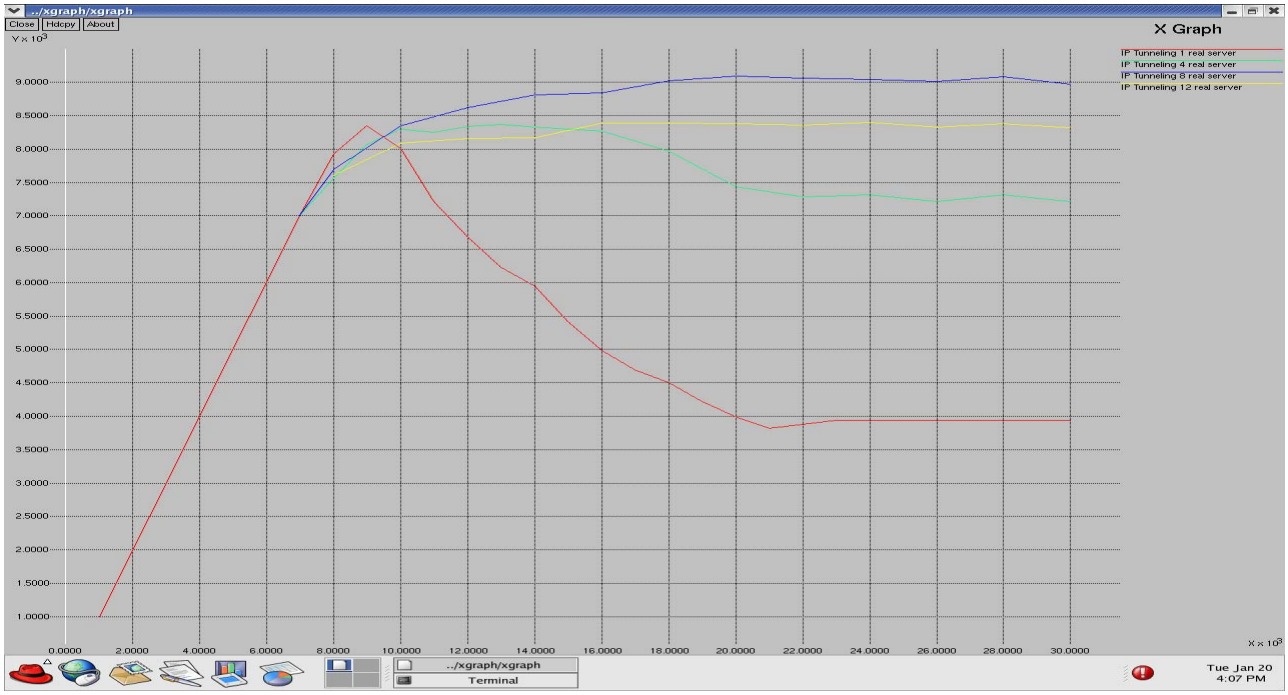

Figure 15 throughput, the IP Tunneling, topology 1, 100M LAN

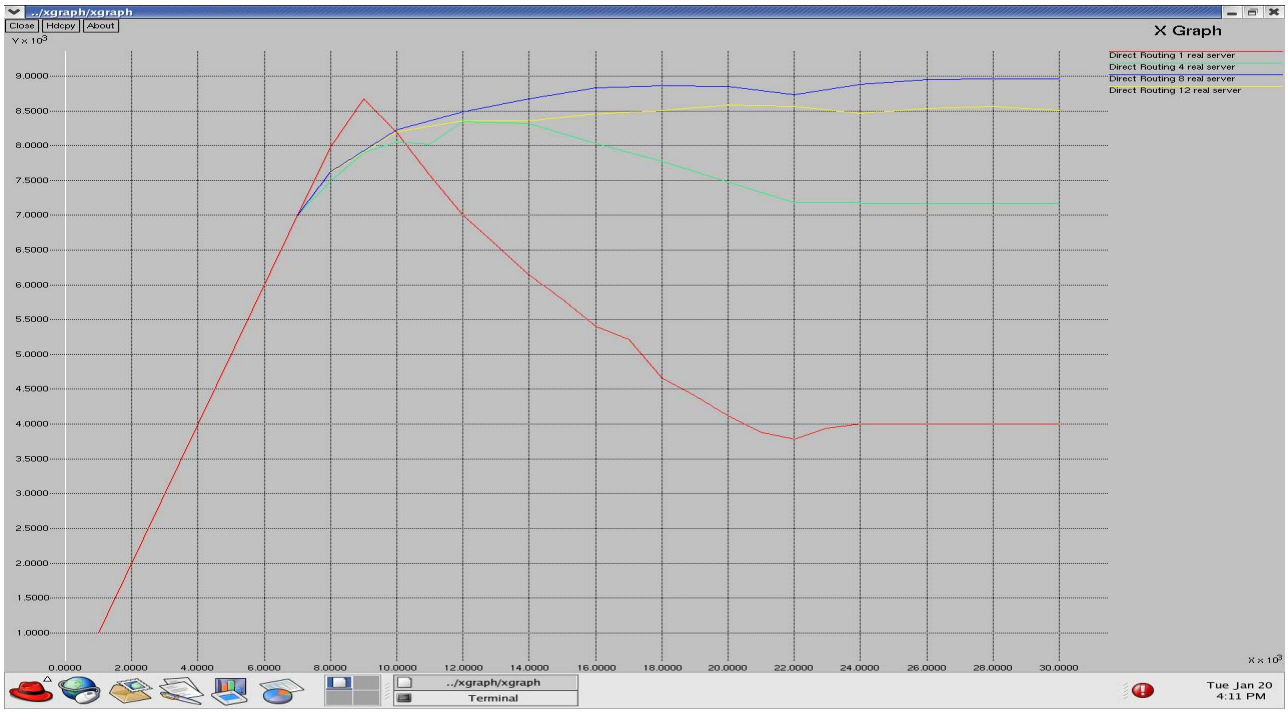

Figure 16 throughput, the Direct Routing, topology 1, 100M LAN

It’s slightly different when using transmitting methods of the IP Tunneling and Direct Routing. The throughputs of having 12 real servers are still lower than that of having 8 real servers. But the throughputs of the virtual server via these two techniques when having 8 real servers are the highest.

This difference between the modified NAT and IP Tunneling, Direct Routing is due to their different processing speed. The Direct Routing runs the fastest, and the speed of the IP Tunneling and Direct Routing is closer than that between the modified NAT and Direct Routing. Since the modified NAT is the slowest, it gives the LAN more time to transmit the packets, thus the LAN under the modified NAT is relatively busier than under the other two methods. When using the modified NAT under 8 real servers, the increasing traffic counteracts the benefit of adding 4 more servers and causes the throughput to decrease. When using the IP Tunneling and Direct Routing, this phenomenon occurs when the number of real servers reaches 12.

The ranking of the speed among these three methods is illustrated by figure 8 and figure 17 below. In figure 8, when the connection rate is lower than 22,000/s, the loss rate of the Direct Routing on the load balancer is the highest, the IP Tunneling is the second, the modified NAT is the last. Given the speed gap between the IPVS agent and the local area network, we can think that the speed of the Direct Routing is the fastest, thus the corresponding IPVS agent pumps more packets into the queue belonging to the local area network VLINK within the same period. So the queue when using the Direct Routing is quicker to be full and causes more packet loss on the load balancer. This is confirmed by getting the first packet dropping time on the load balancer from the trace file. When the connection rate is 9,000/s, the first packet drops at 2.712400s when using the modified NAT, it is at 1.075400s when using the IP Tunneling, and it is at 1.160400s when using the Direct Routing. Furthermore, since more packets are sent to the real server due to the relatively slow processing speed of the modified NAT, the throughput of the modified NAT is even higher than

that of the Direct Routing and IP Tunneling.

The IP Tunneling is a little different from the other two methods, because it must allocate one more IP header, increases the transmitting overhead. In fact, the processing speed of the IP Tunneling and Direct Routing is very close, in a 1000M LAN, with the connection rate 70,000/s, under the IP Tunneling, the first packet drops at the time 1.039900s, the time is 1.038527 under the Direct Routing. Under the modified NAT, it is 1.388400s.

Under different connection rate and different transmitting methods, the first packet loss may happen at the real server or the load balancer. As a whole, the virtual server via the modified NAT drops less number of packets, and has a higher throughput than the other two methods.

It must be notified that the differences between the processing speed of these three techniques are very small. The average round trip time of the modified NAT and Direct Routing when the connection rate is 6000/s is the same----0.007233s. The round trip time of the IP Tunneling is 0.007242, a little larger than the other two techniques, mainly because of adding the extra IP header into the packet when using the IP Tunneling method. We choose a connection rate of 6000/s just because under this configuration no packet loss occurs, every request has been processed and replied by the virtual server under this scenario.

Local area network with a bandwidth 1000M

We know from above simulations that a LAN with a bandwidth 100M becomes the whole system's bottleneck. Simulations below just aims to find how is the situation of the virtual server in a LAN with a 1000M bandwidth.

Below are the figures:

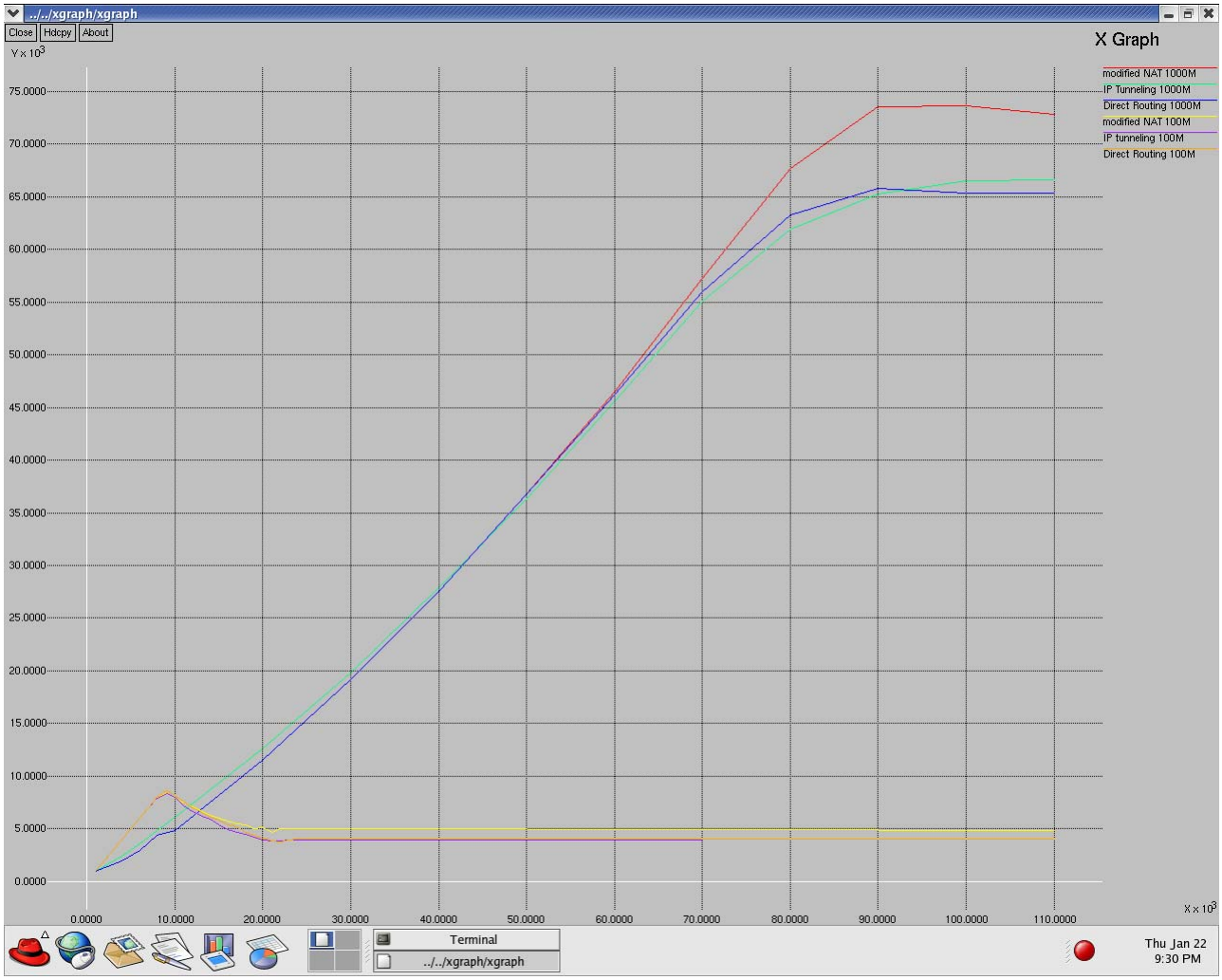

Figure 17 throughput, topology 1, 1000M LAN

Figure 17 shows that the throughput of the virtual server in a 1000M LAN is much higher than that of the virtual server in a 100M LAN. The transmitting capability of the local area network is crucial to the system's performance. And when the connection rate exceeds 30,000/s, the throughput does not increase anymore, but is rather steady. Is this caused by reaching the maximum processing capability of the IPVS agent or by the local area network? Figure 18, 19 and 20 shows that packet loss is still severe in a 1000M local area network.

