Dr. Thomas Dreibholz
Computer Networking Technology Group
Institute for Experimental Mathematics
Ellernstraße 29, 45326 Essen, Germany
☎ +49-201-183-7637
Fax: +49-201-183-7673
✉ dreibh@iem.uni-due.de
☞ http://www.iem.uni-due.de/~dreibh

UNIVERSITÄT DUISBURG ESSEN

June 8, 2010

SCTP and Reliable Server Pooling
A Practical Exercise

Abstract

The intention of this exercise is to obtain basic knowledge of the SCTP protocol [Ste07] and the configuration and application of Reliable Server Pooling (RSerPool) [LOTD08, Dre07, DR08b]. This exercise covers SCTP association setup, data transport, association teardown and multi-homing as well as setting up an RSerPool scenario with the protocols ASAP [SXST08a], ENRP [XSS+08] and example applications.

Contents

1 Lab Setup and Preparations 2

2 The SCTP Protocol 2
2.1 Preparations . . . 2
2.2 Association Setup and Data Transmission . . . 3
2.3 Multi-Homing . . . 3
2.4 Association Teardown . . . 4

3 The Reliable Server Pooling Framework 4
3.1 Setting Up a Basic Scenario . . . 5
3.2 Keeping an Overview of the Scenario . . . 6
3.3 Automatic Configuration . . . 6
3.4 Pool Management . . . 7
3.5 Server Selection . . . 9
3.6 Session Layer . . . 9
3.7 ENRP Handlespace Synchronization . . . 10

4 Application of Reliable Server Pooling 10
4.1 The RSPLIB Scripting Service . . . 10
4.2 The POV-RAY Ray-Tracer . . . 11
4.3 Applying the Scripting Service for POV-RAY . . . 12
4.4 Parallelizing Image Computation . . . 14

1 Lab Setup and Preparations

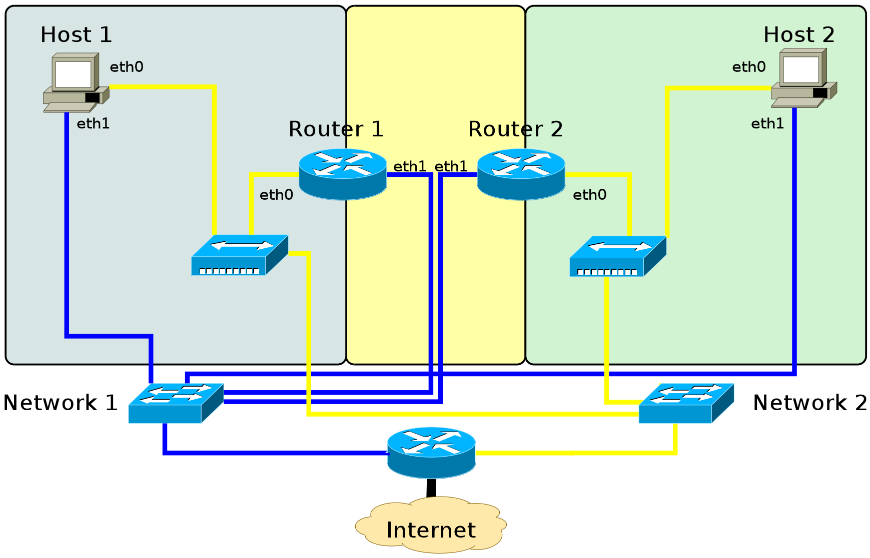

Figure 1: The Basic Networking Lab Setup

Figure 1 illustrates the networking lab setup. The lab PCs are connected to two independent networks: network 1 (blue cables, interface *eth1*) and network 2 (yellow cables, interface *eth0*). The router provides IPv4 (see [Pos81]) and IPv6 (see [DH98]) connectivity to the Internet. IPv4 addresses are provided by a DHCP server, IPv6 prefixes are provided by the router (see [CDG06, NNS98]).

Please take care of the following rules:

- Do not reboot or turn off the PCs. They may be used for simulation runs in background.
- For the same reason, do not disconnect or reconfigure network 1.
- At the end of the exercise, please reconfigure the PCs to the basic setup.

