Live migration of Virtual Machines: Sustaining active TCP-sessions

IK1550 Internetworking
Final Paper
Examinator: Prof. Gerald Q. Maguire Jr.

Elis Kullberg
KTH - Royal Institute of Technology
Stockholm, September 3 2009

1. Introduction

Virtualization and the technology associated with it, has attracted quite a lot of attention during the past years. As of 2009, it is estimated that the market for virtualized X86 server solutions will be worth in the vicinity of $1.8 Billion [1]. So far, the main advantages presented by virtual solution vendors have been related to server consolidation savings. Users of virtualization capable hardware environments enjoy a greater "bang for the buck" since servers with low utilization can be virtualized in the same physical machine. Guest-migration, the process of moving virtual guests between different hardware platforms has been an area of research for more than 20 years [2, page 1]. The process of "live" or "seamless" migrations of virtual machines (or processes) would add scalability to virtualization solutions since virtual guests could be moved freely around different sets of hardware depending on bandwidth, available CPU cycles and latency requirements, while optimizing costs for the service provider. Migrations can also add redundancy to server environments by mirroring active VMs on different sets of hardware. The health of a VM can be monitored via a tailored external active monitoring solution or via passive monitoring in the host OS, and migrated at the first sign of trouble [3, page 18].

The hardware-related issues for offering techniques for seamless live migrations of servers have more or less been solved. Most solution vendors offer some kind of service for migrating guests between different physical hosts seamlessly [4, page 7]. However, these solutions still struggle with network related challenges, mainly the problem of keeping active TCP-sessions connected for an indefinite time after migration. Most solutions today offer no integrated way to ensure that network connectivity is unaffected by a server migration, thus all existing connections are lost for the end user every time a server is migrated.

The purpose of this paper is to assess current methods for ensuring the continuity of TCP-sessions during, and after, live migrations of virtual machines outside the original subnet. The primary purpose of this type of functionality is to enable migrations of servers via wide area networks, but this functionality is also useful for any type of service-provider with ambitions of offering scalable virtualized services to clients by migrating virtual machines or processes between different types of hardware in different geographical or network-topological environments.

The paper will present a summary of current methods for live-migrations, and also review the feasibility and implications of future proposed solutions. The paper also features a test-implementation of a current technique.

2. Theory: Research conducted so far

2.1 An introduction to virtualization

Reasons for why the market for virtualization-solutions has boomed, and buzzwords such as "the cloud" are becoming increasingly used in boardrooms around the globe include reduced costs for bandwidth and widely different costs for IT-competence in different parts of the globe. However, the main reason for the leap is that the main reverse-salient for virtualization – the fact that the X86 architecture is notoriously hard to virtualize – has been solved thanks to CPU-instruction-extensions from AMD and Intel, i.e AMD-V and Intel VT-x respectively [5, page 225]. These extensions make it possible for hypervisors to run unmodified guest operating-systems with minimal overhead. Intel's and AMD's respective 64-bit architecture extensions are also supported for virtualization (and migration between the two platforms is possible).

In this paper I have decided to focus my research on a single virtualization technology, called KVM (kernel based virtual machine). KVM is integrated into the linux kernel and leverages as much of the existing kernel-functions as possible, improving performance and streamlining the development process [6, page 37]. KVM is open source, well documented and free to use and modify. Practically a user-space is set up and managed by a modified version of Qemu (a popular x86 emulator), and the hardware-capabilities of the host are accessed though the /dev/kvm device [7, page 226]. A disadvantage of KVM is that its (obviously) tied to the linux-platform. The KVM module is included in the 2.6.x kernel, and features unique migration features [8].

Network support in KVM is inherited from the Qemu emulator and features a wide range of options. A standard option is Usermode Network Stack, that simply embeds a implementation of IP, TCP, UDP, DHCP, etc. within the Qemu process itself. This has been heavily criticized, but proved to be quite functional for simple applications [9]. The most advanced way to handle network communication between guests, the host, and the external network is to use the tun/tap kernel drivers. TAP-devices work in a similar way as IP-tunnels, but on data-link level. Live migration is supported by KVM, according to the following algorithm [10].

Setup > Transfer memory > Stop the guest > Transfer state > Continue the guest

In the "Transfer memory" phase almost all of the Virtual Machine's virtual memory is transferred, while the virtual machine is still running. In the "Transfer state"-phase the final parts of the memory (that have been changed since the main transfer) are copied. The memory read/write instructions are monitored via dirty page logging to enable this type of functionality.

KVM supports compression of the data-stream and several types of encryption. The migration process does not feature any support for handling the network-related changes for the guest, and the process is designed to take place in a separate subnet. Apart from automatically broadcasting a gratuitous ARP-reply, all other network-related issues need to be handled by the administrator. This makes it a challenge to migrate virtual guests to a new subnet, especially without resetting existing TCP-connections. However, several solutions have been proposed.

2.2 Status quo

One "solution" would be to simply ignore the TCP-issue. Instead focus would only have to be put on either giving the client a new IP address as fast as possible. This can be achieved by configuring a DHCP server with short IP-lease times. The administrator could also use static IP's and identical

network topologies before and after migration. However, this would result in all TCP-connections being reset. For a FTP-server, ERP-server, or just a server being administered via a remote desktop-protocol this would yield very disturbing results for the end user, since the communication would be reset.

For desktop systems, low utilizations servers for backup-purposes a migration without any action to ensure the protection of active sessions would be acceptable. For most modern applications however, this is not the case.

2.3 Using Visualization Devices

A team of researchers from Nortel Labs, Northwestern University, and Universiteit van Amsterdam released a paper in 2006 that addressed live migration of virtual systems [11, page 6]. Part of the paper was dedicated to live migration over WANs. In order to keep TCP-sessions active, the network traffic was tunneled to a "visualization device". The visualization device has the same function as the home-agent in the mobile IPv4 standard (described in section 2.5). It not migrated, has a static IP, and forwards the traffic to the virtual machine via an IP-tunnel. In summary, IP-tunnels work by encapsulating IP-packets in a new IP-packet (IP in IP) [12]. The visualization keeps track of the real IP of virtual machine (for example by running a special service in the virtual operating system that sends heartbeats to the visualization device). It encapsulates traffic for the virtual machine with packets addressed to current host of the virtual machine. This host decapsulates the packets and delivers them to the guest NIC. Traffic in the other direction works in the same way, it is sent via the tunnel and seems to be delivered from the visualization device.