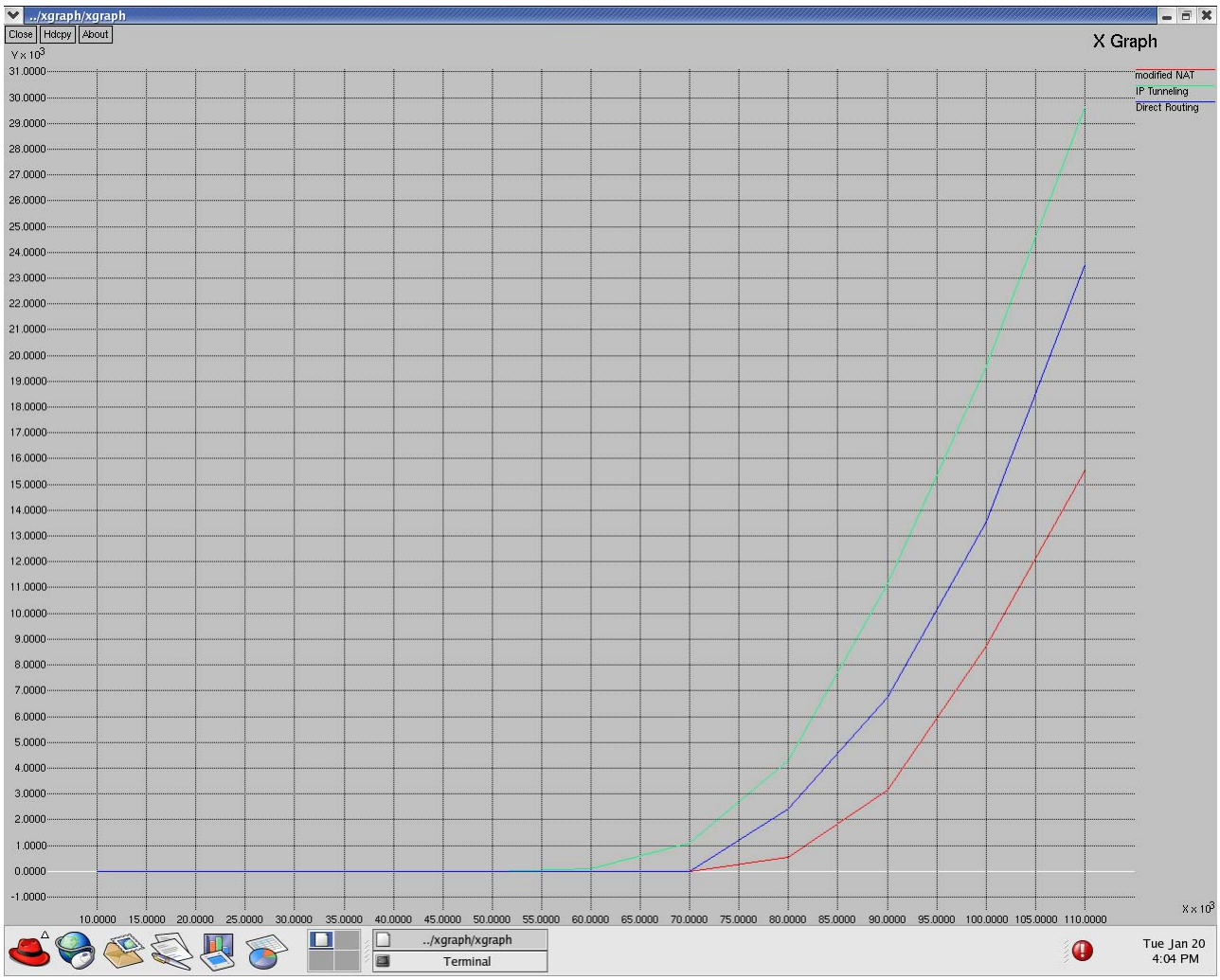

Figure 18 loss rate on the load balancer, topology 1, 1000M LAN

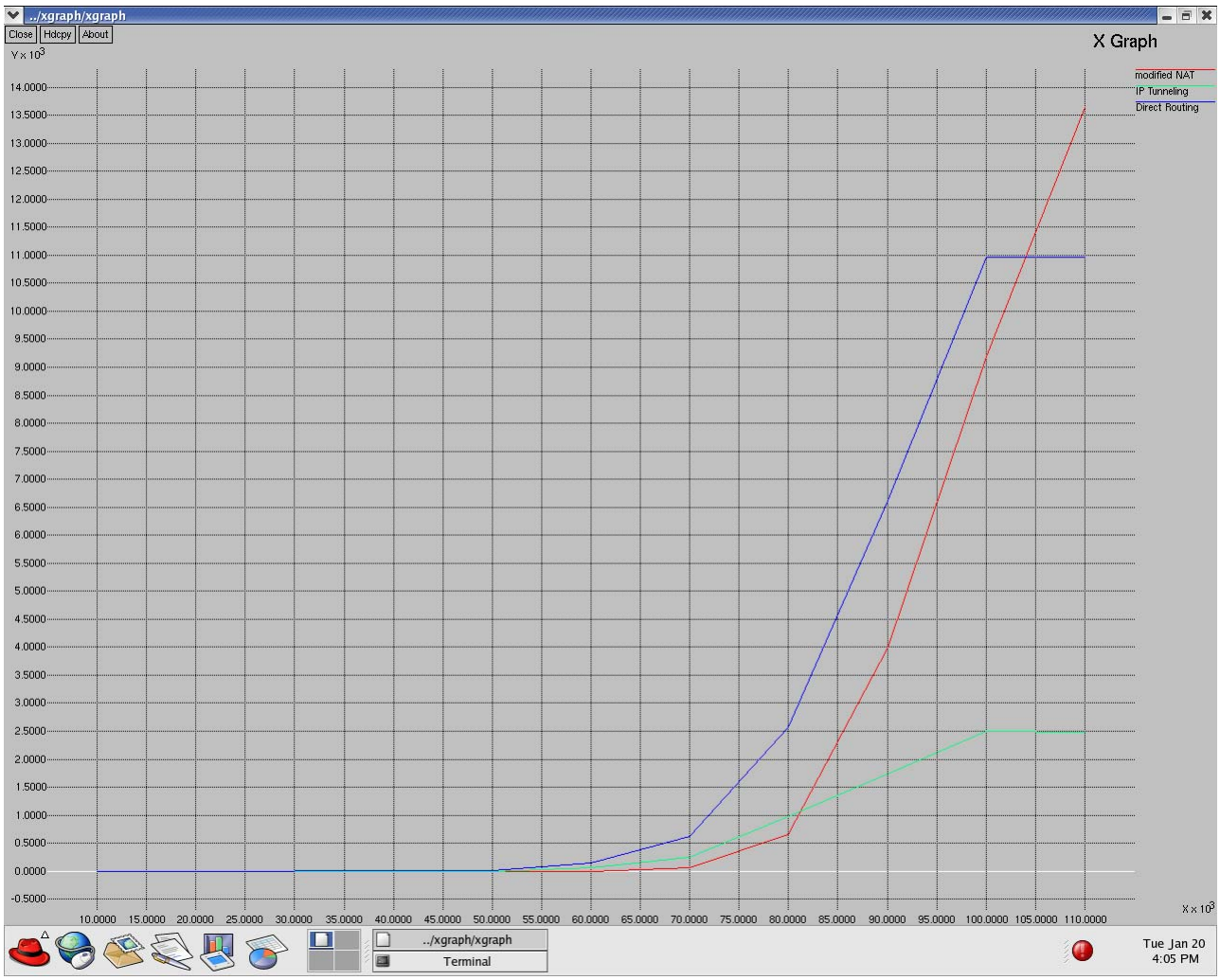

Figure 19 loss rate on the real server, topology 1, 1000M LAN

The packet loss on the load balancer and the real server shows that a 1000M LAN is still the bottleneck of the system. Its transmitting capability can not match that of the IPVS agent. Figure 20 shows the throughput of the virtual server when having 1, 4, 8 and 12 real servers.

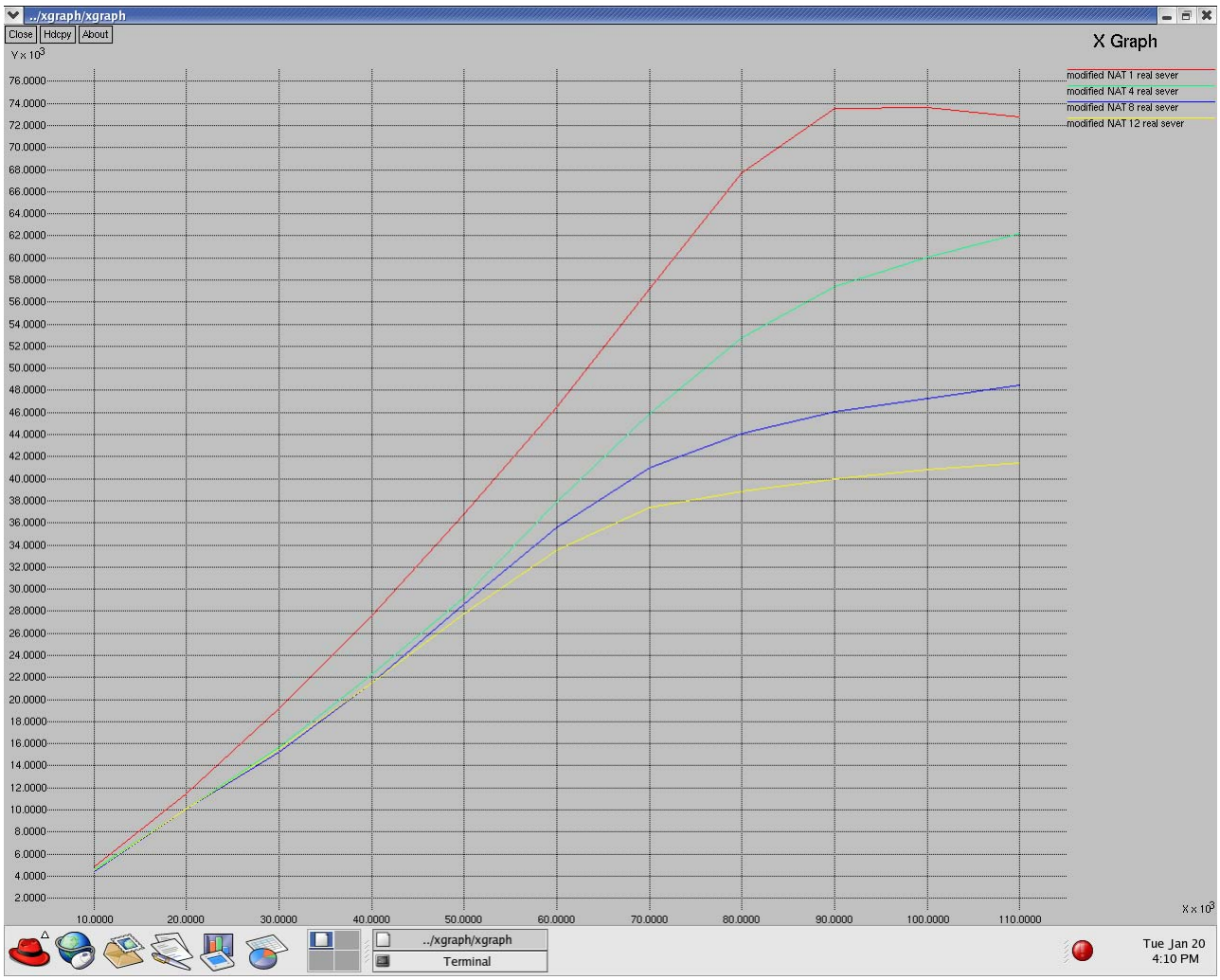

Figure 20 throughput, the modified NAT, topology 1, 1000M LAN

In a 1000M LAN, the throughput of the virtual server via the modified NAT when having one real server is the highest, adding more real servers to the network will degrade the system's performance. This is due to both the higher loss rate on the load balancer, the higher loss rate on the real server, and the higher loss rate due to collision. Relevant figures and those for the IP Tunneling and Direct Routing are included in the appendix. These figures together with figure 20 illustrate that adding new real servers in a 1000M LAN under current configuration and topology will cause more network traffic and even lower the throughput of the virtual server. The local area network with a bandwidth 1000M is still the system's bottleneck.

Another topology

In this topology, the local area network will only handle half of all the traffic, we simulate the scenario that the local area network has the least affection on the virtual server. Below is the structure of this topology:

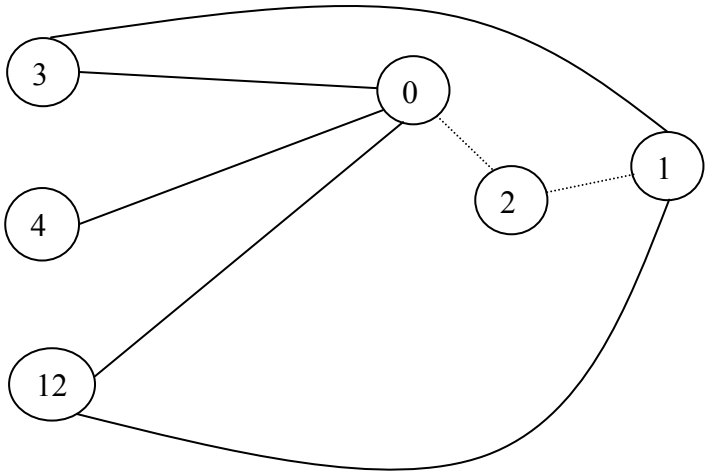

Figure 21 topology 2

In the above figure, node 0 is the load balancer, node 1 is the real server, and node 2 is the VLAN node in ns-2. Node 3 to 12 are the clients(client number varies under different connection rate). The links between the real servers and the clients have a bandwidth of 10M and delay of 1ms. Other parameters remain the same with above simulations.

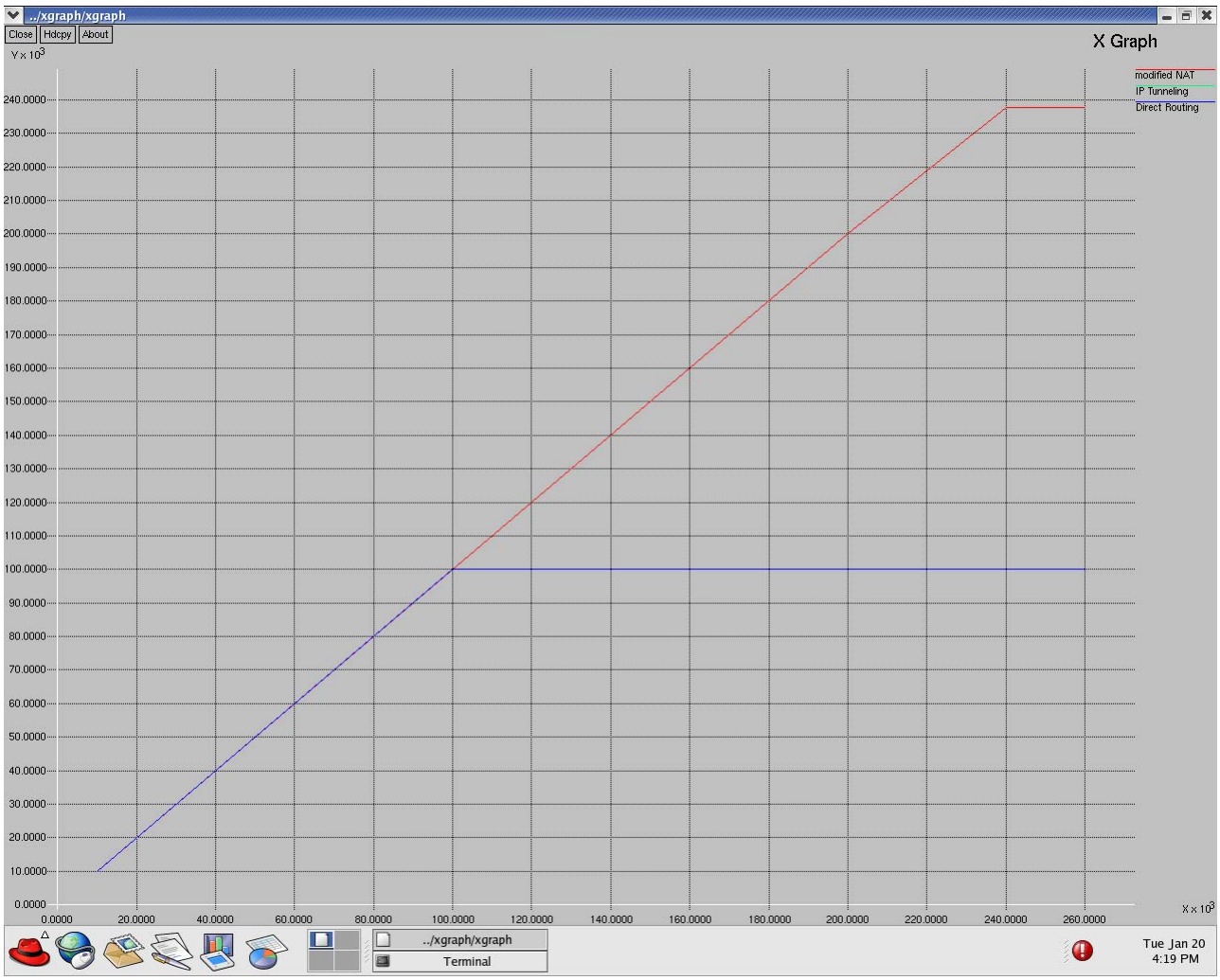

Figure 22 throughput, topology 2, 1000M LAN

The throughput under this topology can be views as the upper bound of the throughput of the virtual server in a 1000M LAN. From figure 22 we can see that the virtual server via the IP Tunneling and Direct Routing have the similar processing speed. The modified NAT is slower, gives more time for the network to transmit the packets, thus has a higher throughput. Under all the three techniques, no packet lost on the real server or due to collision, all packets are lost on the load balancer because of the mismatch of the processing speed of the IPVS agent and the local area network. Figure of the loss rate on the load balancer is included in the appendix.

