Project no. IST-033576

XtreemOS

Integrated Project
BUILDING AND PROMOTING A LINUX-BASED OPERATING SYSTEM TO SUPPORT VIRTUAL ORGANIZATIONS FOR NEXT GENERATION GRIDS

Extended version of the distributed servers platform
D3.2.11

Due date of deliverable: December 1st, 2009
Actual submission date: December 1st, 2009

Start date of project: June 1st 2006

Type: Deliverable
WP number: WP3.2
Task number: T3.2.1

Responsible institution: VUA
Editor & and editor's address: Vrije Universiteit
Dept of Comp. Science
de Boelelaan 1081a
1081HV Amsterdam
The Netherlands

Version 1.2 / Last edited by Jeffrey Napper / December 7, 2009

Project co-funded by the European Commission within the Sixth Framework Programme		
Dissemination Level		
PU	Public	√
PP	Restricted to other programme participants (including the Commission Services)	
RE	Restricted to a group specified by the consortium (including the Commission Services)	
CO	Confidential, only for members of the consortium (including the Commission Services)	

Revision history:

Version	Date	Authors	Institution	Section affected, comments
0.0	15/10/09	Guillaume Pierre	VUA	Document skeleton
0.1	05/11/09	Jeffrey Napper	VUA	First draft
0.2	06/11/09	Guillaume Pierre	VUA	Minor improvements
1.0	06/11/09	Jeffrey Napper	VUA	Ready for internal review
1.1	12/11/09	Jeffrey Napper	VUA	Edited according to feedback from G. Pipan.
1.2	4/12/09	Jeffrey Napper	VUA	Added evaluation and example sections.

Reviewers:
Yvon Jégou (INRIA) and Gregor Pipan (XLAB)

Tasks related to this deliverable:

Task No.	Task description	Partners involved$^\circ$
T3.2.1	Design and implementation of distributed servers	VUA*

$^\circ$This task list may not be equivalent to the list of partners contributing as authors to the deliverable
*Task leader

Executive summary

Distributed Servers provide an abstraction that allows a group of server processes to appear as a single entity to its clients. This deliverable presents the current status of the development of Distributed Servers. It describes the current re-implementation for the XtreemOS 2.0 linux kernel version 2.6.27.

Since the last report, we have re-implemented Distributed Servers as a set of initialization scripts, a separate kernel module for saving and restoring sockets, and a local demon that communicates with the application. The kernel module is based on an open source TCPCP module developed at NEC. We have updated the module for the 2.6.27 kernel and added additional functionality. This module is maintained separately and can be used independently of Distributed Servers for (including IPv4) socket passing. The new version of the kernel module will be released as a separate open source project after Distributed Servers is included in XtreemOS.

The Distributed Servers package also contains a new, local, system-level demon. Requests for local actions related to a handoff are given to the local demon for security checks and processing. By creating a new demon process, it is not necessary to modify the existing Mobile IPv6 implementation for Linux. Therefore, we do not need to maintain any changes to the Linux networking stack and can immediately leverage any speed or stability improvements made therein. This approach has led to a much stabler implementation of Distributed Servers.

Finally, we expect Distributed Servers to be integrated into the minor release XtreemOS 2.1. After completing packaging, further work on the performance and security aspects will continue.

Contents

1 Introduction

The goal of the Distributed Servers package is to help provide the abstraction of a large, scalable server that can handle any number of clients. This package in work package 3.2 concentrates on the network component of the abstraction. A set of nodes providing the scalable server abstraction can handoff clients to each other without the need for client assistance. The client only must support the Mobile IPv6 protocol as described in previous deliverables [7].

