Session awareness issues for next generation cluster based network load balancing frameworks

Narjess Ayari, Denis Barbaron
France Telecom R&D – Lannion, France
{narjess.ayari,denis.barbaron}@orange-ftgroup.com

Laurent Lefèvre, Pascale Primet
INRIA / LIP (UMR CNRS, ENS, INRIA, UCB), France
{laurent.lefèvre,pascale.primet}@ens-lyon.fr

Abstract

While a lot of researches focused on how to efficiently spread the offered network load on the available cluster resources, less interest has been granted to the impact of the used mechanisms on the reliable execution of the upper layer services. On the other hand, emerging NGN services as well as some of the already familiar services involve multiple flows during the lifespan of a single end-to-end session, hence, raising the challenge of session awareness while processing the incoming network traffic.

In this paper, we grasp the need for fine grained session awareness to efficiently allocate the cluster resources to the offered network traffic. The analysis of load balancing scenarios of some representative IP services provides us with solid reasons to use deep packet inspection to achieve fine grained network traffic load distribution, and to meet NAT and firewall traversal constraints as well.

1. Introduction

IP Service scalability is a key issue that has been addressed through scale-up and through scale-out techniques. While the scaling-up approach meets the increasing load constraint in the short run, it does, however, meet neither the transparency nor the availability and the scalability requirements for the long run. In fact, upgrading a server with more processing power may require service interruption while it doesn't prevent against the need for more processing resources in the long run. Thus, this approach has been rapidly replaced by cluster based architectures.
A cluster consists of a set of networked off-the-shelf servers offering a single system image while providing additional processing capabilities. Cluster based architectures take advantage of resource redundancy to meet both scalability and high availability requirements. First, efficient load balancing algorithms are considered to achieve an improved throughput and end-to-end delay under regular load. Second, fault recovery models are invested to provide highly available execution environments, which enable to recover from the failure of a legitimate cluster resource on an available replica.

Emerging NGN services as well as some of the already existing services involve multiple flows during the lifespan of a single end-to-end session. In this work, we grasp the need for session awareness to efficiently allocate the cluster resources to the offered network traffic and to provide a reliable execution of the handled IP services.

The remainder of this article is organized as the following. In section 2, we provide an overview of the network load balancing mechanisms. We cover both of the flow level and application level routing mechanisms. In section 3, we analyze a set of load balancing scenarios for some representative IP services, including bulk file transfer using FTP, and emerging NGN services such as multimedia streaming using RTSP/RTP/RTCP, and Voice over IP using SIP. In section 4, we deal with the NAT and firewall traversal issues. Finally, we conclude summarizing the requirements of the next generation cluster based load balancing frameworks.

2. Overview of the network traffic load balancing

Different approaches have been proposed to allocate the available cluster resources to the offered network traffic. These approaches can be divided into (*i*) client based, (*ii*) server based, and (*iii*) dispatcher based approaches. This work focuses on the dispatcher based mechanisms, where a central load balancing engine operating at the entry point to the cluster allocates its available resources to the offered network traffic.

Network traffic load balancing engines can be classified according to the TCP/IP layer at which they are operating to route the offered network traffic to an available server within the cluster. On the other hand, network traffic load dispatchers can be designed as stateful or as stateless engines.

In the following we'll compare stateful and stateless designs and we'll cover both of the transport level and the application level load balancing frameworks.

2.1. Statefull vs. stateless load balancing engines

A stateless load balancing engine forwards each datagram regardless to its predecessors. Hence, it doesn't need to maintain any flow, session or server state in its memory. Instead, flow or session integrity is achieved by using a hash function which assigns all the datagrams pertaining to the same flow or session, to the same processing server in the cluster. For this reason, the choice of the hash function is crucial to achieve a minimal latency while avoiding collisions.
Since a stateless design is unaware of the cluster servers states, it doesn't prevent neither against datagram loss in case of a node failure, nor against flow or session replay in case of a node addition. Moreover, it fails in achieving an optimal system throughput under normal load, as it is unaware of the resource requirements of the incoming IP sessions and has no means to achieve fairness in distributing the offered network traffic among the available cluster resources.

