

# CAIA Testbed for TEACUP Experiments Version 2

Sebastian Zander, Grenville Armitage  
Centre for Advanced Internet Architectures, Technical Report 150210C  
Swinburne University of Technology  
Melbourne, Australia  
szander@swin.edu.au, garmitage@swin.edu.au

**Abstract**—This technical report describes how we setup a TEACUP v0.8 testbed to conduct automated TCP experiments. Over the last few decades several TCP congestion control algorithms were developed in order to optimise TCP’s behaviour in certain situations. While TCP was traditionally used mainly for file transfers, more recently it is also used for streaming applications, for example YouTube or Netflix [1], [2]. Now there is even an ISO standard called Dynamic Adaptive Streaming over HTTP (DASH) [3]. However, the impact of different TCP congestion control algorithms on TCP-based streaming flows (within a mix of other typical traffic) is not well understood. TEACUP experiments in a controlled testbed allow shedding more light on this issue.

**Index Terms**—TCP, experiments, testbed

## I. INTRODUCTION

Over the last few decades several TCP congestion control algorithms were developed in order to optimise TCP’s behaviour in certain situations. While TCP was traditionally used mainly for file transfers, more recently it is also used for streaming applications, for example YouTube or Netflix [1], [2]. Now there is even an ISO standard called Dynamic Adaptive Streaming over HTTP (DASH) [3]. However, the impact of different TCP congestion control algorithms on TCP-based streaming flows (within a mix of other typical traffic) is not well understood. Experiments in a controlled testbed allow shedding more light on this issue.

This technical report describes how we setup a TEACUP<sup>1</sup> [4] testbed to conduct automated TCP experiments using real machines. This report is an updated version of [5]. Note that in this report we have replaced all host names and public IP addresses our testbed uses with fake names and private IP addresses. The design and implementation of the TEACUP v0.8 software itself is described in [4].

<sup>1</sup> “TCP Experiment Automation Controlled Using Python”

The rest of the report is organised as follows. Section II describes the overall design of the testbed. Section III describes our boot setup. Section IV describes the setup of the testbed hosts that are used as traffic sources and sinks. Section V describes the setup of the testbed router. Section VI describes the setup of other hardware part of the testbed, such as power controllers. Section VII lists a few known issues. Section VIII concludes and outlines future work.

## II. OVERALL TESTBED DESIGN

Our testbed consists of the following hardware:

- 20 PC-based hosts that can be used as traffic sources or sinks;
- Two Mac Minis that can be used as traffic sources or sinks;
- Two PC-based routers that route between testbed networks;
- Two managed switches to which all PCs are connected;
- PC used for testbed control and data collection;
- Five power controllers to control the power of the testbed hosts and routers;
- Two IP KVMs that allow remotely accessing the console of each host and router.

Currently, our testbed is separated into two logical testbeds, where each logical testbed consists of two subnets with PCs and Macs connected by a PC-based router. The separation into two testbeds exists for historical reasons and in the future the configuration could be rearranged easily to have one big testbed. While there are separated experimental networks for both testbeds all hosts are connected to a single control network. The control network and the four different experimental networks are switched Gigabit Ethernet networks, each mapped to a different VLAN on one of our managed switches (one switch is responsible for one logical testbed).

Figure 1 depicts the IP-layer setup of one of the two testbeds (logically both testbeds are identical). All testbed hosts, the testbed router and the data and control server are connected to the common control network (10.1.1.0/24), which is connected to the public Internet via a NAT gateway (running on the data and control server). The NAT gateway connects all testbed hosts to the public Internet. Testbed hosts are connected to a test network 172.16.10.0/24 or a test network 172.16.11.0/24. The testbed router routes between the two 172.16. networks. For clarity we omitted the power controllers and the KVMs in Figure 1.

The 20 PC testbed hosts are installed as triple-boot machines that can run Linux openSUSE 13.2 64bit, FreeBSD 10.1 64bit, and Windows 7 64bit. This allows experiments with all major TCP congestion control variants. The testbed router is dual-boot and runs Linux openSUSE 13.2 64bit or FreeBSD 10.1 64bit. To automatically boot PCs with the required operating system (OS) we use a Preboot Execution Environment (PXE) based booting approach. The two Mac Minis only run Mac OS X and they do not use PXE booting.

The data and control server runs TEACUP [4] to setup and control the experiments. It also runs DHCP and TFTP servers to assign IP addresses for the control network automatically (required for the PXE boot setup that enables us to switch between different OS, see Section III).

Logical testbed 1 uses six HP Compaq dc7800 machines (4GB RAM, 2.33GHz Intel Core™ 2 Duo CPU, Intel 82574L Gigabit NIC for test traffic) as end hosts, and a Supermicro X8STi (4GB RAM, 2.80GHz Intel Core™ i7 CPU, two Intel 82576 Gigabit NICs for test traffic) as the router.

Logical testbed 2 uses fourteen Acer Veriton X6630G (8GB RAM, Intel Core™ i5, Intel 82574L Gigabit NIC for test traffic) as the end hosts and a Supermicro X9SRi-F (16GB RAM, 3.70GHz Intel Xeon® E5-1620v2, Intel I350-T2 dual port Gigabit NIC for test traffic) as the router.

### III. PXE BOOT SETUP

We use a PXE-based boot solution to enable automated switching between OSs on the hosts and the router. All machines have different OSs installed on different hard disk partitions. TEACUP [4] alters the PXE boot configuration files served up by the local HTTP/TFTP

server to each testbed host, thus determining the partition (and therefore the OS) to boot on the fly.

The PXE boot process works as follows. The PXE-enabled client (running on the PCs) contacts the local DHCP server (running on the data and control server). The DHCP server will hand out an IP address and the address of a local TFTP server that provides the boot image. The client will download the boot image from the TFTP server and then boot it. In our case the boot image is GRUB4DOS and the GRUB4DOS configuration selects the hard disk partition to boot from.

We first describe the setup of the different components and then describe the boot process in more detail.