Scheduling algorithms

We have implemented the Round Robin, weighted Round Robin, Least Connection and weighted Least Connection algorithms in this project. The round robin algorithm just chooses the next server, thus it has the least scheduling overhead. The weighted round robin algorithm will schedule the next server which has a weight is larger than current weight(the current weight is initialized to the largest weight among the servers, and will update after each scheduling). The least connection algorithm needs to scan the real server list to find the real server with the least connection number. The weighted least connection also scans the list and chooses a server with the minimum value of (connection number)/weight. The least connection and weighted least connection algorithm needs to scan the whole real server list, thus they have the highest overhead.

A simulation is designed to investigate the impact of the scheduling overhead. In this simulation, the virtual server has 200 real servers, uses the modified NAT transmitting method, all the servers have the equal weight. And there's one client node sends request to the virtual server at an interval of 1ms. The result is that the round trip time when using above four scheduling algorithms is the same----0.006891s. This means that the four algorithms have nearly the same cost, the overhead of searching the real server list is small. It needs a more powerful machine to see the overhead's impact on the system's throughput.

Round Robin vs. Weighted Round Robin

The weighted round robin can treat each real server different. If we can accurately give weights to the real servers, it will perform superior to the round robin. To illustrate this, simulation below uses two real servers, one has the weight 1, and the other server has the weight 3. Server one has a queue size of 50, server two has queue size of 150. One hundred clients send packets to the load balancer at a interval of 1ms. And the local area network queue size on the load balancer is set to be 10,000 to ensure that no packet loses on the load balancer. The LAN has a bandwidth of 1000M.

The IPVS agent is again modified to treat each request as a new one, thus it will schedule a new server for them. Below is the result:

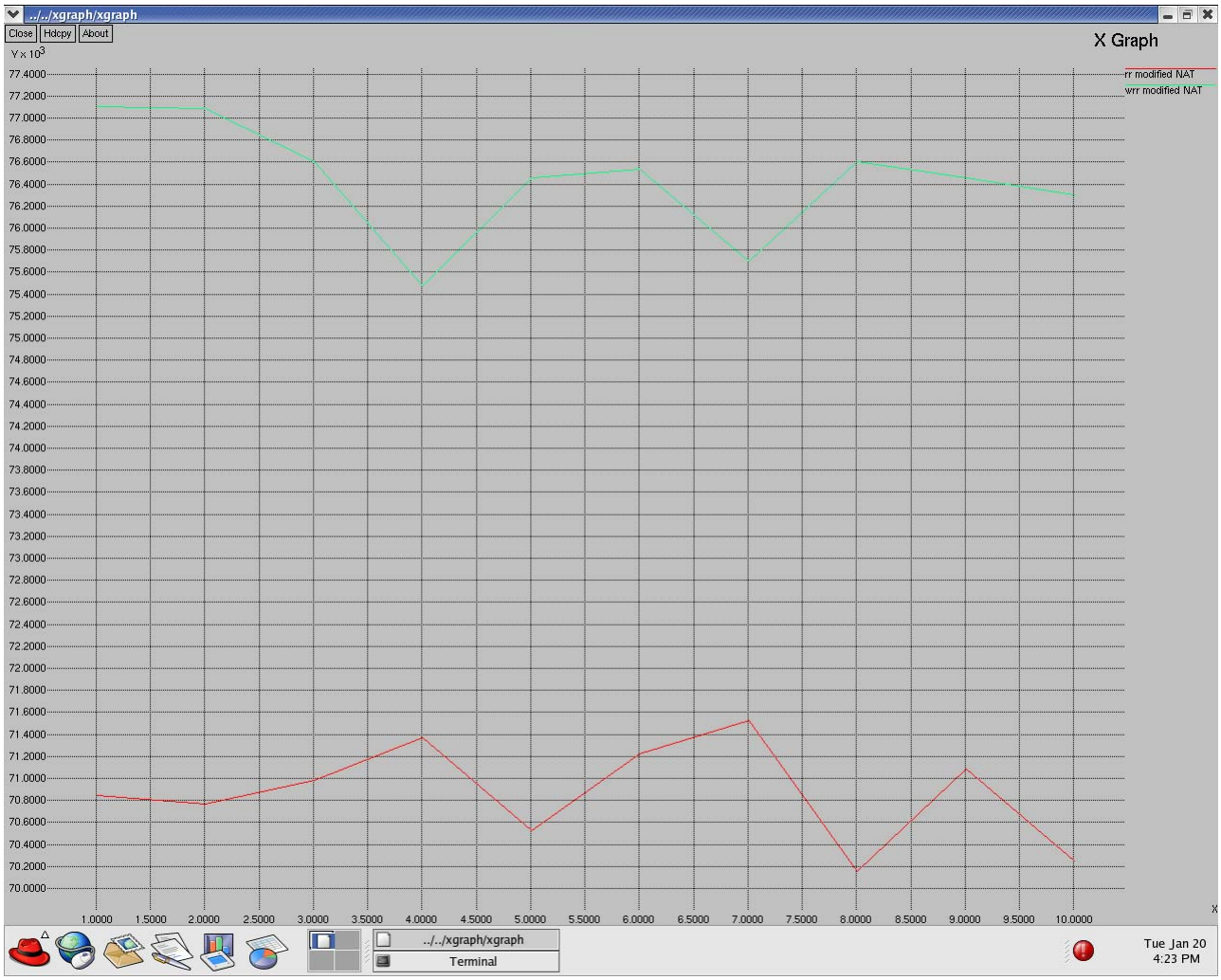

Figure 23 throughput, RR vs. WRR, 1000M LAN

The horizontal axis represents the time(unit is second), the vertical axis represents the throughput of the virtual server. Figure 23 shows that the weighted round robin performs much better than the round robin algorithm. This is because the round robin algorithm distribute packets equally to the real servers, but server one only has a queue size of 50, thus causes a much higher loss rate on this server. The figures of receiving rate and loss rate on each real server are included in the appendix.

Round Robin vs. Least Connection

Unlike round robin, the least connection algorithm is dynamic. When the work load of the requests are all the same, there're no much difference between these two algorithms(given each server has the same weight). However, if the work load of the requests vary, the least connection scheduling algorithm is superior to the round robin algorithm. Figure below illustrates this:

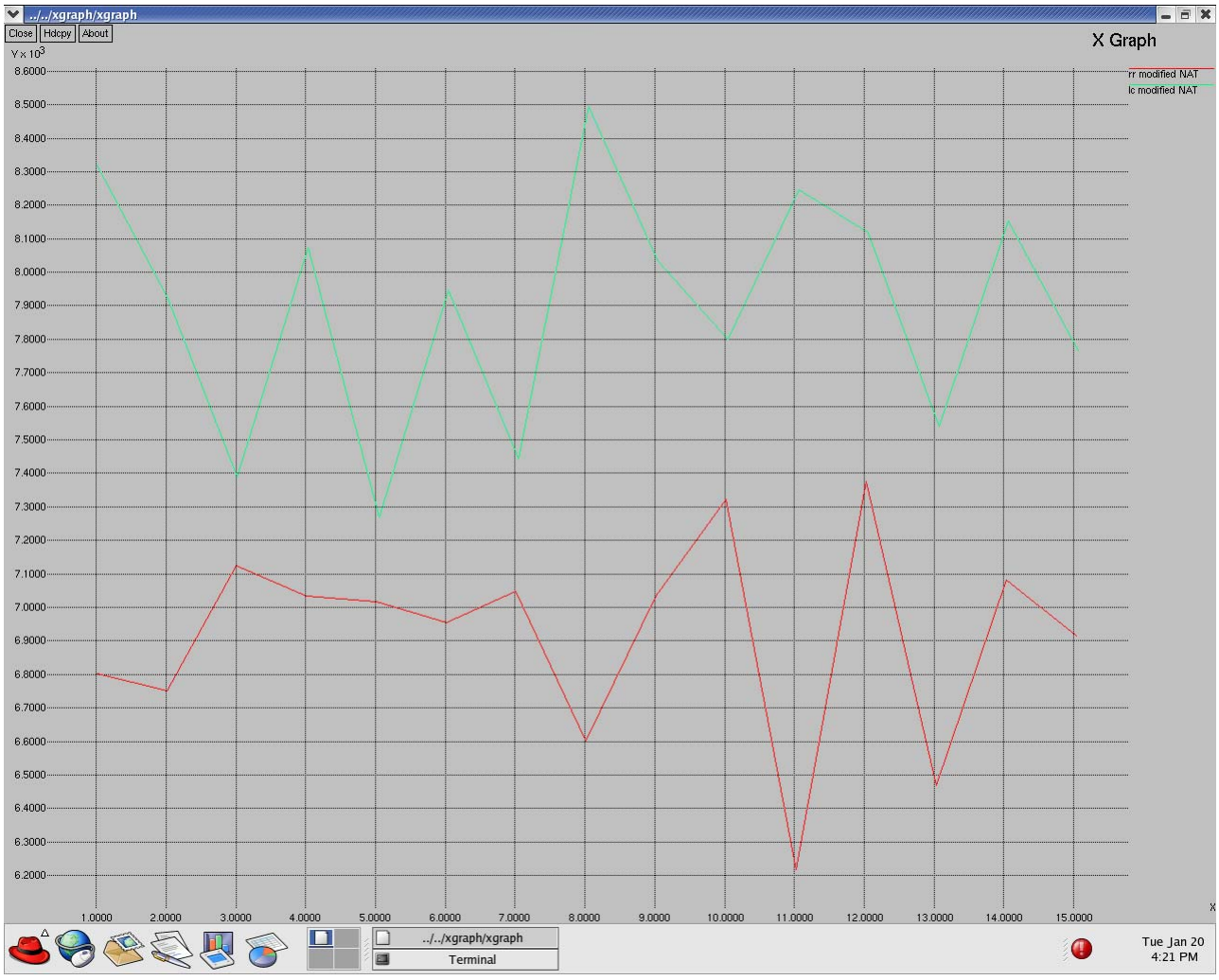

Figure 24 throughput, RR vs. WRR, 1000M LAN

Above simulation uses a designated traffic pattern file. The pattern is that, the only one client sends request to the virtual server at an interval of 0.5ms or 1ms. That is to say, the client sends the next packet randomly after 0.5ms or 1ms. This simulation treats each request different, if the sequence number of the request packet is odd, the sink agent will send two more packets back. The timeout of each connection is set to

be 1ms in the IPVS agent. The bandwidth of the LAN is 100M. Below is the throughput of the virtual server using the round robin and the least connection scheduling algorithms and the modified NAT transmitting method:

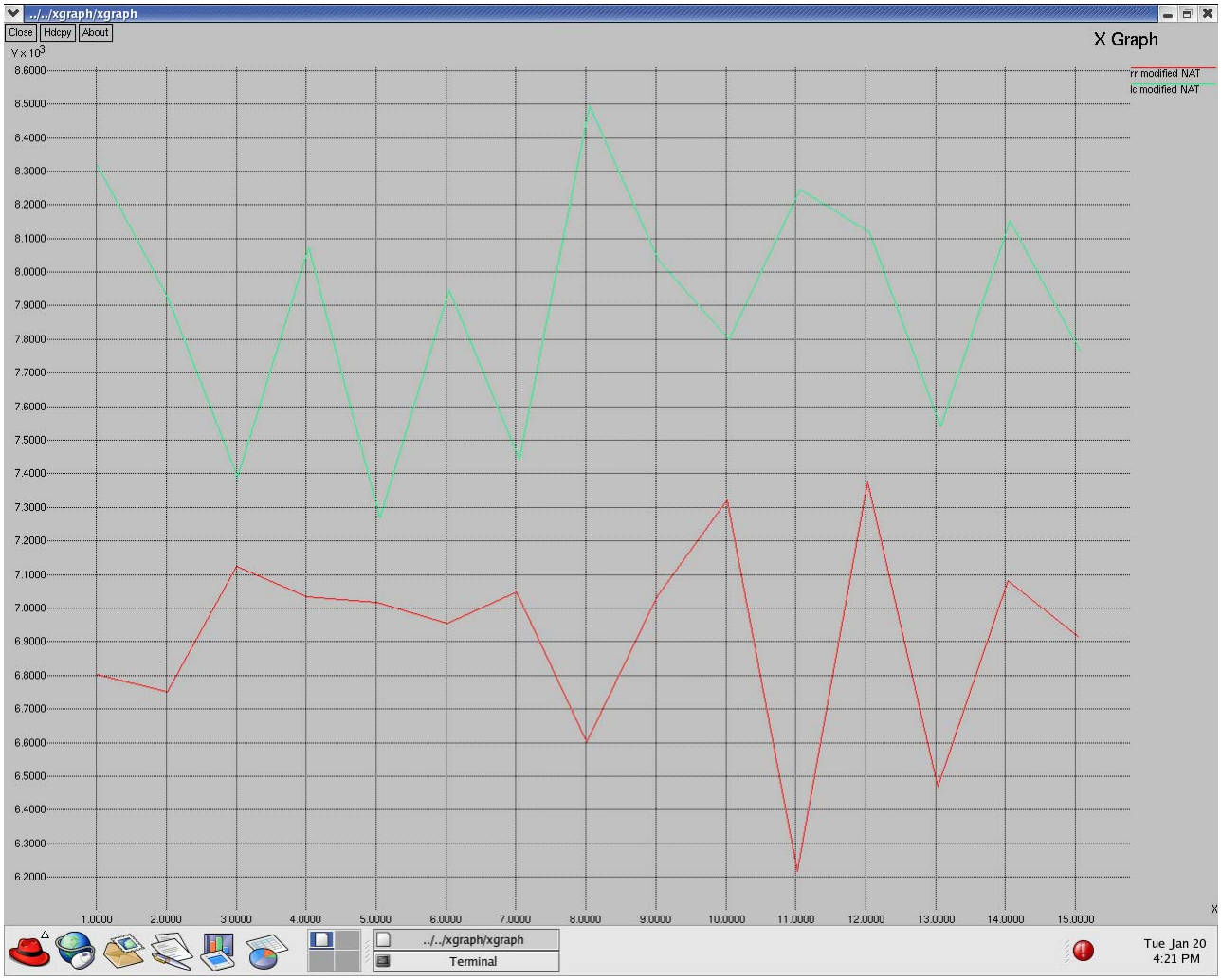

Figure 25 throughput, RR vs. LC, 100M LAN

As in figure 24, the horizontal axis represents the time, the vertical axis represents the throughput. We can see the least connection algorithm performs better than round robin all the time. The reason is that the round robin treats each request and server equal, just deliver packets to different server in turn. Thus the real server 1 is always receiving the packets whose sequence number is odd, and the real server 2 is always receiving the packets whose sequence number is even. So the real server 1 sends twice more number of packets back, the traffic is not evenly distributed between these two servers. While using the least connection algorithm, things are different.