A stateful load balancing engine provides however the framework to fulfil efficient and session aware distribution of the offered network load on the available cluster resources. This can be achieved by maintaining an in-memory mapping between the incoming flows and the available servers within the cluster.

2.2. Network traffic load balancing

Network traffic load balancing frameworks fall into one-way and into two-way architectures. With two-way architectures, the full duplex network traffic flowing between the clients and the cluster servers crosses the entry point to the cluster. This means that the resources of the load balancing engine, as well as the cluster network bandwidth, will be kept busy with the processing of the outgoing traffic, even though this traffic does not necessarily need to be involved in the load balancing procedure.
Since the outgoing traffic is typically more important than the incoming traffic, two-way architecture based clusters suffer a limited scalability and support less simultaneous sessions compared to the one-way based architectures. Hence, we believe that two-way architectures are inappropriate for the distribution of IP sessions involving heavy size outgoing traffic, such as multimedia sessions.
On the other hand, one-way architectures provide the framework for a better usage of the processing capabilities inside, as well as at the entry point to the cluster (CPU, memory, etc.). In fact, they are designed such that the outgoing network traffic is directly sent from the cluster nodes to the clients, bypassing the cluster head. Thus, the cluster throughput, as well as its scaling factor, are no longer limited by the processing capacity of the entry point to the cluster.

2.3. The transport level network load balancing mechanisms

Transport level network load balancing mechanisms aim to provide flow level integrity while distributing the offered network traffic on the cluster resources. The basic idea is to prevent datagram losses, and consequent end-to-end increasing delay, by assigning all the datagrams pertaining to the same flow to the same processing server inside the cluster. Hence, an incoming packet is routed to a backend server in three steps. First, the packet is captured. Then, its flow is looked up and its state is created or updated. Finally, the packet is forwarded to the right processing server inside the cluster.

Transport level network load balancing mechanisms can follow the one-way approach or the two-way approach. The two-way approach includes the packet double rewriting mechanism. The one-way approach includes the packet forwarding mechanism and the packet tunnelling mechanism.

2.3.1. The packet double rewriting mechanism. The packet double rewriting mechanism [1] is a NAT-friendly scheme, according to which the cluster is reached through the public virtual IP address of its head node. Every incoming packet is rewritten such that its source flow identifiers correspond to the cluster head node identifiers, and such that its destination flow identifiers correspond to the processing node identifiers. The outgoing packets are also rewritten accordingly.

While being simple, this approach assumes that the cluster head and the cluster processing nodes belong to the same private network. Moreover, due to the double processing overhead, this mechanism allows only a bounded maximum number of nodes to be deployed inside the cluster. Hence, it suffers from performance limitations in terms of non-optimal throughput and reduced scalability.

2.3.2. The packet forwarding mechanism. The basic idea here [1] is to build the cluster such that its head and its processing nodes share the same public virtual IP address. Each incoming packet is then forwarded to the right processing server inside the cluster by performing a link layer header rewriting. Hence, the link layer destination address of each incoming packet is rewritten to the corresponding processing server link layer address. Since the head and the cluster nodes share the same public IP address, there is no need to modify the TCP/IP headers, and the outgoing traffic will be directly sent to the clients.

This technique applies to nodes clustered within a LAN and requires disabling the ARP modules at the processing nodes. However, it assumes that the dispatcher statically maintains in its ARP table the appropriate link layer address of each cluster node.

2.3.3. The packet tunnelling mechanism. This approach [1,2] proposes to encapsulate each offered packet to the cluster in an IP packet which destination address corresponds to the IP address of the chosen processing server. At the receiver side, the processing server extracts the original packet and learns that its earlier destination IP address corresponds to its tunnel network level address. Hence, it processes the request and sends back the response directly to the client.

While being more scalable, IP tunnelling incurs more processing overhead at both of the dispatcher and the processing server sides and assumes that both sides are IP tunnelling aware.

