Towards a Dependable Architecture for Highly Available Internet Services

Narjess Ayari[1], Pablo Neira Ayuso[2], Laurent Lefèvre[3], Denis Barbaron[1], Rafael M. Gasca[2]
[1]France Telecom R&D - 2, Avenue Pierre Marzin, 22307 Lannion, France, {narjess.ayari,denis.barbaron}@orange-ftgroup.com
[2]QUIVIR Research Group - Department of Languages and Systems, ETS Ingenieria Informatica - Avda. Reina Mercedes, s/n 41012 SEVILLA, Spain, {pneira.gasca}@lsi.us.es
[3]INRIA RESO - LIP - Université de Lyon, - Ecole Normale Supérieure de Lyon - 46, allée d'Italie - 69364 LYON, France, laurent.lefevre@inria.fr

***Abstract*- For an improved QoS provisioning, operators are strongly concerned both with the availability and the reliability of their performance critical servers. Among these entities, we mention the firewalls used to filter the offered network traffic to the operator's information system as well as the end processing servers which service each incoming client request. In this work, we advocate and evaluate a full architecture for highly available services. Performance evaluations show that our proposed framework incurs a minimal overhead to the end-to-end communications during failsafe periods and performs well during failures.**

I. INTRODUCTION

Network operators have always been concerned both with the robustness and the security of the services they provide to the end clients. For an improved QoS provisioning, operators are strongly concerned both with the availability and the reliability of their performance critical servers. Among these entities, we mention the firewalls used to block and to monitor the offered network traffic to the operator's information system as well as the end servers which process each incoming client request. Firewalls are placed between the corporative network and the Internet. They apply filtering policies to determine which traffic is allowed in the network. Filtering policies are defined by means of a rule-set. A rule contains several descriptors that refer to packet header information. Modern firewalls go further and keep track of the evolution of a traffic flow to ensure that it evolves in a standard compliant way. This evolution is stored in a variable so-called state. The state information can be used to define more intelligent filtering policies that neutralize several attacks such as TCP reset.

Firewalls and servers are single point of failures in the network since their failures lead to service disruptions or to the whole network isolation. Their availability is particularly important to guarantee that network services are properly rendered. Existent fault tolerant solutions are not suitable for stateful firewalls. First, these solutions are not tailored to the stateful devices. Second, most of them add severe delays to the client responses.

On the other hand, most of the NGN services are based on a session model which involves different flows for the signalling and for the data exchange all along the session lifespan. Providing reliability means to the whole session requires providing reliability capabilities to every related flow.

In this work, we advocate and evaluate a full architecture for highly available services illustrated in Fig. 1.

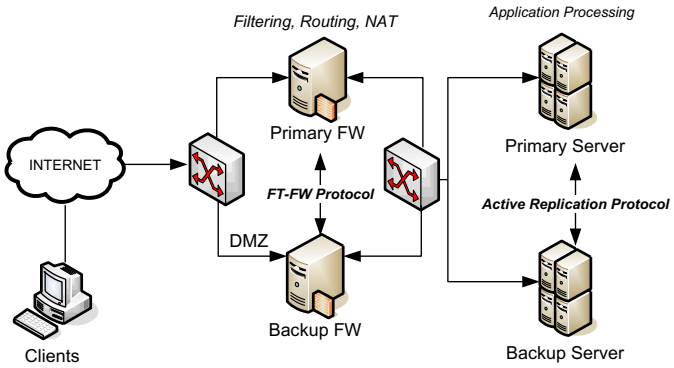


Fig. 1. The highly available framework architecture.

The framework replicates the critical components of the operator information system to provide high availability capabilities. It generalizes previous works [1,2] on high availability for Internet servers and stateful firewalls. Performance evaluations show that the framework incurs a minimal overhead to the end-to-end communications during failsafe periods and performs well during failures.

The rest of this paper is organized as follows. In section II, we describe the related work to high availability. In section III, we give an overview on the framework components. In section IV, we present the performance evaluation results. Finally, we conclude the paper by summarising its potential perspectives.

II. RELATED WORK

Several research works have been proposed to provide service reliability support. This work focuses on the client transparent solutions. An efficient connection failover for web servers was detailed in [3]. Its main idea consists in a backup TCP (BTCP) stack implementation that is a silent version of TCP. The web servers are organized into a ring.