2 The SCTP Protocol

The SCTP protocol [Ste07] is the foundation of Reliable Server Pooling. Therefore, we will have a look at the basics of this protocol first.

2.1 Preparations

To get a practical insight into the functionalities of SCTP, the tool `sctp_darn` will be used in this exercise. Before applying the tool, a few basic settings should be performed. The Linux kernel SCTP module should already be loaded on the lab PCs. If it is not loaded, this can be done manually by:

```
sudo modprobe sctp
```

The default SCTP heartbeat interval is 30000ms. For our exercise, this default is a little bit too large (and the tool cannot set the interval by itself). Therefore, we set the heartbeat interval to 3000ms by:

```
sudo sysctl net.sctp.hb_interval=3000
```

After these settings, the PC is ready for some tests with `sctp_darn`. Using the command

```
man sctp_darn
```

you can get a description of the parameters for `sctp_darn`.

2.2 Association Setup and Data Transmission

At first, run WIRESHARK to capture all traffic on the *any* pseudo-interface. You can use "sctp" as filter rule to see the SCTP traffic only. After having started WIRESHARK, run an SCTP receiver on port 1234 by:

```
sctp_darn -H :: -P 1234 -l
```

The receiver will accept association requests on any of its network interfaces and receive messages.

On another PC, start a sender using the following command:

```
sctp_darn -H :: -P 2345 -h <Remote IP> -p 1234 -s -I
```

The parameter "-I" denotes the interactive mode. In this mode, you can interactively call commands (like sending data) or change parameters (like the primary path). Using "?" as command, you can get an overview of all possible commands.

Now, let the sender transmit a 10,000 bytes message by the following command:

```
snd=10000
```

Since there is no association established yet, SCTP will establish an association first. After that, the message is sent.

Question 1:

Which type of SCTP chunks can you observe on the WIRESHARK trace?

Question 2:

What are the main differences to a comparable TCP session for the data transport?

Question 3:

Have a look into the DATA and SACK chunks! What is the difference between Stream Sequence Number (SSN) and Transport Sequence Number (TSN)? Why is there no SSN or Stream ID necessary in the SACK chunk?

2.3 Multi-Homing

One of the most interesting features of SCTP is the multi-homing (see also [Jun05]). Since all lab PCs have two network interfaces – with each one having an IPv4 as well as an IPv6 address – there are four different paths in each direction.

Question 4:

How are the possible paths signalled at the association setup?

Question 5:

Observe the SCTP association in WIRESHARK for some seconds. How is the usability of each path checked by SCTP?

Ensure that the primary path goes over network 2 (yellow cable). You can explicitly set the primary path by `primary=<Remote IP>` and test this setting by sending a few more messages. After that, unplug the yellow cable and again send some messages.

Question 6:

What can you observe in the WIRESHARK trace?

2.4 Association Teardown

In order to finally perform a shutdown of the association, you can e.g. stop the sender process by typing <Ctrl>+C.

Question 7:

How is the association teardown signalled by SCTP?

3 The Reliable Server Pooling Framework

After learning the basics of the SCTP transport protocol, we will now have a look at an important SCTP application: Reliable Server Pooling (RSerPool), which is described in [LOTD08, Dre07]. As implementation, we use the Open Source RSPLIB [Dre10c, Dre07] package. It is already installed on the lab PCs.

3.1 Setting Up a Basic Scenario

The first thing we need is a registrar[1] (PR). Negotiate with the other participants which PC in the lab should become the first PR. Run WIRESHARK on this PC (sniffing on the *any* pseudo-interface) and start the PR process by:

```
registrar
```

After starting the PR, choose another PC for running a PE process. Also, run WIRESHARK and start a PE for the Fractal Generator Service [Dre07, section 5.7]:

```
server -fractal
```

