On improving the Reliability of Internet Services through Active Replication

Narjess Ayari and Denis Barbaron
France Telecom R&D
2, Avenue Pierre Marzin, 22307 Lannion, France
{narjess.ayari,denis.barbaron}@orange-ftgroup.com

Laurent Lefèvre
INRIA - University of Lyon
Ecole Normale Supérieure, 46, allée d'Italie - 69364 LYON, France
laurent.lefevre@inria.fr

***Abstract*- Fault tolerance can be defined as the capability of a system or a component to continue normal operation, despite the occurrence of a hardware or a software fault. Frameworks providing a fault tolerant service take advantage of resource redundancy to provide high availability capabilities. One of the critical challenges that led to this work is the observation that the existing fault tolerance frameworks are not adapted to current and next generation Internet services. Indeed, they do not provide consistent service-aware failure recovery capabilities. Particularly, little interest has been granted to transport level awareness despite its important partaking in improving the reliability of connection oriented services and stateful devices. In this paper, we evaluate an active replication based framework for highly available Internet services. The proposed framework is fully client/server transparent. Performance evaluations show that it incurs minimal overhead to end-to-end conversations during failsafe periods and performs well during failures.**

Index Terms— High availability, reliability, transport level awareness, session level awareness, connection oriented services, stateful devices.

I. Introduction

Fault tolerance can be defined as the capability of a system or a component to continue normal operation, despite the occurrence of hardware or software faults. Frameworks providing a fault tolerant service take advantage of resource redundancy, to provide high availability capabilities. They are based on two key concepts, which are the fault detection on the one hand, and the fault recovery on the other hand. Moreover, they are subjected to different challenges related to their robustness and performance. One of the critical challenges that led to this work is the observation that the existing fault tolerance frameworks are not adapted to the current as well as to most next generation Internet services. Indeed, they do not provide consistent service-aware failure recovery capabilities. Particularly, little interest has been granted to the transport level aware failure recovery capabilities despite their important partaking in improving the reliability of the connection oriented services and the stateful devices.

Conversely, Internet servers run typically stateful applications. Services based on TCP maintain a server state for each peer involved in the communication. TCP states store all the information required to perform error, sequence and flow control. Due to the explicit association between a service and its physical location for the wired Internet, a failure of the legitimate processing server leads to loosing the states related to all the active conversations, be they transport level or application level.

Network level failover techniques [1] are not sufficient to transparently recover TCP based services. Conversely, TCP tolerates short periods of disconnection not longer than a few RTTs. A typical example is a simple connection to a database server. When the connection terminates, all the uncommitted transactions handled over this connection are aborted, requiring users to restart the lost connection from scratch. A robust transport level failover is then needed to provide reliability to TCP based services.

In this paper, we evaluate an active replication based service-aware highly available framework. The proposed framework is fully client/server transparent. Performance evaluations show that it incurs a minimal overhead to the end-to-end communications during failsafe periods and performs well during failures.

Section 2 briefly presents the related works and section 3 describes the general overview and design space of our proposal. The architecture details are presented in section 4 and some performance evaluation are analysed in section 5. Section 6 concludes this paper and proposes some future works.

II. Related Work

Several research works were proposed to provide Internet servers with reliability capabilities. These works can be classified into client-aware solutions, requiring changes to the client side, and client-transparent solutions, moving most of the fault tolerance work either to the target server or to an intermediate proxy.

Among the client-aware solutions, we first mention Migratory-TCP (M-TCP) [2]. M-TCP transparently migrates the server side endpoint of a live connection and assists the underlying server application in resuming its reliable service, following the connection migration. SockMi [3] is a similar mechanism which moves both the client and the server endpoints of a live connection. MSOCKS [4], Indirect-TCP (I-TCP) [4] and TCP handoff [5] advocate all of them split-connection proxy-based architectures which achieve consistency by rewriting all the traffic flowing between the clients and the servers. Their major drawback consists in the proxy being the major point of failure, and the main source of bottleneck. Reliable sockets (ROCKS) and reliable packets (RACKS) [6] are two systems providing means for transparent network connection mobility. TCP Migrate [7] offers similar capabilities, assuming both connection endpoints to be TCP Migrate aware.