This deliverable describes the progress made in the last year (since [9]). We completed porting Distributed Servers to run on the Linux kernel 2.6.27 on which XtreemOS 2.0 is based. Our new version has several advantages. First, it does not require modifying the Mobile IPv6 implementation in Linux, resulting in much simpler maintenance and better stability. Second, we introduced a system-level demon that handles the privileged aspects of Distributed Servers, providing a secure channel for applications to request actions such as handoffs. Third, we improved the TCPCP kernel module to efficiently drain connections for faster and smaller handoffs [1]. Finally, we also provided an internal API to the Distributed Servers implementation in XOSAGA so that application developers can use a standard API [11]. This is one of the final steps to integrating Virtual Nodes and Distributed Servers as described in [8].

This deliverable is organized as follows. Section 2 describes the current design of Distributed Servers after porting to the XtreemOS 2.0 linux kernel. Section 4 discusses the current status of the implementation and areas for future development. Finally, Section 6 concludes.

2 Design

The Distributed Servers package provides location transparent networked services [4]. Clients connect to a single *distributed server address* for a service and may be moved transparently among multiple locations. Mobile IPv6 (MIPv6) route optimization [3] does the heavy lifting: all IPv6 connections from a client are atomically changed directly to each location, avoiding triangular routing. The distributed server address is simply an IPv6 [2] address. In the terminology of Distributed Servers, a client first connects to a *contact node*. A client may then be transparently *handed off*—the server endpoint of all of the client's connections are transferred—to different servers for load-balancing or for client-specific processing. The ability to transparently migrate client connections depends on the implementation of Mobile IPv6, which typically supports client mobility. However, Distributed Servers can provide server mobility, inverting the functionality. As mentioned in previous deliverables [7, 9], the Distributed Servers functionality depends on all parties—clients and servers—possessing and using IPv6 addresses with support for Mobile IPv6.

The Distributed Servers system was originally built on Linux kernel 2.6.8 (released 2004). In the last year, we have further redesigned Distributed Servers for the 2.6.27 Linux kernel used by XtreemOS 2.0. This kernel has relatively stable support for Mobile IPv6 along with support for new hardware likely to be found in computational grids. An important design goal of the new Distributed Servers package was to leave the MIPv6 implementation in Linux untouched. Our experience modifying earlier versions of the MIPv6 implementation led us to believe that maintenance would be considerably easier if we did not modify the increasingly complex IPv6 implementation.

In order to not touch MIPv6, we created a system that spoofs mobility events to persuade the stock MIPv6 implementation to perform route optimization on client handoff. A system-level demon, called *dsco,* injects spoofed packets and drops other packets as necessary to spoof new client connections or break old ones. Our redesign encompasses three major areas: 1) developing initialization scripts to bring the system up, 2) porting the TCPCP kernel module used for connection passing (handling kernel level socket state), and 3) a system-level demon to perform privileged actions required by handoffs. We describe these components in more detail in the rest of the section and give a detailed description of a handoff in the following Section 3.

2.1 Overview and Startup

At a high level, the current design of Distributed Servers is very similar to previous versions. The standard MIPv6 implementation in Linux is handled by a user-level demon, called *mip6d,* that manages mobility events. All three roles in MIPv6 are handled by different configurations of the same mip6d demon: the Mobile Host (MH) changes network addresses generating mobility events; the Home Agent (HA) manages the MH home address in the home network for connectivity of new connections; and the Correspondent Node (CN) represents the other endpoint of connections with the MH during mobility events [3].

Our design uses the mip6d in a controlled environment to simulate mobility of the server end of a connection to clients. The relationship between the MIPv6 and Distributed Servers terminology is straightforward. The Home Agent is an unmodified component of MIPv6 and serves precisely the same role in Distributed Servers. The Distributed Servers *contact node* accepts new connections from clients and is registered with the HA. Other *server nodes* accept client handoffs from the contact node and other server nodes. Both the contact node and set of server nodes are configured as Mobile Hosts in the MIPv6 terminology. Clients that contact these nodes are configured as Correspondent Nodes.