Since all external traffic goes through the visualization device, and is then dynamically redirected to the current IP of the virtual machine, TCP-sessions do not have to be reinitialized when the virtual machine migrates. The paper delivers a nice analogy where you think of an IP-tunnel, with one end statically attached to the visualization device, and the other end dynamically attached to different hosts as the virtual machine moves around.

In my opinion the main advantage of using a visualization device is simplicity. The host-system will always connect to the same visualization-device making the scripting-process quite simple. Having a static visualization device also makes route-optimization easier since data will always be forwarded through a maximum of one tunnel. Disadvantages include the fact that if the visualization-device fails the virtual machine will be rendered useless. Furthermore, if the route to the visualization device has a low QoS, the visualization device would become a bottleneck. Therefore the geographical position of the visualization device and the infrastructure supporting it is of great importance.

2.4 Using a "temporary redirection scheme"

In 2007 a group of researchers from Deutsche Telekom Laboratories published a paper with a different approach. Instead of routing an IP-tunnel to a static visualization-device, an IP-tunnel is set up between the old host and the new host. Old TCP-sessions will be forwarded via the IP-tunnel, while new sessions are set up via the new host-IP [13, page 4]. This system is more flexible and redundant than the system with visualization devices since no central node exists. Less hardware overhead is required since no visualization device needs to exist. Disadvantages include the fact that this solution would yield a complex net of IP tunnels if a server is migrated several times. Furthermore security could be an issue in this type of implementation. The numerous tunnels could make the data-stream venerable to packet sniffing; man-in-the-middle attacks, etc.; meaning that the use of encryption is important. This can be handled via IPSec tunnels at the network layer level for content protection

though the tunnels. An even more secure approach is use SRTP [14] at the application level layer in order to provide application to application security; this could work especially well in the case of container based virtualization. However implementations such as the one conducted in this paper feature the guest VM bridged virtually at the data link layer, meaning that the RTP-implementation would have to be done within the Guest VM itself. If the end-user decides to run an unmodified operating-system on his VM, he would not be getting encryption automatically, which is a problem.

The authors of the research-paper omit one very important factor in their method description. They fail to describe how they implemented the separation of new TCP-connections established after migration (that are associated with the new external IP), and the already active TCP-connections (that are associated with the old IP) and routed through the tunnel. One alternative would be to use multiple NICs in the guest, that would however require guest OS networking modifications during the migration (for virtual interfaces), or a large number of NICs (emulated by KVM) for consecutive migrations. It is very hard to increase the number of NICs after the virtual machine has been initialized. The other alternative that I looked into when doing my test implementation was state(policy)-routing using iptables, this only works for TCP, and would yield an immensely complicated routing table in the host.

2.5 Mobile IPv4

The visualization implementation is very similar to mobile IPv4-protocol [15]. In the mobile IPv4-protocol each mobile node has a home agent that receives all communication. When a mobile node moves away from its home-node, it communicates with the foreign agent in the new network, and a tunnel is set up between the home agent and foreign agent. For the virtualization-case the host-systems could act as the home agent and foreign agents.

In order to work in compliance with the mobile IPv4 protocol however, some overhead-code (the agents) is needed that could seem unnecessary in a situation where a single node is to be migrated. The size of the overhead is however very manageable since Mobile IP is designed for use in mobile hardware with limited capabilities. For most corporations, and service providers, an IPv4 implementation is beneficial, since a large number of migrations need to be supported, and code can be reused. Finally, if the vendors of integrated virtualization-solutions decided to use mobile IPv4 instead of vendor-specific solutions, the horizon for migration between different hypervisors would improve.

2.6 Mobile IPv6

Mobile IPv6 is has the ability to facilitate the migration of virtual machines [16]. To simply implement IPV6 is not sufficient; in order to be able to migrate over the boundaries of subnets a home agent (with an IPv6 address that the node is always reachable via) is needed. Otherwise connections will break ungracefully when the node receives a new IPv6 address. Improvements compared to mobile IPv4 include NAT-related problems disappear, since there are enough (128-bit) addresses to give virtual machines globally unique addresses. Depending on implementation, VMs are usually bridged with the eth0 interface of the host, and therefore receive a node-adress within the same subnet as the host. The simplest way for the virtual NIC to receive a new IPv6 address after migration is via DHCP with very low lease-times. Furthermore, thanks to movement detection in the nodes via the IPv6 router advertisement foreign agents are not needed; the data can be decapsulated directly in the mobile node. Less noticeable improvements that would be beneficial in large scale implementations include better support for route optimization and reverse tunneling.

There are however few up-to-date open source implementations of the mobile IPv6-protocol available. The Helsinki Institute of Technology created the MIPL-implementation, that was designed for a patched 2.4.22 Linux kernel [17]. The same university also has an open-source implementation named Dynamics available for download, however with sketchy documentation [18]. On the BSD-front support seems to be a bit better – NEC and Ericsson have implementations available via license [19]. With the proper research and implementations mobile IPv6 has the potential to be the standard protocol for mobile nodes in the future. At the time of publication of this paper, the implementations are not mature enough to support real applications.

2.7 HIP – Host identification protocol

A promising complement to Mobile IPv6 is the HIP protocol [20]. HIP addresses the problem of using IP addresses as both locators in network topologies, and network interface identifiers. By adding a host identification layer between the transport layer and network layer, it would be possible to identify and track virtual (and mobile) nodes in a simpler way. TCP sessions would use the Host identity to specify what interface (virtual machine) it wants to communicate with, and the underlying HIP-infrastructure would translate this into the actual IP, that is used to locate the node in the network topology. This seems to be a very promising standard, but at the same time it seems quite unfeasible that the change of introducing a new layer in the internet-stack could be achieved within a reasonable time-horizon. Even though HIP seems to be a very good idea, the Mobile IPv6 protocol has the advantage of being a natural complement to the IPv6-standard that is reaching a rising acceptance level (especially in Asia) [21, page 32-40]. The reason I took interest in the protocol is that it seems that TKK have abandoned its mobile IPv6-implementation for infraHIP – a quite mature HIP infrastructure implementation project [22].