The major drawback of the client-aware solutions is that they require the cooperation of the end server, either with the client side or with an intermediate device. From then, they are not adapted to the current client applications, which require to be rewritten accordingly. Secondly, at the origin, they were not designed to meet the requirements of true failure situations. Indeed, the server side state can be lost before being consistently migrated to a safe server.

Client-transparent fault tolerant frameworks use however replication techniques to backup, during failure-free periods, every service state on a number of replicas. Following a node or a link failure, the service is recovered on an elected replica. A number of approaches exist to coordinate between replicas during failure-free periods. Among these approaches we mention the checkpointing approach [8], the message logging approach [9], the active replication approach [10] and the hybrid approaches [10].

In a previous work [1], we out looked these approaches and showed that the active replication offers the best recovery time compared to any other mechanism. Its main idea is to have all the servers receive and concurrently process the offered network traffic. However, in order to provide efficient failure recovery capabilities, the concept of active replication assumes replicas to generate the same consistent service states when provided with the same input. We provided also a comprehensive overview of the building blocks of a fault tolerance framework. We outlined state-of-the-art failure detection approaches and dealt with service replication approaches and issues. We provided an overview of failure recovery techniques, used to provide network, flow and session level high availability capabilities. ARP spoofing, Virtual Router Redundancy Protocol as well as the Network Address Translation are well known network level failure recovery techniques. They provide an acceptable level of service availability for stateless services by enabling a replica with means to takeover the network level identity of the failed server. However, they do not provide reliability capabilities to stateful services such as stream-based services. In order to avoid restarting the already active flows or sessions from scratch, in the event of failure of the legitimate processing server, both transport level and session level failure recovery approaches are required. FT-TCP [11], transparent connection failover [12] and ST-TCP [13] are representative client transparent frameworks providing transport level failure recovery capabilities. In [1], we showed that these frameworks reduce the performance of the end-to-end conversations during failure-free periods, by leading to smaller throughput or increased delay and legitimate server resources' usage.

III. GENERAL OVERVIEW AND DESIGN SPACE

This proposal is a generalization of a previous work for transparently enhancing the reliability of stream based flows [15]. The active replication-based framework that we advocate aims to provide high availability capabilities to any stateful device, be it a stateful firewall, a deterministic processing server or the entry point to a cluster of less deterministic servers. It enhances the service reliability by avoiding the interruption of the already active sessions, in case of failure of the legitimate processing server. The framework handles both stream-based and connectionless oriented services.

Let us first have an insight into the framework design space [Fig. 1]. The idea consists in a replica non-intrusively tapping the network traffic legitimately flowing between the primary node and the clients. The backup node is engineered such that it actively replicates the states maintained by the primary node.

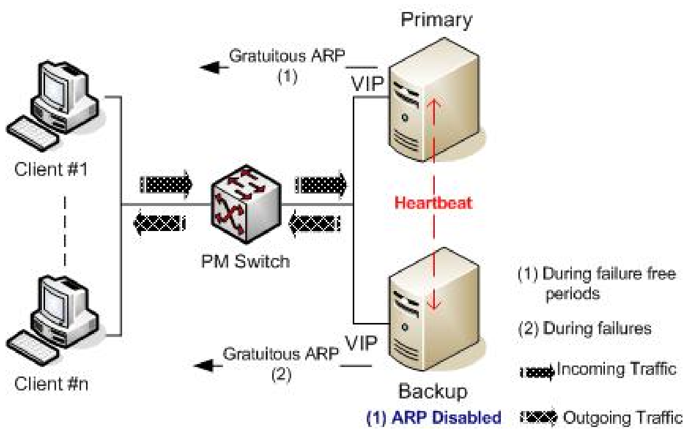

Figure 1. - Design Space of the active replication-based Framework

In order for our infrastructure to cope with a wider range of services, while incurring a minimal processing to the backup node during failure-free periods, we chose to share the virtual service IP address between replicas. This approach is particularly interesting, for instance, in case we actively replicate the entry point of a cluster of non-deterministic application servers. The effective association between the service virtual IP address and the primary's link level address is provided by means of ARP flooding. In essence, during failure-free periods, the primary node initiates a gratuitous ARP flooding. The backup node is on the other hand engineered such that it ignores any ARP request. In case the primary node is subjected to a failure, the backup node takes over the ownership of the network level service identity by performing, on its turn, gratuitous ARP flooding.