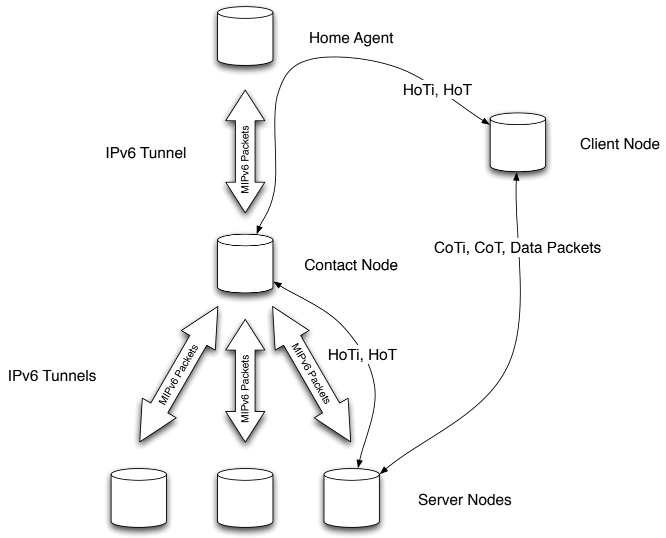

Figure 1: Overview of Distributed Servers design architecture. All MIPv6 control packets from server nodes are routed through IPv6 tunnels to a spoofed Home Agent on the Contact Node. The route optimization packets (HoTi,HoT) are then routed to the real Home Agent, which in turn routes them to the client node.

Figure 1 gives an overview of the new design architecture of the system. The system is composed of several parts: 1) the unmodified mip6d demon, 2) the ds startup script used to modify static network routes, 3) the TCPCP kernel module used to enable connection passing between nodes, 4) the dsco demon that manages application requests and dynamic network routes, and 5) the Gecko framework that uses the dsco demon to provide applications with a simple API for client migration [5]. The interaction of these systems is shown in Figure 2, which provides a single node-level view of the system.

We describe the architecture from the top down. The contact node runs an extra mip6d in Home Agent mode on a dummy subnet to allow server nodes to register with an HA. Startup scripts configure the network settings of each server node to route connections to this spoofed Home Agent on the contact node instead of the real network HA. These connections are automatically made by the mip6d on the server nodes configured as Mobile Hosts. The contact node is configured specially so that its mip6d demon connects to the real Home Agent for the distributed server address. Hence, the unmodified Home Agent for the distributed server address sees only one connection from a Mobile Host responsible for the server address—the contact node.

In our design, all MIPv6 control messages from server nodes to the HA are routed through the contact node because the contact node hosts the spoofed HA.