2.4. The application level network load balancing mechanisms

Application level network load balancing techniques aim to forward the datagrams belonging to the same application level session to the same processing node. The basic idea is to identify all the data pertaining to the same client session by inspecting the application level data of the traffic flowing between the clients and the servers. This is essential to enable the association of the data pertaining to a given session to the same processing node inside the cluster. Hence, the offered network traffic is routed in almost four steps. First, in case the incoming session involves a connection oriented signalling flow, a connection between the client and the load dispatcher is first established. Then, the data is buffered and its application level content is inspected. Third, the session states are created or updated. Finally, an appropriate processing node is chosen, and the data is sent to that node over a connection established between the dispatcher and that node.

Content inspection is achieved by parsing the application layer data using a specific application protocol parser. Typical parsers are implemented in the user-space, hence, session awareness through deep packet inspection (DPI) incurs an additional latency compared to the transport level routing mechanisms. This overhead is due to the several user-space to kernel-space context switching, consequent to calling the several parsing functions, as well as to the data copies needed between the client socket, the dispatcher socket and the processing node socket.

When the involved flows are TCP based, TCP splicing and TCP handoff mechanisms have been provided to reduce this overhead.

Application level routing techniques can follow the one-way approach or the two-way approach. The two-way approach includes the TCP Gateway and the TCP splicing variants. The one-way approach includes the TCP handoff variants.

2.4.1. TCP Gateway. The TCP gateway approach is the standard mechanism, where the entry point to the cluster operates as a proxy. It establishes connexions with the clients on the one hand, and with the processing nodes on the other hand. Then, it relays the data sent by the clients on the appropriate server side sockets.

2.4.2. TCP splicing variants. The TCP splicing approach [3] is a NAT-friendly mechanism which reduces the number of user-space to kernel-space context switches by splicing the client side and the server side connexions [Figure 1].

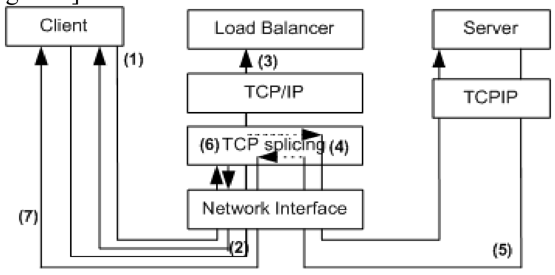

Figure 1. TCP splicing based routing

Hence, once the data flowing on the client side connection is inspected (1,2), the appropriate server is chosen and the server side connection is established (3). Finally, a splicing kernel module is involved in rewriting the IP and the TCP headers of both of the incoming and the outgoing traffic (4,6). The header fields concerned by the splicing operation include the source and the destination addresses and port numbers, the TCP sequence and acknowledgment numbers, as well as the corresponding checksum values.

In order to reduce the end-to-end delay, fully pre-splicing [4] proposes to pre-establish a pool of TCP connexions between the dispatcher and the processing nodes and to keep these server side connections alive for further splicing processes.

Connection binding [5] improves fully pre-splicing by un-splicing the server side connections and by keeping them alive after the corresponding sessions terminate. The dispatcher splices any new incoming client side connection with the fastest opened server side connexion, which it identifies as the connection holding the smallest RTT value in its TCP control block structure.

Other alternatives have been proposed to reduce the splicing overhead and to improve the cluster throughput as well. SpliceNP [6,7] is an implementation of the TCP splicing mechanism on an IXP2400 network processor Intel chip. The experiments, conducted on a Xeon 3.0 Ghz Dual processor using a 1Gbps Intel Pro 1000 (88544GC) NIC, show that the process latency is reduced by up to 83%, and that the dispatcher throughput is improved by up to 5.7 times [Figure 2] [6].

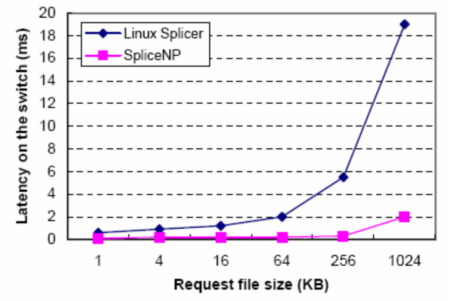

Figure 2. Linux versus NP based splicing latency for various request sizes

2.4.3. TCP handoff variants. The main goal of the TCP handoff protocol [8] is to reduce the resource usage at the entry point to the cluster and to provide the processing servers with the capability to send back the outgoing traffic directly to the clients. The basic idea is to handoff each client side connexion to the chosen processing server. The handoff mechanism requires that the TCP/IP stack at the dispatcher and at each processing node is handoff aware. In a load balancing scenario, the TCP handoff protocol operates as the following [Figure 3].

