A Survey on High Availability Mechanisms for IP Services

Narjess AYARI, Denis BARBARON, Laurent LEFEVRE, Pascale VICAT PRIMET
narjess.ayari@ens-lyon. fr, denis.barbaron@francetelecom.com, laurent.lefevre@ens-lyon. fr, Pascale.Primet@ens-lyon. fr

Abstract*. This work aims to make a survey on high availability mechanisms in both local and distributed cluster based architectures. We address both scalability and failover mechanisms for IP Network data applications. We investigate in comparing the major layer 4 and layer 5 level switching solutions as well the undergoing load balancing policies used to achieve a better distribution of the incoming requests.*
We raised issues for both stateless and staltefull proxy designs. These issues do concern essentially session's integrity and fair load distribution.
Key words and phrases*: High availability, scalability, load balancing, layer 4 switching, layer 5 switching, session integrity, failover, IP Services.*

1. Motivations

One of the most ever been challenging issues for all service architectures is high availability. Basic high availability addresses service non interruption by reducing the impact of failures thourough redundancy and failover techniques. But service downtimes can occur when a growing number of demands cannot be handled due to hardware or software limitations. Hardware and software scaling up is a mean to overcome such limitations.

Scalability can be implemented on cluster based architectures to improve the throughput as well as the processing speed within the highly available architecture. Moreover, when implemented on distributed architectures, it addresses site availability.

In both architectures, it is important to improve ressource utilization. Load balancing techniques aim to distribute efficiently and fairly inbound requests on the available ressources, by reducing the processor's idle time. Nonetheless, maximizing the throughput should keep reasonable response times.

Many load distribution policies have been proposed, ranging from static to dynamic algorithms. Dynamic policies are based on the load evaluation on each node of the cluster. The load evaluation takes into consideration the usage of the different node's ressources, like the CPU, the memory, etc. Some predictive schemes, basing their load evaluation on the server's response time, have also been proposed.

Basic comparison metrics used to evaluate these policies are all about measuring the CPU overhead introduced by the load evaluation and the accordingly best matching decision, vs. the reached throughput improvement while keeping reasonable response times. These comparisons are conducted regardless from the effectiveness of the load distribution operation on the upper layer traffic handled.

These upper layer flows of data can be load balanced over homogeneous or heterogeneous servers offering the same service processings. They can be classified in *signaling non elastic data*, and in *non signaling elastic data*.

Typical elastic flows include file transfers such as those in email and in the world wide web services. Typical signalling non elastic flows are used to control voice or video sessions established over different types of networks. These are different from elastic signaling protocols used within session oriented Internet protocols such as ftp or telnet, in that they are latency and jitter sensitive. A sample of signaling protocols used to establish and control real time streams over IP networks, is the Session Initiation Protocol [1].

The most widely used request distribution of these flows of data happens at the layer 4, because the TCP/IP headers contain the ultimate necessary information needed to identify a service.

Regardless from the distribution nature, which may be statefull or stateless, things would have done if each handled packet was processed at the upper layers as a single independent transaction. But this is not the case for most of the application level protocols. Thus, applying the dispatching policies blindly to the inbound packet's content may result in some session non integrity scenarios. Typical non integrity uses cases that we will discuss later, are relevant to routing SIP control messages in a cluster based VoIP architecture based on layer 4 switching schemes. They result in a retransmission behavior at the client side, as if the original messages were lost, leading to an additional latency that, typically, VoIP architectures should avoid to compete with the Public Switched Telephone Networks, known for their efficient routing.

This work addresses high availability in both local cluster based and distributed based architectures. We first address scalability and efficient load balancing within layer 4 and layer 5 switching mechanisms. Then we investigate in failover requirements and solutions for IP Network data applications.