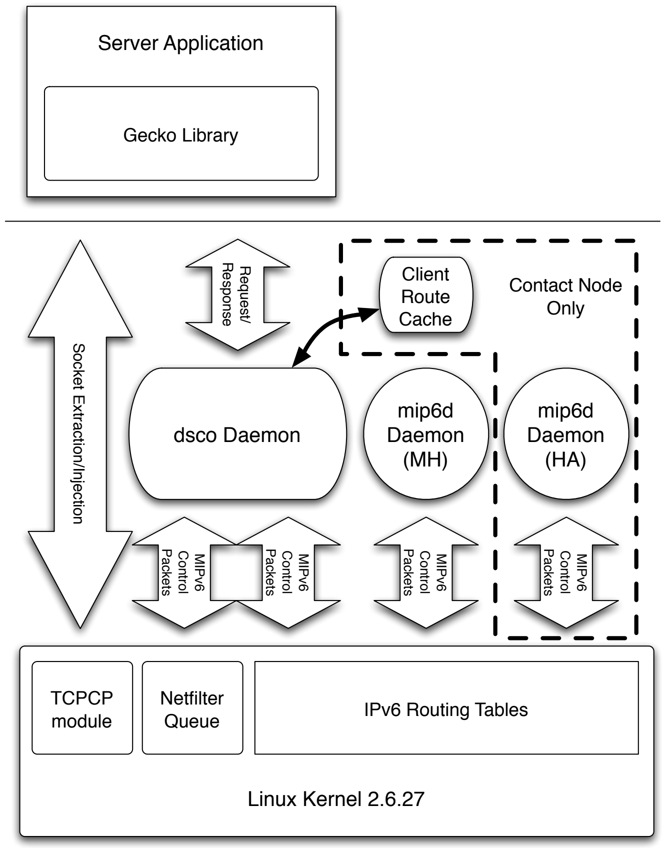

Figure 2: Node view of current Distributed Servers architecture. The server application manages the Distributed Servers framework using the Gecko library, which in turn communicates with the dsco demon for privileged actions and to perform TCPCP actions (above the horizontal line is unprivileged space). The dsco demon can inject or drop MIPv6 control packets and change the IPv6 routing if the contact node changes. Note the mip6d demon is unmodified, and client IPv6 traffic is handled as normal by the kernel.

However, client IPv6 traffic still routes directly to the client after route optimization. The purpose of the dsco demon at the contact node is to specially route the MIPv6 control messages to the real Home Agent to which the contact node is connected normally. The dsco demon caches routes for the control messages so that replies (from clients) can be correctly routed to the proper server node. Access to the packets before local processing is achieved using the Netfilter Queue interface in Linux [6]. Further discussion of the dsco demon appears in a later subsection. The combination of mip6d and dsco demons handles all client handoffs provided by the Distributed Servers service.

If the contact node fails or needs to be rotated for any reason, a new contact node can takeover for a failed contact node using the *ds* script provided in the package. The script starts the spoofed Home Agent, reconfigures the local network settings to accept packets from server nodes, and informs the local dsco demon to begin performing actions as the contact node. The ds script must also be used on all server nodes to reconfigure the network settings to route MIPv6 control packets to the new contact node. We envision that some of these actions will eventually be moved into the dsco demon itself so that reconfiguration can be handled with the same API as client handoffs.

The initial configuration of Distributed Servers is determined by a configuration file for the dsco demon. Since the demon performs privileged actions, configuration of Distributed Servers requires privileges. Applications can query the configuration through requests to the dsco demon. Changes to non-privileged settings can also be performed through requests to the dsco demon.

2.2 TCPCP kernel module

In addition to the high level routing of MIPv6 control messages, Distributed Servers needs to migrate the open connection to the client from either the contact node or a server node to another node in the system. The TCP connection passing system (TCPCP) provides migration of open sockets by serializing the network stack state in the kernel, which can then be transferred to a different node [1]. The TCPCP package is composed of a user-level library that invokes functions in an additional kernel module through extensions to the `getsockopt` system call.

The TCPCP package works in a straightforward manner. Sockets are first frozen; that is, they will not accept new data for sending. Next, the kernel socket state is copied into a user-level buffer, which can then be transferred to another node. Finally, at the new node the socket state is transferred back into the kernel network stack to create a new copy of the socket, and the new socket is activated to accept incoming and outgoing data.

Our completed migration consists of 1) migrating the implementation from Linux kernel 2.6.15 to the XtreemOS 2.0 Linux kernel 2.6.27, 2) adding an extra

API function to drain a socket before handoff, and 3) modifying the Distributed Servers API to use the new API provided by TCPCP. We intend to release our newer version of TCPCP back to the open source community after integrating Distributed Servers into XtreemOS.

The port is partially a reimplementation as the Linux kernel network stack has changed significantly with increasing IPv6 support. Some bugs in the TCPCP implementation were fixed, and extra functionality was added to support Distributed Servers. For example, we fixed a bug that would send the wrong timestamp for the activated socket, eventually causing local end of the connection to slow start the socket. We also added a function `tcpcp_flush` to the TCPCP library (and kernel module) that waits until a given timeout or until the kernel send buffers of a frozen socket are empty. This approach results in much less kernel socket state transferred to the user level and subsequently over the network. As shown in [4], smaller send buffers result in faster handoff times. Currently, the implementation polls inside the kernel using a very short timeout to wait for the buffers to empty. We believe this is an acceptable approach because the overall timeouts are already very short so that the time spent polling is also very short.