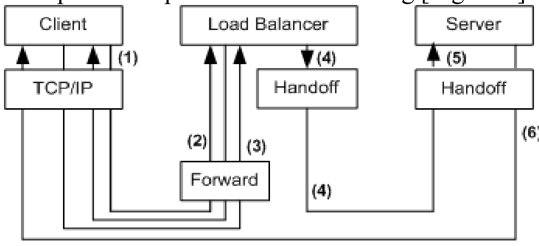

Figure 3. TCP Handoff based load balancing

Once the client side connexion is established, its state is captured and a handoff request is sent to the receiver side, where a faked three-way handshake is performed to reconstruct the socket soft structure inside the kernel of the processing node. If the socket handoff procedure succeeds, the receiver side builds an acknowledgment message and sends it back to the dispatcher. This latter enters then to the forwarding mode so as to be able to route the packets belonging to each forwarded flow to the appropriate processing node.

For a reduced latency, the handoff messages are exchanged over a UDP channel, assuming the reliability of the underlying infrastructure.

To achieve a better performance, the handoff operations can be offloaded on specialized network processor chips.

2.4.5. Application aware routing using MPLS. Multi-Protocol Label Switching is a protocol designed for connection oriented routing. Packets having the same label share the same Forwarding Equivalent Class (FEC), and are routed the same way in the network. The MPLS labels are injected in the link layer and in the network layer headers. In a load balancing scenario, it is assumed that all the intermediate routers, as well as all the cluster nodes, support the MPLS protocol. The idea [9], hence, is to provide all the datagrams pertaining to the same session with the same label.

3. Session aware network traffic load distribution

3.1. Sessions versus flows

An application typically manages several client sessions. A session is an association between two communicating end points, and is related to an activity undertaken by a user, such as web browsing, e-mail consulting, networked game playing, file transfer, and so on.

A given session can span over a single, or over multiple connection-oriented and/or connectionless flows [Figure 4].

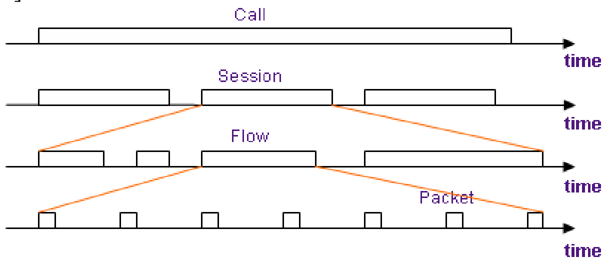

Figure 4. Flows versus Sessions

NGN services as well as some of the already familiar services are multiple-flow based. They first involve an opening flow to establish the end-to-end session. This signalling flow is then used to negotiate the identifiers and the parameters of the subsequent flows pertaining to that same session.

The association of the flows pertaining to the same IP session to different processing servers inside the cluster leads to a possible interruption of the session or to its QoS degradation. On the other hand, the state of the art

network load balancing frameworks, such as LVS [10] and IBM Network Dispatcher [11], are exclusively flow-aware. In order to ensure the reliable execution of services, these frameworks need to provide application level integrity. LVS provides a client based persistency model, by which all the data issued from a given client is marked at the IP level, and assigned to the same processing node inside the cluster.
While providing session integrity, this approach fails to achieve an optimal throughput because it doesn't efficiently spread the offered network load among the available cluster resources. Hence, this solution doesn't truly meet the session awareness requirements, especially, when the clients generate unequal traffic volumes. In the following, we'll analyse some load balancing scenarios of representative IP services. We'll argue for the use of deep packet inspection (DPI) techniques to provide a fine grained distribution of the offered sessions on the available cluster resources.