2. Scalability requirements and solutions

The earliest efforts toward scaling architectures were done through hardware and software scale-up over a single node, when bottlenecks were observed. Hardware scale up is done by upgrading machines with larger and faster components, by incrementally adding ressources like CPUs, disks, memory or network cards. Software scale up is operating system and application parameter's tuning. A walk through the *proc* virtual filesystem or the use of the *sysctl* facility are means to tune kernel runtime parameters under Linux. But some of these upgrading scenarios need temporary service interruption and still fail under pick loads. Thus, scaling-out servers was introduced. Scaling-out achieved both service and site availability respectively thourough cluster and distributed based architectures [Figure 1].

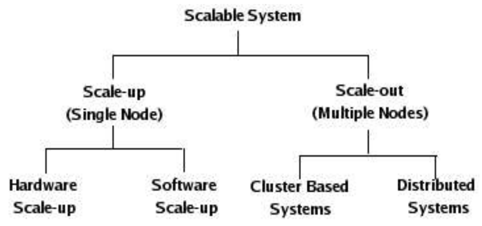

Figure 1 – Scalable Architectures

2.1 Efficient switching in cluster-based architectures

Different types of clusters exist, ranging from massive parallel processing clusters to clusters of individualy working nodes.

MPP clusters provide a scalable environment thourough solving heavy computational problems whose components can be worked in parallel. SMP [2] and NUMA [2], and their variations, provide larger clusters, respectively via CPU and memory bus interconnects. The effectiveness of scalability over these clusters depends on the job scheduling and migration operation effectiveness, as well as on fault detection and checkpointing.

Clusters of independent working nodes are based on commodity hardware and on general purpose kernels. They provide scalability through intelligent request switching to efficiently use cluster ressources.

Switching designs can be statefull or stateless, and applying to both layer 4 and layer 5 switching.

2.1.1 Stateless switching design

Stateless switching aims to achieve a better latency by processing each inbound PDU independently from its predecessors and regardless from any state information.

To achieve session integrity, a stateless design is based on a hash function, that computes the same destination cluster node for all PDUs issued from the same source IP address and port number, within the same transport protocol. Robust hash functions avoid collisions [3]. Among the main limitations of stateless switching is the fair load distribution. In fact, stateless switching is unable to state upper layer session average duration and does not implement any mechanism to prevent from forwarding PDUs belonging to the same long sessions to the same cluster nodes, resulting in an unfair load distribution. A typical sample of long sessions are SIP sessions, where the average duration is closely relevant to client's practice.

The second main limitation of stateless designs is node's fault handling. In fact, the hash function should be able to learn about a node's crash whenever it happens. This is very important to avoid datagram losses. In fact, when switching handles TCP based traffic, datagram losses cause the client sending rate to fall, due to the TCP AIMD processings. Besides, the hash function should be robust enough to avoid replaying all sessions when processing a node crash. In fact, suppose that datagrams originated from client c_i whithin the session s_i were forwarded to node d_i. If the hash function depends on the number of active nodes, it may forward datagrams c_i belonging to the session s_i to some other node d_j where $i \neq j$.

2.1.2 Statefull switching design

The main idea behind statefull switching is to address both upper layer session's integrity as well as fair distribution of inbound PDUs, by keeping in memory state informations. State informations are connection state or application layer session informations, depending on whether switching is done at layer 4 or 5.

Connection state informations include at least the source and destination IP address, the source and destination port number, and the underlying transport protocol.

For UDP based traffic, there is not a particular semantic to identify the beginning and the end of a UDP session, thus, upper layer protocol dependant header informations may also be kept for the safe of session integrity. Additional fields, such as multiple purpose timers, may also be kept as part of the state informations. They are used as an optimisation of the memory usage or as DDoS counter measures, by avoiding keeping state informations for inactive sessions. Timers can also be used to maintain statistics on the average of a client session's duration.

The load balancing decision is taken only for the first datagram within a connection or a session. All the subsequent packets need to be looked up within the state information structure. Thus, the performance of the statefull design depends on the lookup operation, which we can speed up using a hash function.