2.3 dsco Demon

This subsection describes the dsco demon that is an important part of the Distributed Servers package. The demon both routes MIPv6 control packets and handles requests from applications using Distributed Servers. It is composed of two major parts: a packet filter to handle control packets and a process request handler to handle application requests. The packet filter uses Netfilter queueing mechanism to route, drop, and inject packets [6]. For example, 1) the contact node must reroute HoTi packets from a server node performing route optimization (as described in [4]) to the real Home Agent to which it is connected; 2) the contact node must drop bogus Home Agent rejection messages sent by the mip6d demons configured for the Mobile Host (as opposed to the spoofed Home Agent mip6d); and 3) server nodes must inject a new, spoofed connection request from a client received in a handoff to force the mip6d demon to begin route optimization for the client. The actions are presented in detail in the context of a client handoff in the next section.

The packet filter is the primary reason that the dsco demon requires privileged execution. The demon must be allowed access to raw packets before they are routed or delivered. The Netfilter queueing mechanism provides this access, but requires special privileges because access to raw packets is necessarily limited. To prevent users from seeing all packets (including those destined to other users), access to the raw streams is obviously restricted.

```
// Delete Binding Update with specified client, dropping
//   client from local binding cache used for route
//   optimization.
bool deleteBU(const struct in6_addr *client_addr);

// Get current Binding Update sequence number.
int getBUSeqNo(const struct in6_addr *client_addr);

// Set current contact node address.
bool setContactAddr(const struct in6_addr *contact_addr);

// Inject packets to begin route optimization for the
//   specified client address.
bool startRO(const struct in6_addr *client_addr,
             int BUSeqNo);

// Add all nodes to provided container. Returns true
//   on error.
bool getAllNodes(std::vector<const DSNode*>& container);

// Return anycast address for this distributed server.
//   Returns true on error.
bool getAnycastAddr(struct in6_addr *anycast_addr);

// Returns iterator pointing to local node (self) in
//   Vector containing all nodes.
std::vector<const DSNode *>::const_iterator
   findLocalNode(std::vector<const DSNode*>& container);
```

Figure 3: Description of the current API provided to access the dsco demon. Functions send requests to and receive replies from the demon over a local UNIX socket.

The process request handler listens on a UNIX socket for local process requests. These requests perform the privileged actions of the API of the dsco demon. For example, the Gecko framework can, on behalf of an application, request that a client is started or dropped, the contact node is changed, or to request Distributed Servers configuration information such as the distributed server address. Although not all privileged actions are currently performed by the dsco demon (for example, calls to the tcpcp module are not), we intend to centralize all privileged actions required by Distributed Servers in the dsco demon to allow non-privileged (that is, non-root accounts in Linux) processes to use Distributed Servers. The use of UNIX sockets allows us to eventually move invocations of TCPCP, which require root privileges similar to access to raw packets, to the dsco demon, passing open sockets created by TCPCP to applications. The current dsco demon API used by the Gecko library [5] and applications is shown in Figure 3.

Currently, support for multiple dsco demons per machine is not present. We believe that to host multiple Distributed Servers applications on the same node, we can further specialize the packet filtering mechanism by specific distributed server address to allow multiple dsco demons per node. However, further development is required to determine whether multiple mip6d demons (configured as mobile hosts) can execute simultaneously on the node, or whether a single mip6d can support multiple distributed server addresses.

3 Handoff Example

This section provides a detailed example of a simple client handoff to help the reader better understand our new design. We consider the following scenario: A client, which we call Client, connects first to the contact node called ContactNode. After due consideration, ContactNode decides to handoff the client to the distributed server called ServerNode. The following steps occur in order:

1. Client connects to ContactNode by sending a TCP SYN packet to the distributed server address. The SYN packet is picked up by the home agent (HA) located in the address's corresponding network and forwarded to ContactNode through an IPv6 tunnel (setup by mip6d) between the HA and ContactNode. Replies from ContactNode to Client are sent back through the tunnel with the HA and are then forwarded by the HA to Client.