3.2. Distributing the offered FTP sessions in a cluster of servers

FTP [12] is a typical example of a multiple flow based service. An FTP signalling flow is first established between a client and a server. Then one or more distinct flows are established to carry the bulk data from the server to the client.
In a basic flow aware load balancing scenario, the control and the data flows pertaining to the same incoming passive FTP session are assigned independently to the available processing servers. Client persistency based schemes can help solving this issue, but still provide no means to fairly allocate the cluster resources to the incoming sessions.
Fine grained distribution of the offered FTP sessions can be achieved by identifying every FTP session through inspecting the PASV responses flowing from the FTP servers to the FTP clients on the signalling flow [Figure 5].

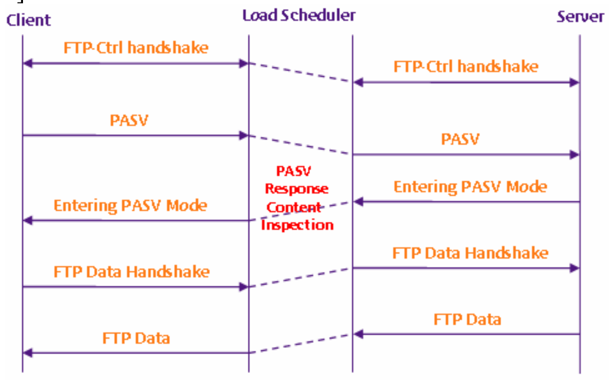

Figure 5. Passive FTP sessions load balancing scenario

3.3. Distributing multimedia streaming sessions in a cluster of servers

Video streaming is a typical example of a multiple and heterogeneous flow based service. The delivery of continuous media involves three protocols. First the RTSP protocol [13] is used to establish a connexion oriented control flow between a client and the streaming server. Then, the video streaming server wraps the requested media into RTP [14] datagrams and sends them to the client over a UDP channel. RTCP datagrams [15] provide flow control like functions and are periodically exchanged between the transmitter and the receiver of the media. The following illustration shows how RTSP, RTP and RTCP are typically involved in a unicast streaming session [Figure 6].

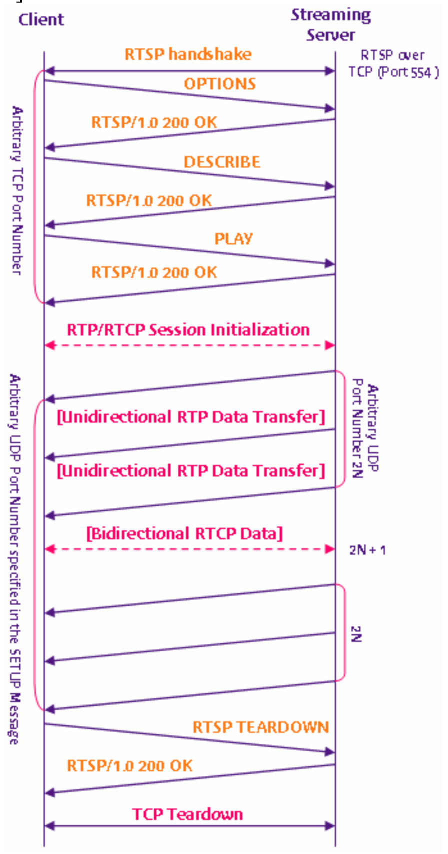
Client
Streaming Server
RTSP handshake
RTSP over TCP (Port 554)
OPTIONS
RTSP/1.0 200 OK
DESCRIBE
RTSP/1.0 200 OK
PLAY
RTSP/1.0 200 OK
Arbitrary TCP Port Number
RTP/RTCP Session Initialization
[Unidirectional RTP Data Transfer]
[Unidirectional RTP Data Transfer]
Arbitrary UDP Port Number 2N
[Bidirectional RTCP Data]
2N + 1
2N
Arbitrary UDP Port Number specified in the SETUP Message
RTSP TEARDOWN
RTSP/1.0 200 OK
TCP Teardown

Figure 6. A streaming media session using RTSP, RTP and RTCP

Unless a persistency scheme is used, a basic flow aware load balancing scenario distributes the RTCP control packets independently from their corresponding RTSP signalling flow, leading to the QoS degradation of the rendered service. Hence, fine grained load balancing needs to identify the RTCP and the RTSP flows pertaining to the same streaming session. To achieve this goal, there is a need to inspect the content of the SETUP responses flowing from the streaming server to the clients on each RTSP signalling flow at the entry point to the cluster.