3. Making it real – Implementation test

3.1 Justification / Goals

As stated before, quite a few implementations of WAN-migrations exist, with a varying degree of documentation. In the interest of getting a better idea of the opportunities and challenges regarding live migrations of virtual machines over wide area networks, I decided to create a simple test-environment. The goal of the implementation was not to benchmark ground-breaking performance, nor to design a Mobile IPv6-killer protocol. My aim was instead to demonstrate that exciting protocols such as the redirection scheme proposed by Deutsche Telekom Labs, and Mobile IPv4/6 are both feasible, realistic, and can be supported by existing network infrastructure and hypervisor technology.

3.2 Method

I decided to create the environment on a linux platform, using the KVM hypervisor. I chose the Linux platform because of the superior tools for network configuration, the transparent open-source architecture, and the opportunity to create powerful scripts efficiently. I chose KVM mainly because it is well-documented open-source, has leading migration features [23], and has very customizable networking features thanks to the Qemu front-end and the TUN/TAP drivers.

My initial plan was to script in python with the libvirt API python bindings [24]. I quickly decided to abandon this plan because of lacking support for KVM's migration features in the libvirt API, instead I decided to use simple shell-scripting as much as possible. The issue of automating the interaction with the KVM-console was solved by simply routing the Qemu console to a temporary unix-socket, and using SOCAT [25] to send/receive commands.

The implementation requires at least two Intel VT-x (or AMD-V) enabled host-machines. My first plan was to connect my hosts to the tele2-UMTS network and migrate over a wireless connection. Sadly I had to abandon the idea; there are too many unknown factors in their network (including their usage of NAT). Instead I decided to use a PC-router running Slackware Linux, interconnecting the two machines, to represent an ISP.

The implementation logic is shown in Appendix A.2. Highlights include the heavy reliance on bridging TAP-interfaces (link level tunnels) together with virtual bridges in order to create advanced virtual network topologies. I decided to load the virtual routers from a virtual memory image (pseudo migration), this improves start-up times significantly. Both the virtual routers and the virtual guests use live-cd linux-technology, meaning that they need no hard-drive image access via nfs. The main reason for this decision was not the issue of setting up a NFS-server. The qcow harddrive-image format has a tendancy of becoming corrupt when the KVM-process is terminated (the analogy is pulling the plug on a physical machine), which often happens in the testing-environment – thus it was useful to not have an actual virtual hard disk drive. The full script code, and a detailed environment-description is available in appendices A.3 and A.4.

The main difference between my implementation and the Deutsche telekom-implementation (that served as a initial point of reference) is the fact that all the traffic related to the client is tunneled to the initial host – I did not manage to implement policy routing in a manner sophisticated enough to allow for the separation of established and new TCP-sessions.

3.3 Benchmarking results

After successfully scripting and setting up a successful environment, my first test was simple: to migrate a virtual machine with an active SSH-session without resetting the connection. In order for the session to be able to be sustained and reestablished directly from the guest at its new location source routing is required, this was something I initially overlooked. After this tweak the migration worked seamlessly from the virtual guest, TCP-sessions were sustained during migration.

After completing my initial goal I decided to benchmark the performance and availability of the guest using iperf [26] in tcp-mode. I used an extra virtual guest in the initial host as a client, and the migrating guest as the server.

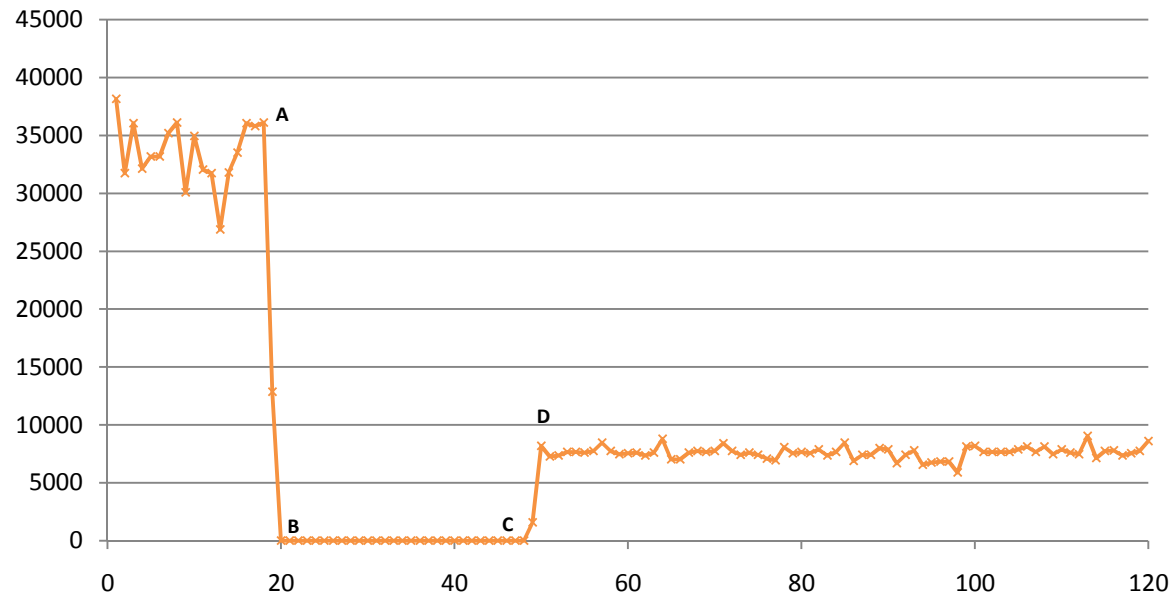


Figure 1: Virtual Guest throughput (Kbit/s) as a function of time (s)

The large downtime (between point B and C) can be traced to the fact that the tunnels and routes are initialized directly following migration decision (at point A), combined with the time required for the network topology changes to take effect. I found no simple, reliable way to get a migration confirmation from the KVM monitor. The bad throughput after migration (at point D) is most probably a consequence of the questionable NICs used in the PC-router (acquired pre-2001, benchmarked to around 10Mbit/s bandwidth). More detailed output is available in Appendix A.5.