#### A. DHCP server

Our DHCP server runs on the FreeBSD control host. For setting up a DHCP server refer to the technical report [6]. For each PXE-enabled testbed host some DHCP configuration needs to be written. The following example shows the entry for testbed host testhost1 – the entries for other hosts only differ in the MAC and IP addresses.

```
host testhost1 {
    hardware ethernet 00:1e:0b:9d:8f:fb;
    fixed-address 10.1.1.2;
    next-server 10.1.1.254;
    if exists user-class and
        option user-class = "iPXE" {
        filename "conf.ipxe";
    } else {
        filename "undionly.kpxe";
        option routers 10.1.1.1;
    }
}
```

A host is identified by its MAC address (hardware ethernet) and handed out a static IP address (fixed-address). The next-server specifies the TFTP server that serves the boot files, in our case the control host 10.1.1.254. If the client is an iPXE [7] client, it is given the name of the configuration file to download from the TFTP server (conf.ipxe). If the client is not an iPXE client, it is given the name of the iPXE ROM to download from the TFTP server and is also told its gateway. Since our testbed hosts and router are not natively iPXE they will first download the iPXE ROM, reboot, execute the iPXE ROM and then download conf.ipxe from the TFTP server.

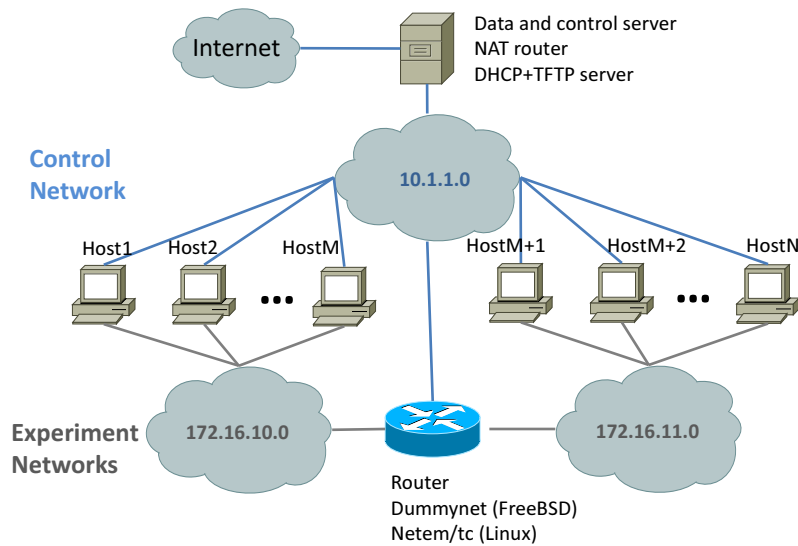


Figure 1. Logical testbed setup

### B. TFTP server

The control host also runs the TFTP server. We enabled the TFTP server as follows. In `/etc/inetd.conf` we removed the `#` in front of the line: `tftp dgram udp wait root /usr/libexec/tftpd tftpd -l -s /tftpboot`. The directory `/tftpboot` is the directory with the boot files. In `/etc/rc.conf` we added `inetd_enable="YES"` to start `inetd` at boot time.

### C. HTTP server

For downloads from within iPXE configuration, we actually decided to use an HTTP server because it is faster and more configurable. We installed `thttpd` from the ports tree, in `/etc/rc.conf` we added `thttpd_enable="YES"` to start `thttpd` at boot time, and we configured it as follows (`/usr/local/etc/thttpd.conf`):

```
user=www
dir=/tftpboot
port=8080
chroot
logfile=/var/log/thttpd.log
pidfile=/var/run/thttpd.pid
```

The iPXE ROM image `undionly.kpxe` and the ipxe configuration files are placed in `/tftpboot` (the iPXE image can be downloaded from [7]). We use GRUB4DOS [8] to select the boot partition. The file `grub.exe` must also be placed in `/tftpboot`.

### D. Clients

We enabled PXE booting in the BIOS. For most of our PCs this was straight-forward, we just needed to put network boot first in the boot options list. For the testbed router, which is a different type of machine, we first had to enable an option "Enable Option ROM". Then after a reboot a new network boot entry appeared in the boot devices list, and we set this as first entry.

### E. Boot process

Assuming PXE boot is enabled in the BIOS, first the client machine will contact the DHCP server. The DHCP server will hand out the client's IP address and tell the client the IP address of a TFTP server. If the client is not an iPXE client (our case) the client will then download the iPXE ROM (`undionly.kpxe`) from the TFTP server, reboot and execute the iPXE ROM. Then the client will download the iPXE configuration file (`conf.ipxe`) from the TFTP server.

The file `conf.ipxe` looks as follows:

```
#!ipxe
echo "Welcome to the IPXE loader"
chain http://10.1.1.254:8080/conf-macaddr_
${net0/mac}.ipxe || goto unknown
:unknown
chain http://10.1.1.254:8080/grub.exe
--config-file="rootnoverify (hd0);
chainloader +1"
```

Based on `conf.ipxe` the client will then attempt to download the file `conf-macaddr_${net0/mac}.ipxe` from the HTTP server, where `${net0/mac}` is substituted by the client's MAC address. If this file is not present the client will download and use `grub.exe` to boot from the first hard disk. This will activate GRUB2 installed in the Master Boot Record (MBR) of the first hard disk, which allows to select the OS via the GRUB2 menu, if one has terminal access. By default after a timeout Linux will be booted.

The host specific ipxe files look as follows. The following is the file for the host `testhost1` (`conf-macaddr_00:1e:0b:9d:8f:fb.ipxe`):

```
#!ipxe
chain http://10.1.1.254:8080/grub.exe
--config-file="root (hd0,2); chainloader +1"
```

In this case the client first downloads `grub.exe` and then uses the GRUB configuration (value of `--config-file`) to boot from the 3rd partition on the first hard disk. This partition has FreeBSD installed and the `chainloader +1` command will invoke the FreeBSD boot loader installed in that partition. Note that in GRUB4DOS (based on GRUB1) the partition numbers start with 0 in contrast to GRUB2 where numbers now start with 1. The file `conf-macaddr_00:1e:0b:9d:8f:fb.ipxe` looks very similar for booting Windows 7. The only difference is that Windows is installed on the first partition. Again, we use the GRUB `chainloader` command to hand over to the Windows boot manager.

```
#!ipxe
chain http://10.1.1.254:8080/grub.exe
--config-file="root (hd0,0); chainloader +1"
```

For booting Linux, the file `conf-macaddr_00:1e:0b:9d:8f:fb.ipxe` looks a bit different. Not only does the partition number change (Linux is installed on the second partition), but also for Linux we use the GRUB commands `kernel` and `initrd` to specify the Linux kernel and the initial RAM disk to boot. One could install a Linux boot loader, such as GRUB2, in the second partition and hand over with the `chainloader` command (as for FreeBSD and Windows). However, our approach has the advantage that we can select the Linux kernel to boot from the iPXE file.

```
#!ipxe
chain http://10.1.1.254:8080/grub.exe
--config-file="root (hd0,1);
kernel /boot/vmlinuz-3.17.4;
initrd /boot/initrd-3.17.4"
```

Our TEACUP control framework [4] writes the `conf-macaddr_${net0/mac}.ipxe` files based on a template and the user's choice of OS and then sends the shutdown command to the machines to be rebooted. The machine will then reboot the desired OS as explained above.