Since the least connection algorithm chooses the server with the least number of connections, and randomly arrival time of the packets(since we have a random sending interval), at given time, the connection number on the two real servers are different. Thus the least connection algorithm will not always choose server 1 for packets with an odd sequence number. The network traffic is rather evenly distributed among the servers. Relevant figures are included in the appendix.

Throughput Improvement

Recall that in the simulations above, we see that the local area network is the whole system's bottleneck. To improve the system's throughput, the best way is to improve the performance of the local area network and eliminate this bottleneck. But it's also helpful to improve the virtual server's performance under current LAN technology.

This project uses a hierarchical virtual server structure to improve the virtual server's throughput. Though it's rather simple and straightforward, no similar work has been seen in the development and application of LVS. The hierarchical structure is illustrated in the figure below:

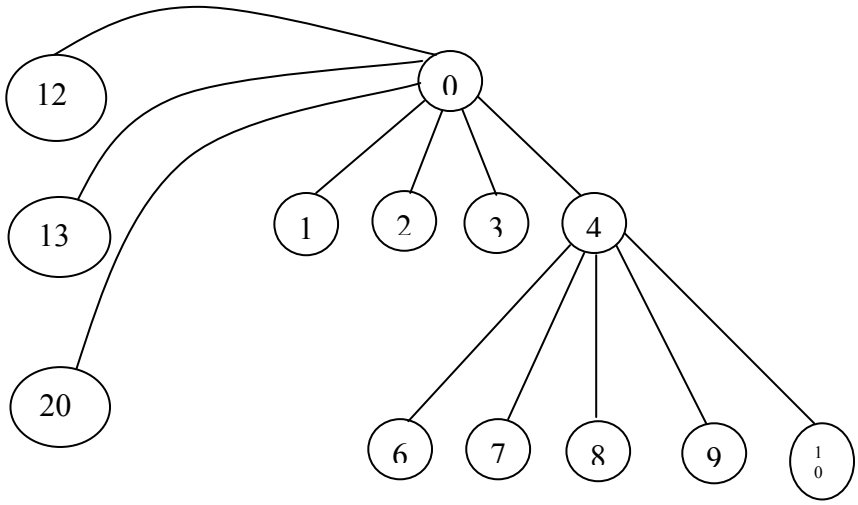

Figure 26 A hierarchical virtual server structure

In figure 26, node 12 to 20 are clients. Node 0 and 4 are load balancers. Node 1, 2, 3

and 4 are real servers of load balancer 0. Node 6 to 10 are real servers of load balancer 4. Node 4 acts both as a real server of load balancer 0 and load balancer of node 6 to 10. Node 0 to 4 consists a LAN, node 4 and node 6 to 10 consists another LAN. These two LAN further consists the virtual server.

When a client, for example node 12, send a request to the virtual server, it first arrives at node 0. Node 0 chooses a real server, modifies the packet and forward it to the chosen real server. When node 0 chooses node 4, whose role is both a real server and a load balancer, node 4 will further modify the packet and forward the packet to one of its real servers.

The advantage of this hierarchical structure is that, it has fewer nodes in each level. From previous simulation, we can see that adding more machines will degrade the system's performance provided that the real server can respond promptly and the requesting connection rate is high. We can expect that this structure will improve the system's throughput. And it is verified by figure below:

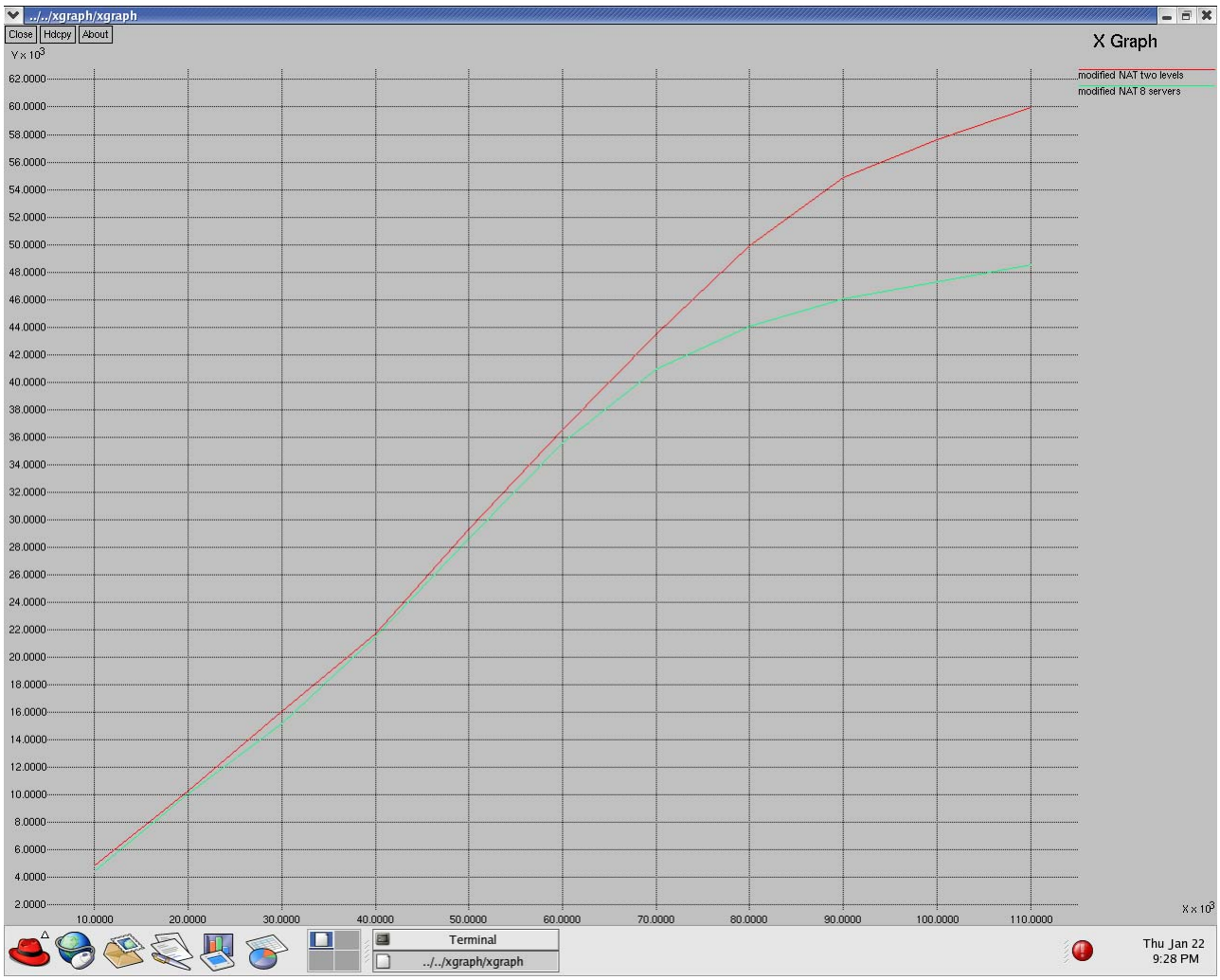

Figure 27 throughput, NAT, 8 servers, hierarchical structure

Figure 27 is a comparison of the throughput of the virtual server using the modified NAT transmitting method under these two structures. Other parameters just are the same with in figure 21, a 1000M LAN, queue size of 100, etc.

We can see in figure 27 that the hierarchical structure performs better. The reason is just that fewer packets are dropped on the load balancer. It can be seen from figure below:

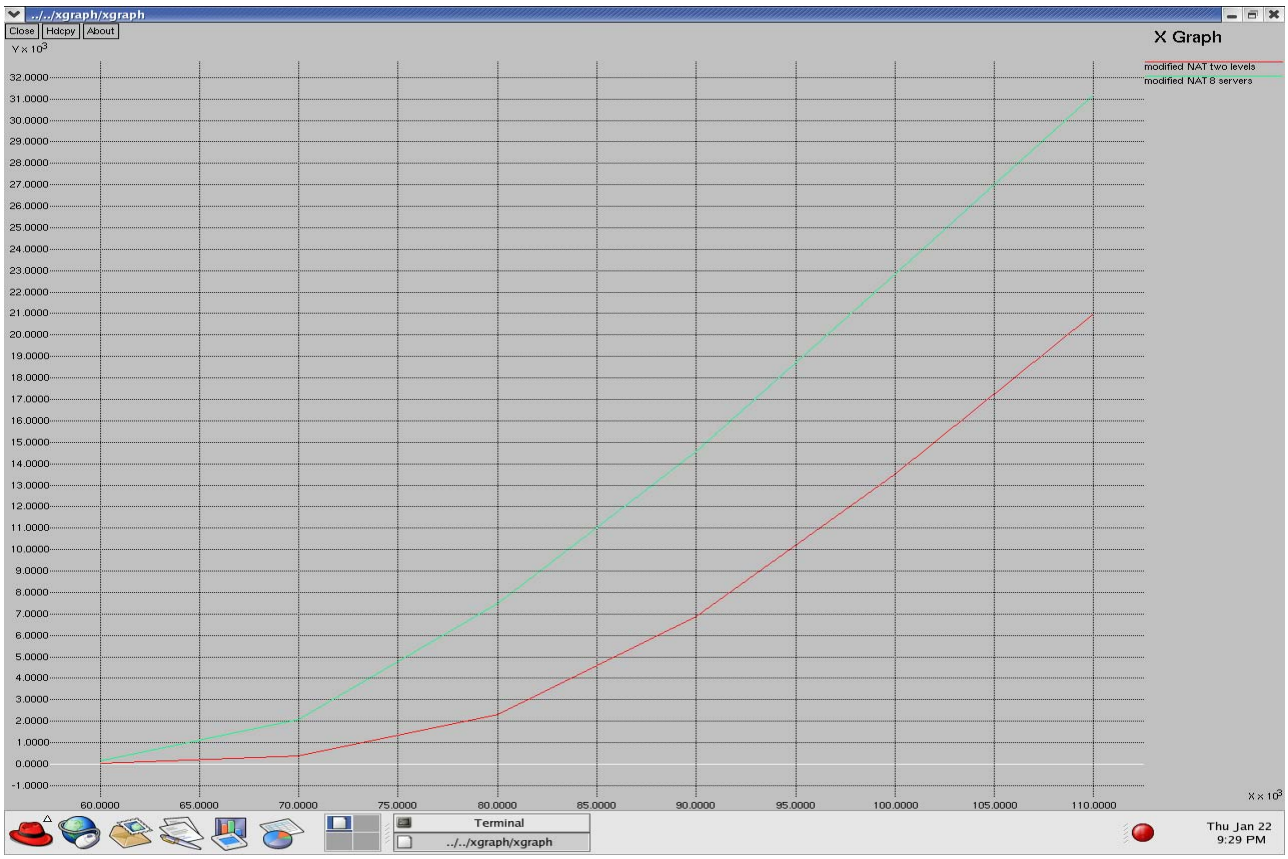

Figure 28 loss rate on the load balancer, NAT, 8 servers, hierarchical structure

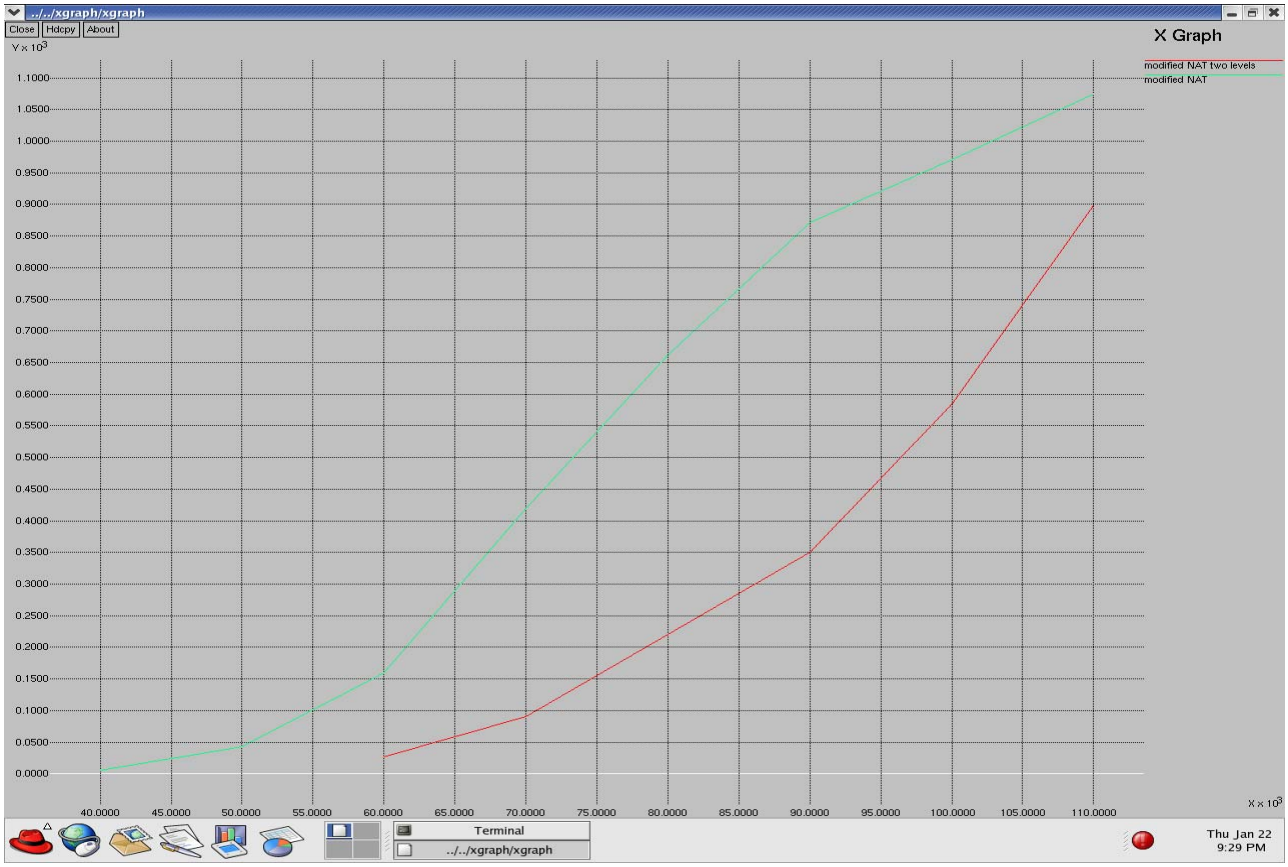

Figure 29 loss rate on the real server, NAT, 8 servers, hierarchical structure

In figure 28, loss rate on the load balancer(just node 0) with a hierarchical structure is much smaller. In figure 29, the loss rate on the real server with a hierarchical structure is higher, but the difference is small. In fact, the loss rate on the real server isn't high under the two structures, for we have 8 servers to share the work load. The loss rate on node 0 is the main factor influencing the throughput.

Currently the approach to deal with the extremely website is to configure several clusters, and use a round robin DNS to distribute requests among the different clusters. But as we have pointed out, the DNS redirection approach may lead to severe load imbalance, and under this approach, the scheduling granularity is per cluster. This may cause some clusters to a very busy state, while others are not.