Each time a request is received, a balancer forwards it to the primary as well as to the backup servers. The backup keeps a backup socket structure that contains the request state. Since the BTCP stack never sees incoming traffic, some important information has to be inferred, so the authors propose a complete logic to infer such information. This solution requires however server side modifications. A similar solution is FT-TCP [4] which key concept are the loggers, a software component that stores in the backup replica all the packets processed by the primary replica via network tapping techniques. In case of failure of the primary node, the backup replays the complete communication until the last consistent state before the failure is reached. However, this solution does not scale well for long connections with high data exchange rates. Moreover, the authors of FT-TCP do not cover the loggers' machine failure. In ST-TCP [5,6], the backup node relies on intercepting passively the traffic flowing between the primary node and the clients. The TCP layer receive queue on the primary node is modified such that it keeps a copy of any TCP segment already read by the application, unless an acknowledgment is sent back by the backup node to inform the primary that the segment has been successfully processed. The main disadvantage of ST-TCP is its cost during failsafe periods. In fact, when the backup is not as fast as the application server on the primary node, the ST-TCP failover mechanism affects seriously the TCP flow control on the primary node. This latter would advertise a reduced congestion window which leads to a less end-to-end throughput. Moreover, ST-TCP leads to inconsistent connection states on the backup node when the primary node fails jointly with a TCP segment loss by the backup node. In [7], the authors discuss an architecture providing availability capabilities to stateful firewalls. The architecture is based on an event-driven model and describes a library baptized Stateful Networking Equipments (SNE) library, which allows the implementation of highly available firewalls and routers. However, the proposed work assumes server side dependencies.

III. General Architecture

This work generalizes previous works on the high availability of stateful firewalls [1] and connection oriented flows [2]. In the following, we give an overview of the building blocs of the proposed architecture.

A. FT-FW

FT-FW is a complete solution for cluster-based fault tolerant stateful firewalls that keeps in mind simplicity, transparency, fast responses to clients and low cost. Basically, the solution must guarantee negligible delay in client responses so that bandwidth performance is not reduced. Our solution is composed of two parts: the hardware and the software architectures. The main idea of the hardware architecture is the firewall cluster. This cluster is composed of two or more firewalls replicas that are deployed in the local area network always coupled two by two. These replicas are connected through a dedicated link that is used to propagate state changes. We assume that, at least, one firewall replica acts as Primary, so that deploys the traffic filtering, and the other acts as Backup.

On the other hand, the software architecture follows an event-driven model (EDM) whereby any change in the state variable is propagated through an event. These events are produced by the stateful firewall and consumed by the state proxy (SP). The SP is an application that propagates state changes to other replicas through the dedicated link. The EDM suits well for distributed systems since share many of the same characteristics such as modularity and loose-coupling, and whose asynchronous nature suits well for the performance requirements of stateful firewalls. This architecture is not dependent of the failure detection schema. So, we assume a failure detection software, i.e. an implementation of VRRP, that works in cooperation with the SP.

The stateful firewall provides a framework to subscribe to state change events, dump the state of all existent traffic flows, and inject states to the stateful firewall to implement state recovery during failures. The SPs use this framework to interact with the stateful firewall.

At start-up, every SP dumps the existent states and stores them in a cache, so-called internal cache, and subscribes to events of state change to keep the cache up to date. The SP also maintains another cache to store foreign state changes that comes from other replicas. Once an event of state change occurs, the SP that runs in the Primary updates its internal cache and propagates the state change to the SP that runs in the Backup so that it updates its external cache. If the Primary fails, the failure software detection notifies the SP that invokes the inject method to insert the states stored by the external cache into the stateful firewall. We represent the FT-FW architecture and the information flow in Fig. 2.

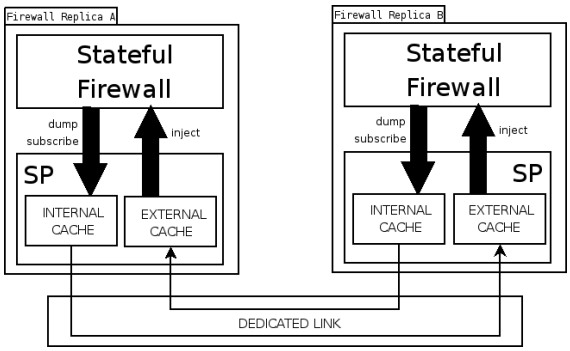


Fig. 2. The FT-FW architecture.