## IV. HOST SETUP

In this section we describe the setup of the testbed hosts. Since we had to install many testbed hosts, we choose the following approach. We installed and configured one host in a virtual machine running under VirtualBox [9]. We then cloned this host to multiple physical machines. After the cloning a few post-installation tasks must be carried out for each OS on each host.

### A. BIOS

PXE boot needs to be enabled. The option is called something like "Enable Option ROM" or "LAN boot". After enabling PXE we need to reboot once before we can select the network card connected to the control network as first boot device. Then under "Boot devices selection" we move the network interface to the first position. We also need to make sure the PCs start automatically after a power outage or power cycle. On our PCs this could be enabled under "Power options" (enable "Turn on after power outage").

### B. Hard disk partitioning

Our disk layout is designed around the 80 GB hard disks in our older hosts. We use MBR partitioning instead of GPT, because it creates less problems with older software, e.g. the PXE boot solution we implemented does only work with MBR partitions since GRUB4DOS cannot handle GPT. Also the advantages of GPT are not really needed in our testbed. With MBR we can only have three primary partitions and booting an OS from an extended partition is problematic. We use the partition layout in Table I. The reasons for putting the swap spaces in extended partitions (instead of making them part of the Linux or FreeBSD partitions) is that then we can still adjust the sizes. It is relatively easy to change the sizes of the last three (extended) partitions and reformat the FAT32 partition in case we need more/less swap space.

First, we used Windows 7 `fdisk` to create the Windows partition. Then after installing Windows we used Linux

Table I  
HOST PARTITION LAYOUT

Type	Size	Purpose
Prim	30GB	Windows 7
Prim	10GB	Linux
Prim	10GB	FreeBSD
Ext	1GB	Linux Swap
Ext	1GB	FreeBSD swap
Ext	28GB	FAT32 data exchange

`fdisk` to create all other partitions. Initially we tried to create all partitions with Linux `fdisk`, but Windows 7 refused to use the Windows partition created.

If Windows 7 is installed into an existing partition it will not create the separate 100 MB System reserved partition it creates when installed on an unpartitioned hard disk. In our case we needed to avoid this extra partition because with MBR we only have three primary partitions (booting from an extended partition is problematic).

### C. Boot manager

We use the GRUB2 boot manager from openSUSE 13.2 (installed into the MBR). The configuration is created with the following command:

```
> grub2-mkconfig -o /boot/grub2/grub.cfg
```

This will add all bootable Linux partitions (in our case the second partition) and all available kernels to the boot menu. At the top level the “openSUSE” entry will boot the default kernel. Under “Advanced options for openSUSE” we can select any of the installed kernels. The first kernel in the list is the default kernel. Bootable Windows 7 partitions are recognised automatically and added to the boot menu, when running the above command. There is no need to manually create boot menu entries for Windows 7. However, GRUB2 does not automatically recognise FreeBSD installations. The FreeBSD partition must be added manually by adding the following to the file `/etc/grub.d/40_custom`:

```
menuentry "FreeBSD 10.1" {
  insmod ufs2
  set root=(hd0,msdos3)
  chainloader +1
}
```

After changing `/etc/grub.d/40_custom` we must rerun `grub2-mkconfig`.

### D. Windows 7 64bit

We installed Windows 7 on the first partition following the Windows setup prompts. We set language to English, set timezone to Sydney/Melbourne, create a user named ‘root’ with the same root password we will use for the root user on Linux/FreeBSD. We skipped security settings (select “Ask me later”), and set location to ‘Work network’. We installed the following applications:

- Chrome
- Wireshark
- WinPcap
- WinDump (put it in a directory in PATH, e.g. C:\Windows)
- Windows updates (all available at the time)
- VirtualBox guest additions
- win-estats-logger (our TCP statistics logger)
- Cygwin 64bit
- Traffic generation tools iperf, httpperf, nttcp (see Section IV-H)
- NTPd (pre-compiled package from <http://www.meinbergglobal.com/english/sw/ntp.htm>)

We installed the following packages for Cygwin (using the Cygwin setup tool<sup>2</sup>):

- cygrunsrv
- openssh
- wget
- procps
- psmisc
- make
- autoconf
- automake
- gcc
- lighttpd
- openssh
- pcre-devel (needed by lighttpd)
- patch

Note that on Cygwin `lighttpd` is installed under `/usr/sbin` but that directory is not in PATH for non-interactive shells (and it is hard to add that path as normal user, since files such as `.profile` are ignored by non-interactive shells). We created a symlink so that `lighttpd` is in the PATH for a non-interactive shell:

<sup>2</sup>Command line installation for Cygwin should work as follows: `setup-x86_64.exe -q -P <package>,<package>,...` but in our case this command still opened a pop-up window requiring user input.

```
> cd /usr/bin
> ln -s /usr/sbin/lighttpd lighttpd
```

We configured Windows 7 as follows:

- Turn off firewall
- Set host name
- Configure control NIC with DHCP (and turn off IPv6, NetBIOS, all discovery)
- Configure secondary testbed NIC with static 172.16. address (and turn off IPv6, NetBIOS, all discovery)
- Configure static route into other testbed network:

```
# if host connected to 172.16.10.0
> route add 172.16.11.0 mask
255.255.255.0 172.16.10.1 -p
# otherwise
> route add 172.16.10.0 mask
255.255.255.0 172.16.11.1 -p
```