Another approach may be just group the real servers into different LAN, and all addressable by the load balancer. But the scalability of this approach is not as good as the hierarchical structure.

The disadvantage of the hierarchical structure is that, a request may unfortunately go through several levels, this introduces extra overheads. But we can carefully choose the level of the hierarchical structure, and connect each load balancer to the outside world directly. By doing so, replying packets need not go up through the tree and can be forwarded to the users directly.

So this structure seems a good solution, though it needs further consideration. In Linux, it seems that the virtual server via the IP Tunneling can support this structure directly. It needs further tests under LINUX, if it can not, it will be helpful to add such a support. The virtual server via the NAT can not support the structure. So it's another important reason that LVS should modify its current NAT technique.

Conclusions and future work

Through simulations of virtual server via the modified NAT, IP Tunneling and direct routing, we find that the performance of the modified NAT is comparable to other two techniques under all simulation scenarios in this project. Thus although the modified NAT needs to change the codes on the real servers, it is worthwhile to implement the virtual server using the modified NAT. Or at least, it is worthwhile to add the support for the modified NAT and provide an option to users.

Also through simulation, we find the implementation of the IPVS is not the bottleneck of the virtual server. Even a LAN with a bandwidth 1000M can not catch up with its processing speed, and the LAN becomes the system's bottleneck.

Simulation results show that the Direct Routing method is not so good as the first look. Its speed is slightly faster than the other two techniques, but it requires all real servers on the same physical segment. Given the bottleneck is the local area network, its faster speed is even not so attractable. Compared with the Direct Routing, the modified NAT and IP Tunneling have nearly the same speed and other more benefits. The modified NAT can use private IP addresses, and the IP Tunneling can be used in geographically distributed environments.

Finally, simulation results show that the hierarchical virtual server structure can improve the system's throughput. But in this project, all the real servers are assumed that they have the enough capability to process and respond all the requests. It needs further consideration and simulation with using a distributed file system.

References

[1] Wensong Zhang, “Linux Virtual Server for Scalable Web Services”, http://www.linuxvirtualserver.org/ols/lvs.ps.gz

[2] Wensong Zhang, Shiyao Jin, Quanyuan Wu, “Creating Linux Virtual Servers”, http://www.linuxvirtualserver.org/linuxexpo.html

[3] Internet Appliance INTERNET pro Enterprise Stack: Performance & failover testing, http://www.veritest.com/clients/reports/internetapp/internetapp.pdf

[4] Cisco local director: Cisco Systems, Inc., 1998, http://www.cisco.com/Warp/public/751/lodir/index.html

[5] T.Brisco, Dns support for load balancing, http://www.ietf.org/rfc/rfc1794.txt

[6] M. Wangsmo. White paper: Piranha – load-balanced web and ftp clusters. http://www.redhat.com/support/wpapers/piranha/

[7] P. Braam and et al. The intermezzo project. http://intermezzo.org/

Appendix A

Below are the files included in the implementation:

Modifications in ns-2

Classifier/classifer-hash.h

Classifier/classifier-has.cc

Common/agent.h

Common/agent.cc

Common/ns-process.cc

Common/packet.h

Mac/lanRouter.h

Mac/lanRouter.cc

Mac/ll.h

Mac/ll.cc

Tcl/lib/node.tcl

IPVS Agent:

ip_vs.h

ip_vs_core.cc

ip_vs_conn.cc

ip_vs_rr.cc

ip_vs_lc.cc

ip_vs_wrr.cc

ip_vs_wlc.cc

jhash.h

list.h

types.h

IPVS sink agent:

ip_vs_sink.cc

Simulation scripts:

Nat1.tcl

Tunnel1.tcl

Dr1.tcl

Nat3.tcl

Tunnel3.tcl

Dr3.tcl

Nat-new.tcl

File for generating simulation results:

gen.cc

File for generating simulation traffic:

trafgen.cc

Appendix B Additional Figures

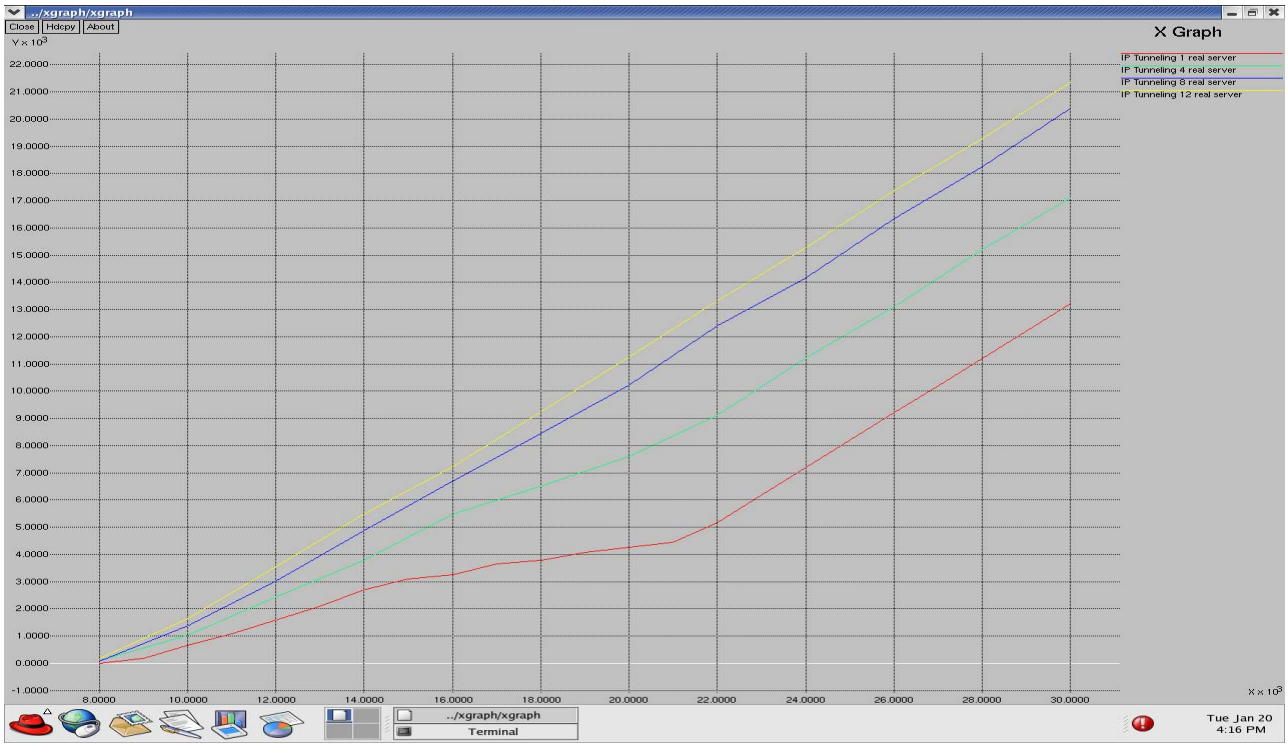

Figure 30 loss rate on the load balancer, IP tunneling, topo1, 100M

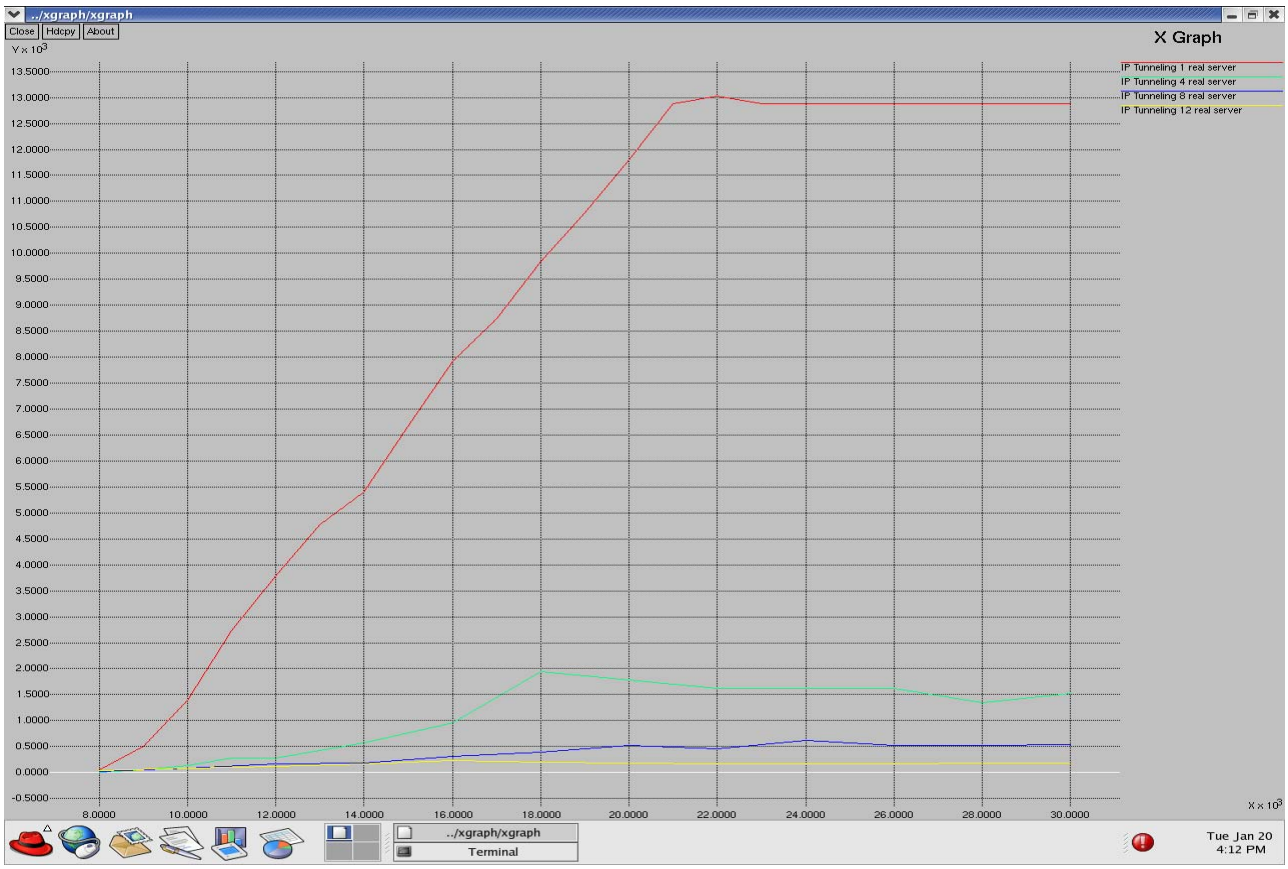

Figure 31 loss rate on the real server, IP tunneling, topo1, 100M

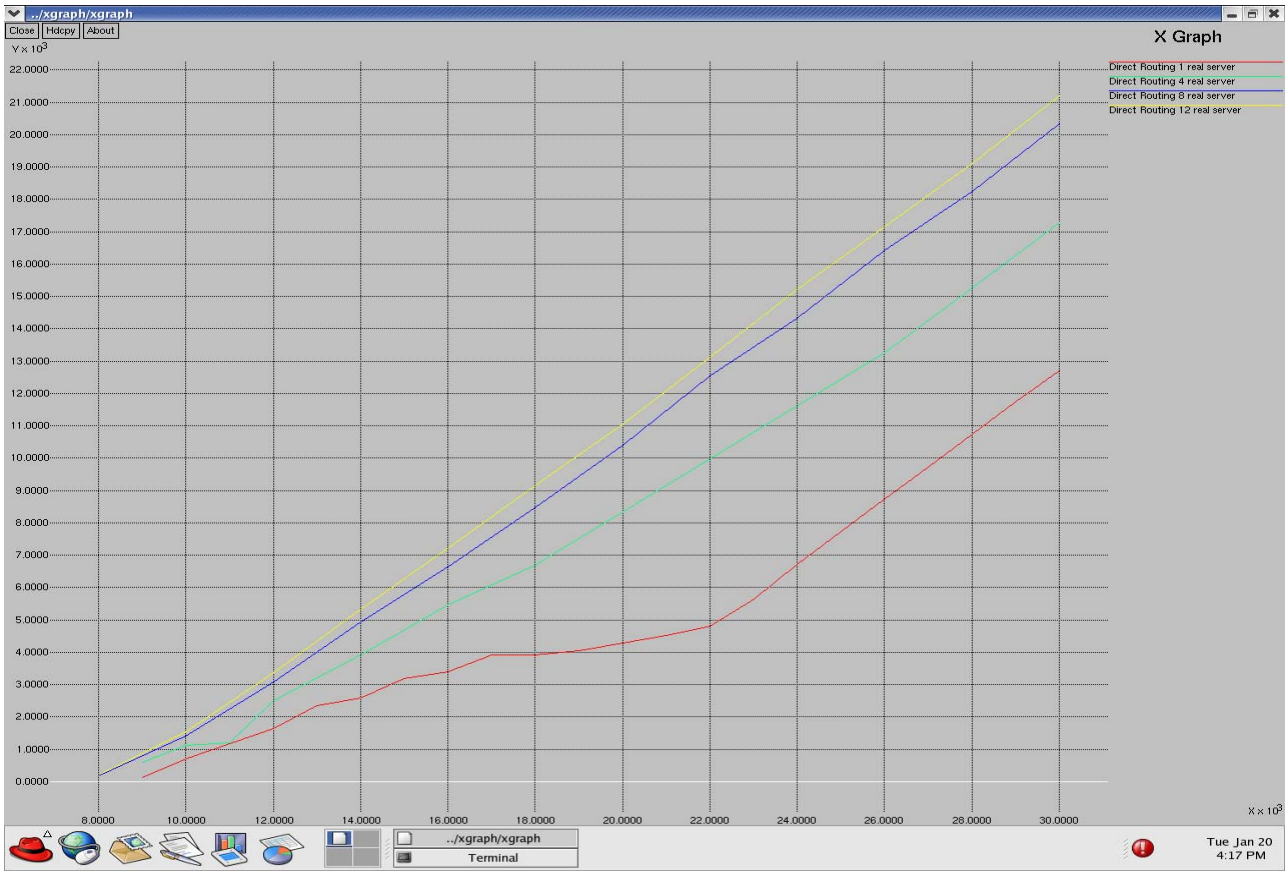

Figure 32 loss rate on the load balancer, Direct Routing, topo1, 100M

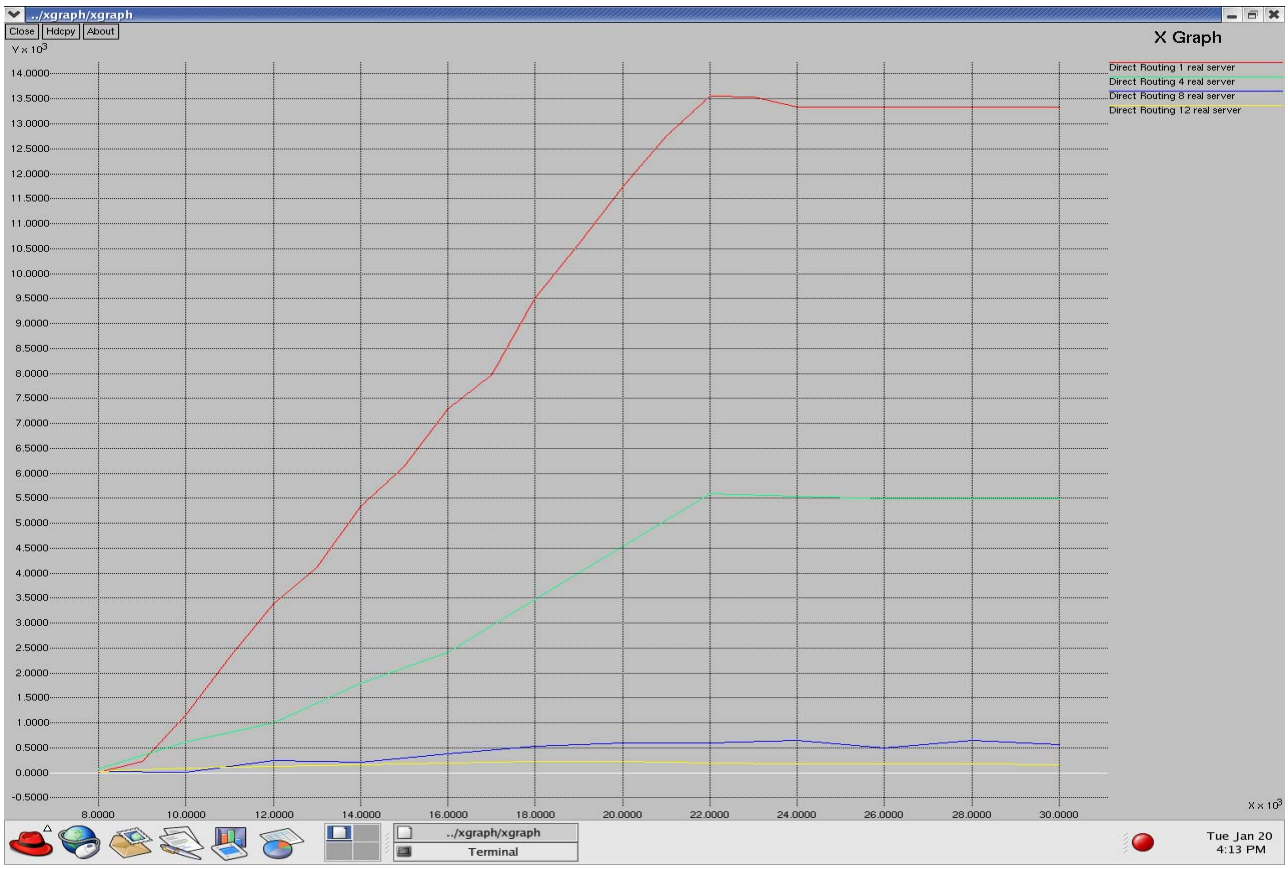

Figure 33 loss rate on the real server, Direct Routing, topo1, 100M

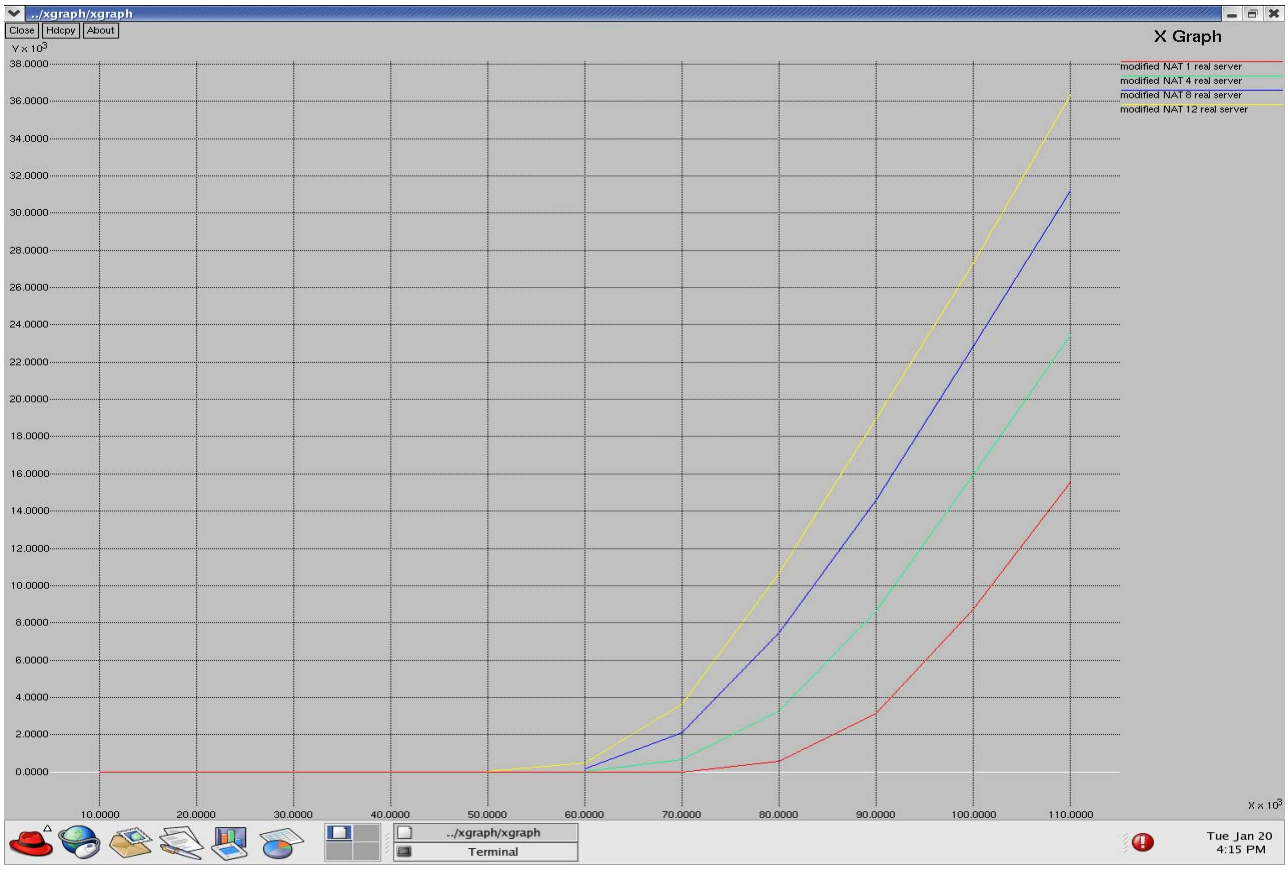

Figure 34 loss rate on the load balancer, modified NAT, topo1, 1000M