3.4 Implementation Analysis

The implementation does what it is supposed to do, even though the performance is not earth-shattering. It demonstrates that seamless live migrations of virtual guest operating systems while maintaining active TCP-connections are quite feasible, and also quite easy to implement in a scalable manner. I've been thinking quite about a lot about ways to minimize the server downtime, such as calculating an estimated migration time[1], flooding the socket with info-requests, etc. Asynchronous notification is not an option since KVM does fails to route any console output to the socket upon successful (or failed) migration, the VM is simply paused if a successful migration takes place. The purpose of the implementation was a proof of concept, so I will leave this as a recommendation for further study (or API-binding project), along with an encryption implementation.

The scalability of this implementation should be quite good, compared to DNAT/SNAT-solutions that could work equally well in a small environment. It also has the potential to follow the mobile IPV4-protocol more strictly with further modifications. During the process of designing and scripting the implementation I encountered several problems, almost all of them related to NAT. After going through this process I have been convinced that an IPv6-transition is crucial for the continued development of the internet.

[1] Iissue a "sleep"-command to the script after migration, for a time period equaling (guest RAM-size / NIC-throughput).

4. Conclusion

After reviewing the literature, and implementing my test, I am convinced that live migration of virtual machines with sustained TCP-connectivity is a problem that can be solved quite easily. However, for this type of migration to get a higher level of acceptance and utilization, more research and development needs to be conducted. There are two areas of research I believe crucial for continued innovation within the field. A standardized and accessible solution for the issue of separating outgoing traffic, needs to be developed (switching to mobile IPv6 may be a good solution). More importantly a much more robust and hypervisor independent virtualization API needs to be developed and maintained. In my opinion, the libvirt API does not offer the capabilities and configurability needed to be able to program an efficient migration-script, because of insufficient bindings. The API is however a step in the right direction, and the ability for programmers to make simple but robust hypervisor-independent GUI-frontends and scripts will make the live-migration process much more accessible to system administrators and the general public in the future. If organizations like SUN Microsystems (that are already heavily involved in open source development of their Virtualbox decided to) endorse and support a single platform-independent API, the standardization process could perhaps be speeded up. Access to a suitable API could have optimized the downtime of my implementation considerably.

It is interesting that the problem of mobility – that has been addressed by mobile service providers for decades – is now beginning to be applicable to virtual machines (VMs) and processes moving between physical hosts. The fact that IP-addresses are used both as locators in the network-topology and interface identifiers, is becoming a problem (once again). The IPv6-solution is a step in the right direction thanks to the huge increase in the number of available IP addresses. However, I believe that the HIP-layer has quite a few advantages that are needed for a streamlined development-process of cloud services, and that this protocol is worth keeping an eye on during the upcoming years.

The development of cloud-systems such as Microsoft's Azure and Amazon EC2 represents only the tip of the ice berg regarding what is possible thanks to VM (and process) migration. In the future, I believe that CPU-cycles and bandwidth will be commodities, just like electricity – with clear and important market prices. Combined with SaaS-offerings[2] and SOA-service[3] contracts, the "main thought of the year" for the past few years– that our computing needs will be moving into "the cloud", might become a reality in the upcoming decade.

[2] Software as a Service – offering software online for a license fee.
[3] Service Oriented Architecture Service Contract – Standardized contracts for the specifications of services.

Appendix A.1 – References and Software used

A1.1 References

[1] www.computerworld.co.nz, 2009-08-30
http://computerworld.co.nz/news.nsf/tech/EDAD5A16489D8FB4CC25721B0006B9EA

[2] G. Q. Maguire Jr. and J.M Smith, *Process migration: Effects on Scientific Computations, Columbia University, New York, 1988*

[3] Peterström Dan, *IP Multimedia for Municipalities: The supporting architecture, Royal Institute of Technology,* Stockholm, August 2009

[4] Christopher Beal, *xVM Server and xVM Virtualbox*, (published online by Sun Microsystems), http://opensolaris.org/os/project/losug/files/October2008/xVM-losug-oct-08.pdf 2009-08-30

[5] Kivity Avi, Kamay Yaniv, Laor Dor, Lublin Uri, Ligouri Anthony, "KVM: the Linux Virtual Machine Monitor" published in *proceedings of the Linux Symposium Volume one*, Ottawa Canada, 2007

[6] Shah Amit, "Deep Virtue: Kernel Based Virtualization with KVM", published in *Linux Magazine,* Munich, 2009

[7]Kivity Avi, Kamay Yaniv, Laor Dor, Lublin Uri, Ligouri Anthony, "KVM: the Linux Virtual Machine Monitor" published in *proceedings of the Linux Symposium Volume one*, Ottawa Canada, 2007

[8] www.linux-kvm.org, 2009-08-30
www.linux-kvm.org/page/Migration

[9] www.nongnu.com, 2009-08-30
www.nongnu.com/qemu

[10] www.linux-kvm.org, 2009-08-30
www.linux-kvm.org/page/Migration

[11] Travostino Franco, Daspit Paul, Gammans Leon, Joj Chetan, de Laat Cees, Mambretti Joe, Monga Inder, van Oudenaarde Bas, Raghunath Satish, Wang Phil, "Seamless Live Migration of Virtual machines over the MAN/WAN", published in "*Future Generation computer systems*" USA, 2006

[12] lartc.org, 2009-08-30
http://lartc.org/howto/lartc.tunnel.ip-ip.html

[13] Bradford Robert, Kotsovinos Evangelos, Feldmann Anja, Schiölberg Harald, *Live Wide-Area Migration of Virtual Machines Including Local Persistent State,* Deuche Telecom Laboratories, 2006

[14] Bauger M, Mcgrew D, Naslund M, Carrara E, Norrman K, Ericsson Research, Cisco Systems Inc, "The Secure Real Time Transport Protocol", *RFC 3711*, Stockholm, Sweden, NYC, USA, 2004 http://www.ietf.org/rfc/rfc3711.txt 2009-08-30

[15] Perkins C (Ed.), "IP Mobility support for IPv4", *RFC 3344*, Mountain view USA, 2002 http://www.ietf.org/rfc/rfc3344.txt, 2009-08-30

[16] Johnson D, Perkins C, Arkko J, "Mobility support for IPv6", *RFC 3775* Houston USA, Mountain View, USA, Jorvas, Finland, 2004
http://www.ietf.org/rfc/rfc3775.txt, *2009-08-30*