- Turn off automatic updates
- Turn off automatic go to sleep under power management (set to “Never”)
- Enable remote desktop access
- Disable Windows default SNTP (set W32Time service start to “Disable”)
- Enable installed NTPd with local NTP servers (edit ntp.conf), set iburst option for servers
- Turn off hibernation to remove hibernation file (execute: powercfg.exe /hibernate off)

We configured SSH under Cygwin based on the instructions at [10]:

- Open a Cygwin shell (run as administrator)
- Run: `ssh-host-config -y`
- Type in a password for the user that runs `sshd`
- Run: `mkpasswd -l -u root >> /etc/passwd`
- Login as root: `ssh root@localhost` (home directory and home directory config files are created when logging in for the first time)

### E. Linux openSUSE 13.2 64bit

We selected the Linux partition as installation target. We set language to English, set timezone to Sydney/Melbourne, under Desktop selection selected “Minimal Server Selection (Text Mode)”, and set the root password. In the Installation Settings we clicked on “Software” and in the software selection screen, besides the already selected patterns, we also selected the “Enhanced Base System”, “Network Administration” and “Base Development” patterns. We followed the open-

SUSE installer instructions to set up language, time zone, root user etc.

We installed the following patched kernels:

- Linux kernel 3.16.6 (default for openSUSE 13.2) with web10g patch;
- Linux kernel 3.17.4 (latest stable at the time of installation) with web10g patch.

We also kept the old kernels installed when we installed the first version of the testbed [5]:

- Linux kernel 3.7 (default for openSUSE 12.3) with web10g patch;
- Linux kernel 3.9.8 with web10g patch (at the time the latest kernel for which web10g patches were available was 3.9).

Since the hard disk space is limited on our testbed hosts we built the modified kernels on a different machine with more hard disk space. Then we installed the patched kernels on the testbed machines by copying the kernel and the `initrd` files into `/boot` and the kernel modules into `/lib/modules/<kernel>` (by tar’ing the modules, copy the tar file and then untaring the modules) from the build machine to the testbed host. After copying new kernels GRUB2 must be updated as explained above.

We updated all packages with `zypper update` and installed the following additional applications:

- `lighttp` (`zypper install lighttp`)
- Traffic generation tools `iperf`, `httperf`, `nttcp` (see Section IV-H)

We configured Linux as follows:

- Turn off firewall (in Yast)
- Turn on `sshd` (in Yast)
- Add “PermitRootLogin yes” to `/etc/ssh/sshd_config` (and restart `sshd` as necessary)
- Set host name (in Yast)
- Configure control network interface to use DHCP (in Yast)
- Configure testbed network interface with static 172.16. address (in Yast)
- Set up default route and static routes for testbed networks (in Yast)
- Set up DNS servers (in Yast)

- Disable boot splash (edit `/etc/default/grub` and change `splash=silent` to `splash=0` in `GRUB_CMDLINE_LINUX_DEFAULT`)
- Turn automatic updates off
- Enable NTP with local NTP servers (in Yast)
- Add GRUB2 FreeBSD entry (see Section IV-C)

1) *web10g patch*: We patched all kernels with the web10g kernel patch from [11]. First one needs to copy an existing kernel source or download a kernel source and untar it under `/usr/src`:

```
# copy kernel
> cd /usr/src && cp -a linux-3.16.6-2
linux-3.16.6-2-web10g
# unpack kernel
> cd /usr/src && tar -xvf linux-3.17.4.tar.xz
```

Then we applied the web10g patch, for example:

```
> cd linux-3.17.4
> patch -p1 < /home/'echo $USER'/src/
web10g-userland-2.0.9/web10g-0.11-3.17.diff
```

Then we configured the kernel (copy existing `.config` to new kernel directory and run `make oldconfig`).

Then we ran `make menuconfig`:

- 1) Go to “Networking support”->“Networking options” and enable “TCP: Extended TCP statistics (TCP ESTATS) MIB”. This will automatically enable “TCP: ESTATS netlink module”.
- 2) Go to “General setup”->“Local version” and change the string to ‘-vanilla-web10g’

Then we built the kernel and the modules:

```
> make mrproper
> make
> make INSTALL_MOD_STRIP=1 modules_install
> make install # creates initrd as well
```

Finally, we added the new kernel to the boot manager:

```
> grub2-mkconfig -o /boot/grub2/grub.cfg
```

Since web10g version 2.0.8 the tool `web10g-logger` that logs data for all flows with a configurable frequency in CSV format is part of the standard web10g distribution and no modifications are required for web10g. The following steps are required to install the web10g user space library and applications.

First, we needed to install `libmnl`:

```
> zypper install libmnl0 libmnl-devel
```

Then we built the web10g userland tools:

```
> ./configure --prefix=/usr
> make
> make install
```

The following commands can be used to test if web10g is installed:

```
> modprobe tcp_estats_nl
# list connections
> web10g-listconns
# get state for connection 1
> web10g-readvars 1
# output stats every 500ms
> web10g-logger -i 500
```

## F. FreeBSD 10.1 64bit

First we made the FreeBSD partition usable for FreeBSD. We booted the FreeBSD DVD into `rescue/shell` mode and executed:

```
> gpart modify -i 3 -t freebsd-ufs ada0
> bsdlabel /dev/ada0s3
> newfs /dev/ada0s3
```

Then we booted the FreeBSD DVD again, entered the installation mode and made `ada0s3` the root with type “freebsd-ufs”. We followed the installation instructions, select the “src” and the “ports” packages to install, enabled SSH and NTP daemons, and selected Melbourne/Sydney as timezone.