2.1.3 Layer 4 switching mechanisms

Layer 4 switching works at the TCP/IP level and performs a content blind switching of inbound packets. A broad classification of layer 4 switching mechanisms distinguishes between one way and two way architectures, where PDUs are respectively handled once and twice by the switching entity [Figure 2].

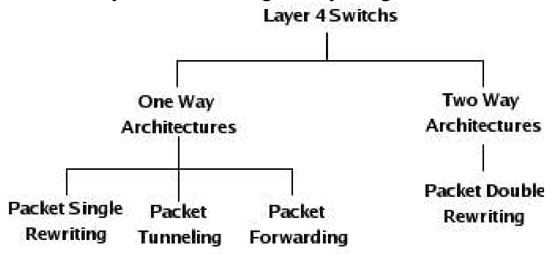

Figure 2 – Layer 4 switching taxonomy

2.1.3.1 Double packet rewriting

Double packet rewriting is based on Network Address Translation [4]. The layer 4 switch rewrites the source IP address and port number of each inbound packet to the virtual IP address of the cluster and to an arbitrary port number available on the switching entity. The destination IP address and the port number of each inbound packet, initially set to the VIP address of the cluster and the service port number, are rewritten to the IP address and port number of the choosen real node. The consequent checksum updates are applied to headers before delivering the packet to the outgoing interface. The reverse operations are performed on outbound PDUs. Thus, double packet rewriting becomes rapidly the bottleneck due to this double processing overhead.

2.1.3.2 Single packet rewriting

One way architectures use single packet rewriting to reduce the overhead of handling twice a datagram. The switch operations are simpler because changes are made only on the destination IP address and port number of the inbound packet, keeping the source identifiers of the connection unchanged. Real servers are then able to send back responses directly to the original clients after rewriting the source connection identifiers, set to theirs, to the VIP identifiers of the cluster.

2.1.3.3 Packet tunneling

IP tunneling consists of encapsulating IP packets within IP packets, allowing packets destinated to one IP address to be wrapped and redirected to another IP address [5]. The old header and data are the new IP packet's payload.
Tunnel switches wrap inbound packets into IP packets where the source and destination addresses are respectively the VIP and the real server's IP address. The receiving side would strip the IP packet's header off and check whether the encapsulated packet has as destination IP address the VIP address already configured on its tunnel device. If so, it processes the request and sends back the response directly to the client.

2.1.3.4 Packet forwarding

Packet forwarding can be performed only within a LAN. The nodes of the cluster must share the same VIP through aliasing on their secondary IP addresses. In order to ensure that all inbound frames are received by the switching entity, the ARP protocol must be disabled on the cluster's nodes. When a cluster node is choosen to process an inbound packet, the corresponding frame is generated such that the destination MAC address is the MAC address of the target real server. Then the frame is transmitting on the wire. Thus, the processing node have all the needed informations to send back responses directly to clients.

2.1.4 Kernel implementations of layer 4 switching

The IP Virtual Server [6] implemented the major layer 4 switching techniques as addon modules in the networking layer of the linux kernel. IPVS is a collection of kernel patches that turns an IPVS enabled node into a layer 4 load balancer. It implements *NAT*, *Direct Routing,* and *Tunneling,* in conjunction with load balancing policies. Its implementation is based on both the linux packet filtering and on the linux kernel routing capabilities.
The Linux Virtual Server [7] is a cluster of independantly working nodes, using the IPVS load balancer to handle inbound and outbound traffic.

2.1.4.1 The Netfilter capabilities

Netfilter [8] is a packet filtering engine in the Linux kernel from version 2.4, that works outside the Berkley socket interface. It provides extensible NAT and packet mangling facilities and allows filtered or modified packets, by userspace processes, to be reinserted back again into the kernel. These capabilities are implemented by including additional sanity checks within the kernel routing processings. The Netfilter architecture [Figure 3] defines 5 hooks for IPv4 [Table 1].