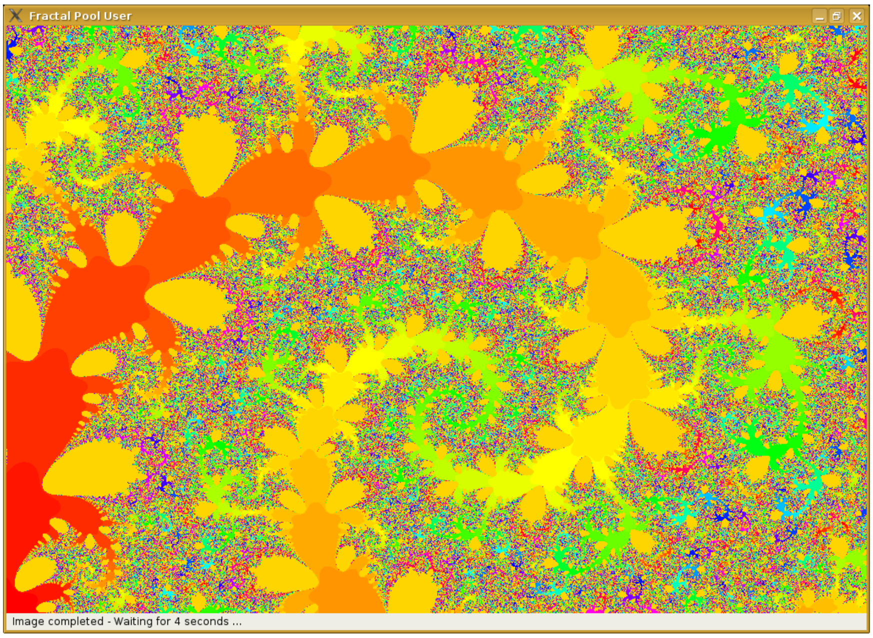

Figure 2: Screenshot of the Fractal Pool User

Finally, choose a third PC to run WIRESHARK and the Fractal Generator Service PU:

```
fractalpooluser
```

Note: There are manual pages for all RSPLIB programs, describing their possible options. Just have a look using `man <program>`!

You should now observe that the initial scenario is running and the PU should display the progress of the image calculation, as illustrated in figure 2. Also, on the WIRESHARK outputs, you should see the protocols ASAP [SXST08a], ENRP [XSS+08] and Fractal Generator Protocol (FGP). Use the pre-defined filters to select specific types of packets. Also, useful colouring rules are provided to make observing the RSerPool and application traffic illustrative.

[1] In the drafts, "registrar" is denoted as ENRP server. This terminology is in fact wrong – since ENRP has peers, but no designated clients or servers. However, due to "standardization tactics", the term "registrar" – which is also used in SIP signalling – had to be avoided for this reason.