A particular processing is required during failure-free periods on the one hand, and during failures, on the other hand. First, during failure-free periods, the framework provides the means to achieve a consistent and a transparent replication of every service state, be it kernel level or application level. Kernel level states are states built and maintained by the kernel for each service running in the system, during its lifetime. These states can be the flow level states, the connection tracking states, etc. Application level states are however the states associated to the application running on the highly available node.

Our framework ensures service consistency by guaranteeing that only one replica is providing the service at once. Otherwise saying, during failure-free periods, the backup server is silent: the outgoing traffic it generates consequently to the active replication process is dropped.

Should the primary server fail, the available replica takes over the service network level identity. Next, it leaves the silent mode and enters the master mode, where it fully provides the service to the end clients.

IV. ARCHITECTURE DETAILS

The proposed infrastructure is fully client/server transparent. Indeed, no changes are required at the client and at the legitimate server. In particular, the active replication components are exclusively deployed at the backup node. The advantage of this approach is that, during failure-free periods, the primary node suffers neither a worse performance, for instance in terms of reduced end-to-end throughput, nor an increased resource's usage.

The key concepts of the proposed service aware highly available framework are based on three main components (Figure 2). The first component provides the state replication capabilities for the handled stateful services. The second component provides the means to detect a potential failure of the highly available service or node. The third component provides failure recovery capabilities.

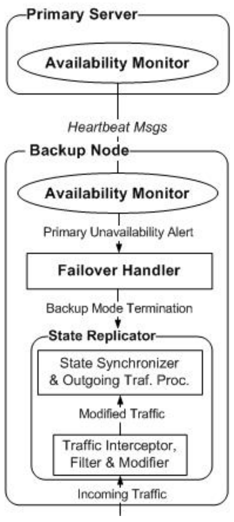

Figure 2. - Active Replication Architecture

A. *The active replication-based component for state replication*

The active replication-based component for state replication is split between a set of user and kernel space programs which together provide the backup node with means to receive and to process the offered network traffic to the primary node. Our implementation covers UDP-based and TCP-based services. It distinguishes between three steps. First, packets legitimately destined to the primary node are intercepted by the replica's network interface. Second, they are filtered at the link level. Only the traffic pertaining to the highly available service(s) is delivered to the next engine, which will rewrite it at the link level and inject it into the replica's kernel. Since replicas share the same network level address and are similarly configured and tuned, the injected traffic is guaranteed to successfully bypass the upper layers sanity checks, in particular at the network level.

The steps of intercepting and filtering the legitimate traffic are fundamental to the active replication process. We used the BSD packet filter-based Libpcap capabilities [14] to tap the traffic flowing between the clients and the legitimate server. Firstly, Libpcap puts the Ethernet interface into the promiscuous mode to enable non-intrusive copies of the legitimate packets to be obtained. These packets are then filtered according to our settings, based for instance, on the transport level service identity. In order to reduce the replica's load and help it lower its drop rate, only the relevant subset of the captured packets is copied to a Libnet-based user space program for traffic modification and rewriting. Libnet [14] is a powerful portable open source C-library for creating and injecting network traffic. It supports packet creation at all the TCP/IP network levels and provides a granular access to the network packets at each level of the protocol stack. The user space module for traffic modification smartly parses and rewrites each packet protocol header. The resulting traffic is then delivered to the backup's network stack and completes its journey up to the application layer for processing.

A.1. State synchronization for stream-based services

The initial sequence number used by the server is the fundamental source of non-determinism at the TCP level. Other sources of TCP level non-determinism can be handled by similarly tuning the replica's TCP/IP stacks, particularly, at the transport level. In order to guarantee similar flow states over the replicated servers, we chose to synchronize replicas at the three-way-handshake rather than synchronizing the replicated TCP states all along the TCP flow's lifespan.

The early TCP state synchronization requires the backup node to non-intrusively intercept the primary's outgoing traffic. In particular, it intercepts segments having their SYN/ACK flags set. A client/server-like package is deployed at the backup node(s) to perform the early state synchronization. The client side of the package is a user space application which intercepts and analyses the primary's outgoing SYN/ACK segments. From the intercepted traffic, it builds a minimal connection state, described in figure 3.