[17] http://fivedots.coe.psu.ac.th, 2009-08-31
http://fivedots.coe.psu.ac.th/~ple/ipv6/mipl/mipl-howto.html#setup

[18] dynamics.sourceforge.net, 2009-08-30
http://dynamics.sourceforge.net

[19] www.eurescom.de, 2009-08-30
http://www.eurescom.de/~public-web-deliverables/P1100-series/P1113/D1/41_mip.html

[20] Moscowitz R, Nikander P, Jokela P, Henderson T, "Host Identity Protocol", *RFC5201*, Jorvas, Finland, Seattle, USA, 2008
http://www.ietf.org/rfc/rfc5201.txt, 2009-08-31

[21] Hain Tony (Cisco Systems), "IPv6 Acceptance", published at *IPv6 Forum Beijing*, China, 2008
http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6553/Tony_Hain_IPv6Forum_200804151.pdf, 2008-08-31

[22] Infrahip.hiit.fi, 2009-08-30
http://infrahip.hiit.fi/index.php?index=how

[23] www.linux-kvm.org, 2009-08-30
www.linux-kvm.org/page/HOWTO

[24] Libvirt – Virtualization API
www.libvirt.org, 2009-08-30

[25] Socat – Multipurpose relay
www.dest-unreach.org/socat, 2009-08-30

[26] iperf – measure TCP performance
http://sourceforge.net/projects 2009-08-30

A.1.2 Software Used

Ubuntu Linux – Debian Based Linux Distrobution
www.ubuntu.org, 2009-08-30

Knoppix Security Tools Edition – Live CD with useful network tools
www.knoppix-std.org 2009-08-30

Scientific Linux – Live CD with iperf pre-installed
www.scientificlinux.com 2009-08-30

Damn Small Linux – Live CD, smaller than 50Mb
www.damnsmalllinux.org, 2009-08-30

KVM – Kernel-based Virtual Machine
www.linux-kvm.org, 2009-08-30

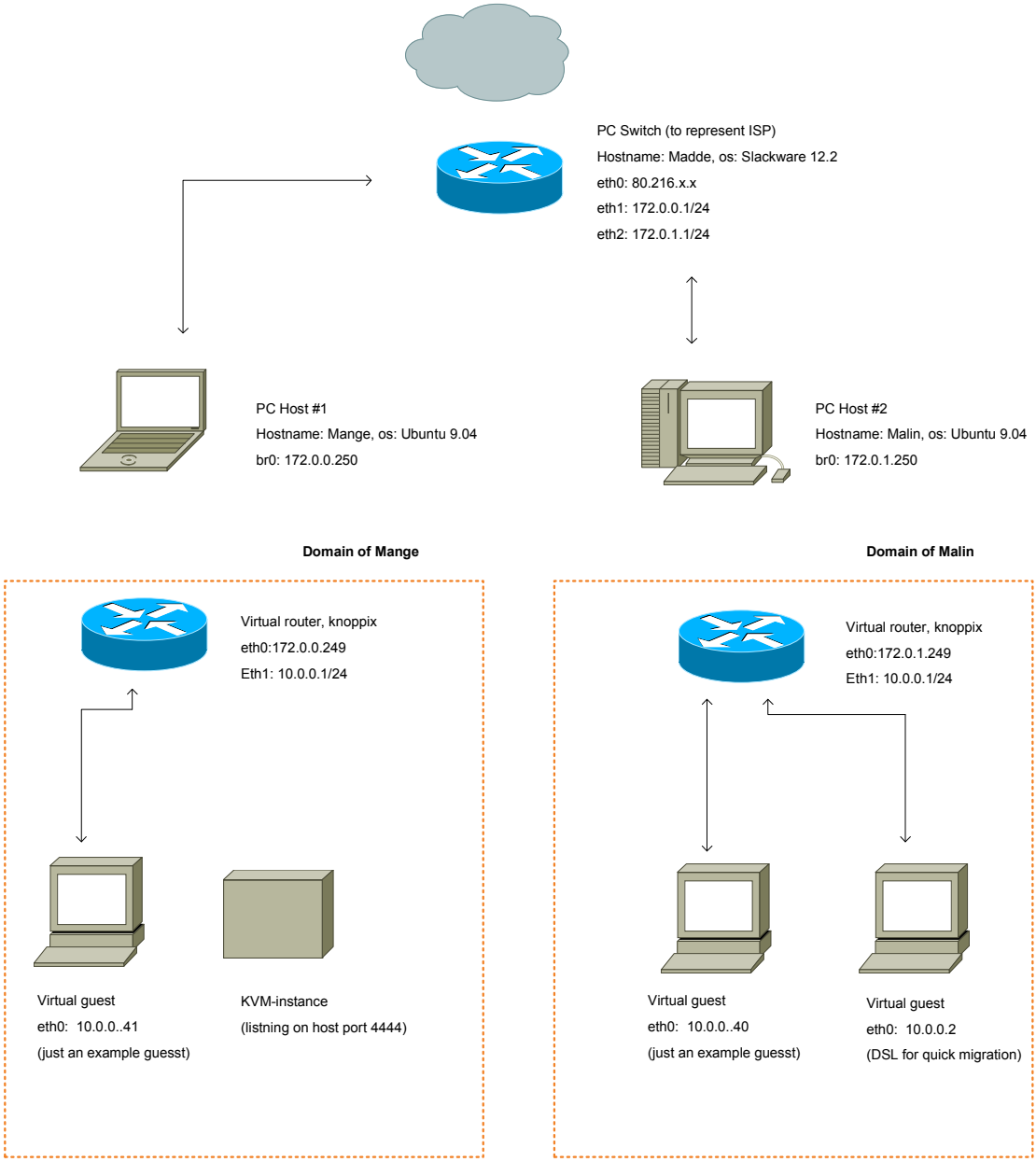
iperf – measure TCP performance
http://sourceforge.net/projects 2009-08-30