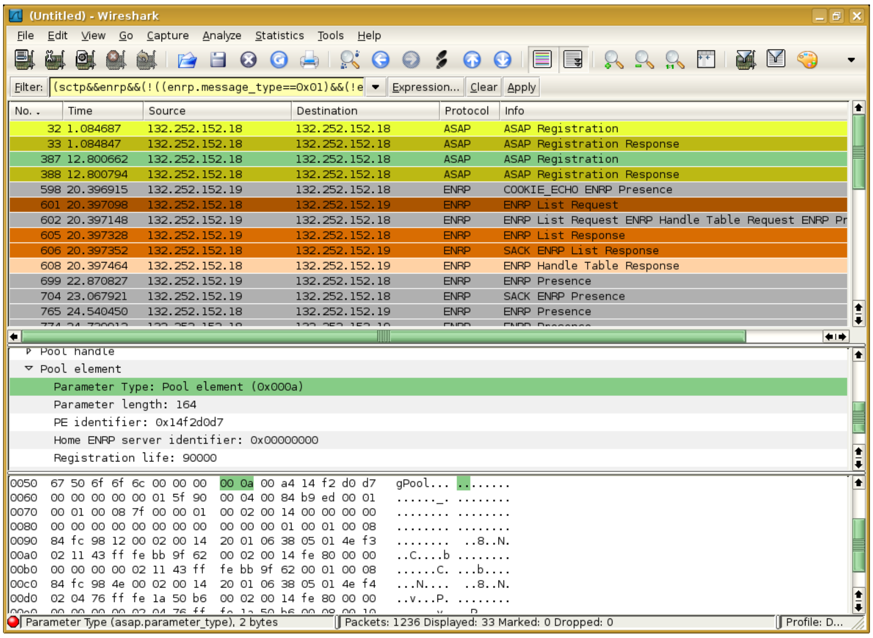

Figure 3: Observing RSerPool Traffic with WIRESHARK

Note: The filters and colouring rules are provided as part of the RSPLIB source package. Just copy the files `colorfilters`, `dfilters` and optionally `preferences` from `rsplib/wireshark` to your WIRESHARK configuration directory (usually: `~/.wireshark` or `/root/.wireshark`). The output should look similar to the example in figure 3.

3.2 Keeping an Overview of the Scenario

Before starting further components, it is useful no ensure not loosing the overview of running components. For this reason, the RSPLIB components support the Component Status Protocol[2] (CSP), which provides regular status information over UDP to a monitoring component. First, choose another PC to provide the monitoring output and start the monitor program:

```
cspmonitor
```

Now, restart the other components with two additional parameters, i.e.:

```
registrar -cspserver=<Monitor Address> -cspinterval=500
server -fractal -cspserver=<Monitor Address> -cspinterval=500
fractalpooluser -cspserver=<Monitor Address> -cspinterval=500
```

The interval gives the inter-report time in milliseconds. 500ms should be useful for our scenario.

Now, you should be able to keep an overview of your RSerPool scenario. Start a second PR and some additional PEs and PUs to test your setup.

3.3 Automatic Configuration

Have a look at the ASAP and ENRP traffic over UDP. You can apply the filter rule "(asap||enrp)&&udp" in WIRESHARK to view exactly this kind of packets.

[2]CSP is not part of RSerPool itself, but a quite useful tool provided by RSPLIB.

Question 8:
Which ASAP message type can you see here?

Question 9:
What can you say about the destination address of these ASAP messages?

Question 10:
Can you imagine the reason why SCTP cannot be used for these messages?

Question 11:
Where can you find the IP address and SCTP port number of the PR sending these messages?

Question 12:
Can you also find out the sender's PR ID?

Question 13:
Have a look at the ENRP Presence messages. What is the difference to the ASAP Server Announces?

Question 14:
Now, have a look at the messages sent over SCTP by applying the filter "(asap||enrp)&&sctp". Can you imagine why you cannot see ASAP Server Announces but only ENRP Presences via SCTP?

3.4 Pool Management

Now, have a look at the WIRESHARK output at a PE's PR-H. In particular, observe the ASAP Registration and ENRP Update messages.

Question 15:
Which two parameters can you observe in ASAP Registration and ENRP Update messages?

Question 16:
Which information about the PE to register can be found in an ASAP Registration message's parameters?

Question 17:
Does the ENRP Update contain exactly the same information about the PE as the ASAP Registration message? Why or why not?

Question 18:
What is the ASAP response to an ASAP Registration message? Can you imagine why there is no Pool Element Parameter included in this type of message?

Question 19:
Now, try registering a new PE into the existing pool, using `-policy=LeastUsed` as additional parameter. This parameter sets the pool policy (see also [DT08]) to Least Used (LU). Why is the registration not successful? Also have a look at the response message!

Question 20:
Stop all PEs of the pool. After the pool is completely empty, restart them with the Least Used policy. Does the pool now have the desired policy (check the WIRESHARK output)? Why or why not?

Question 21:
Now, deregister one of the PEs and monitor the message sequence. Which message types do you see?

What is the difference in the ENRP Update message(s)?

3.5 Server Selection

After the pool management, we will now have a look at the PU side. Observe the message flow at one of the lab PCs running the PU process.

Question 22:
Which type of ASAP message is used to request a server selection? What are its contents?

Question 23:
What is the response?

Question 24:
What happens upon request for a non-existing (equal to "empty") pool? Try to run the PU for a non-existing PH using `-poolhandle=<Name>`!