In order to communicate, the SPs use a replication protocol which must guarantee high current state durability (CSD), which is the probability that current states can survive failures [8]. Nevertheless, the protocol must

introduce a negligible delay in client responses. For that reason, we discard any synchronous replication protocol used in database environments that would guarantee CSD of 1 but, in return, introduce a severe penalty in client responses. Therefore, the replication protocol must be asynchronous. However, the use of an asynchronous solutions does not guarantee that the firewall replicas contain the same set of states at any time, ie. we cannot guarantee that replicas are one-copy equivalences. Nevertheless, our target is to improve CSD without harming client responses. Therefore, the use of an asynchronous turns out the only feasible solution. We have already proposed a reliable UDP-based replication protocol that addresses the problems exposed above [1].

B. The Active Replication Building Blocs

The used active replication based framework is a generalization of a previous work on the reliability of connection oriented flows. It achieves its goals by providing particular processing during failure-free and during failure periods. First, during failure-free periods, it provides means to achieve a consistent and a transparent replication of every service state, be it kernel level or application level. The kernel level states include for instance the flow level states, the connection tracking states, etc. The application level states are however the states associated to the application running on the highly available node.

Second, the framework guarantees service consistency during the active replication process by guarantying that only one replica is providing the service at once. All the other replicas are however silent.

In case a failure occurs at the primary server, the replica first takes over its network level identity. Then it leaves the silent mode and enters the master mode where it fully provides the service to the end clients.

The proposed framework assumes first the test bed illustrated by Fig. 3. According to it, a replica is able to non-intrusively tap the traffic legitimately flowing between the primary node and the end clients.

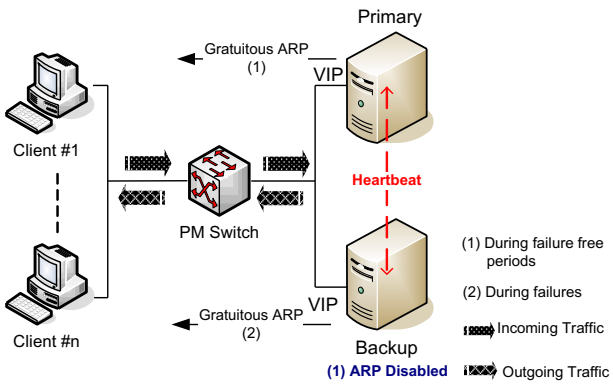


Fig. 3. The active replication infrastructure.

The backup is engineered such that it actively replicates the primary's states. In order to our infrastructure to cope with a wide range of applications and services, we opted to let both the primary and the backup nodes share the same virtual IP address of the service. This approach is particularly interesting in case we want to actively replicate a server which explicitly involves network level identifiers at the upper layer protocols during the regular processing of the offered requests. This is typically the case of network address translation (NAT) devices and application level gateway (ALG) devices which handle signaling traffic. The effective association between the virtual IP address of the service and the link level identifier of the node is provided by means of ARP gratuitous flooding. Indeed, during failure-free periods, the primary node initiates the flooding. The backup node is on the other hand engineered such that it ignores any ARP request. Static entries are however required in its ARP table. Once a failure occurs, the backup node performs gratuitous ARP flooding in order to take over the network level identity of the service. Finally, a consistent end-to-end service is provided during failure-free periods by dropping the outgoing traffic generated by the backup consequently to the active replication.

IV. Performance Evaluation

In this section, we evaluate the performance of the FT-FW and the active replication based protocols.

Since firewalls are subject to a heavy load, we choose to evaluate FT-FW using an AMD Opteron dual core 2.2GHz hosts connected to an 1G Ethernet network. The active replication framework is however evaluated using less powerful machines.

A. FT-FW protocol Results

The schema is composed of four hosts: Host A and B that act as workstations and FW1 and FW2 that are the firewalls (Fig. 1). We have adopted a Primary-Backup configuration for simplicity. Thus, FW1 acts as Primary and FW2 acts as Backup. In order to evaluate the solution, we reproduce a very hostile scenario in which one of the hosts generates lots of short connections. Thus, generating loads of state change messages. Specifically, the host A requests HTML files of 4 KBytes to host B that runs a web server. We created up to 2500 GET HTTP requests per second (maximum connections rate reached with the test bed used). For the test case the Apache web server and a simple client HTML suite have been used.

To evaluate FT-FW, we have implemented a state proxy daemon for stateful firewalls so-called *conntrackd* (connection tracking daemon) [9]. This software is a user-space program written in C that runs on Linux. We did not use any optimization in the compilation.