TUN/TAP Kernel Drivers
http://vtun.sourceforge.net/tun 2009-08-30

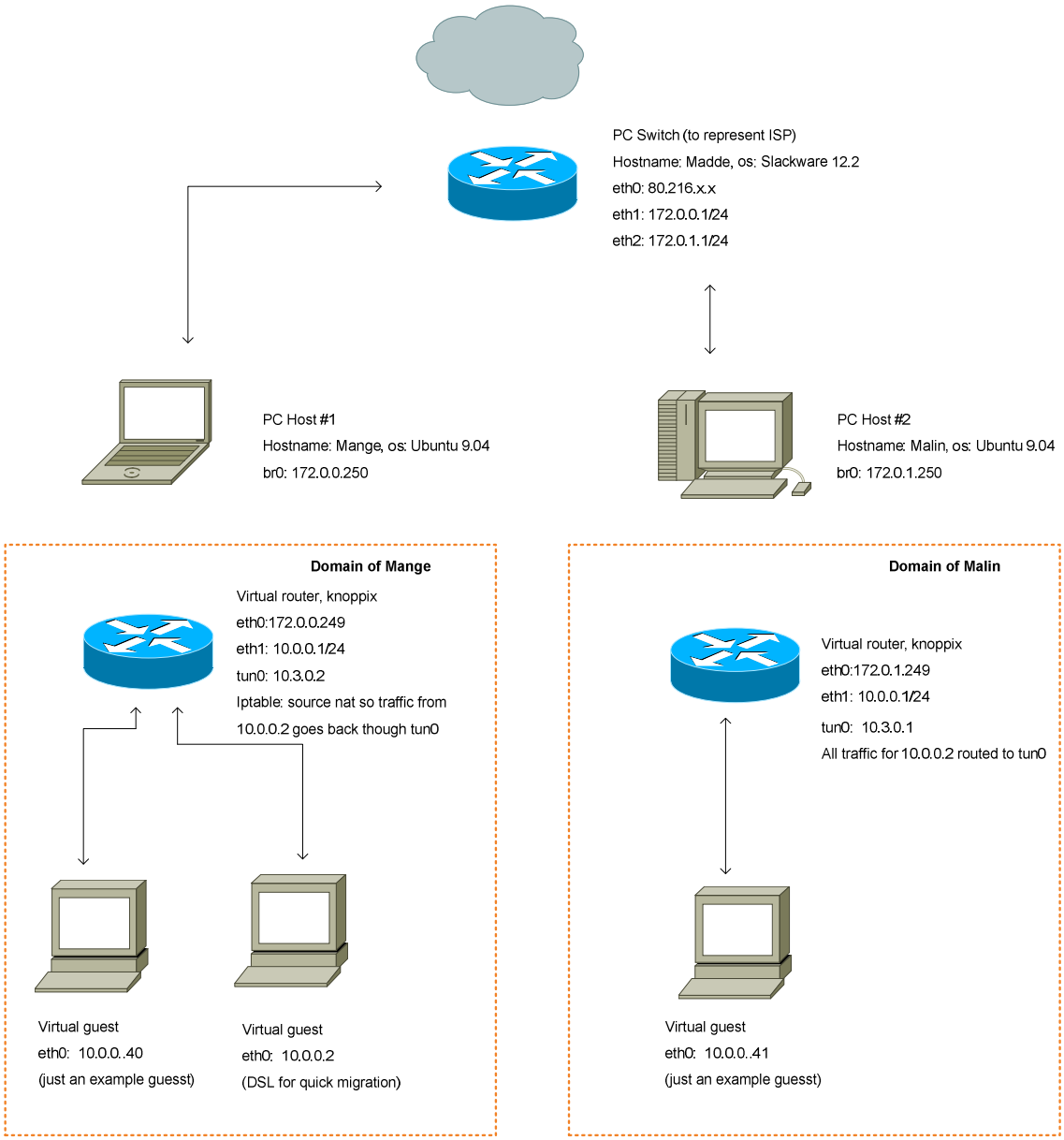
Appendix A.2 Implementation Logic

Appendix A.3 Network Topology

A.3.1 Before Migration



A.3.2 After migration



A.3.3 Hardware Used

Malin

Intel Core2Quad Q9400
4 GB RAM (only 3.2 GB addressed by OS)
Asus P5Q-motherboard
Nvidia-graphics
Ubuntu 9.04 – 32bit

Mange (Dell XPS M1330)

Intel Core2Duo T7250
2GB RAM
Generic Dell motherboard
Nvidia Graphics
Ubuntu 9.04 – 32bit

Mange

Intel Celeron 300 Mhz @ 466 Mhz
64 MB RAM
Unknown motherboard
Integrated Graphics
Debian Linux (minimal installation)
2x NIC – Unknown vendors and speed

