# L50 Labs

Dr Noa Zilberman & Prof Andrew W. Moore

Michaelmas, 2019/2020

This document provides useful information required for the practicals. All the information is located at: `https://github.com/cucl-srg/L50`; as we find bugs we will do updates - so `git pull` regularly.

## 1 Test Machines

Each experiment uses two machines, known as Machine A and Machine B. The setup of Machine A and Machine B is different, however all the A machines and all the B machines are the same.

You will be assigned one pair of machines to use per lab session and the machines are located in the Practical Classroom (back room of SW02). This provides you physical access to the machines, so you can change the physical connectivity according to the experiment.

You will interact with the machines via ssh:

1. On a computer in ACS/Part III classroom, SW02, log in using your own UIS credentials, then
2. `ssh -X root@<hostname>.nf.cl.cam.ac.uk` and enter the password. Hosts ending in .cl.cam.ac.uk or .cst.cam.ac.uk are permitted to ssh into these machines. The `-X` enables X11 forwarding, allowing you to run graphical applications on the nf-test machines from your SW02 machine. The bash prompt and Firefox theme are red on Machine A, and blue on Machine B.

| Machine | hostname | IP Address | Machine | hostname | IP Address |
|---|---|---|---|---|---|
| A1 | nf-test108 | 128.232.82.68 | B1 | nf-test102 | 128.232.82.62 |
| A2 | nf-test103 | 128.232.82.63 | B2 | nf-test104 | 128.232.82.64 |
| A3 | nf-test111 | 128.232.82.71 | B3 | nf-test110 | 128.232.82.70 |

**Important: The IP addresses noted above should only be used to communicate (from the ACS/Part III hosts) with the machines. The network interfaces assigned for the tests use different IP addresses.**

## 2 Repository Structure

**Important: As multiple teams may work on the same machines (using the same `root` account, clone the repository to a local folder under your crsid, e.g., ∼/nz247/L50**

- `Jupyter`
  - Notebook templates (`.ipynb`)
    You will use Jupyter Notebook to run the experiments and save the results.
  - `useful`
    The 'useful' scripts include functions which you will use in the Notebooks, to simplify data processing. These are documented later in Section 5.
    `generator.lua` is an exception, as it is a prepared script used in Lab 3 by the packet generation tool "MoonGen".
- `handouts`
  Practical sessions handouts.
- `lab-report`
  A template for lab reports.
- `lectures`
  Lecture slides.
- `pcap_files`
  pcap files are packet capture files, which are loaded into traffic generation tools to be replayed. You can view the details of the pcap files by opening them in, e.g., wireshark. For these prepared files, the filename reflects the packet size in bytes within the host (packet size on the wire adds an additional +4 bytes).
- `setup`
  This folder contains various scripts for setting up tools and creating directories in which to save results.

## 3 Jupyter Notebooks

The Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations and explanatory text.

You will use Jupyter Notebook to run the experiments and save the results. The templates are designed to run on Machine A and results are saved to Machine A.
To copy a remote directory onto your local machine, `sftp root@<hostname>.nf.cl.cam.ac.uk` and `get -r <directory>`.
Exporting a Notebook as `.tex` will save graphs as separate files, which you can then

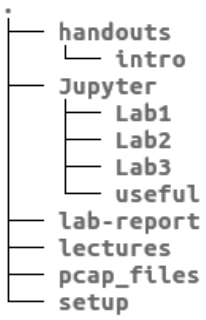


Figure 1: Repository Structure

include in your lab report.

Starting a Jupyter session: after you `ssh -X` into Machine A, type the following:
`cd L50/Jupyter && jupyter notebook --allow-root`
(`pkill firefox` if Firefox is already running.)

There are several places where you can find more information on using Jupyter:

- Jupyter-Notebook page on the course's wiki.
- guide.ipynb under the handouts folder.
- Jupyter documentation: `https://jupyter-notebook.readthedocs.io/en/latest/notebook.html`
- Jupyter-Notebook Basics page: `http://nbviewer.jupyter.org/github/jupyter/notebook/blob/master/docs/source/examples/Notebook/Notebook%20Basics.ipynb`

# 4 Test Setup

The following special equipment is used in the classes:

- Solarflare network interface cards (NIC) - SFN6122F (Machine A)
  Solarflare network ports are marked *slf0* and *slf1*.
- Endance DAG card, used for traffic capture (Machine A)
  DAG network ports are marked *dag0* and *dag1*.
- Intel NIC - X520 (Machine B)
  Intel network ports are marked *intl0* and *intl1*.
- NetFPGA SUME board, used as a traffic generator (Machine B)
  NetFPGA network ports are marked *nf0* to *nf3*.

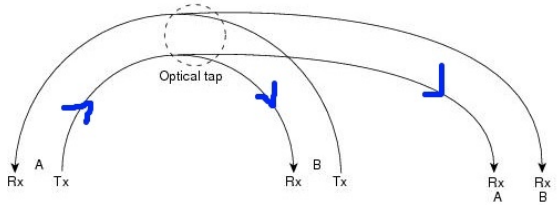


Figure 2: Using the tap as a splitter, in one direction.

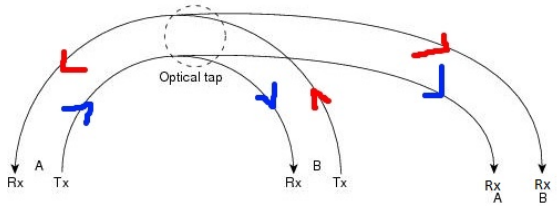


Figure 3: Using the tap to capture traffic originating from two interfaces.