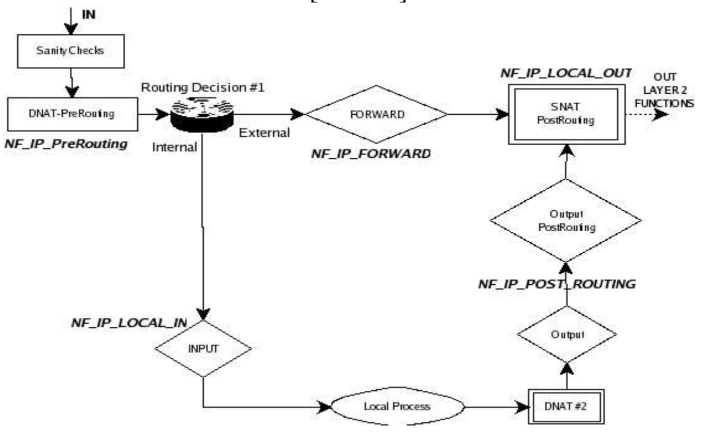

Figure 3 – The Linux Netfilter architecture

Hook	Use
NF_IP_PRE_ROUTING	After sanity checks, before routing decisions.
NF_IP_LOCAL_IN	After routing decisions if packet is for this host.
NF_IP_FORWARD	If the packet is destined for another interface.
NF_IP_LOCAL_OUT	For packets coming from local processes on their way out.
NF_IP_POST_ROUTING	Just before outbound packets "hit the wire".

Table 1 - Available IPv4 hooks

Any kernel module can register a callback function that will be called every time a packet traverses the corresponding hook. A kernel module can then lookup, manipulate any packet before it continues its path down the routing chain. Then, it must returns one of the predefined Netfilter return codes [Table 2].

Return Code	Meaning
NF_DROP	Discard the packet
NF_ACCEPT	Keep the packet
NF_STOLEN	Forget about the packet
NF_QUEUE	Queue packet for userspace
NF_REPEAT	Call this hook function again

Table 2 - Netfilter return codes

2.1.4.2 The IPVS architecture

IPVS is based on only three of the five Netfilter hooks. They are the *PRE_ROUTING,* the *LOCAL_IN* and the *POST_ROUTING* hooks [Figure 4]. The routing rules inside the kernel send all incoming packets to the *LOCAL_IN* hook, where the registered IPVS callback function checks that the packet holds a request to a virtual service and causes the packet continues its path down the *POST_ROUTING* hook. The corresponding registered callback function would then perform the alteration of the packet according to the choosen layer 4 switching mechanism.

The LVS Director can work statefully to ensure level 4 integrity. By maintaining a connection tracking table, IPVS ensures that all packets related to an already established connection are sent to the same node. Performance measures have been conducted [9] and showed that LVS-DR performs better than LVS-NAT.

2.1.4.3 Persistancy design

Persistancy ensures that connection requests originated from a client go to the same real server. It can be achieved using the Netfilter marked packets [Figure 4].

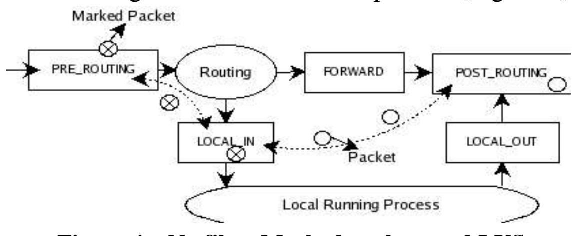

Figure 4 – Netfilter Marked packets and LVS

When the packet hits the PRE_ROUTING hook, the Netfilter mark number is placed inside the sk_buff structure. Then, the packet completes with the routing process and is delivered to the LOCAL_IN hook, where the corresponding registered callback function uses the Netfilter mark to determine which IPVS to use.

2.1.4.4 Load balancing policies

Layer 4 routing mechanisms smooth out peack loads using static and dynamic content unaware request distribution policies. Static policies do not need server's load information. They include the Random, the Round Robin, the Weighted RR, and the source hashing partitionning policies. Their use within layer 4 switching results in a rapid disptaching, since they do not rely on any external entity to make a decision. However, their stateless nature may result in poor assignments as discussed in section 2.1.1. Some other issues remain open and are close to the signaling protocol handled by the switching entity. As a representative example, let's assume a SIP session where two parties, a caller and a callee, are made able to exchange voice streams over the unreliable IP network [Figure 5].