3.6 Session Layer

After the basic functionalities, we will have a look at the Session Layer functionality of RSerPool. On a PC running a PU, filter for all SCTP traffic and observe the protocol flow of an image calculation using the Fractal Generator Service. Also, during calculation, shut down the PE the PU is connected to in order to observe the failover. Alternatively, you can turn on the "failure mode" for the PEs using the parameter `-fgpfailureafter=<Packets>`, which turns on breaking the association after the given number of FGP Data packets.

Question 25:
How is a session failover handled for the Fractal Generator Service?

You can tell the PU program to use multiple sessions simultaneously, using the PU's command line parameter `-threads=<Sessions>`. Each PE by default accepts four sessions simultaneously. You can change this behaviour by `-fgpmaxthreads=<Sessions>`.

Question 26:

What do you observe on WIRESHARK when you start more sessions than there is PE capacity to process them? Is this a problem?

3.7 ENRP Handlespace Synchronization

In the following, we are going to make some tests with the ENRP protocol. First, we would like to observe the PR initialization. Therefore, run WIRESHARK and start a new PR. The new PR will detect the existence of the other PRs by the ENRP Presences over UDP multicast and establish associations.

Question 27:

How is the initial sequence of the ENRP messages over SCTP?

Question 28:

Which type of message contains the Peer List?

Question 29:

What type of information is transported by the ENRP Handle Table Response? Can you imagine the meaning of the "M" bit in the flags field?

4 Application of Reliable Server Pooling

In the last part of this exercise, we have a look at an RSerPool application: the RSPLIB Scripting Service (SS). This service can e.g. be used to process OMNET+ [Var09] simulation runs (see [DR08a, DZR09] for details). In the following, we utilize it to perform ray-tracing image calculations using the ray-tracer POV-RAY [POV10].

4.1 The RSPLIB Scripting Service

The Scripting Service works as follows: the Scripting Service PE accepts a Tar/GZip file ("work package"), which is unpacked into a temporary directory. Within the archive, there is a script named `ssrun`. `ssrun` is executed with the name of an output archive as its first argument. The `ssrun` script may do something useful with the data provided within the archive (e.g. processing a simulation run) and finally write a Tar/GZip archive as output ("results package") – using the provided output archive name. If `ssrun` returns 0, the processing has been successful. Otherwise, there has been a problem and a failover should be performed.

A user simply has to provide the work package to the Scripting Service PU and eventually gets back the results package. Using a pool of multiple PEs and starting several PU sessions in parallel, RSerPool can be used for efficient load balancing and pool management (see also [Dre10a]).

4.2 The POV-RAY Ray-Tracer

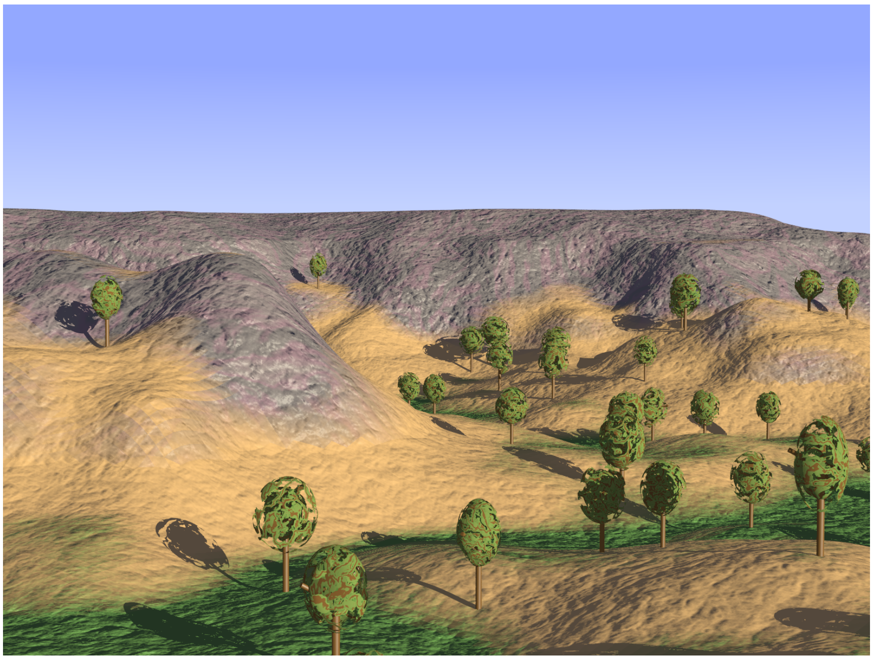

Figure 4: The POV-RAY Example "`landscape.pov`"