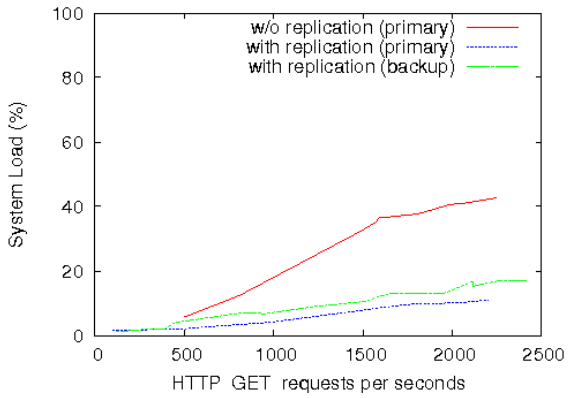
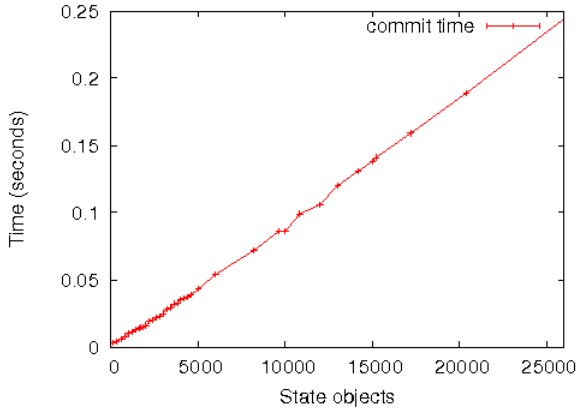


Fig. 4. CPU overhead.

We have measured CPU usage in FW1 and FW2 with and without full state replication. The tool cycle soak has been used to obtain accurate CPU consumption measurements. For the sake of simplicity, we use a reduced version of the graph of TCP states that consists of six states: syn_sent, syn_recv, established, time_wait, fin_wait, close. This means that we have generated up to 6 * total number of requests. The results obtained in the experimentation has been expressed in a graph (Fig.~\ref{load}). The full state replication is CPU consuming, reaching up to 42.7\% of CPU load. Not surpringsly, the full replication of short connection is costly due to the amount of states propagated.

Fig. 5. Time required to inject objects



In order to obtain the delay that FT-FW introduces in client responses, we have created up to 1700 HTTP GET requests per second while measuring the round trip time of an ICMP echo request/reply (ping pong time) with and without replication enabled. The results have been expressed in the following table (in milliseconds) (Tab. 1).

TABLE I

HTTP GET rps	w/o replication	replication
1500	238.5	246
1700	243	247

As we can observe, the increment in the round trip time is between 4 and 7 milliseconds so that we can say that the delay introduced in client's responses is negligible.

We have also measured the time to inject state objects that represents connections from the proxy daemon external cache to the in-kernel CTS. The results (Fig.. 5) show that the injection of 20000 state objects is close to 180 ms, that is an affordable delay. Another obvious conclusion extracted from the results obtained is that the time required to inject objects increases linearly.

We can conclude from the results that FT-FW requires extra CPU power in order to improve CSD of CBSF but have negligible impact in terms of client response time.

B. Active replication results

We run our experiments using three separate machines, one each for the client, the server and the replica as shown in Fig. 3. On the backup server, we installed the active replication user and kernel space modules. The primary and the backup are 2089 MHz AMD Athlon XP 2800+ PCs with 512 KB of cache and an SDRAM 512 MB memory. On both machines, the Linux 2.6.18 kernel is running. The client is a 1.60 GHz Pentium(R) Laptop with 512 MB of memory and also running Linux, although it could be running any other OS and TCP/IP stack. The client used a Broadcom NetXtreme Gigabit Ethernet card. Each replica uses three 100/1000 Ethernet PCI cards. The three machines are connected to the same LAN using a 100 Mb Ethernet port mirroring capable Catalyst 3500 series XL CISCO switch.

All the active replication modules are required on the backup node during failure-free periods and no modifications are required either to the client or to the legitimate server to achieve state replication. As a result, no active replication overhead is incurred to the active sessions during failure-free periods. During failure-free periods, we evaluated the active replication process performance based on measures conducted at the backup node. The first metric we considered is consistency. First, we checked that the infrastructure correctly rewrites the incoming traffic to the backup's NIC. We checked also that this traffic is properly delivered to the upper layers which consequently generate an outgoing traffic destined to the legitimate client. We checked that the legitimate outgoing traffic, used to build consistent TCP states is also correctly intercepted and analyzed (Tab. 2).