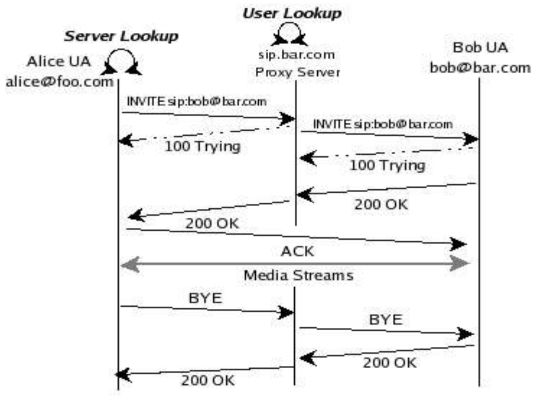

Figure 5 – A simple SIP call flow

Since a SIP session is based on different SIP transactions, it may involve different transport connections, that would be handled differently by the LVSDirector. Besides, the random nature of clients make it impossible to get help thourough persistancy. Thus, routing SIP messages based on the layer 4 switching could result in non integrity, when for example, the shutting down message happens to be sent to the wrong SIP proxy server during the session.

Hence, to prevent upper layer non integrity, the use of application level information may be advantageous, because additional informations, such as the session identifier, may be used within the statefull switch design to achieve a content aware distribution of the requests.

2.1.5 Layer 5 switching mechanisms

It is also called delayed binding because of the additional overhead due to analysing the upper layer headers and to the context switchings between kernel and user spaces. One way and two way layer 5 switch designs exist, where packets are handled respectively once and twice by the switching entity.

2.1.5.1 TCP Gateway

An application level proxy running on the layer 5 switch mediates the communication between the client and the server by making *separate* TCP connections to client and server [Figure 7].

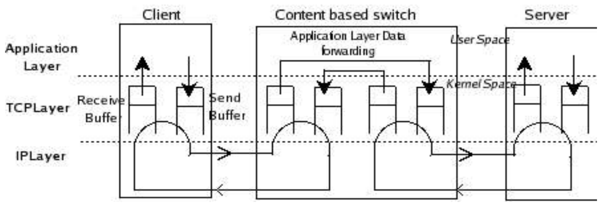

Figure 7 – Feedback loop of a TCP Gateway connection

2.1.5.2 TCP Splicing

TCP splicing is a two way mechanism [10] that reduces TCP Gateway overhead by splicing the client and the server side sockets [Figure 8], by mapping sequence number space of client-switch packets to that of switch-server packets. Changes also affect IP header fields as well as socket options. TCP Splicing has also been implemented on hardware-based switches [11].

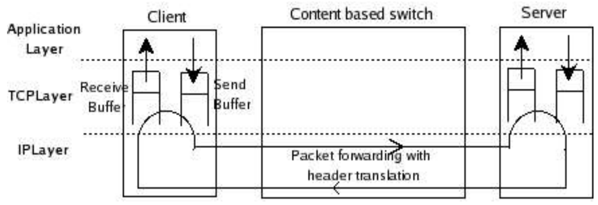

Figure 8 – Feedback loop of a spliced TCP connection

2.1.5.3 TCP connection hop

It is a one way proprietary solution proposed by [12] that handles session persistancy. Once the client side connection is established with the switch, the latter encapsulates the IP packets in an RPX packet and send it to the server.

2.1.5.4 TCP Handoff

TCP Handing off [13] is a one way mechanism. It requires that the receiving host supports the handoff protocol. It is implemented by setting up a new TCP connection with sequence number specified by the proxy in a TCP option during handshaking.