The POV-RAY [POV10] ray-tracer provides the command-line program `povray`, which takes an input file (`.pov`) and calculates the resulting image. For us, the following parameters are relevant:

-w Image width (e.g. `-w1024`),

-h Image height (e.g. `-h768`),

+a Use anti-alias (e.g. `+a0.3`),

+FN8 Use PNG output format (8 bits per colour, i.e. 24 bits for RGB),

+I Specifies input file name (e.g. `+Iinput.pov`),

+O Specifies output file name (e.g. `+Ooutput.png`),

-D Turns off X11 preview.

There are various other options described in the manual page of `povray`. The output of the example `landscape.pov` is shown in figure 4.

The directory `/usr/share/doc/povray/examples/advanced` contains a set of advanced example `.pov` files (GZip-compressed). For convenience reasons, copy them to a new directory and unpack them:

```
mkdir raytracing-images
cd raytracing-images
find /usr/share/doc/povray/examples/advanced | xargs -i§ cp § .
gzip -d *.gz
```

Now, we would like to calculate wallpapers (e.g. 1024x768) of all `.pov` files in the new directory. Of course, we would like to utilize the computation power of the complete lab pool for this task.

4.3 Applying the Scripting Service for POV-Ray

We now write a script `povray-distribute`, which performs the workload distribution task. This new script takes image width and height as well as a `.pov` file name as arguments. First, the arguments have to be processed:

```
#!/bin/bash
# ====== Get arguments ============================================
if [ $# -lt 3 ] ; then
   echo >&2 "ERROR: Usage $0 [Width] [Height] [Input POV]"
   exit 1
fi
WIDTH=$1
HEIGHT=$2
INPUT=$3
OUTPUT="`echo $INPUT | sed -e "s/.pov/-$WIDTH-$HEIGHT.png"/g`"
```

For the new image calculation task, we create a temporary directory and store the input `.pov` file as `input.pov` into this directory. Include files (`*.inc`) are also copied.

```
# ====== Create temporary directory =================================
TEMPDIR="temp-$INPUT-$WIDTH-$HEIGHT"
umask 077
rm -rf $TEMPDIR
mkdir $TEMPDIR
cp $INPUT $TEMPDIR/input.pov
find -name "*.inc" | xargs --no-run-if-empty -n1 -i§ cp § $TEMPDIR
```

Furthermore, we need a `ssrun` file, which we write using `echo` commands. `ssrun` is also stored into the temporary directory. The `ssrun` script will call `povray` on the input file, with the appropriate parameters. The resulting output image will be called `image.png`. The variable `SUCCESS` contains the result of `ssrun`. It is set to 1 (i.e. "failed") if something goes wrong. Furthermore, the text output of `povray` is written to `output.txt` for debugging in the case of something going wrong.

```
# ====== Write ssrun script ========================================
(
   echo "#!/bin/sh"
   echo "OUTPUT_ARCHIVE=\$1"
   echo "SUCCESS=1"
   echo -n "povray -w$WIDTH -h$HEIGHT +a0.3 -D +FN8 +Ooutput.png "
   echo    "+Iinput.pov >output.txt 2>&1 || SUCCESS=0"
   echo "tar czvf \$OUTPUT_ARCHIVE output.png output.txt || SUCCESS=0"
   echo "exit $SUCCESS"
) >"$TEMPDIR/ssrun"
chmod +x "$TEMPDIR/ssrun"
```