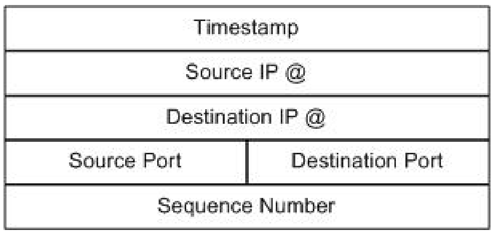

Figure 3. - Structure of the Connection Synchronization Message

This minimal state is then sent to a peer kernel space module through a Netlink socket. The Netlink-based kernel module is responsible for performing the TCP early state synchronization of the replicated flows.

A.2. Service consistency during failure-free periods

In order to ensure that only one server is processing client requests at once, all the data generated by the backup are dropped at the network level using Netfilter-based rules. In addition, to improve effectiveness, we used the sysctl interface to similarly tune the replica's TCP stack parameters such as the read and write buffer sizes, the options, etc.

B. The failure detection component

Two properties are desirable for the failure detection component. First, it should detect failures as soon as they occur so as to rapidly trigger the resumption of the service on an available backup. Second, it should be robust enough to ensure that only one error free instance of the service is running at once. Our infrastructure provides failure detection through the explicit and the secure exchange of heartbeat messages between replicas, over redundant links. The used failure detection mechanism obeys the push model, described in the previous chapter. In essence, each replica monitors its peer by sending a Hello message each fixed time interval over a unicast channel, while expecting the receipt of an acknowledgement within a timeout period. Both the timeout and the time separating the sending of consecutive regular heartbeat messages are adjustable.

C. The failure recovery component

The failure recovery component is triggered by the failure notification of the primary server. It aims to achieve a transparent failure recovery of the service. The network level identity takeover provides service availability. Reliability capabilities consist however in avoiding the interruption of the already active sessions during the resumption of the service on the available replica.

The main operations of this component are summarized as follows. First, the replica disables all the modifications required by its kernel during the failure-free period. It disables the routing and the link level customizations, such as enabling ARP and disabling all the active replication related filtering rules. It also stops all the active replication related processes, such as traffic interception, filtering, modification and rewriting. Next, it leaves the silent mode by disabling the filtering rules triggered on its outgoing traffic. Finally, it takes over the ownership of the network level identity, by carrying out an ARP gratuitous flooding, similarly to what the legitimate server performs at the framework's start-up, to confirm its ownership of the service virtual address.

V. Performance evaluation

An implementation of the above-described active replication-based framework is provided for the Linux 2.6.18 kernel. The prototype is checked using typical services having different communication patterns. In this section, we quantitatively evaluate the performance of the implemented prototype in enhancing the high availability capabilities of interactive stream oriented service. It consists in a client and a server reliably exchanging messages over a TCP channel. The echo conversation has similar communication patterns to the telnet and the secure shell applications.

A. Experiment setup

We run the experiments on the stream oriented services using the test bed described in figure 1. Both replicas run the same stream oriented service during the experiments. At the backup server, we additionally deployed the active replication related user and kernel space modules. The primary and the backup nodes are 2089 MHz AMD Athlon XP 2800+ PCs, with 512 KB of cache memory and 512 MB of SDRAM. At both machines, the Linux 2.6.18 kernel is running. The generic client is a 1.60 GHz Pentium(R) Laptop with 512 MB of memory. It also runs 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. In order to allow the backup node to tap the full traffic flowing between the clients and the legitimate node, we enabled the full duplex port mirroring on the port to which the primary server is connected. To increase effectiveness, the replica's kernels were tuned similarly. In particular, the replica's TCP stacks were set to use their optimal values, such as the TCP buffers, the memory limits, etc. Finally, replicas are synchronized through the NTP service.

B. Results and discussion

The conducted experiments consist of a backup node non-intrusively intercepting the legitimate traffic destined to the primary server. The backup node actively replicates the states maintained by the primary server, both kernel level and user level. Should the primary node fail, the backup node takes over the newly offered sessions, as well as the already active ones, while avoiding the interruption of the ongoing conversations. We perform various sets of measurements. The first set measures the performance of the prototype during failure-free periods. The second set of tests evaluates the performance of the prototype during failures.

All the measurements are repeated at least three times and their average value is considered in the results.