2.1.5.5 Load balancing policies

Server state aware policies provide a better request assignment. Indeed, special care have to be done to avoid that load updates become the bottleneck. Typical server state informations include the CPU, the memory, the disk and I/O storage utilization, as well as the number of instantaneous active connections and processes. Predictive policies use an approximate value of the amount of time needed to complete the request processing. Typical dynamic policies are the least connection and weighted least connection scheduling, the shortest expected delay scheduling, the minimum miss scheduling, and so on. None of the policies above does prevent from overloading cluster nodes. Indeed, admission control policies [14] do this by scheduling requests to nodes with an utilization value under a given threshold. They include locality based least connection and locality based least connection with replication scheduling.

3. Failover and Scalability through redundancy

3.1 Site availability

Site availability was addressed in the DNS ressource records, the Reliable Server Pooling, the Multicast and the Unicast schemes. It was also used to achieve failover in distributed architectures.

3.1.1 Using the DNS Ressource Records

DNS RR resolves machine's names into their IP addresses. DNS SRV RR is used in service location, such as localizing available SIP proxies [Figure 9][15]. DNS request assignments use the round robin policy as well as the server priority handler, and achieve scalability through site redundancy. Indeed, the effectiveness of DNS based scalability and failover are corrupted by the DNS cache updates frequency.

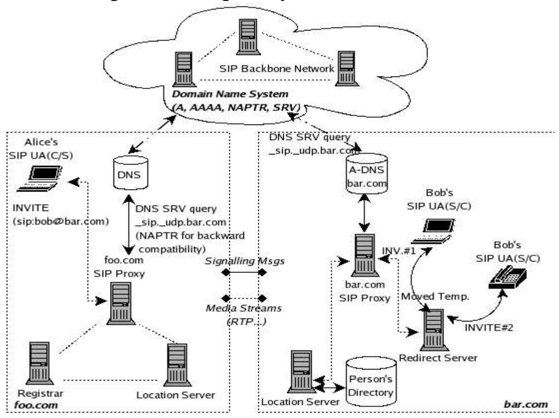

Figure 9 – SIP Session establishment using DNS SRV

3.1.2 Using the RSerPool architecture

The reliable server pool architecture [Figure 10][16] is a framework for load sharing and fault tolerance in small domains rather than in the Internet.

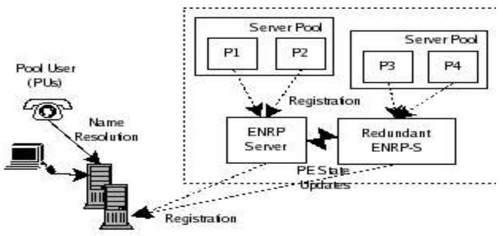

Figure 9 – RSerPool architecture

Pool Entities are a set of one or more servers providing the same application, each identified by a pool handle. Each PE registers with a specific ENRP server, called the home Endpoint haNdlespace Redundancy Protocol Server that will supervises its state. PEs can provide load informations to the ENRP Server. The latter is redundant. ENRP servers exchange informations about the SPs, such as the addition of a PE, the status change of an existing PE, etc. Besides, an ENRP server monitors the health of its peers, and takes over the responsibility of being the home ENRP server for a set of PEs when the previous corresponding ENRP server has failed.

3.1.3 Using the Multicast

In some publications about fault tolerance [17], multicast is used for communication between redundant entities. Unfortunately, multicast needs explicit support of all routers in the path between the communicating hosts. Thus, its use is restricted to LANs or specifically prepared networks.

3.1.4 Using IP Anycast

An anycast address is an IP address that may be bound to one or more network endpoints [18]. Different servers that are providing the same service can all have the same anycast address on one of their interfaces. If a server fails, some routers will update their tables to route packets to the nearest remaining server with the same IP address.

3.2 Failover in cluster based architectures

Failure detection through keepalive mechanisms and the corresponding failover operations make redundant nodes, subsystem nodes, and applications highly available. Channel bonding, heartbeat messages exchange, transport level connection establishments, application level messages exchange, hardware watchdog timer are some of the failover mechanisms.