TABLE II
STATE ACTIVE REPLICATION CONSISTENCY EXPERIMENT

Application	Traffic Interception and Rewriting at the Backup	Outgoing Traffic Generation at the Backup
Echo	Correct frames (Headers + Data)	Correct replies (Headers + Data)

In order to provide service consistency, the duplicated outgoing traffic generated by the backup node is however dropped at the backup's network layer using Iptables based rules.

The next dimension of the active replication process cost during failure-free periods is expressed in terms of the latency overhead incurred both to the incoming and the outgoing traffic. A run of the application sending 100 messages of 100 bytes each to the legitimate server provides us with a mean latency overhead of about 1,5 ms in a 100 Mb LAN. The following illustration describes the latency overhead incurred to the incoming and to the outgoing frames (Fig. 6).

We can observe that the packet rewriting incurs a fairly minimal overhead to the incoming traffic. The gap experienced by the outgoing traffic depends however on the hardware and on the software tuning as well as on the possible network congestion.

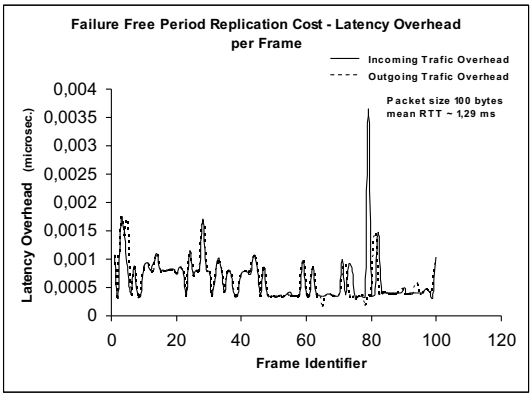


Fig. 6. Packet rewriting and packet generation latency overhead.

Second, we evaluated the resource's usage overhead due to the active replication at the backup node during failure-free periods in terms of overhead on the CPU, memory and network buffers occupancy.

Following are the illustrations of this overhead in time for a run of 100 sec. Fig. 7 depict the CPU usage at both replicas during failure-free periods.

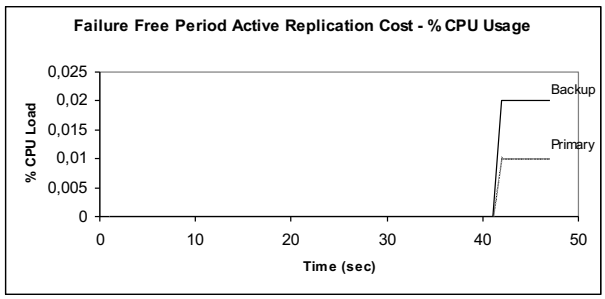


Fig. 7. CPU usage during failure-free periods.

Fig. 8 describes however the network buffer's usages at both nodes during failure-free periods.

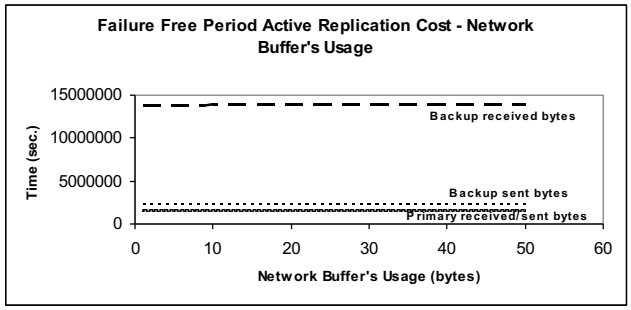


Fig. 8. Network buffer's usage during failure-free periods.

These illustrations show that the active replication resource's usage overhead is fairly minimal during failure-free periods, meaning that an active/active replication scenario would require no more than optimized hardware and software tunings of the replicas.

Next, we evaluated the effectiveness of the described setup in providing availability capabilities to a simple stateless service. Failure period tests are based on two components, which are the failure detection and the failure recovery modules. The failure recovery time is indeed defined as the sum of the failure detection and the traffic takeover latencies. A measure of the takeover latency advocates a mean value of 360 ms. This value depends basically on the processing capabilities of the backup node. Next, we measured the partaking of the failure detection granularity on the average recovery time. The following table gives the average recovery time for a failure detection interval (FDI) of 1, 3 and 5 sec respectively (Tab. 3). One hundred packets of 100 bytes each were exchanged over a network having a mean RTT of 69 ms.