B.1. Performance during failure-free periods

State replication effectiveness

This test aims to check the effectiveness of the active replication modules, enabled at the backup node. The incoming traffic is properly rewritten to the backup's kernel. It completes its journey in the TCP/IP stack up to the application layer, which generates responses and delivers them back to the transport layer. The bottom-up processing results in the kernel maintaining all the related service states.

Delay overhead incurred to the incoming and to the outgoing traffic

Recalling the definition of the delay as the time from when a client issues a packet to the time this packet is received by the destination, we define the delay overhead as the additional time incurred to a legitimate frame before it reaches the backup's kernel. This overhead is measured at the backup node and is due to the link level packet rewriting.

In the case of the stream-based experiment, this overhead is incurred only to the incoming traffic. Figure 4 depicts the average packet rewriting latency incurred to the stream-based traffic as a function of the received frame identifier. The mean size of the traffic flowing over the TCP stream is 75 bytes. The mean overhead is evaluated to almost 6 ms.

As we can see, in the event of failure of the legitimate node, the average contribution of the packet rewriting operation in a possible QoS degradation is fairly low compared to the perceived end-to-end delay value.

Netlink communication performance

The Netlink-based communication is used during failure-free periods to synchronize flow states among replicas. In essence, in order for any fake mini-socket, maintained at the backup node, to be successfully promoted to the ESTABLISHED state, it is of prime importance that its state is updated before the backup node receives the final TCP ACK, issued by the client to complete the TCP three-way-handshake. Otherwise saying, the time needed by the backup node to intercept the minimal legitimate state, added to the time required by the Netlink-based kernel module to update the fake connection state, must be less than the average round trip time delay.

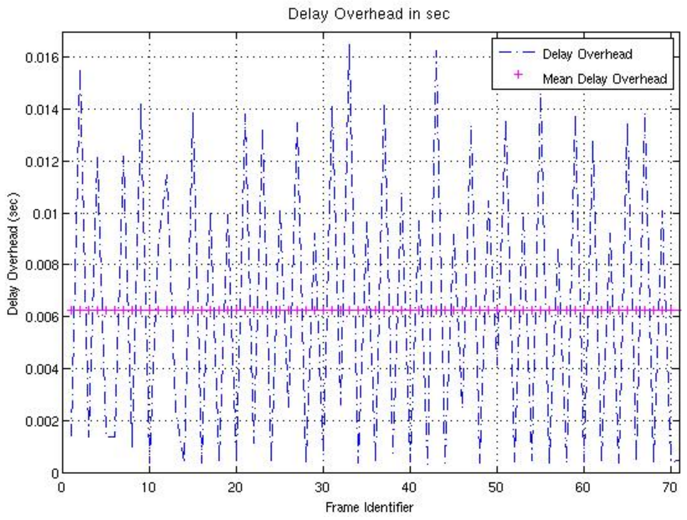

Figure 4. Delay Overhead Incurred to the Offered Stream-based Traffic

We define the kernel-to-user Netlink Delay as the time required by the kernel space synchronization module to receive the minimal legitimate connection state, sent by its peer user level program, added to the time required to update the fake connection state accordingly. Figure 5 shows that the kernel-to-user netlink delay remains on average less than the average round-trip time.

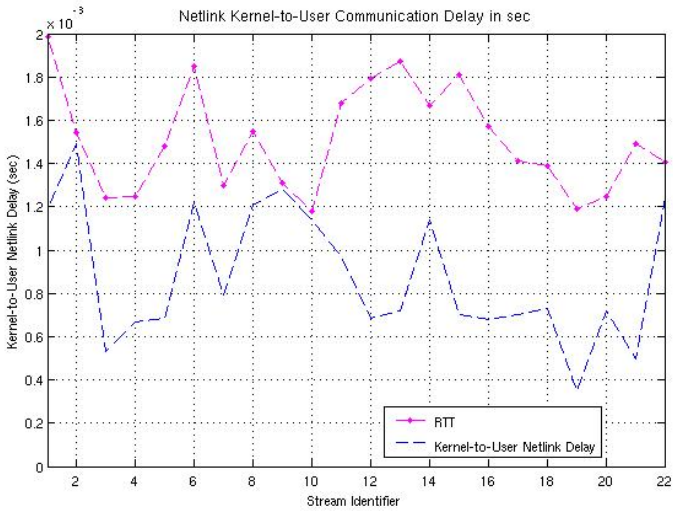

Figure 5. - Average Kernel-to-User Space Netlink Delay versus Average Round Trip Time Delay

These measures considered the worst case, i.e. where clients and servers share the same 100 Mb Ethernet network. Knowing that the round trip time delay is fairly more important over the Internet, it turns out that the fake mini-sockets maintained at the backup node are sure to be promoted to the ESTABLISHED state. Moreover, similarly tuning the replica's TCP stacks is of prime importance so that they exhibit the same behavior over time.