Then we updated the ports tree with the `portsnap` tool (see FreeBSD handbook on how this is done) and installed the following applications:

- portmaster
- vim (portmaster editors/lighttp)
- lighttp (portmaster web/lighttp)
- Traffic generation tools iperf, httperf, nttcp (see Section IV-H)s

We installed the SIFTR ERTT [12] patch for the SIFTR kernel module (the patch will add the ERTT as another column in SIFTR’s output):

```
> cd /usr/src
> patch < FreeBSD-9.x-10.x_siftr_log_ertt.patch
> cd sys/modules/siftr
> make
> make install
```

We configured FreeBSD as follows:

- Add swap to `fstab`: `/dev/ada0s6 none swap sw 0`  
`0 /dev/ada0s7 /data msdosfs rw 1 2`
- Add “PermitRootLogin yes” to `etc/ssh/sshd_conf` and restart `sshd` (service `sshd restart`)

- Set host name (if not set during installation)
- Edit `/etc/rc.conf` to setup host name, primary and secondary NICs and static routes:

```
hostname="testhost1"
ifconfig_em0="DHCP"
ifconfig_em1="inet 172.16.10.2/24"
defaultrouter="10.1.1.1"
static_routes="inet1"
route_inet1="-net 172.16.10.0/24 172.16.11.1"
sshd_enable="YES"
ntpd_enable="YES"
dumpdev="NO"
firewall_enable="YES"
firewall_type="OPEN"
ntpd_sync_on_start="YES"
```

- Edit `/etc/resolv.conf` to add a search entry for the domain and the name servers
- Edit `/etc/ntp.conf` to setup the local NTP servers

### G. Mac OS X 10.9.5

First we updated our Mac Minis to Mac OS X 10.9.5. Then we configured the following:

- Enabled root login (from the command line use: `dsenableroot`) and `ssh`
- Set the admin user password to same password as root (from the command line execute: `dscl localhost -passwd /Search/Users/System\ Administrator`)
- Disabled power management and enabled "Start automatically after a power failure" via the system config GUI (Energy Saver)
- Turned off "automatic check updates" under System Preferences, App Store
- Turned off the firewall via Set Firewall: Off in the system config (under Security & Privacy)
- Set computer name in the system config (under Sharing) or on command line (`sudo scutil -set HostName <host_name>`)
- Enabled VNC in system config (Sharing, Screen Sharing)
- Turned Wifi off under System Preferences Network Wifi
- Set mouse tracking speed close to fast in System Preferences Mouse
- Created a `/root` directory (not really necessary, only for consistency with Linux and FreeBSD)
- Install working NTP as described in Section IV-G1 below

The following software necessary for TEACUP was installed:

- Install `ipperf`, `httperf`, `nttcp` from patched source (as for other OS, see IV-H); the first configure executed from a shell will trigger the automatic download of the developer tools necessary to compile the traffic generator tools
- Install MacPorts from <http://www.macports.org> (version for 10.9)
- Install `lighttpd` from the ports:
 

```
> port install lighttpd
> cd /usr/bin
> ln -s /opt/local/sbin/lighttpd lighttpd
```
- Install the DTrace-based tcp logger we implemented (`dsiftr-osx-teacup.d`) by copying it to `/usr/bin`

We configured the network interfaces and the static routes:

- Configure control interface with DHCP
- Configure testbed interface with static IP
- Configure persistent static route into other testbed subnet by creating a new script in `/Library/StartupItems` as described below in Section IV-G2

1) *Install working NTP*: Accurate time synchronisation on Mac OSX version 10.9.2 to 10.9.5 does not work because the default installation of NTP is broken [13]. We followed the instructions from [13] to install our own version of NTP and disable the problematic pacemaker application:

- Download the NTP source, build and install it (by default it will be installed under `/usr/local/bin`)
- Modify `/usr/libexec/ntpd-wrapper` to point to new `sntp` and `ntpd` (which are under `/usr/local/bin`)
- Disable pacemaker (`cd /System/Library/LaunchDaemons && launchctl unload -w com.apple.pacemaker.plist`)
- Restart `ntpd` (`launchctl stop org.ntp.ntpd && launchctl start org.ntp.ntpd`)
- Check that new `ntpd` works (with `ntpq -4 -p`)

2) *Creating persistent static routes*: First, create a new directory under `/Library/StartupItems`. Then in this directory created two files named `StartupParameters.plist` and `AddRoutes`, and changed the permissions of the `AddRoutes` script so it is executable:

```
> mkdir /Library/StartupItems/AddRoutes
> cd /Library/StartupItems/AddRoutes
> vi StartupParameters.plist
> vi AddRoutes
> chmod 755 AddRoutes
```

The file `StartupParameters.plist` contains:



```

{
Description = "Add static routing tables";
Provides = ("AddRoutes");
Requires = ("Network");
OrderPreference = "None";
}

```

The file `AddRoutes` contains the following (subnet address and gateway IP need to be adjusted as required):

```

#!/bin/sh
. /etc/rc.common
StartService()
{
ConsoleMessage "Adding Static Routing Tables" \
sudo route add 172.16.10.0/24 172.16.11.1
}
StopService() { return 0 }
RestartService() { return 0 }
RunService "$1"

```

## H. Traffic generators

This section provides detailed information on how we installed the modified `iperf`, `httperf` and `nttcp` tools we use as traffic generators.

1) *iperf*: This sub section explains how to compile and install `iperf` with the latest CAIA patch.

Download and unpack `iperf` source 2.0.5 from [14]. Install `gcc`, `make`, `automake` and `patch` (if not installed). Download the CAIA patch for `iperf` 2.0.4 from [15] and patch the `iperf` 2.0.5 sources:

```

> cd <iperf_source_directory>
> patch -p0 < <path_to_caia_iperf_patch>

```

Configure `iperf` (set the prefix to `/usr` so `iperf` executable is installed in `/usr/bin/`, because Cygwin does not have `/usr/local` in the `PATH` by default). Note that on Cygwin we need to replace the `config.guess` from the `iperf` source with the `config.guess` installed on the system under `/usr/share` (check with `automake --version` which `automake` version is used). Then we built `iperf` as follows:

```

# copy config.guess necessary on Cygwin
> cp /usr/share/automake-1.9/config.guess
> ./configure --prefix=/usr
> make
> make install

```

2) *httperf*: This sub section explains how to compile and install the modified `httperf`.

Our modified `httperf` is based on the already modified `httperf` 0.8 (with added DASH-like traffic generation and statistics output extension) source from [16]. We added further statistics output and modified the source so that

statistics are always printed when `httperf` is terminated with a `SIGTERM` signal.

