Chapter 1

Hardware

The hardware specifications of testbed machine are as follows.

CPU	
processor_no	8
vendor_id	GenuineIntel
cpu family	6
model	26
model name	Intel(R) Xeon(R) CPU E5504
cpu MHz	2G MHz
cache size	4096 KB
Hard drive	
Capacity	500 GB, 500107862016 bytes
Memory	
TotalMem	24GB (24612632 KB)

Chapter 2

Installation

The Xen Haggle Simulator (XHS) can be installed on the Debian Linux 5.0 (Lenny) with kernel 2.6.26-2-xen-amd64. Debian was chosen due to its sustained support for Xen. The more information on installation of Debian (Lenny) can be found via http://www.debian.org/releases/stable/amd64/index.

2.1 Xen

In order to install Xen on Debian system, the following packages are required to be installed from Debian repositories.

- linux-headers-2.6.26-2-common-xen - Common header files
- linux-headers-2.6.26-2-xen-amd64 - Header files
- linux-image-2.6.26-2-xen-amd64 - Linux 2.6.26 image on AMD64
- linux-modules-2.6.26-2-xen-amd64 - Linux 2.6.26 modules on AMD64
- xen-linux-system-2.6.26-2-xen-amd64 - XEN system
- xen-docs-3.2 - Documentation for Xen
- xen-hypervisor-3.2-1-amd64 - The Xen Hypervisor on AMD64
- xen-utils-3.2-1 - XEN administrative tools
- xenstore-utils - Xenstore utilities for Xen
- xen-utils-common - XEN administrative tools - common files
- xen-shell - Console based Xen administration utility
- xen-tools - Tools to manage Debian XEN virtual servers

You probably need to add the following non-free sources to /etc/apt/sources.list.

```
deb http://ftp.uk.debian.org/debian/ lenny main contrib non-free
deb-src http://ftp.uk.debian.org/debian/ lenny main contrib non-free
```

Xen configuration file can be found in /etc/xen/xend-config.sxp. In this file, one line with (network-script network-bridge) need to be uncommented. In order to make linux bridge work, you need install `bridge-utils` package from repos. After changing the network configuration, it is required to reset all xend and network settings to activate the new settings with the following commands.

```
$ sudo /etc/init.d/xend restart
$ sudo /etc/init.d/networking restart
```

2.2 Haggle

First step you need to install *Mercurial* from Debian repos. Then you have to download the latest haggle source code from *Mercurial* repositories by the following command.

```
hg clone https://haggle.googlecode.com/hg/ haggle
```

For the latest LABEL, RANK, and BUBBLE source code, you have to check out the repositories by the following command.

```
svn checkout http://haggle-cambridge.googlecode.com/svn/trunk/ haggle
```

In order to successfully compile Haggle code on Linux, you are required to install standard development environment, including gcc, g++, standard development headers and libtools. Additional development packages for OpenSSL, Sqlite, and LibXML2 are also needed depending on your configurations.

A typical set of packages for Debian required for Haggle is as follows :

- build-essential (including libc6-dev, gcc, g++, make, etc.)
- automake
- autoconf
- libtool
- libsqlite3-dev
- libxml2-dev
- libbluetooth-dev
- libdbus-1-3

- libdbus-1-dev
- libssl-dev

Please install all of them with `apt-get` or `aptitude`.

In order to compile and installl haggle on the Xen Haggle Simulator, you need to go to the downloaded souce code directory and do the following commands.

```
./autogen.sh
./configure --enable-debug--prefix=/usr/local/haggle
make
sudo make install
```

```
note:  The prefix should be consistent with HAGGLE_INSTALL_PATH
 in create_testbed.sh in testbed folder.
```

To run haggle, you need to go to `bin` directory in the source code and type `./haggle`. Or you need to add `/usr/local/haggle/bin` in your PATH environment.

2.3 Haggle testbed

Download Haggle testbed source[1] in the any folder and unzip them. You will find there are 4 directories under the haggle testbed folder:

```
controller - used for remote control
scripts - used to run the testbed
testbed - used to create the testbed
vendetta - is the java GUI for testbed (not working at the moment)
```

To create testbed, you have to do as follows.

```
run_create_testbed.sh
```

The script will automatically use debootrap to create the image file and xen configuration file. Finally, the program will ask you for the password for the user and root for the virtual nodes.

[1]The source code for haggle testbed can be found in http://code.google.com/p/haggle-cambridge/downloads.

Tips:
It is better to create a user without sudo password. Otherwise, you have to give the password every time when the testbed is manipulated (create nodes, start/stop nodes, filter traffic, etc.)
To do this, you need to add the following line into your /etc/sudoers file.

```
yourusename ALL=NOPASSWD: ALL
```

Then you need to go to the scripts folder. You will see a lot of shell files which are used to manipulate the testbed. Most of time you can guess their functions from their names.