Apendix 4 – Script

```
#!/bin/sh
#########################################################################################################################
# Live migration script for KTH course IK1550 (Internetworking)
# Elis Kullberg, August 2009
# Appendix to final report
#
# Before starting
#  -br0 needs to be configured in /etc/network/interfaces (and bridged with your physical nic)
#  -tun/tap and socat packages needed
#  -rp_filter should be off (echo 0 > /proc/sys/net/ipv4/conf/all/rp_filter as root)
#  -ip-forward should be activated (echo 1 > /proc/sys/net/ipv4/ip_forward as root)
#  -Script tested in Ubuntu 9.04
#  -Intel-VX or AMV-V required... otherwise change "kvm"-commands to qemu (significantly lowers performance)
#  -script designed to be run with root-privelege
#
# Example router-images and summary of configuration available on www.eliskullverg.com/livemigrationdocs
#
# A Thank You to the Qemu support forum (http://qemu-forum.ipi.fi/) and the Ubuntu user community (http://help.ubuntu.com)
#
# Available under Creative Commons ShareAlike 3.0 License - http://creativecommons.org/licenses/by-sa/3.0/
#########################################################################################################################

USERID=`whoami`

startup()
{
# Thanks to author of https://help.ubuntu.com/community/KVM/Directly
# Add TAP-devices, belonging to active user
# Initialize two TAP-devices belonging to me
# Also initialize one TAP-device for each virtual guest

iface1=`tunctl -b -u $USERID`
iface2=`tunctl -b -u $USERID`
iface3=`tunctl -b -u $USERID`
iface5=`tunctl -b -u $USERID`

# Add first tap-device to existing bridge (connected to the world)
ifconfig $iface1 0.0.0.0 up
brctl addif br0 $iface1

# Add second TAP-device to a new bridge
ifconfig $iface2 0.0.0.0 up
brctl addbr br1
ifconfig br1 0.0.0.1 up
brctl addif br1 $iface2

# Add more TAP-devices (depending on desired number of guests)
ifconfig $iface3 0.0.0.0 up
brctl addif br1 $iface3
ifconfig $iface5 0.0.0.0 up
brctl addif br0 $iface5

# Generate some random macs
# A special thanks to pheldens @ qemu forums for this (released via help.ubuntu.com license)
ranmac1=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac2=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac3=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)
ranmac4=$(echo -n DE:AD:BE:EF ; for i in `seq 1 2` ; \
do echo -n `echo ":$RANDOM$RANDOM" | cut -n -c -3` ;done)

# Start our virtual router from RAM-dump.
# It will obtain external IP from external router (or ISP)
# Internal IP will be 10.0.0.1
# We need static MAC addresses in order to resume a virtual dump glitch-lessley
```

```
gzip -c -d /home/elis/skola/IK1550/testenv/KSTD.gz | kvm -m 512 -net nic,vlan=0,macaddr=$ranmac1 -net tap,vlan=0,ifname=$iface1 -net nic,vlan=1,macaddr=$ranmac2 -net tap,vlan=1,ifname=$iface2 -cdrom /home/elis/skola/IK1550/testenv/knoppix-std-0.1.iso -incoming "exec: cat"&

# Start other virtual guests
# These need to be configured with static IP's in same subnet as virtual router (DHCP in virtual router - risk for IP-crash after migration!)
# ifconfig eth0 10.0.0.x broadcast 10.0.0.255 netmask 255.255.255.0, route add default gw 10.0.0.1, should do the trick
startoptions="-localtime -m 512 -cdrom /home/elis/skola/IK1550/testenv/mini_livecd_SL53_2009-03-26.iso"
incomingoption="-incoming tcp:0:4444"
# Initialization of guest #1 (this is the one that will be migrated)
# Thanks francesco @ qemu forums for info on syntax for routing the Qemu monitor to a unix socket
kvm -net nic,vlan=3,macaddr=$ranmac3 -net tap,vlan=3,ifname=$iface3 $startoptions -monitor unix:/tmp/socket,server,nowait&
# Print the mac-adress of the first machine - to make identification possible from within the virtual OS
echo macaddr=$ranmac3
# Initialization of guest #2
kvm -net nic,vlan=4,macaddr=$ranmac4 -net tap,vlan=4,ifname=$iface5 $startoptions&
# For let any process or person acces to socket-file that we routed the Qemu monitor to
# chmod -v 0777 /tmp/socket (optional, not needed if script run as root)
}

listen()
{
# Get the router IP at sending site
echo "Enter senders router IP"
read senderip
# Also current host router IP is needed
echo "Enter current router IP"
read receiverip
# Get guest IP
echo "Enter the guest IP"
read guestip

# Create TAP Device
iface4=`tunctl -b -u $USERID`
ifconfig $iface4 0.0.0.0 up
# Add it to the second bridge
brctl addif br1 $iface4

# Set up receiving side of the tunnel
# Need to configure autologin to server (ssh)
ssh knoppix@172.16.0.249 "sudo ip tunnel add tunnel mode ipip remote $senderip local $receiverip dev eth0 ttl 255"
ssh knoppix@172.16.0.249 "sudo ip link set tunnel up"
ssh knoppix@172.16.0.249 "sudo ip -family inet addr add 10.3.0.2 peer 10.3.0.1 dev tunnel"
# Source routing needed if guest wants to re-initializa communication
ssh knoppix@172.16.0.249 "sudo ip route add default via 10.3.0.2 dev tunnel table 200"
ssh knoppix@172.16.0.249 "sudo ip rule add from $guestip table 200"

#Start kvm in listning-mode
#startoptions="-localtime -m 48 -cdrom /home/elis/skola/IK1550/testenv/current.iso -monitor unix:/tmp/socket,server,nowait"
incomingoption="-incoming tcp:0:4444"
#kvm -net nic,vlan=4 -net tap,vlan=4,ifname=$iface4 $startoptions $incomingoption&
kvm -net nic,vlan=4 -net tap,vlan=4,ifname=$iface4 -localtime -m 512 -cdrom /home/elis/skola/IK1550/testenv/mini_livecd_SL53_2009-03-26.iso $incomingoption&
}


migrate()
{
# Migration requres a destination IP (for the host-system)
echo "Enter destination host IP"
read destip
# Also current host IP is needed
echo "Enter current host-router IP"
read hostip
echo "Enter the upcoming host-router IP"
read nextrouterip
# Finally the guest IP is needed
echo "Enter the guest IP"
read guestip

#Send migration command to QEMU
```

```
echo "migrate tcp:$destip:4444" | socat stdio unix:/tmp/socket

# Set up sending side of the tunnel
# Binding the session-keys to the router is possible for password-less ssh
# Change ip and username if not using example image
ssh knoppix@172.16.1.249 "sudo ip tunnel add tunnel mode ipip remote $nextrouterip local $hostip dev eth0 ttl 255"
ssh knoppix@172.16.1.249 "sudo ip link set tunnel up"
ssh knoppix@172.16.1.249 "sudo ip -family inet addr add 10.3.0.1 peer 10.3.0.2 dev tunnel"
ssh knoppix@172.16.1.249 "sudo route add $guestip dev tunnel"


}

teardown()
{
# This is really useful - cleans up after unsuccesful trials
# Remove all TAP-interfaces
# This is not smooth code, but it works
tunctl -d tap0 &> /dev/null
tunctl -d tap1 &> /dev/null
tunctl -d tap2 &> /dev/null
tunctl -d tap3 &> /dev/null
tunctl -d tap4 &> /dev/null
tunctl -d tap5 &> /dev/null
echo "TAP 0-5 gone"
# Remove redundant virtual bridge
brctl delbr br1
echo "Attempted to remove br1"
}

################################################################################
########### MENU ###############################################################

echo "Welcome, [S]tartup a VM-environment, [L]isten [T]eardown or [M]igrate?"
read selection

case "$selection" in

                "S" | "s" )
                startup
                ;;

                "L" | "l" )
                listen
                ;;

                "M" | "m" )
                migrate
                ;;

                "T" | "t" )
                teardown
                ;;

                                * )
                echo "Bad selection"
                ;;

esac

exit
```