3.2.1 IP address takeover using channel bonding

Channel bonding consists into aggregating multiple network interfaces at the link layer, giving to the networking protocol stacks the illusion of a single interface. This technique is usually implemented in Unix servers and networking hardware like ethernet switches and IP routers. The same concept is called Etherchannel by Cisco, and Trunking by Sun.

3.2.2 The Linux watchdog timer interface

A watchdog timer is a hardware circuit that can reset the computer system in case of a software fault. A userspace daemon notifies the kernel watchdog driver via the *dev/watchdog* special device file that userspace process is still alive, at regular intervals. If userspace fails, due to a RAM error, a kernel bug, etc, the driver will inform the hardware watchdog about the failure and the hardware watchdog will reset the system, causing a reboot after the timeout occurs. This solution does cause service interruption for a period of time.

3.2.3 Linux Heartbeat subsystem

A heartbeat subsystem monitors the presence of nodes in the cluster through a series of status messages that can be broadcast, unicast or multicast, which length is not more than 150 byte. Three types of heartbeat messages are handled by heartbeat daemons. They are status messages, cluster transition messages and retransmission messages. Status messages use sequence numbers to ensure that packets are not dropped or corrupted. Retransmission messages does not include sequence numbers to avoid flooding the network with control messages. Cluster transition messages are actions against a cluster member entry or left within the cluster, handled by an event trigger within the cluster management software.

4. Conclusion

In this work, we addressed both sclability and failover to build high available architectures. We raised the importance of layer 4 and 5 server state persistency, as well as the effectiveness of synchronisation schemes to achieve application integrity and to avoid long time service interruption.

REFERENCES

[1] J. Rosenberg, H. Schulzrinne, U. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler, "RFC3261-SIP", June 2002.

[2] David E. Culler, J. P. Singh, A. Gupta, "Parallel Computer Architecture: A Hardware/Software Approach", 2001.

[3] Hashing for Dynamic and Static Internal Tables, Lewis and Cook. *IEEE Computer*, October, 1988.

[4] K. Egevang, P. Francis, "RFC 1631 - The IP Network Address Translator (NAT)", Network Working Group, May 1994.

[5] R. Housley, S. Hollenbeck, "RFC 3378 - EtherIP: Tunneling Ethernet Frames in IP Datagrams", Network Working Group, September 2002.

[6] K. Kopper, "The Linux Entreprise Cluster", NoStrach press, 2005.

[7] The Linux Virtual Server Project, [http://www.linuxvirtualserver.org/].

[8] T. Eastep. "Netfilter Overview", March 2004. [http://www.shorewall.net/NetfilterOverview.html].

[9] P. O'Rourke, M. Keefe, "Performance evaluation of Linux Virtual Server", April 2001.

[10] D. Maltz, P. Bhagwat, "TCP Splicing for application Layer proxy performance", March 1998.

[11] Nortel Switch, [http://www.nortel.com/].

[12] Resonate Inc., "TCP Connection Hop", April 2001.

[13] R. Kokkuz, R. Rajamonyy, L. Alvisiz, H. Vinz, "Half pipe Anchoring: An Efficient Technique for Multiple Connection Handoff", IEEE ICNP 2002.

[14] C. H. Chen, "An Admission Control and Load Balancing Mechanism for Web Cluster Systems", China University, 2003.

[15] J. Rosenberg, H. Schulzrinne, "RFC 3263 - Session Initiation Protocol (SIP): Locating SIP Servers", Network Working Group, June 2002.

[16] M. Tuexen, R. Stewart, M. Shore, L. Ong, J. Loughney, M. Stillman, "RFC 3237 - Requirements for Reliable Server Pooling", Network Working Group, January 2002.

[17] V. Pappas, B. Zhang, "Fault-Tolerant Data Delivery for Multicast Overlay Networks", 2004.

[18] Anycast R. Engel, V. Peris and D. Saha, "Using IP Anycast for load distribution and server location", 2001.