FDI (sec)	Average Failure Detection Latency (micro sec)	Avg Takeover Latency (micro sec)	Avg Recovery Time (micro sec)
1	2 153 039	0 359 052	2 512 091
3	3 655 465	0 371 398	4 026 863
5	5 141 020	0 374 115	5 515 135

TABLE III
SERVER SIDE AVERAGE RECOVERY TIME FOR DIFFERENT FAILURE DETECTION TUNINGS.

As shown above, the failure detection procedure should be fine grained enough to detect a failure as soon as it occurs so as to incur a minimal overhead to the failure recovery latency. On the other hand, the mean packet loss under the above conditions has been evaluated to 1%.

Next, we measured the service throughput in bytes per seconds before and following an injected failure, as shown in the Figure 9.

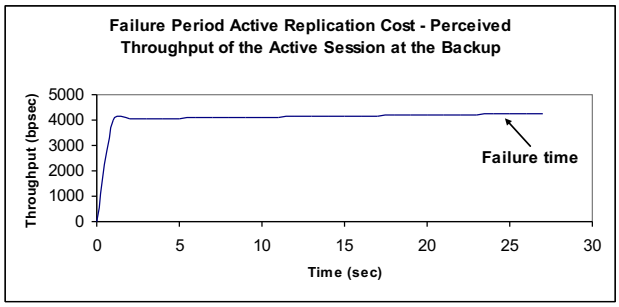


Fig. 9. Throughput of the highly available session.

The failure was injected 25 sec after the service starts running. The results show that no service degradation should be perceived by client following a failure recovery.

V. CONCLUSION AND FUTURE WORKS

In this work, we have proposed a complete architecture for the availability of Internet services that concerns both the firewall and the servers. This architecture is suitable for low cost off-the-shelf equipments. The conducted experiments show that the solution provides a sustained performance both during failure-free and failure periods. Also fast client responses and quick recovery are guaranteed. As future works, we plan to provide more integrated evaluation of both approaches and study multi-primary scenarios in which more than one replica deploys the service at the same time.

REFERENCES

[1] P.Neira, R.M.Gasca, L.Lefevre. "FT-FW: Efficient Connection Failover in Cluster-based Stateful Firewall". In 16th Euromicro International Conference on Parallel, Distributed and network-based Processing, Toulouse, France, feb 2008. Available: http://dune.lsi.us.es/~pablo/fw.pdf

[2] N. Ayari, D. Barbaron, L. Lefèvre, P. Primet, "T2CPAR: A system for Transparent TCP Active Replication", The IEEE 21st International Conference on Advanced Information Networking and Applications (AINA-07), 20th-24th May 2007.

[3] N. Aghdaie and Y. Tamir, "Client-transparent fault-tolerant web service," in 20th IEEE International Performance, Computing, and Communication conference, 2001, pp. 209–216.

[4] Dmitrii Zagorodnov, Keith Marzullo, Lorenzo Alvisi, Thomas C. Bressoud, "Engineering Fault-Tolerant TCP/IP Servers Using FT-TCP", Proceedings of the International Conference on Dependable Systems and Networks, DSN 2003.

[5] M. Marwah, S. Mishra, C. Fetzer, "TPC server fault tolerance using connection migration to a backup server", Proceedings of the International Conference on Dependable Systems and Networks, DSN 2003.

[6] Manish Marwah, Shivakant Mishra, Fetzer, C., "A system demonstration of ST-TCP", Proceedings of the International Conference on Dependable Systems and Networks, DSN 2005.

[7] P. Neira, L. Lefevre, and R. M. Gasca, "High availability support for the design of stateful networking equipments," in *IEEE proceeding ARES'06: The First International Conference on Availability, Reliability* and Security, Vienna, Austria, apr 2006.

[8] X. Zhang, M. A. Hiltunen, K. Marzullo, and R. D. Schlichting, Customizable service state durability for service oriented architectures, in IEEE Proceedings of EDCC-6: European Dependable Computing Conference, Coimbra, Portugal, oct 2006.

[9] P. Neira, conntrack-tools: The netfilter's connection tracking userspace daemon, http://people.netfilter.org/pablo/conntrack-tools/.