2. ContactNode chooses ServerNode to handoff all connections to Client. ContactNode calls the Gecko library, which freezes all connections to the Client using the TCPCP module. Then, again using TCPCP, the state of all frozen connections is extracted from the kernel and passed to ServerNode by the

functions in the Gecko library. At the ServerNode, the Gecko library receives these sockets and uses TCPCP to create new versions of the sockets in the kernel of ServerNode.

3. ServerNode then requests the dsco demon to initiate MIPv6 route optimization for the client to redirect the client from ContactNode. The MIPv6 route optimization procedure requires sending two packets to Client from ServerNode: the CoTi, which is sent directly, and the HoTi, which is sent through the HA. The HoTi is routed to ContactNode by virtue of the IP6 tunnels set up at the server nodes connecting them with the spoofed home agent on ContactNode. The dsco demon at ContactNode then routes the HoTi to the real HA, which forwards it to Client per the MIPv6 standard protocol. The reverse path is used for the HoT response to the HoTi that Client sends to ServerNode. After receiving both the HoT and CoT (which is sent directly from Client to ServerNode using the unique IPv6 address of ServerNode), the two nodes exchange a Binding Update and Binding Ack to commit the new route. These messages travel directly between the Client and ServerNode using their unique IPv6 addresses. At the end of this final exchange, all traffic to the distributed server address sent by Client will be routed by the IPv6 stack to ServerNode, effecting the handoff. The TCPCP module then handles realigning the sequence numbers and timestamps of the connections moved during the handoff.

4. Finally, ServerNode acknowledges the handoff to ContactNode, and the Gecko library signals to the application that the handoff is complete.

4 Current status

Distributed Servers are implemented and running on the Linux 2.6.27 kernel, but have not yet been integrated with XtreemOS 2.0. However, it is scheduled for release with the next minor revision, XtreemOS 2.1. The current implementation is stable and performs handoffs of clients on an active IPv6 network. We intend to release the migrated and improved TCPCP module separately back to the open source community after Distributed Servers is integrated with XtreemOS.

Work remains to be done. We intend to centralize the privileged actions required in Distributed Servers to the dsco demon so that unprivileged applications can perform client handoffs. We also will explore further optimizations, support for multiple Distributed Servers on a node, and make additional performance evaluations for Deliverable 4.2.6 [10]. Finally, we are also working with work package WP3.1 on the Distributed Servers implementation of XOSAGA so that applica-

	LAN handoff	Host-local handoff
Old	18.4 ms	9.4 ms
New	620 ms	13.4 ms

Figure 4: Handoff latency of connection as seen by the client. Time given is between arrival of last packet from the server donating the handoff and the first packet from the server receiving. The Old values are from [4] (ignoring RTT for Route Optimization), while New values were obtained using the design described in this document.

tion developers and other XtreemOS services such as Virtual Nodes [8] can use a standard API to manage Distributed Servers [11].

5 Evaluation

We evaluated Distributed Servers on a testbed of four machines connected by a 100 Mb/s ethernet switch. The machines were identical with a single 1.5 GHz AMD Athlon CPU and 500 MB of memory. One machine was configured as the client. Two were configured as distributed servers (one contact node and one server node). The remaining one acted as the network's home agent. Note that there is only one home agent for both servers, but the servers are configured to be in a different (foreign, in MIPv6) network from the home agent. We evaluated the new implementation of Distributed Servers for latency and throughput and have included measurements from the previous implemenation for comparison. For a more complete evaluation of the previous version, please see [4].

5.1 Handoff Latency

We first evaluate the handoff latency seen by the client to determine the minimum observed disruption to service. By providing a lower bound, application developers can determine whether such disruptions are manageable for their particular cases. The times for the old and new systems are provided in Figure 4. For the new design, we use tcpdump to determine the time of the last and first packets arriving from the donor and receiver of the handoff, respectively.