Appendix 5 – iperf session output

```
iperf -c 172.16.1.10  -i 1 -w 2000 -d -f -k

------------------------------------------------------------
Client connecting to 172.16.1.10, TCP port 5001
TCP window size: 3.91 KByte (WARNING: requested 1.95 KByte)
------------------------------------------------------------
[  5] local 10.0.0.2 port 39907 connected with 172.16.1.10 port 5001
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0- 1.0 sec  4656 KBytes  38142 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  1.0- 2.0 sec  3872 KBytes  31719 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  2.0- 3.0 sec  4400 KBytes  36045 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  3.0- 4.0 sec  3920 KBytes  32113 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  4.0- 5.0 sec  4048 KBytes  33161 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  5.0- 6.0 sec  4048 KBytes  33161 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  6.0- 7.0 sec  4296 KBytes  35193 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  7.0- 8.0 sec  4408 KBytes  36110 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  8.0- 9.0 sec  3672 KBytes  30081 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  9.0-10.0 sec  4264 KBytes  34931 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 10.0-11.0 sec  3912 KBytes  32047 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 11.0-12.0 sec  3872 KBytes  31719 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 12.0-13.0 sec  3280 KBytes  26870 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 13.0-14.0 sec  3880 KBytes  31785 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 14.0-15.0 sec  4088 KBytes  33489 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 15.0-16.0 sec  4400 KBytes  36045 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 16.0-17.0 sec  4368 KBytes  35783 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 17.0-18.0 sec  4408 KBytes  36110 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 18.0-19.0 sec  1568 KBytes  12845 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 19.0-20.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 20.0-21.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 21.0-22.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 22.0-23.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 23.0-24.0 sec  8.00 KBytes  65.5 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 24.0-25.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 25.0-26.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 26.0-27.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 27.0-28.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 28.0-29.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 29.0-30.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 30.0-31.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 31.0-32.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 32.0-33.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 33.0-34.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 34.0-35.0 sec  8.00 KBytes  65.5 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 35.0-36.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 36.0-37.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 37.0-38.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 38.0-39.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 39.0-40.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 40.0-41.0 sec  0.00 KBytes  0.00 Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 41.0-42.0 sec  0.00 KBytes  0.00 Kbits/sec
```

```
[ ID] Interval       Transfer     Bandwidth
[  5] 42.0-43.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 43.0-44.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 44.0-45.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 45.0-46.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 46.0-47.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 47.0-48.0 sec  0.00 KBytes  0.00
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 48.0-49.0 sec    192 KBytes  1573
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 49.0-50.0 sec  1000 KBytes  8192
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 50.0-51.0 sec    888 KBytes  7274
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 51.0-52.0 sec    896 KBytes  7340
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 52.0-53.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 53.0-54.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 54.0-55.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 55.0-56.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 56.0-57.0 sec  1032 KBytes  8454
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 57.0-58.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 58.0-59.0 sec    912 KBytes  7471
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 59.0-60.0 sec    920 KBytes  7537
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 60.0-61.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 61.0-62.0 sec    896 KBytes  7340
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 62.0-63.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 63.0-64.0 sec  1072 KBytes  8782
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 64.0-65.0 sec    856 KBytes  7012
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 65.0-66.0 sec    856 KBytes  7012
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 66.0-67.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 67.0-68.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 68.0-69.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 69.0-70.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 70.0-71.0 sec  1024 KBytes  8389
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 71.0-72.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 72.0-73.0 sec    904 KBytes  7406
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 73.0-74.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 74.0-75.0 sec    904 KBytes  7406
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 75.0-76.0 sec    864 KBytes  7078
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 76.0-77.0 sec    848 KBytes  6947
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 77.0-78.0 sec    984 KBytes  8061
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 78.0-79.0 sec    920 KBytes  7537
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 79.0-80.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 80.0-81.0 sec    920 KBytes  7537
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 81.0-82.0 sec    960 KBytes  7864
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 82.0-83.0 sec    896 KBytes  7340
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 83.0-84.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 84.0-85.0 sec  1032 KBytes  8454
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 85.0-86.0 sec    840 KBytes  6881
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 86.0-87.0 sec    904 KBytes  7406
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 87.0-88.0 sec    904 KBytes  7406
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 88.0-89.0 sec    976 KBytes  7995
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 89.0-90.0 sec    960 KBytes  7864
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 90.0-91.0 sec    816 KBytes  6685
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 91.0-92.0 sec    904 KBytes  7406
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
```

```
[  5] 92.0-93.0 sec    952 KBytes  7799
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 93.0-94.0 sec    800 KBytes  6554
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 94.0-95.0 sec    824 KBytes  6750
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 95.0-96.0 sec    832 KBytes  6816
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 96.0-97.0 sec    832 KBytes  6816
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 97.0-98.0 sec    720 KBytes  5898
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 98.0-99.0 sec    992 KBytes  8126
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 99.0-100.0 sec  1000 KBytes  8192
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 100.0-101.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 101.0-102.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 102.0-103.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 103.0-104.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 104.0-105.0 sec    960 KBytes  7864
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 105.0-106.0 sec    992 KBytes  8126
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 106.0-107.0 sec    936 KBytes  7668
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 107.0-108.0 sec    992 KBytes  8126
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 108.0-109.0 sec    912 KBytes  7471
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 109.0-110.0 sec    960 KBytes  7864
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 110.0-111.0 sec    928 KBytes  7602
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 111.0-112.0 sec    912 KBytes  7471
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 112.0-113.0 sec  1104 KBytes  9044
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 113.0-114.0 sec    872 KBytes  7143
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 114.0-115.0 sec    944 KBytes  7733
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 115.0-116.0 sec    952 KBytes  7799
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 116.0-117.0 sec    896 KBytes  7340
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 117.0-118.0 sec    920 KBytes  7537
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5] 118.0-119.0 sec    944 KBytes  7733
Kbits/sec
Waiting for server threads to complete.
Interrupt again to force quit.
[ ID] Interval       Transfer     Bandwidth
[  5] 119.0-120.0 sec  1048 KBytes  8585
Kbits/sec
[ ID] Interval       Transfer     Bandwidth
[  5]  0.0-120.0 sec  141472 KBytes  9657
Kbits/sec
```