B.2 Performance during failures

Failure recovery latency

In this experiment, we aim to measure the effect of the failure detection granularity on the failure recovery latency. We define the failure recovery latency as the time needed to detect and to recover from a failure of the legitimate processing server. It is the sum of the fault detection latency and the takeover latency. The fault detection latency measures the time separating the failure occurrence and the failure detection. The fault recovery latency is defined as the time required by the elected replica to takeover both the network and the service level identities of the failed primary.

Table 1 summarizes the collected average failure recovery latencies, for different failure detection time intervals (FDTI).

As we can notice, the takeover time is only few hundreds of ms. During that period of time, any offered data to the legitimate server is lost. However, from an end-to-end point of view, the effect of this loss remains insignificant since most client applications retransmit the lost data. Also, as one could have expected, the finer is the granularity of the failure detection process, the faster is the traffic takeover operation.

FDTI (sec)	Average Failure Detection Latency (micro sec)	Average Takeover Latency (micro sec)	Average 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 1 - Average Failure Detection and Average Takeover Latencies

VI. Conclusion and Future Works

In this paper, we have advocated an off-the-shelf active replication-based framework, which enhances the reliability capabilities of Internet servers and stateful devices. The framework is fully client/server transparent and provides the means to achieve a consistent service state replication of the handled services during failure-free periods. The conducted performance evaluations have shown that the framework provides a sustained performance, during both failsafe and failure periods.

As future works, we are interested in covering a wider range of applications and in handling highly non-deterministic applications. While combining different replication techniques is a possible alternative for addressing this issue, we believe that providing truly client/server transparent frameworks, remains an interesting perspective to operators.

VII. References

[1] N. Ayari, D. Barbaron, L. Lefèvre, P. Primet, "A Survey on Fault Tolerance Approaches and Issues for Highly Available Services", To appear in IEEE Communications Surveys & Tutorials, June 2008.

[2] Z. J. Haas, P. Agrawal, "Mobile TCP: an asymmetric transport protocol design for mobile systems", IEEE 1997.

[3] http://sockmi.sourceforge.net

[4] D. A. Maltz, P. Bhagwat, "MSOCKS: an architecture for transport layer mobility", IEEE DSN 1998.

[5] A. Bakre, B. Badrinatdh, "I-TCP: Indirect TCP for mobile hosts", June 1995.

[6] http://pages.cs.wisc.edu/~zandy/rocks

[7] A. Snoeren, H. Balakrishnan, "Fine-Grained Failover Using Connection Migration", 3rd USENIX USITS, March 2001.

[8] O. Laadan, D. Phung, J. Nieh, "Transparent checkpoint-Restart of distributed applications on commodity clusters", Proceedings of the IEEE International Conference CLUSTER 2005.

[9] E.N. Elnozahy, W. Zwaenepoel, "On the use and implementation of message logging", Digest of papers of the 24th Fault-Tolerant Computing Symposium FTCS-24, 1994.

[10] L. Wang, W. Zhou, W. Jia, "The design and implementation of an active replication scheme for distributing services in a cluster of workstations", The Journal of systems and software, 2001.

[11] D. Zagorodnov, K. Marzullo, L. Alvisi, T.C. Bressoud, "Engineering fault-tolerant TCP/IP servers using FT-TCP", Proceedings of the International Conference on Dependable Systems and Networks, DSN 2003.

[12] R. Koch, S. Hortikar, L.E. Moser, P.M. Melliar, "Transparent TCP connection failover", Proceedings of the International Conference on Dependable Systems and Networks, DSN 2003.

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

[14] N. Dhanjani, J. Clarke, "Network Security Tools", O'Reilly Media,March 2005.

[15] 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.