- Optical fibers - duplex fibres with separate strands for transmitting and receiving.
- Optical 10G transceivers (SFP+) - converting electrical signals to optical signals and vice versa.
- Optical Tap - extracts the signal sent from one point to the other, without breaking the connection, and transmits the signal on a third port. Figures 2 and 3 illustrate the tap's operation.

The network port markings noted above apply only to Figure 4 and Figure 5. The name of the port on each machine may differ, (e.g.,*eth1*).


Figure 4: Machine A


Figure 5: Machine B

# 5 'Useful' functions

## 5.1 useful.py

This script provide useful functions for all practical sessions.

`local_cmd(command)`
Executes Terminal command on local machine. Returns after the command exits.
Input: command, eg. 'echo blah'
Output: Terminal output as a string

`ssh_connect(host)`
Connect to a remote machine using ssh.
Input: IP address of remote machine, eg '128.128.128.128'
Output: paramiko.SSHClient instance (paramiko is a python implementation of ssh v2)

`ssh_cmd(command, ssh)`
Executes a command on remote machine.
Input: command = command. ssh = paramiko.SSHClient instance.
Output: Terminal output (if any).

## 5.2 useful1.py

This script provide useful functions for Lab 1

`getrtt(ffile,crsid,num)`
Parses a file with RTT results from a ping command.
Input: ffile = '[folder]/[file]', eg. '1/exp1a_0'. crsid - user crsid. num = number of pings.
Output: list of RTTs in microseconds

`graph1(exp,crsid,div,num)`
Plots graph1 in Lab 1.
Input: exp = experiment name, eg. 'exp1a'. crsid - user crsid. div = gaps between error bars. num = number of pings.

`graph5_000001(exp,crsid,usecs,num)`
Plots graph5 for an interval of 0.000001 seconds.
Input: exp = experiment name. crsid - user crsid. usecs = list of rx-usecs. num = number of pings.

`graph5_001(exp,crsid,usecs,num)`
Plots graph5 for an interval of 0.001 seconds.
Input: exp = experiment name. crsid - user crsid. usecs = list of rx-usecs. num = number of pings.

`data_iperf(fexp,crsid)`
Parses the iperf test results.
Input: '[folder]/[exp]', eg. '1/exp1a'. crsid - user crsid.
Output: list of 5 bandwidth values in Gbits/sec for each measurement interval
[ [5x bw for 0-1 sec] , ... , [5x bw for 9-10 sec] ]

`data10(crsid)`
Parses the data for experiment 10.
Input: crsid - user crsid. Output: tuple containing iperf bandwidth valuess in Gbits/sec of 1. server 2. client
( [ [5x bw for 0-1sec (ser)] , ... , [5x bw for 9-10sec (ser)] ] ,
[ [5x bw for 0-1sec (cli)] , ... , [5x bw for 9-10sec (cli)] ] )

`data11(windows,crsid)`
Parses the data for experiment 10.
Input: windows - list of specified window sizes, in KB . crsid - user crsid.
Output: tuple containing 1. list of actual window sizes in KB 2. list of 5 bandwidth values in Gbits/sec for each window size
( [sz1, ... , szn] ,
[ [5x bw for sz1] , ... , [5x bw for szn] ] )

`data_band(fexp,crsid,band)`
Provides a list of effective bandwidth values, based on a parsed results file.
Input: fexp = 'folder/experiment name'. crsid - user crsid.. band = list of effective bandwidth values in Mbits/sec.
Output: tuple containing 1. list of effective bandwidths in Mbits/sec 2. list of 5 percentiles for each bandwidth value.
( [bw1, ... , bwn] ,
[ [5x % for bw1] , ... , [5x % for szn] ] )

`data13b(windows,crsid)`
Parses the results of experiment 13b.
Input: windows - list of specified window sizes in KB. crsid - user crsid.
Output: list of 5 bandwidths in Gbits/sec for each window size
[ [5x bw for sz1] , ... , [5x bw for szn] ]
`graph_error(data)`
Calculates error bars values, for plotting error bars.
Input: list containing n repetitions for each data point.
[ [n x point 1], [n x point 2], ... , [n x point m]]
Output: tuple containing 1. list of median y-values 2. correctly formatted yerr for plotting error bars.
( [y1, y2, ... , yn] ,
[ [below1, below2, ... , belown], [above1, above2, ... , aboven] ])

### 5.3 useful2.1.py

This script provide useful functions for Lab 2.1

`gettimes(exp)`
Parses a trace file and extracts a list of timestamps.
Input: experiment name, eg. 'exp1a'.
Output: list of timestamps in seconds as strings.

`getdiff(exp)`
Parses a trace file and provides a list of time gaps between timestamps.
Input: experiment name
Output: list of timestamp differences in nanoseconds

### 5.4 useful2.2.py

This script provide useful functions for Lab 2.2

`gettimes(exp)`
Parses an trace file and extracts a list of timestamps.
Input: experiment name, eg. 'exp1a'
Output: list of timestamps in seconds as strings

`getdiff(exp)`
Parses a trace file and provides a list of time gaps between timestamps.
Input: experiment name
Output: list of timestamp differences in microseconds

`getrtt(fname)`
Parses a trace file and provides a list of round trip times.
Input: filename Prints examples of RTTs.
Output: list of RTTs in microseconds.

### 5.5 useful3.py

`getdeltas(fexp, num)`
Parses a trace file and provides a list of inter arrival times.
Input: exp = 'folder/experiment name', eg. '3.1/exp1a'. num = number of packets sent
Output: list of inter-arrival times in microseconds (float)