We can now create the work package Tar/GZip file `input.tar.gz` – containing `input.pov`, includes and `ssrun`. This work package can be processed by the Scripting Service PU, i.e. `scripting-client`. The output archive will be written to `output.tar.gz`.

```
# ====== Create and distribute work package ========================
cd "$TEMPDIR"
find . -name "ssrun" -or -name "input.pov" -or -name "*.inc" | \
   xargs tar czf input.tar.gz
cd ..
scriptingclient -quiet -input=$TEMPDIR/input.tar.gz \
                       -output=$TEMPDIR/output.tar.gz
```

If an output file has been written, we can unpack it. If there is also a PNG file, our run has succeeded. Otherwise, there should be the log output of `povray`, which can be printed for debugging:

```
if [ -e "$TEMPDIR/output.tar.gz" ] ; then
   cd "$TEMPDIR"
   tar xzf output.tar.gz
   cd ..
   if [ -e "$TEMPDIR/output.png" ] ; then
     mv $TEMPDIR/output.png $OUTPUT
     rm -rf $TEMPDIR
   else
      echo >&2 "ERROR: No image has been created. Check log:"
      echo "------ LOG ----------------------------------------"
      cat "$TEMPDIR/output.txt"
      echo "---------------------------------------------------"
   fi
fi
```

In order to perform workload distribution, some PEs have to be in the Scripting Service pool. That is, start some scripting PEs by:

```
server -scripting -policy=LeastUsedDegradation:0.5 \
       -ssmaxthreads=2 ...
```

The policy is set to Least Used with Degradation (LUD). The load degradation by accepting a new request is 50%. Up to two sessions are processed simultaneously – since our lab PCs are dual-core machines.

Test your new script with a small test run, e.g. computing `landscape.pov` (see figure 4) in the resolution 128x96. Also record the session's message flow using WIRESHARK.

Question 30:

How is the message flow of the Scripting Service Protocol (SSP)?

Question 31:

Can you imagine how failovers are handled?

4.4 Parallelizing Image Computation

Now, we would like to process all `.pov` files in parallel. Therefore, we use another script – named `run-povray-for-files` which simply calls `povray-distribute` in background for each file provided as argment:

```
#!/bin/bash
POV_FILE_LIST=`\
( \
while [ x$1 != "x" ] ; do \
   echo $1 && \
   shift ; \
done \
) | sort -u`

for POV_FILE in $POV_FILE_LIST ; do
   ./povray-distribute 1024 768 $POV_FILE &
done
```

Execute this script in the `.pov` files directory by:

```
./run-povray-for-files *.pov
```

It will start processing all files in parallel. Not having enough PE capacity is no problem. A PU will try another PE after some delay. However, it is recommended to add all lab PCs to the Scripting Service pool. Otherwise, the computation can take a long time. . . .

References

[CDG06] A. Conta, S. Deering, and M. Gupta, *Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification*, Standards Track RFC 4443, IETF, March 2006, http://www.ietf.org/rfc/rfc4443.txt.

[DH98] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6)*, Standards Track RFC 2460, IETF, December 1998, http://www.ietf.org/rfc/rfc2460.txt.

[DR08a] T. Dreibholz and E. P. Rathgeb, *A Powerful Tool-Chain for Setup, Distributed Processing, Analysis and Debugging of OMNeT++ Simulations*, Proceedings of the 1st ACM/ICST OMNeT++ Workshop (Marseille/France), March 2008, http://tdrwww.iem.uni-due.de/dreibholz/rserpool/rserpool-publications/OMNeT++Workshop2008.pdf.

[DR08b] ______, *Towards the Future Internet – An Overview of Challenges and Solutions in Research and Standardization*, Proceedings of the 2nd GI/ITG KuVS Workshop on the Future Internet (Karlsruhe/Germany), November 2008, http://tdrwww.iem.uni-due.de/dreibholz/rserpool/rserpool-publications/FutureInternet2008.pdf.

[DR09] ———, *Overview and Evaluation of the Server Redundancy and Session Failover Mechanisms in the Reliable Server Pooling Framework*, International Journal on Advances in Internet Technology (IJAIT) **2** (2009), no. 1, 1–14, http://tdrwww.iem.uni-due.de/dreibholz/rserpool/rserpool-publications/IJAIT2009.pdf.