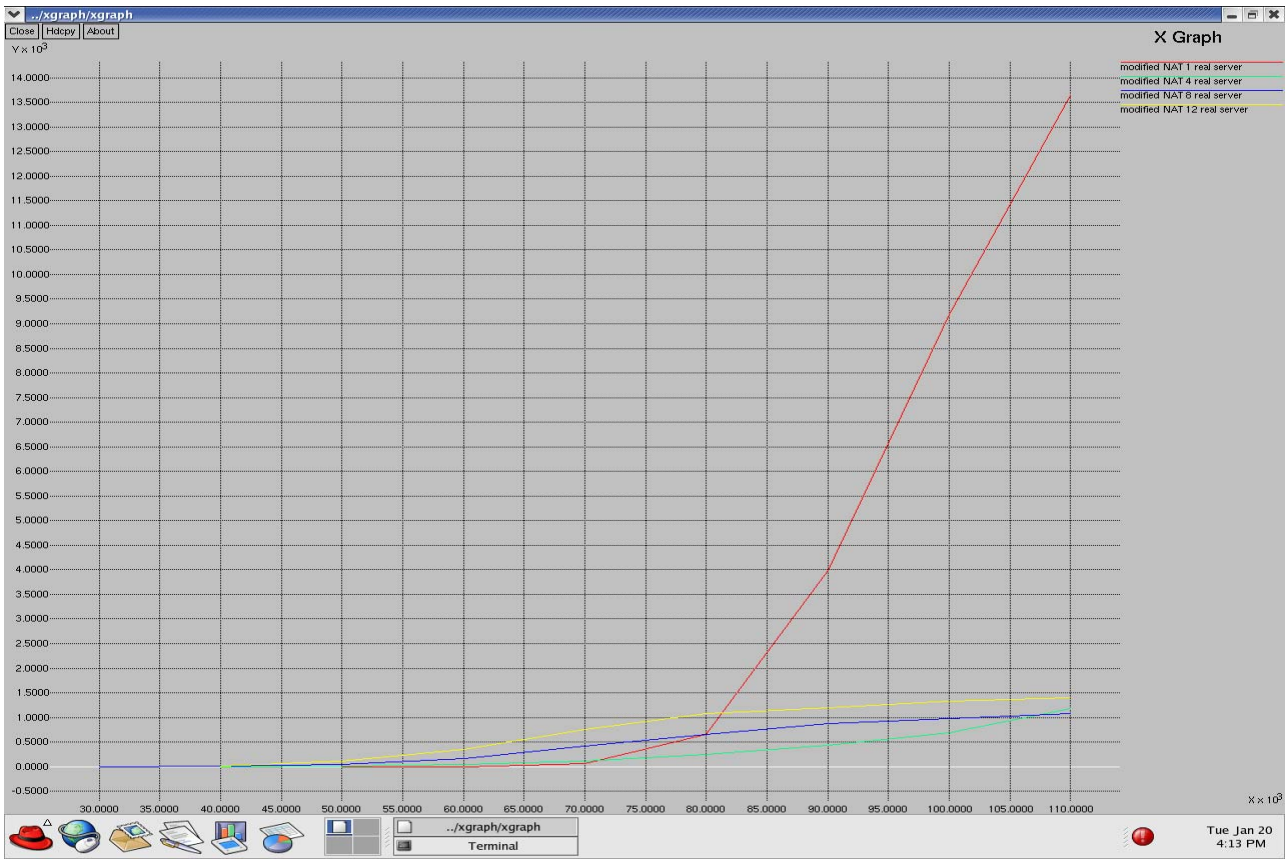

Figure 35 loss rate on the real server, modified NAT, topo1, 1000M

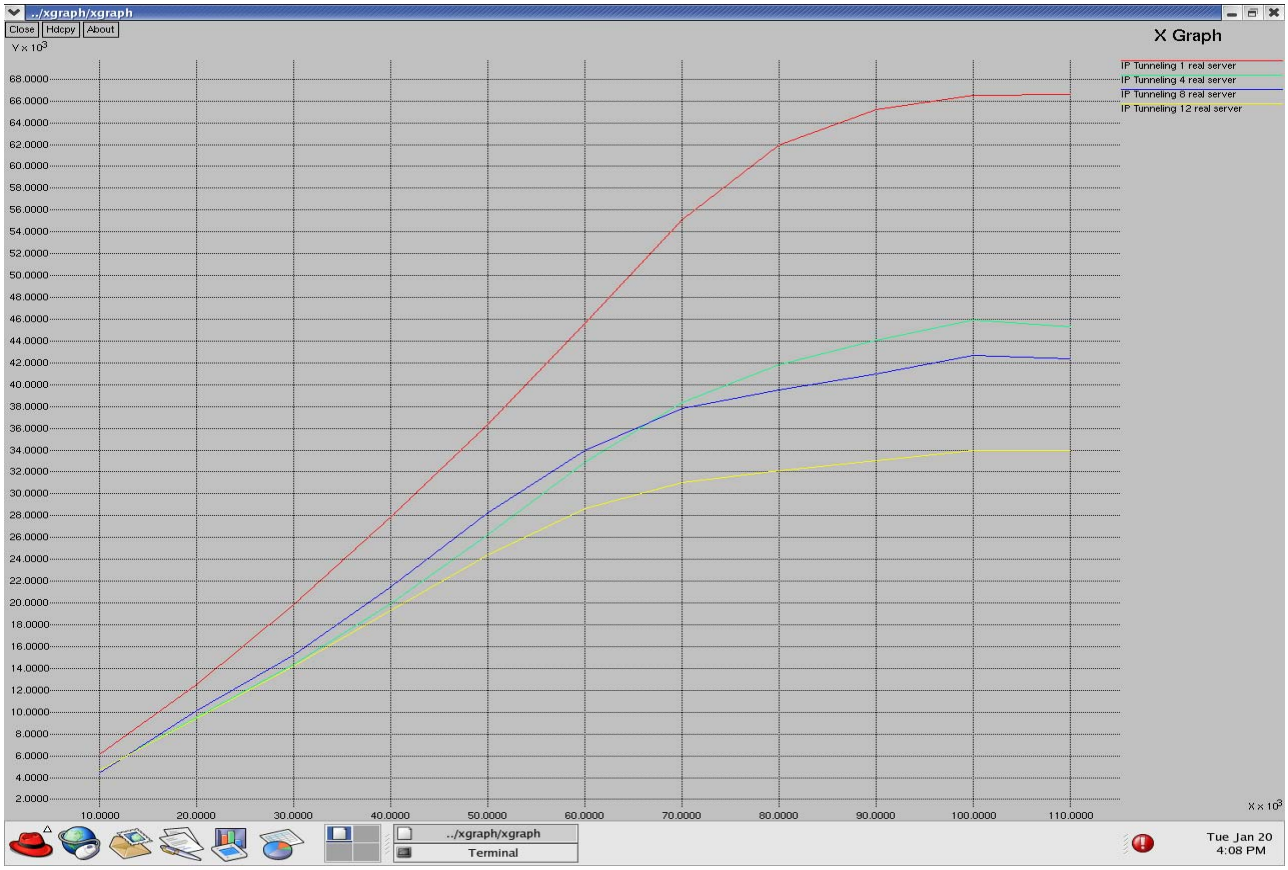

Figure 36 throughput, IP Tunneling, topo1, 1000M

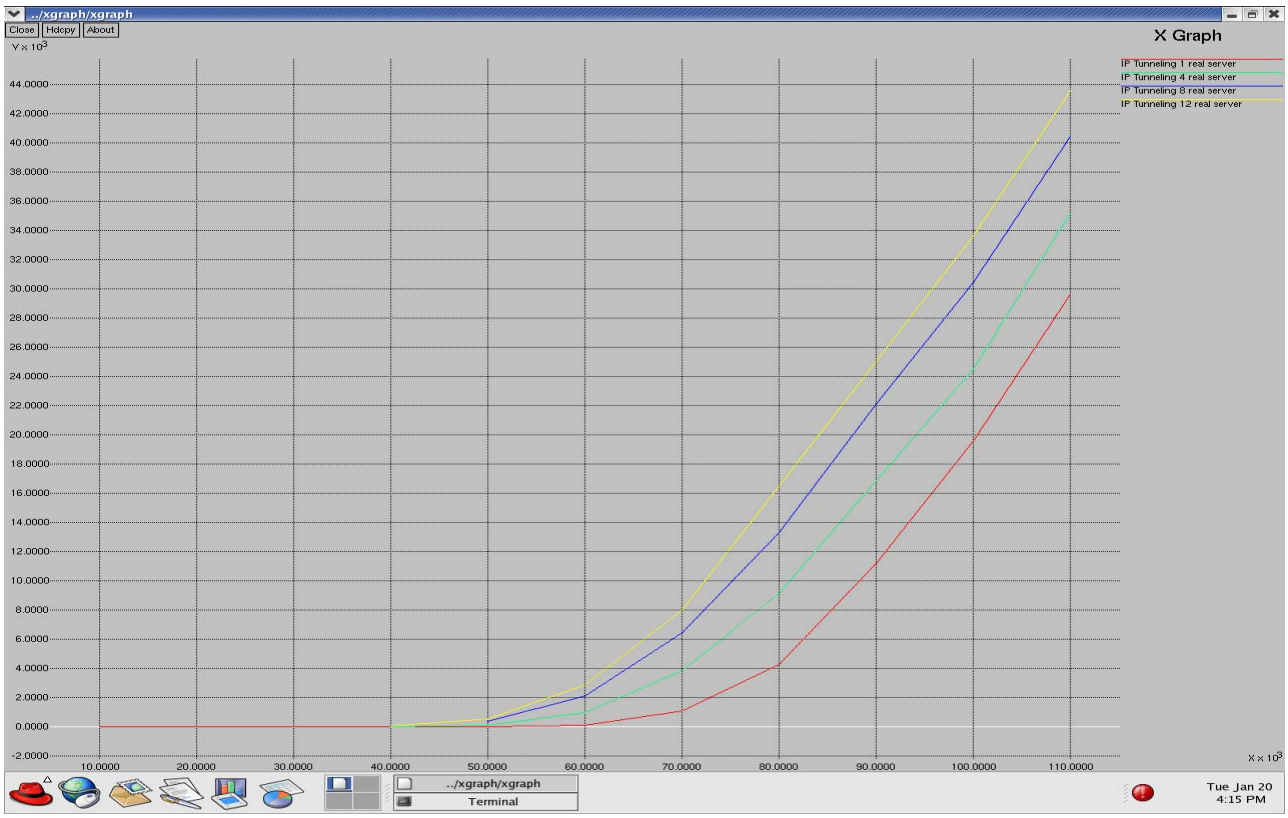

Figure 37 loss rate on the load balancer, IP Tunneling, topo1, 1000M

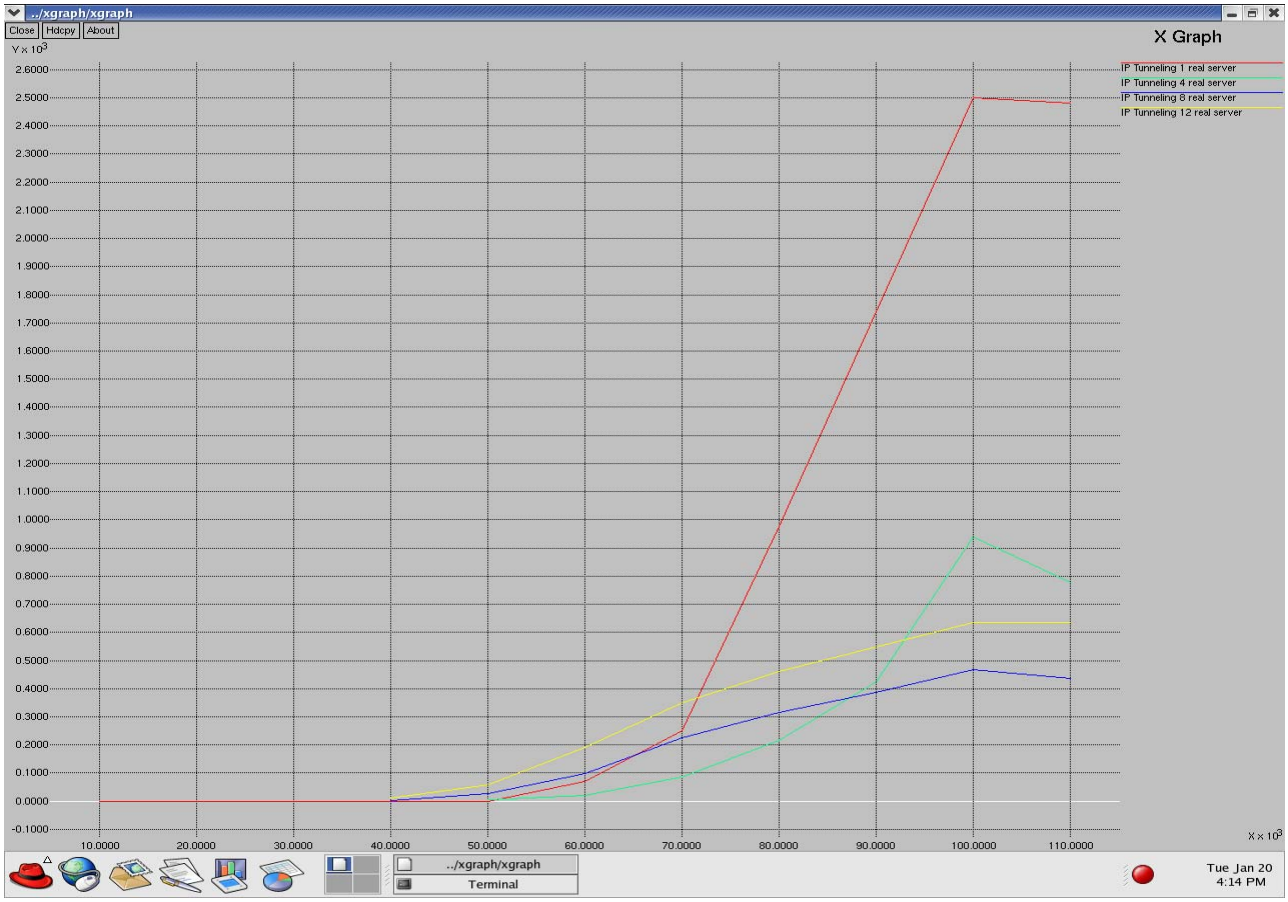

Figure 38 loss rate on the real server, IP Tunneling, topo1, 1000M

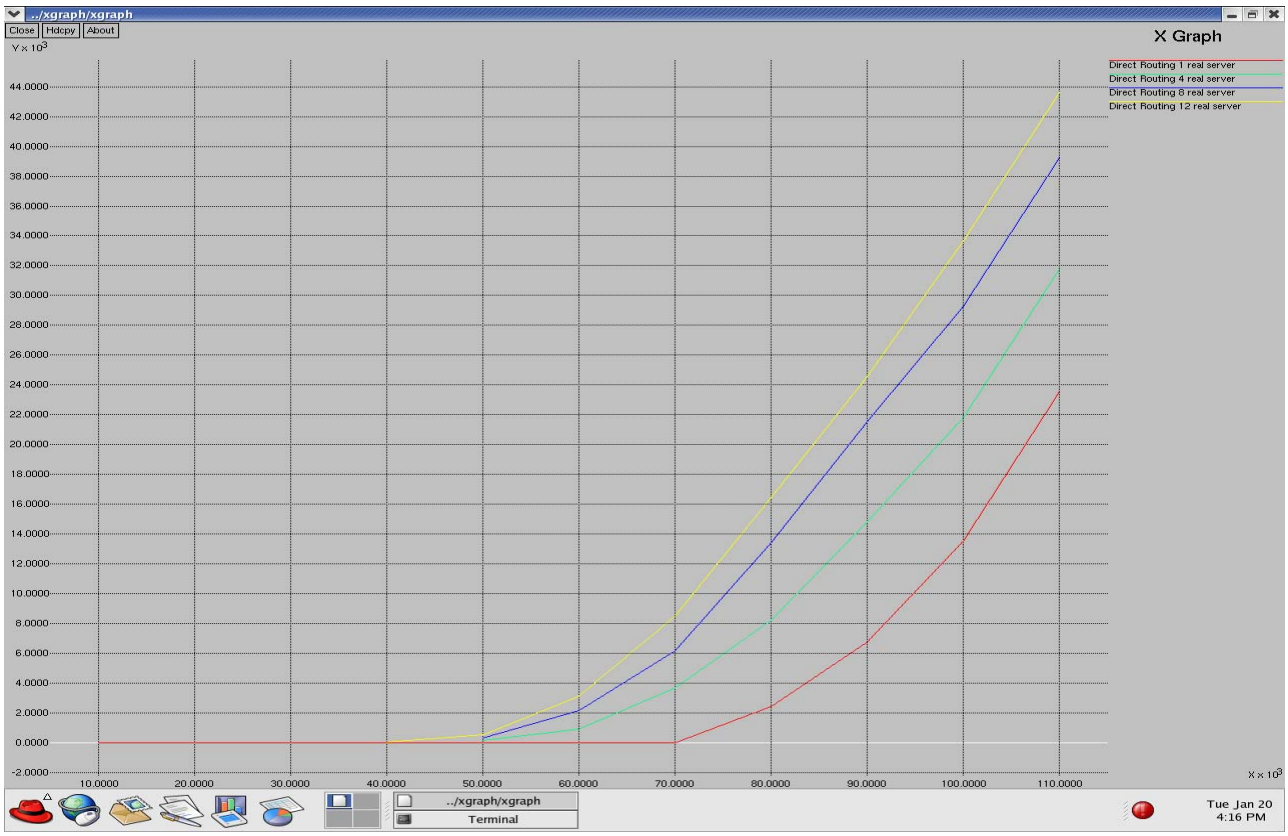

Figure 39 loss rate on the load balancer, Direct Routing, topo1, 1000M

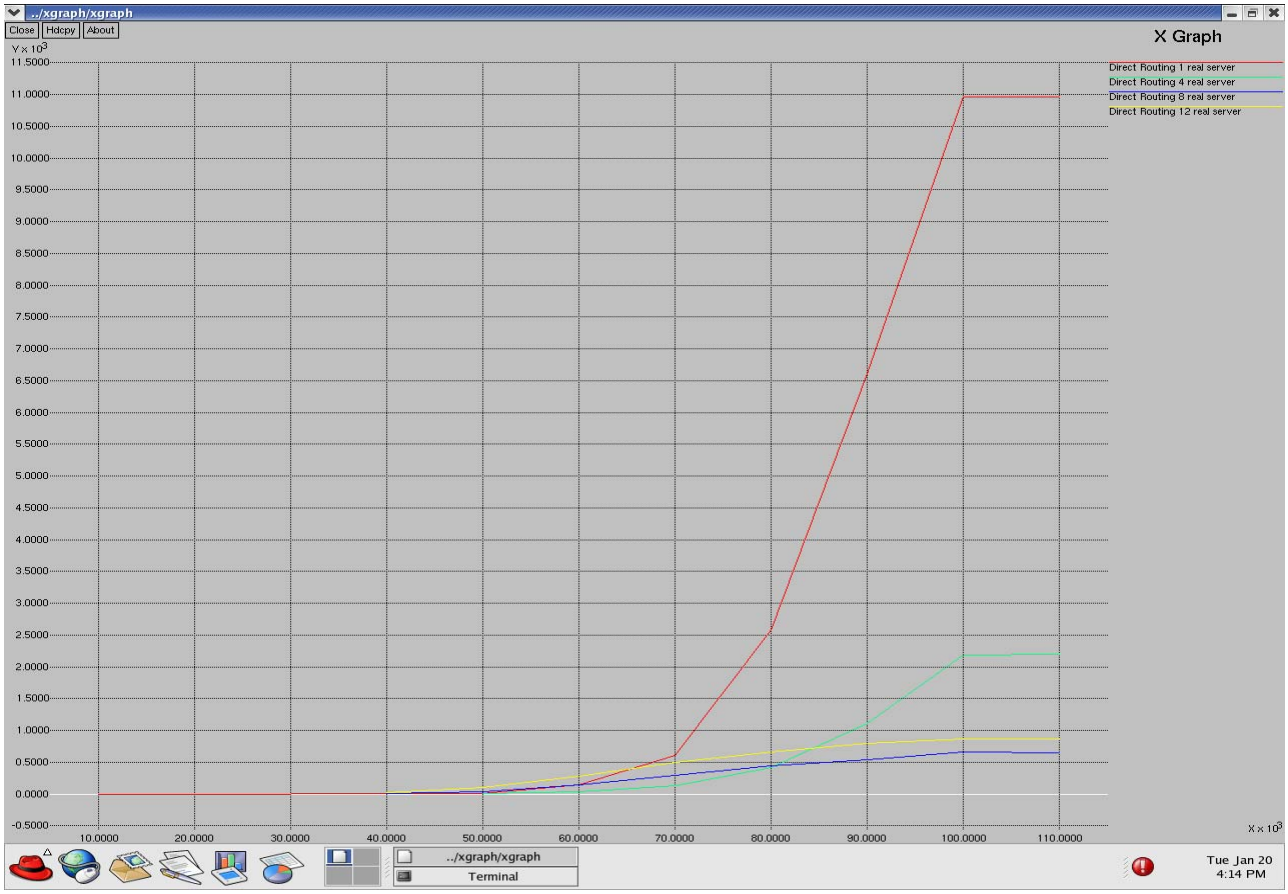