3.4. Distributing SIP based VoIP sessions in a cluster of servers

The Session Initiation Protocol [16] has been chosen by 3GPP as the signalling protocol for the IP Multimedia Subsystem architecture. Scalability is an issue of a great importance for operators providing Voice over IP services using SIP, because SIP proxies are expected to handle heavy load.
The SIP signalling architecture is built around SIP user agents and SIP servers. SIP servers include (***i***) a SIP registrar, with which client user agents register at their bootstrap, (***ii***) SIP proxy servers, which handle SIP requests, (***iii***) SIP redirection servers, which redirect a SIP request to an available SIP user location, and (***iv***) SIP location servers, which resolve the SIP user location.
SIP signalling messages can be wrapped into TCP or into UDP datagrams. Most of the current implementations are UDP based.
Unless DPI mechanisms are called at the entry point to the cluster, a basic flow level load balancing scenario may route the SIP signalling messages pertaining to the same SIP session to different SIP proxies [Figure 7].

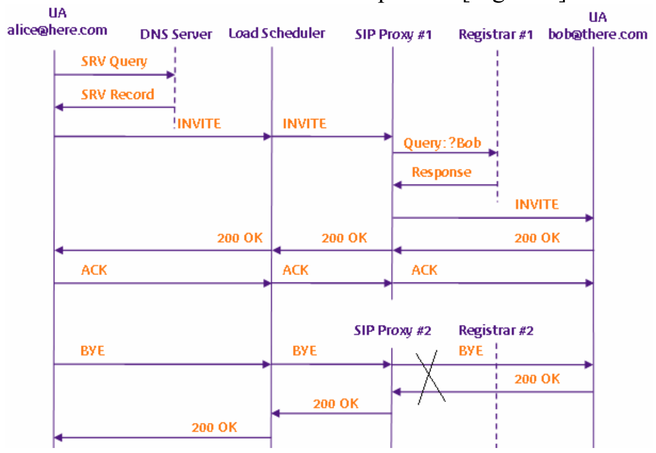

Figure 7. A SIP session load balancing scenario

The inspection of the content of each incoming SIP message at the entry point to the cluster allows the association of the messages pertaining to the same SIP call to the same SIP proxy.

4. Specific Scenarios: DPI for NAT and firewall traversal

Session awareness through content inspection is also useful to solve both of the NAT and firewall traversal issues.

4.1. The NAT traversal issues

NAT issues arise when the communicating user agents are behind NAT devices. For instance, SIP INVITE responses, as well as RTP packets are miss-routed to the source user agent, when the IP address specified in the SDP offer corresponds to a private IP address.
The SIP specifications use content inspection to solve the SIP NAT issue. Each SIP proxy needs to compare the *Via* header field value of the handled message with the originating IP address. When these values differ, the public address is added to the *received* field in that message SDP header. Nonetheless, this solution remains limited to solve the static NAT traversal issue.
Other solutions have been proposed to address the general case SIP NAT traversal issues [17,18,19,20].

4.2. The Firewall traversal issues

Firewalls maintain security rules reflecting the operator security policy. The TCP/IP header of each incoming datagram is checked against these rules in order to decide to accept or to reject the incoming datagram. Typical firewalls prohibit all but the legitimate UDP traffic, such as DNS and NTP messages.
On the other hand, while most of the standard services use well known ports, typical NGN services such as peer-to-peer and streaming video involve flows which identifiers are negotiated on the fly over the corresponding opening flow. Hence, in order to allow the operator to better control the traffic generated by such services, DPI techniques must be used to inform the firewall to dynamically open the legitimate ports.

Secondly, firewalls are sensitive devices which may be offered a heavy network load. In order to reliably scale firewalls, we need to ensure that datagrams related to the same flow as well as flows pertaining to the same session are safely forwarded to the same firewall [Figure 8].