First, download the modified `httperf` 0.8 source from our repository. Install `automake`, `gcc`, and `make` (if not installed). On FreeBSD or Cygwin edit `Makefile.in` to disable (comment out) `sched` functions that *only* work on Linux: `#DEFS += -DHAVE_SCHED_AFFINITY`

Configure `httperf` (set the prefix to `/usr` so `httperf` executable is installed in `/usr/bin/`, because Cygwin does not have `/usr/local` in the `PATH` by default). Note that on Cygwin we need to replace the `config.guess` from the `httperf` source with the `config.guess` installed on the system under `/usr/share` (check with `automake --version` which `automake` version is used by default). Build `httperf` as follows:

```

# copy config.guess necessary on Cygwin
> cp /usr/share/automake-1.9/config.guess
> ./configure --prefix=/usr
> make
> make install

```

3) *nttcp*: This sub section explains how to compile and install `nttcp`.

Note that we have modified `nttcp` slightly to avoid a long delay after starting `nttcp`. Build `nttcp` as follows. First, download the modified `nttcp` 1.47 source from our repository. The `nttcp` `Makefile` has different settings for different OS, but the default settings (for FreeBSD) work not only on FreeBSD but also on Linux and Cygwin. Then run:

```

> make

```

Finally, copy the `nttcp` executable to `/bin`.

## I. TCP logging

1) *Linux*: We explained how to install `web10g` for Linux in Section IV-E.

2) *Windows*: The tool `win-estats-logger` is the TCP EStats logger for Windows implemented by us. The source code and Visual Studio project files can be downloaded from our code repository. The repository also contains 32bit and 64bit binaries. We installed `win-estats-logger` by simply copying the 64bit binary `win-estats-logger.exe` to `c:/windows/system32`. Note that 64bit `win-estats-logger` needs the 64bit MS Visual C++ 2010 Redistributable, but this was already installed on our freshly installed 64bit Windows 7. If you want to install the 32bit version of `win-estats-logger` then you

first need to install the 32bit MS Visual C++ 2010 Redistributable from [17].

3) *Mac OS X*: The DTrace-based tcp logger we implemented can be downloaded from our code repository. It is installed by simply copying the script dsiftr-osx-teacup.d to /usr/bin.

#### J. Virtual image to physical machine

This section explains how we cloned the single testbed host image to multiple physical machines.

1) *Sysprep Windows*: Windows cannot simply be moved onto a new machine if the hardware is different. It needs to be sysprepped before moving, using the following steps:

- Start a command shell as administrator
- Use `regedit` to edit registry
- Set `HKEY_LOCAL_MACHINE\Software\Microsoft\Windows NT\CurrentVersion\SoftwareProtectionPlatform\SkipRearm` to 1
- Set `HKEY_LOCAL_MACHINE\SYSTEM\Setup>Status\SysprepStatus` to 7
- Set `HKEY_LOCAL_MACHINE\SYSTEM\Setup>Status\CleanupState` to 2
- Optionally deinstall IE10 (this was suggested in case sysprep does not work, but was *not* necessary in our case)
- Make the Windows partition the active partition using Linux `fdisk` (very important)
- Start sysprep as admin:

```
c:\windows\system32\sysprep\sysprep
```

- Select OOB and check generalize, select reboot
- Turn off the machine after Windows has shut down completely

2) *Convert VB disk image to raw image*: Ideally we would have created a virtual disk size of exactly the right size given the physical disk we want to move the virtual disk to. However, when creating a disk with VirtualBox we can only chose its size, but not the number of sectors. There are two possibilities:

- Choose exactly the right size based on the size of the physical disk (*unclear* if this works)
- Choose a larger size in VirtualBox and adjust the size later (this *is* the approach we took)

Our approach for resizing the disk assumes that the last partition on the disk can be resized such that the target

disk size can be achieved by shrinking the last partition. In our case the last partition is the extended partition containing only the swap partitions and the empty data partition. We can set the virtual disk size to exactly the size of a target physical disk as follows:

- With Linux `fdisk` remove the last three partitions, resize the extended partition to the exact size given the number of sectors of the physical disk (the extended partition can only extend until the last sector of the physical disk).
- Recreate both swap partitions and the data partition. Given our layout the location and size of the swap partitions should not change. The location of the data partition is also the same but its size has reduced.
- Reboot into Linux and adjust the file system on the resized data partition:

```
> umount /dev/sda7
> mkfs.vfat /dev/sda7
```

- Convert the VDI file to a VMDK file:

```
> VBoxManage clonehd --format VMDK
<inputfile>.vdi <output_file>.vmdk
```

- Use an editor that can handle very large files<sup>3</sup> to edit the VMDK file and set the number of sectors to the correct value. At the start of the VMDK file there is a line `RW xxxxxxxx SPARSE "name"`, where xxxxxxxx are the number of sectors. Change xxxxxxxx to the number of sectors required.
- Export the VMDK image to a raw format image:

```
> VBoxManage internalcommands
converttoraw <disk>.vmdk <disk>.raw
```

3) *Copy raw image to physical machine*: We copied the raw image to a physical hard disk as follows. First, we booted the openSUSE DVD in Rescue mode on the target machine. Then we configured the network interface with DHCP (IPv4) and used `dd` to copy the remote image onto the machine's hard disk:

```
> dhcpd eth0
> ssh user@remote "dd if=/path/image.raw" |
dd of=/dev/sda -
```

One can check the progress by switching to a second console, running `kill -USR1 <dd_pid>` and switching back to the console running `dd`.

<sup>3</sup>For example, `lfhex` (<http://stooipidsimple.com/lfhex>).

4) *Post-install tasks*: In general on all OS we need to do the following steps:

- Change host name;
- Change NIC config;
- Regenerate SSH host key;
- Make sure we set the right static route depending on testbed network a machine is connected to.

For Linux we first removed the existing NICs in Yast and then edited `/etc/udev/rules.d/70-persistent-net.rules` (changing the NIC names to `eth0` and `eth1` and changing the MAC addresses to the addresses of the NICs in the host). This is to ensure that we do not have any non-existing NICs and the two NICs are consistently named `eth0` and `eth1`. After a reboot, we configured the NICs and host name with Yast and created SSH keys:

```
> ssh-keygen -t rsa1 -f /etc/ssh/ssh_host_key
> ssh-keygen -t rsa -f /etc/ssh/ssh_host_rsa_key
> ssh-keygen -t dsa -f /etc/ssh/ssh_host_dsa_key
> ssh-keygen -t ecdsa -f /etc/ssh/ssh_host_ecdsa_key
```

We copied the new keys to `/data/ssh`, so we can use the same host keys for all other OS as well.