The reliable way to run the testbed is based on preconfigured scenarios. There is one java file called `scenariorunner2.java`, which is the main program to run the testbed in non-interactive way in the scripts folder. To run the experiments, you have to create your own scenario.xml file as follows.

```
<Scenario>
 <Magic>haggle</Magic>
 <Architecture>haggle -c -f -I</Architecture>
 <Configuration>config.xml</Configuration>
 <Tracefile>simple.trc</Tracefile>
 <Iterations>1</Iterations>
 <Warmup>300</Warmup>
 <Application>luckyMe</Application>
</Scenario>
```

You have to create your own trace file to describe your scenarios. The trace file must be in the format as follows.

```
nodeA    nodeB    linkup_time  linkdown_time
```

Then you are able to run the scenario with following command.

```
java scenariorunner2 scenario.xml
```

In addition, there is a automatic way to run the testbed by using the `queue_executer.sh`, which will take care of logging the performance of the testbed, execute scenarios, and save logs with a timestamp. To run a scenario you only have to create a tar.gz file containing the following files.

```
config.xml - configuration for haggle
markov.trc - trace file
scenario.xml - scenario file
```

You have to put the*tar.gz* file to the queue/ folder in your home directory on the testbed. You can copy several files there and they will be executed in the order they were copied. Observe that you cannot copy a file with the same name twice to the queue folder. That will overwrite the current file. (The tar file has to have the same name as the folder inside the tar file.)

2.4 Run LABEL, RANK and BUBBLE

We have implemented three forwarding[2] algorithms in this testbed, namely LABEL, RANK, and BUBBLE.

In order to simulate LABEL forwarding algorithm, you need to install the haggle executive file into */usr/local/haggle/bin/* and rename it to haggle-label then change the *Architecture* tag in scenario.xml as follows.

```
<Scenario>
 <Architecture>haggle-label -c -f -I</Architecture>
</Scenario>
```

As for RANK forwarding algorithm, haggle-rank executive file is required to be installed into */usr/local/haggle/bin/*then change the *Architecture* tag in scenario.xml as follows.

```
<Scenario>
 <Architecture>haggle-rank -c -f -I</Architecture>
</Scenario>
```

Finally for BUBBLE forwarding algorithm, the haggle executive file is needed to be into */usr/local/haggle/bin/* and rename to haggle-bubble then change the *Architecture* tag in scenario.xml as follows.

```
<Scenario>
 <Architecture>haggle-bubble -c -f -I</Architecture>
</Scenario>
```

2.5 Community configuration

To enable community configuration you need to replace Forwarder configuration in config.xml with the following line and rename config.xml to config.template.xml.

```
<Forwarder label=$LABEL rank=$RANK />
```

[2]All code can be found via http://code.google.com/p/haggle-cambridge/

New *ComConfig* tag need to be added to scenario.xml in order to update community structure to forwarding algorithm in hagglekernel. You have to add one line to scenario.xml as follows.

```
<Scenario>
 <ComConfig>true</ComConfig>
</Scenario>
```

In this way, the testbed will run *upload.py* to update the community structure according to predefined configuration. The RANK value for each node should be covert into a dictionary format, i.e. {node:rank}, as indicated in *upload.py* file. All community structure information should be added in the format as indicated in *upload.py* file below. Finally you can run the simulation just as normal.

```
community=[[94,2,63,97,82,47],
[26,65,93],
[20,46,84,35,50,73,92],
[12,19,20,23,31,4,72],
[29,14,16,18,6,81,83,85,86,96,39,57,75,95,37,49],
[21,51,7,74,54],
[85,28,71,89],
[15,76,80,91,78,32,37]]

#Rank average
dictRank={1:30,
2:55,
3:76,
....
36:28}

def sedfunc( rep , ch ):
if ch=="label":
sedstr="sed \"s/\$LABEL/"+rep+"/g\""
if ch=="rank":
sedstr="sed \"s/\$RANK/"+rep+"/g\""
return sedstr

os.system("rm "+localName+"node-*.xml")

for x in range(commNum):

nodeLabel="label"+str(x+1)

if debug==1:
print community[x]
```

```
for i in community[x]:
node_name="node-"+str(i)

nodeRank=str(dictRank[i])

fileName=localName+node_name+".xml"

cmd_genxml=sedfunc(nodeLabel,"label")+" "+configTemplate+
" | "+sedfunc(nodeRank,"rank")+" > "+fileName

cmd_mkdir="ssh "+userName+"@"+node_name+" mkdir .Haggle"

cmd_mkscp="scp "+fileName+" "+userName+"@"
+node_name+":.Haggle/"+remoteName

if debug ==1:
print node_name
print cmd_mkdir
print cmd_mkscp
print cmd_genxml

os.system(cmd_genxml)
os.system(cmd_mkdir)
os.system(cmd_mkscp)
```