Figure 40 loss rate on the real server, Direct Routing, topo1, 1000M

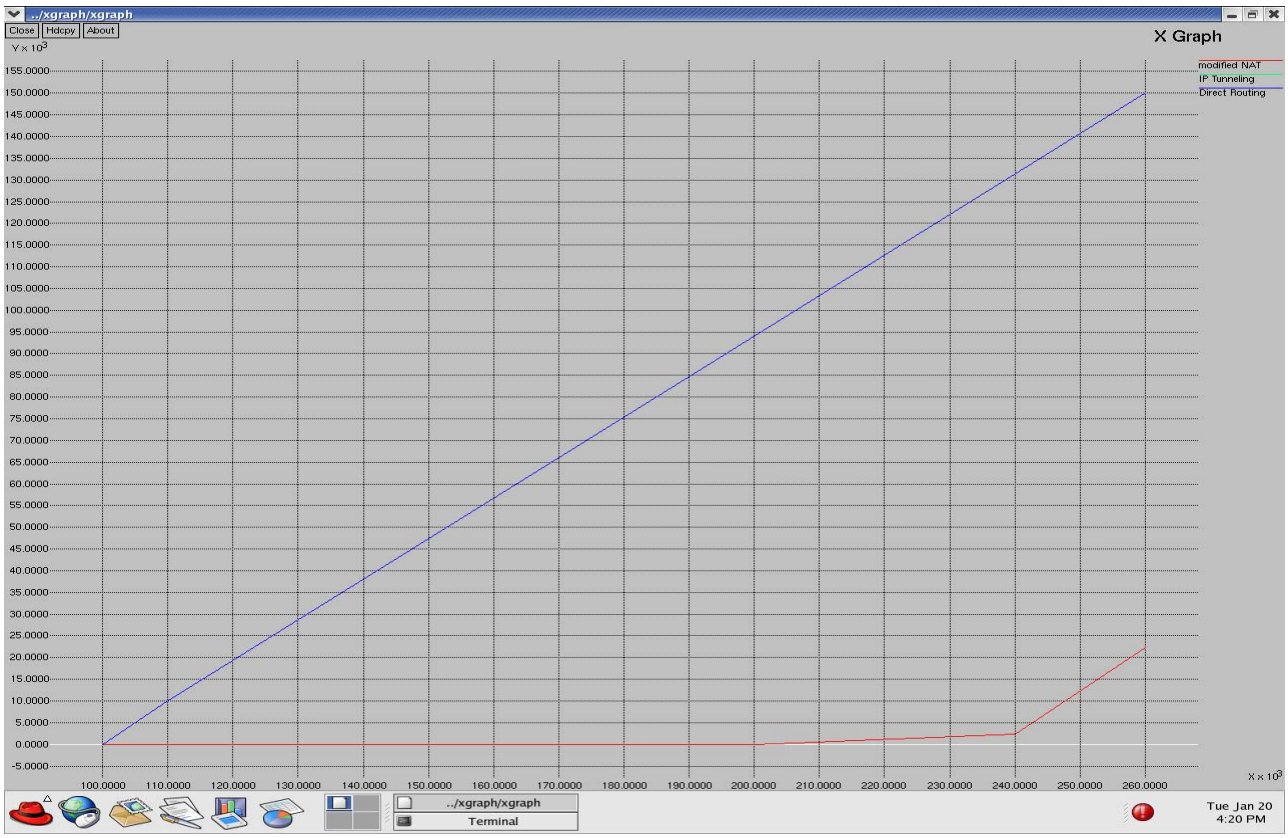

Figure 41 loss rate on the load balancer, topo2, 1000M

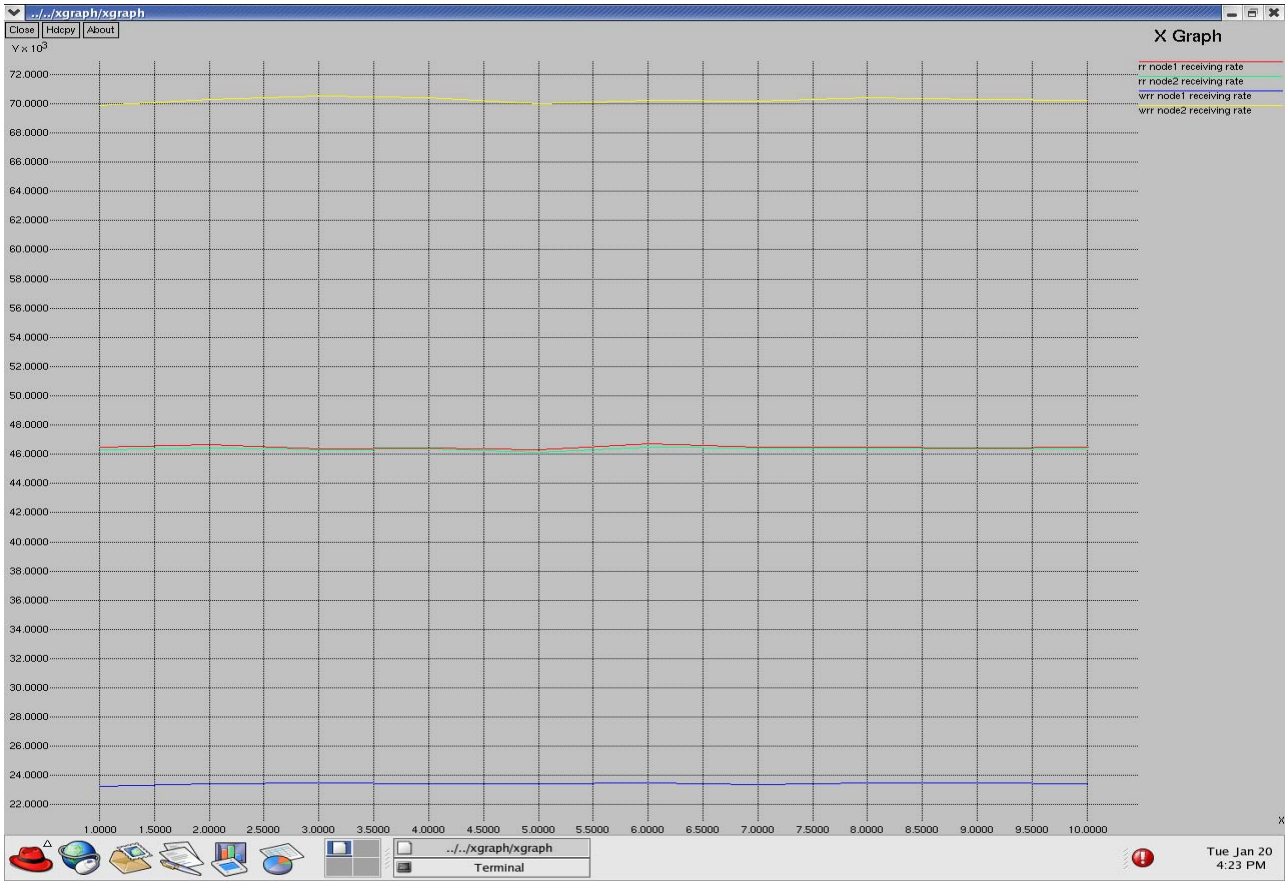

Figure 42 receiving rate on the real servers, RR vs. WRR, 1000M

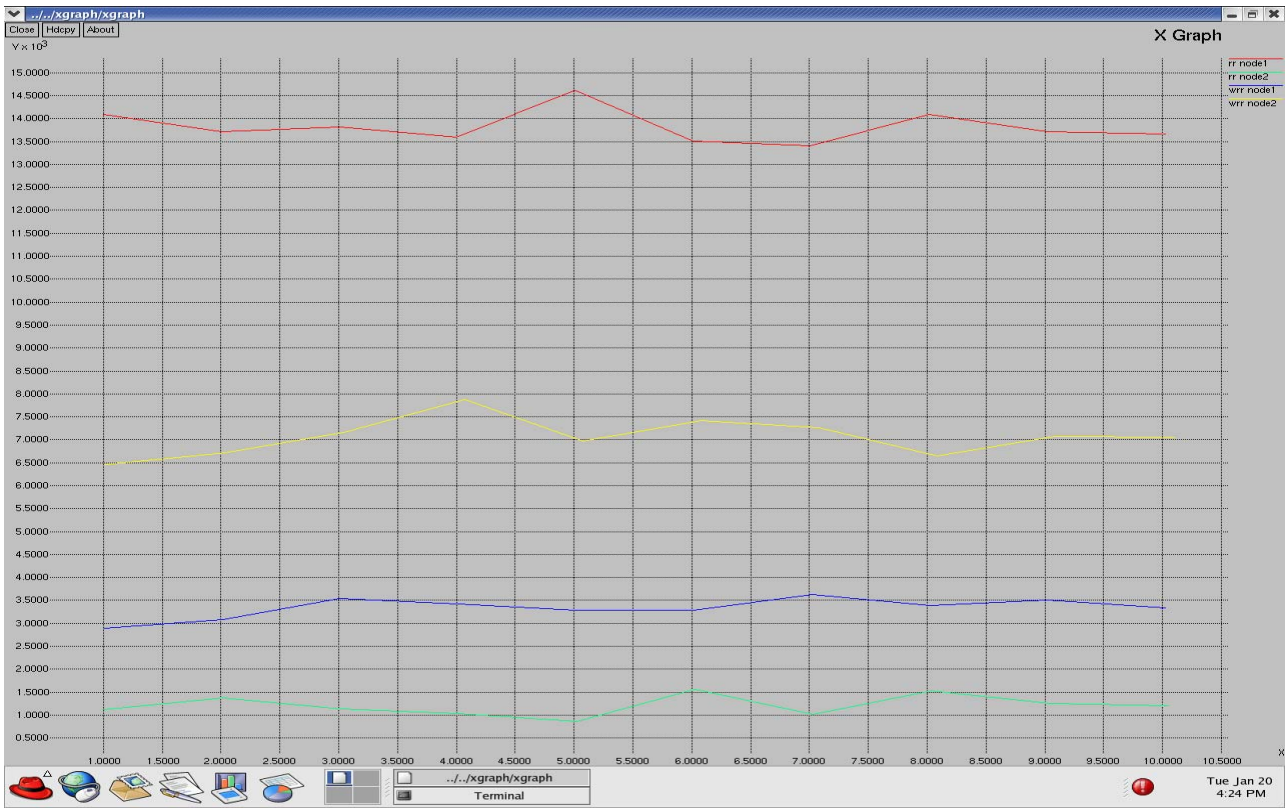

Figure 43 loss rate on the real servers, RR vs. WRR, 1000M

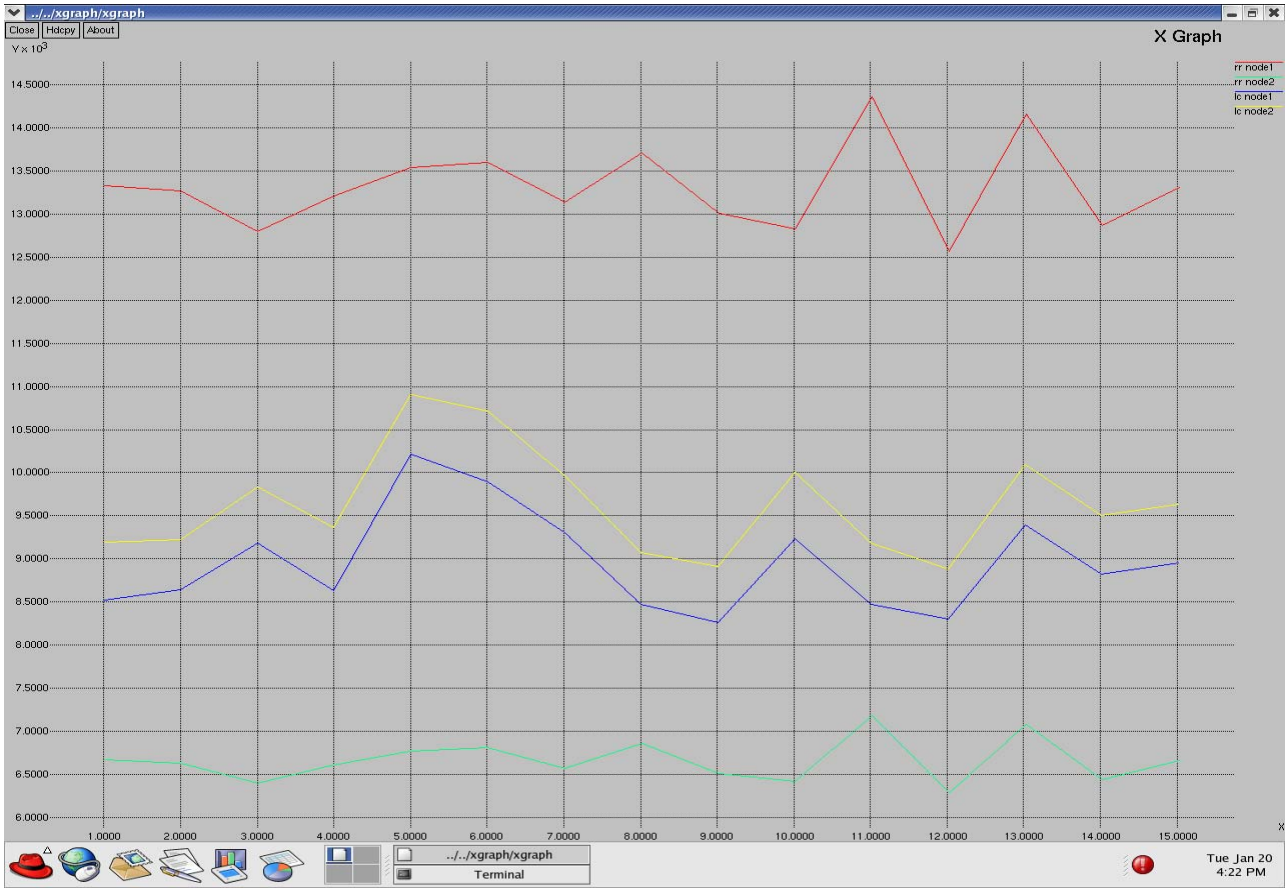

Figure 44 receiving rate on the real servers, RR vs. LC, 1000M

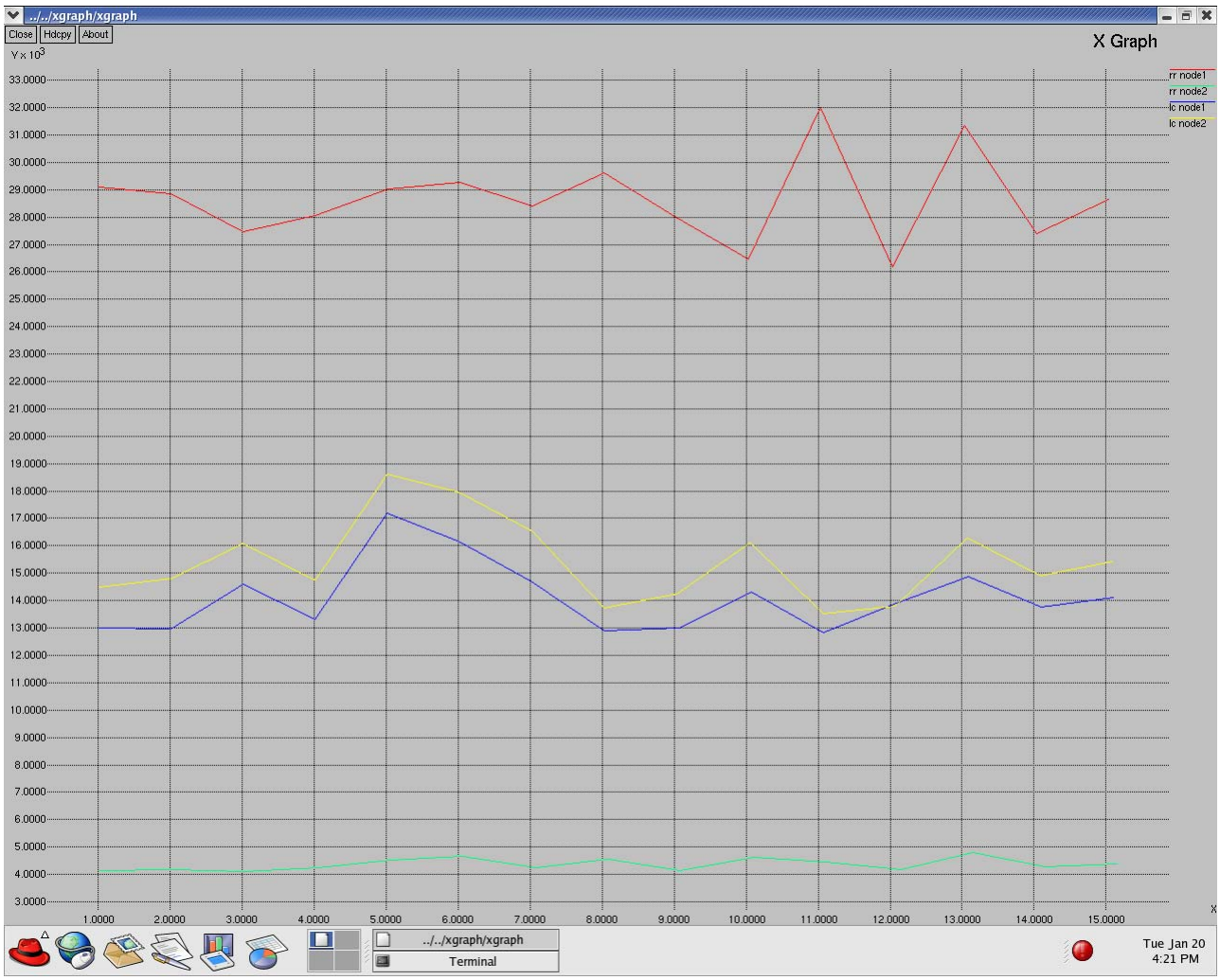

Figure 45 loss rate on the real servers, RR vs. LC, 1000M