For comparison with the old system, we used the minimum times of the old system given when the socket is first drained of data (that is, all data buffered in the kernel is sent to the client) before performing the handoff. Our additional method *tcpcp_flush* provides this functionality in the new implementation instead of the fixed wait time used in the previous implementation.

	Total Length	Avg. Throughput
Donate	264 ms	3.8 conn. / sec
Receive	384 ms	2.6 conn. / sec

Figure 5: Handoff CPU time and estimated throughput as seen by the server in an application. The time given is the CPU time spent in both the system and user levels, not real-time. Donate and receive times are time spent by Accept and Handoff functions, respectively, in the Gecko library.

Finally, the local handoff times are also presented in Fig. 4 when a host hands-off the connection to itself, which does not require route optimization with the client or communication with another server node. These numbers thus provide the network-independent overhead and are quite similar to the previous design of Distributed Servers.

5.2 Handoff Throughput

In addition to the client perceived handoff latency, we measured the cost of Distributed Servers at the server side of the connection. To answer how many connections could the server handoff per second, we measured the CPU time spent at both user and kernel level on either receiving or donating a handoff. The results given in Figure 5 show that our current design is fairly expensive. Note that the latency for donating or receiving is not given and may be longer depending on network latencies. Fig. 5 provides the times spent on our rather dated 1.5 GHz AMD Athlon CPU and can be expected to be lower on modern processors. Handoff throughput is estimated between 3-4 clients per second, and we believe this will also improve as we optimize the new implementation.

6 Conclusion

The Distributed Servers platform is ready for inclusion in XtreemOS. The new architecture for Linux 2.6 does not modify the standard Linux Mobile IPv6 implementation. Instead, it uses a collection of scripts, a kernel module, and a system-level demon to provide the Distributed Servers abstraction to applications. Applications use the Gecko framework [5], which communicates in turn with the system-level dsco demon. This demon then injects or drops packets to manipulate the Mobile IPv6 implementation to implement Distributed Servers behavior. Deliverable [10] will describe the performance of the current implementation. Finally, after inclusion in the XtreemOS release, we will contribute the new version of the TCPCP kernel module for socket passing back to the open source community. Work on Distributed Servers remains to improve the performance and enhance the security of the package.

References

[1] NTT Corporation. TCP connection passing 2. Available on WWW, 2006. http://tcpcp2.sourceforge.net/.

[2] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6). RFC 2460, December 1998.

[3] D. Johnson, C. Perkins, and J. Arkko. Mobility Support in IPv6. RFC 3775, June 2004.

[4] Michał Szymaniak, Guillaume Pierre, Mariana Simons-Nikolova, and Maarten van Steen. Enabling service adaptability with versatile anycast. *Concurrency and Computation: Practice and Experience*, 19(13):1837–1863, September 2007. http://www.globule.org/publi/ESAVA_ccpe2007.html.

[5] Willem van Duijn. A versatile anycast framework for distributed servers. Master's thesis, Vrije Universiteit, Amsterdam, The Netherlands, February 2008. http://www.globule.org/publi/VAFDS_master2008.html.

[6] Harald Welte. Libnetfilter queue. Available on WWW, 2009. http://www.netfilter.org/projects/libnetfilter_queue/.

[7] XtreemOS. First prototype version of ad hoc distributed servers. Deliverable D3.2.2, November 2007.

[8] XtreemOS. On the feasibility of integration between distributed servers and virtual nodes. Deliverable D3.2.10, November 2008.

[9] XtreemOS. Reproducible evaluation of distributed servers. Deliverable D3.2.6, December 2008.

[10] XtreemOS. Evaluation report. Deliverable D4.2.6, January 2010.

[11] XtreemOS. Second set of engine extensions covering XtreemOS functionality from d3.1.5. Deliverable D3.1.9, January 2010.