For FreeBSD 10.1 we edited `/etc/rc.conf` and changed the NIC configuration, the testbed static route, and the host name. Finally, we copied the SSH keys from `/data/ssh` to `/etc/ssh`.

For Windows 7 because of the sysprep we needed to run the initial setup. First, we specified the region/keyboard, created a dummy user without password (because a user must be created during setup), set the computer name, skipped the security configuration (select “Ask me later”), and set the network to “Work network”. Then after a reboot we removed the dummy user, configured the NICs (disabled IPv6, Windows discovery and Net-BIOS), set a suffix for DNS, and set the static routes for the testbed. Finally, we copied the SSH keys from `/data/ssh` to `/etc`.

It may be necessary to fix the root password for SSH logins. This can be done by starting a Cygwin shell as administrator and running:

```
> mkpasswd -l -u root >> /etc/passwd
```

Afterwards use a text editor to edit `/etc/passwd` and remove all entries for root, *except the last entry*.

## K. Physical machine to VB image

We can also convert the hard disk of a physical machine back to a virtual machine. This works as follows:

- Sysprep Windows 7 as explained in Section IV-J1
- Boot the machine from a DVD, e.g. openSUSE in rescue mode, and enable a network interface
- Copy the entire hard disk to the remote machine running VirtualBox:

```
> dd if=/dev/sda | ssh user@remote "dd
of=/path/image.raw"
```

- Convert the raw image into a VirtualBox VDI file:

```
> VBoxManage convertfromraw
/path/image.raw /path/image.vdi --format
vdi
```

- Compact the VDI file:

```
> VBoxManage modifyhd --compact
<name>.vdi
```

- Create a new virtual machine in VirtualBox for Windows 7 64bit with 1.5 GB of RAM and two Ethernet cards (Intel PRO/1000 MT Desktop)
- Set the hard disk of the new virtual machine to be the created VDI file
- Boot and install the sysprepped Windows 7
- Change the NIC configuration for all OS
- Change the SSH host keys

## V. ROUTER SETUP

This section explains the setup of the testbed router. We again installed and configured the router in a virtual machine and then moved the hard disk partitions of the virtual machine onto the physical machine’s hard disk.

### A. BIOS

PXE booting needs to be enabled. The option is called “Enable Option ROM” or “LAN boot”. After enabling PXE we needed to reboot once before we can select the network card connected to the control network as first boot device. Then under “Boot devices selection” we moved the network interface to the first position. We also made sure the router starts automatically after a power outage or power cycle. This could be enabled under “Power options” (enable “Turn on after power outage”).

Table II  
ROUTER PARTITION LAYOUT

Type	Size	Purpose
Prim	4GB	Linux Swap
Prim	10GB	Linux
Prim	10GB	FreeBSD

### B. Hard disk partitioning

On the router we created relatively small FreeBSD and Linux partitions to make it possible to image these. Again, we use MBR because our PXE boot solution only works with MBR partitions. The partitioning is simple and shown in Table II. We made the Linux swap partition the first partition so the Linux and FreeBSD partitions have the same numbers as on the other testbed hosts.

First, we used Linux `fdisk` to create the Linux swap and Linux partitions. Then after installing Linux we used the FreeBSD installer to create the FreeBSD partition.

### C. Boot manager

We use the GRUB2 boot manager from openSUSE 13.2 (installed into the MBR). To get a boot menu entry for FreeBSD we added the following to `/etc/grub.d/40_custom` (same approach as in Section IV-C):

```
menuentry "FreeBSD 10.1" {
  insmod ufs2
  set root=(hd0,msdos3)
  chainloader +1
}
```

Then to update the boot configuration we ran:

```
> grub2-mkconfig -o /boot/grub2/grub.cfg
```

### D. Linux openSUSE 13.2 64bit

We used the openSUSE installer to create Linux swap and root partitions. We selected the software pattern: “Minimal Server Selection (Text Mode)”. In the software selection screen, besides the already selected patterns we also selected the “Enhanced Base System”, “Network Administration” and “Base Development” patterns. We then followed the openSUSE installer instructions to set up the language, time zone, root user etc.

For the router we installed Linux kernel 3.17.4 as default, because this was the latest kernel version available at the time (Linux kernel 3.16.6, is also installed, as it is the

openSUSE 13.2 default kernel). To get high timer precision for queuing and delay emulation we recompiled the kernel with 10000 Hz. By default for Linux the highest frequency is 1000 Hz. To be able to run with 10000 Hz we needed to patch the kernel.

We downloaded the CK patches from [18].<sup>4</sup> Then we applied the `hz-raise_max.patch` as follows:

```
> cd /usr/src/linux-3.17.4
> patch -p1 < hz-raise_max.patch
```

Afterwards we rebuild the kernel as explained earlier.

We configured the Linux router as follows:

- Run: `zypper update`
- Turn off firewall (in Yast)
- Turn on `sshd` (in Yast)
- Add `"PermitRootLogin yes"` to `/etc/ssh/sshd_config` (restart as necessary)
- Set host name (in Yast)
- Configure control NIC to use DHCP (in Yast)
- Configure testbed NIC with static 172.16. address (in Yast)
- Set up default route and static routes for testbed networks (in Yast)
- Set up DNS servers (in Yast)
- Disable boot splash: edit `/etc/default/grub` and change `splash=silent` to `splash=0` in `GRUB_CMDLINE_LINUX_DEFAULT`
- Turn automatic updates off (should not be on by default)
- Enable NTP with local NTP servers (in Yast)
- Add GRUB2 FreeBSD entry (see above)

We edited `/etc/udev/rules.d/70-persistent-net.rules` (changing the NIC names to `eth0`, `eth1` and `eth2` and changing the MAC addresses to the addresses of the NICs in the host). This is to ensure that we do not have any non-existing NICs and the three NICs are consistently named `eth0`, `eth1` and `eth2`.