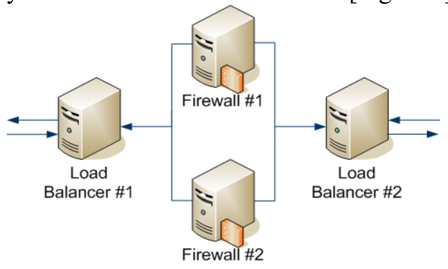

Figure 8. Firewall load balancing

Hence, the use of DPI techniques is a must to reliably scale firewalls by preventing the interruption of the legitimate sessions.

5. Conclusion and lessons learned

In this paper, we questioned the appropriateness of using flow aware network load balancing frameworks to distribute the offered network traffic among the cluster available resources. We showed that multiple flow based sessions require fine grained load balancing, through deep packet inspection, to achieve session integrity while efficiently spreading the offered network load among the cluster available resources.

Next generation load balancing frameworks require to follow a one-way architecture, in order to achieve an optimal resource usage at the entry point to the cluster on the one hand, and to provide a better scaling factor inside the cluster on the other hand. These factors, as well as investing adaptive resource allocation methods, are essential to provide an improved throughput under normal load. This approach reduces also the probability that an accepted session at the cluster head gets refused by the chosen processing server, due to its instantaneous bottleneck.

The QoS of the rendered service under heavy load is also a relevant issue for the next generation load balancing frameworks. Hence, we believe that session awareness should be considered while admitting or rejecting the offered network traffic to the cluster.

Session awareness is quite relevant when scaling firewalls as well. We believe that the use of DPI techniques is a must to reliably scale sensitive network devices by preventing the interruption or the QoS degradation of the legitimate sessions. The DPI techniques are also useful to allow the traversal of specific NAT devices.

6. References

[1] C.Z. Xu, "Scalable and secure Internet services and architectures", Chapman & Hall, 2005.

[2] C. Perkins, "IP Encapsulation within IP", Internet RFC, October 1996.

[3] D. A. Maltz and P. Bhagwat, "TCP Splicing for Application Layer Proxy Performance", IBM research report, March 1998.

[4] W. H. Cheng, "Design and implementation of a web switch", research report, National Cheng Kung University, 2004.

[5] M-Y. Luo, C-S. Yang, "Persistent Connections Management in Web Server Cluster", Proceeding of the IEEE International Conference on Web Technologies, Applications, and Services, 2005.

[6] L. Zhao, Y. Luo, L. Bhuyan, "SpliceNP: A TCP Splicer using a Network Processor", Proceedings of the symposium on Architecture for networking and communications systems, ANCS 2005.

[7] W. Tang, L. Cherkasova, M.W. Mutka, "Modular TCP Handoff Design in STREAMS based TCP/IP Implementation", Proceedings of the First International Conference on Networking, 2001.

[8] J. Lu, J. Wang, "Analytical Performance Analysis of Network-Processor-Based Application Designs", Proceedings of the 15th International Conference on Computer Communications and Networks, Oct. 2006.

[9] R. Dragos, S. Dragos, M. Collier, "Design and implementation of an MPLS based load balancing architecture for Web switching", Internet draft, 2002.

[10] W. Zhang, "Linux virtual server for scalable network services", 2000.

[11] G. Goldszmidt, G. Hunt, "Netdispatcher: A TCP connection router", IBM research report, 1997.

[12] J. Postel, J. Reynolds, "File Transfer Protocol (FTP)", Internet RFC 959, October 1985.

[13] H. Schwarzbauer, Q. Xie, K. Morneault, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, "Stream Control Transmission Protocol", Internet RFC 2960, October 2000.

[14] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications", Internet RFC 1889, January 1996.

[15] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, "RTP Control Protocol", Internet RFC 3550, July 2003.

[16] J. Rosenberg, H. Schulzrinne, U. G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, E. Schooler "Session Initiation Protocol (SIP)", Internet RFC 3261, June 2003.

[17] J. Rosenberg, J. Weinberger, C. Huitema, R. Mahy, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)", Internet RFC 3489, March 2003.

[18] Rosenberg, Weinberger, Huitema, Mahy, "Traversal Using Relay NAT (TURN)", Internet Draft, March 2003.

[19] NewPort Networks white paper, "NAT Traversal for Multimedia over IP", August 2005.

[20] P. Mächler, "SIP Network Address Translation (NAT), SIP Architecture with NAT", Siemens white paper, 2004.