[Dre07] T. Dreibholz, *Reliable Server Pooling – Evaluation, Optimization and Extension of a Novel IETF Architecture*, Ph.D. thesis, University of Duisburg-Essen, Faculty of Economics, Institute for Computer Science and Business Information Systems, March 2007, http://duepublico.uni-duisburg-essen.de/servlets/DerivateServlet/Derivate-16326/Dre2006-final.pdf.

[Dre10a] ———, *Applicability of Reliable Server Pooling for Real-Time Distributed Computing*, Internet-Draft Version 08, IETF, Individual Submission, January 2010, http://www.watersprings.org/pub/id/draft-dreibholz-rserpool-applic-distcomp-08.txt.

[Dre10b] ———, *Handle Resolution Option for ASAP*, Internet-Draft Version 06, IETF, Individual Submission, January 2010, http://www.watersprings.org/pub/id/draft-dreibholz-rserpool-asap-hropt-06.txt.

[Dre10c] ———, *Thomas Dreibholz's RSerPool Page*, 2010, http://tdrwww.exp-math.uni-essen.de/dreibholz/rserpool.

[DT08] T. Dreibholz and M. Tüxen, *Reliable Server Pooling Policies*, RFC 5356, IETF, September 2008, http://www.ietf.org/rfc/rfc5356.txt.

[DZR09] T. Dreibholz, X. Zhou, and E. P. Rathgeb, *SimProcTC – The Design and Realization of a Powerful Tool-Chain for OMNeT++ Simulations*, Proceedings of the 2nd ACM/ICST OMNeT++ Workshop (Rome/Italy), March 2009, http://tdrwww.iem.uni-due.de/dreibholz/rserpool/rserpool-publications/OMNeT++Workshop2009.pdf.

[Jun05] A. Jungmaier, *Das Transportprotokoll SCTP*, Ph.D. thesis, Universität Duisburg-Essen, Institut für Experimentelle Mathematik, August 2005, http://miless.uni-duisburg-essen.de/servlets/DocumentServlet?id=12152.

[LOTD08] P. Lei, L. Ong, M. Tüxen, and T. Dreibholz, *An Overview of Reliable Server Pooling Protocols*, Informational RFC 5351, IETF, September 2008, http://www.ietf.org/rfc/rfc5351.txt.

[NNS98] T. Narten, E. Nordmark, and W. Simpson, *Neighbor Discovery for IP Version 6 (IPv6)*, Standards Track RFC 2461, IETF, December 1998, http://www.ietf.org/rfc/rfc2461.txt.

[Pos81] J. Postel, *Internet Protocol*, Standards Track RFC 791, IETF, September 1981, http://www.ietf.org/rfc/rfc791.txt.

[POV10] POV-Team, *POV-Ray – The Persistence of Vision Ray Tracer*, 2010, http://www.povray.org.

[Ste07] R. Stewart, *Stream Control Transmission Protocol*, Standards Track RFC 4960, IETF, September 2007, http://www.ietf.org/rfc/rfc4960.txt.

[SXST08a] R. Stewart, Q. Xie, M. Stillman, and M. Tüxen, *Aggregate Server Access Protcol (ASAP)*, RFC 5352, IETF, September 2008, http://www.ietf.org/rfc/rfc5352.txt.

[SXST08b] ———, *Aggregate Server Access Protocol (ASAP) and Endpoint Handlespace Redundancy Protocol (ENRP) Parameters*, RFC 5354, IETF, September 2008, http://www.ietf.org/rfc/rfc5354.txt.

[Var09] A. Varga, *OMNeT++ Discrete Event Simulation System*, 2009, http://www.omnetpp.org.

[XSS+08] Q. Xie, R. Stewart, M. Stillman, M. Tüxen, and A. Silverton, *Endpoint Handlespace Redundancy Protocol (ENRP)*, RFC 5353, IETF, September 2008, http://www.ietf.org/rfc/rfc5353.txt.