### E. FreeBSD 10.1 64bit

We booted the FreeBSD DVD and started the FreeBSD installer. Then we created a FreeBSD partition with swap and data slices and selected the data slice as

<sup>4</sup>There is no version of the 10 kHz patch for the latest Linux kernels, but the patch for version 3.10 applied fine to the 3.17.4 kernel source code.

install target. We installed FreeBSD as usual (see Section IV-F). We updated the ports tree, installed portmaster as well as other tools (see Section IV-F). We added the search domain and name servers in resolv.conf and edited rc.conf as follows:

```
hostname="testrouter"
ifconfig_em1="DHCP"
defaultrouter="10.1.1.1"
ifconfig_igb0="inet 172.16.10.1/24"
ifconfig_igb1="inet 172.16.11.1/24"
gateway_enable="YES"
sshd_enable="YES"
ntpd_enable="YES"
dumpdev="NO"
firewall_enable="YES"
firewall_type="OPEN"
ntpd_sync_on_start="YES"
```

Finally, we added the line kern.hz="10000" to /boot/loader.conf to set HZ to 10 kHz.

## VI. OTHER HARDWARE SETUP

Here we describe the setup of other hardware, such as the switches, the power controllers and the KVMs.

### A. Switches

We use two Gigabit Ethernet switches to connect the hosts to the control network (10.1.1.0) and the two data networks (172.16.10.0 and 172.16.11.0). A separate VLAN is configured for each of the three networks and switch ports are allocated to the VLANs. The switches are also connected to the control network directly (have their own IP address in 10.1.1.0/24).

### B. Power controllers

The main leads of all testbed hosts (including the routers) are connected to either AVIOSYS 9258HP or Serverlink SLP-SPP1008-H power controllers. This allows to control the power, e.g. perform power cycling, via a web interface or via TEACUP. We configured the access permissions, network interface, time synchronisation and email alerts on each power controller.

### C. KVMs

All testbed hosts (including the routers) are connected to IP-KVM switches via VGA and USB. This allows accessing the console of each host using a monitor and keyboard in the testbed room or from a remote machine.

We configured the access permissions and the network interface of the KVMs.

## VII. KNOWN ISSUES

The console on the Linux router behaves abnormally with a kernel running with 10000 Hz. Sometimes it does not update, but switching between different consoles with CTRL+ALT+<number> solves this.

## VIII. CONCLUSIONS AND FUTURE WORK

In this report we described the design and setup of the CAIA TCP testbed that is used by TEACUP [4] to conduct automated TCP performance experiments.

## ACKNOWLEDGEMENTS

TEACUP v0.8 was developed as part of a project funded by Cisco Systems and titled “Study in TCP Congestion Control Performance In A Data Centre”. This is a collaborative effort between CAIA and Fred Baker of Cisco Systems.

## REFERENCES

- [1] A. Finamore, M. Mellia, M. M. Munafò, R. Torres, and S. G. Rao, “Youtube everywhere: Impact of device and infrastructure synergies on user experience,” in *Proceedings of the 2011 ACM SIGCOMM Conference on Internet Measurement Conference*, ser. IMC '11, 2011, pp. 345–360.
- [2] A. Rao, A. Legout, Y.-s. Lim, D. Towsley, C. Barakat, and W. Dabbous, “Network characteristics of video streaming traffic,” in *Proceedings of the Seventh Conference on Emerging Networking Experiments and Technologies*, ser. CoNEXT '11, 2011, pp. 25:1–25:12.
- [3] “Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats,” ISO, 2012, iSO/IEC 23009-1:2012. [http://www.iso.org/iso/iso\\_catalogue/catalogue\\_tc/catalogue\\_detail.htm?csnumber=57623](http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=57623).
- [4] S. Zander, G. Armitage, “TEACUP v0.8 – A System for Automated TCP Testbed Experiments,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 150210A, 2015. [Online]. Available: <http://caia.swin.edu.au/reports/150210A/CAIA-TR-150210A.pdf>
- [5] —, “CAIA Testbed for TCP Experiments,” Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 140314B, 2014. [Online]. Available: <http://caia.swin.edu.au/reports/140314B/CAIA-TR-140314B.pdf>

- [6] C. Holman, "Netbooting Microsoft Windows 7 and XP," Centre for Advanced Internet Architectures, Swinburne University of Technology, Tech. Rep. 130226A, 2013. [Online]. Available: <http://caia.swin.edu.au/reports/130226A/CAIA-TR-130226A.pdf>
- [7] "iPXE – Open Source Boot Firmware." [Online]. Available: <http://ipxe.org/>
- [8] "GRUB for DOS." [Online]. Available: <https://gna.org/projects/grub4dos/>
- [9] "VirtualBox." [Online]. Available: <https://www.virtualbox.org/>
- [10] "Cygwin SSHD HowTo – How to run the OpenSSH SSHD server on Windows using Cygwin." [Online]. Available: <http://www.noah.org/ssh/cygwin-sshd.html>
- [11] "The Web10G Project." [Online]. Available: <http://web10g.org>
- [12] D. Hayes, "Timing enhancements to the FreeBSD kernel to support delay and rate based TCP mechanisms," Centre for Advanced Internet Architectures, Swinburne University of Technology, Melbourne, Australia, Tech. Rep. 100219A, 19 February 2010. [Online]. Available: <http://caia.swin.edu.au/reports/100219A/CAIA-TR-100219A.pdf>
- [13] Scott, "WHAT TIME IS IT?" March 2014, <http://www.atmythoughts.com/living-in-a-tech-family-blog/2014/2/28/what-time-is-it>.
- [14] "iperf Web Page." [Online]. Available: <http://iperf.fr/>
- [15] "NEWTCP Project Tools." [Online]. Available: <http://caia.swin.edu.au/urp/newtcp/tools.html>
- [16] J. Summers, T. Brecht, D. Eager, B. Wong, "Modified version of httpperf," 2012. [Online]. Available: <https://cs.uwaterloo.ca/~brecht/papers/nossdav-2012/httpperf.tgz>
- [17] "Microsoft Visual C++ 2010 Redistributable Package (x86)." [Online]. Available: <http://www.microsoft.com/en-au/download/details.aspx?id=5555>
- [18] "Kernel patch homepage of Con Kolivas." [Online]. Available: <http://users.on.net/~ckolivas/kernel/>