Finally, a huge thank you to Eben Upton, Raspberry Pi Foundation, and PiHut people for enabling this incarnation of the module at incredibly short notice.
Lessons from Lab1

• Ping isn’t the best tool for latency measurements
• Iperf isn’t the best tool for bandwidth measurements

• Control, variability, accuracy, ….

• If you didn’t conclude that; go back and look at your results….
• What should you expect? (do the math)
Ethernet switch
(48 x 1Gbps)

1m Twisted pair (UTP) 1Gbps

<0.4ns

Latency < 10 µs

<0.4ns
Ethernet switch (48 x 1Gbps) 

Latency < 10 µs

<0.4ns
Ethernet switch (48 x 1Gbps) 1m Twisted pair (UTP) 1Gbps

Latency < 10 µs

<0.4ns

Ping reported ~90% results ~20 µs so ~2 x switch…. plus a bit….
Ethernet switch (48 x 1Gbps) 1m Twisted pair (UTP) 1Gbps

Latency < 10 µs

Ping reported ~90% results ~20 µs so ~2 x switch…. plus a bit….
• So even if the end hosts added no extra time (they do); what about the rest of the 150µs?
  
• Systematic errors: clocks, speed of light

• Non-systematic errors: scheduling in host, competing demands on resources,
  
  • BUFFERING in the switch

• “Where has my time gone?” https://www.repository.cam.ac.uk/handle/1810/263038
Example: Network Congestion
How to control generated traffic?

- What is the packet format? (e.g. protocol, payload)
- How many packets?
- What is the packet size(s)?
- What is the average data rate?
- What is the peak data rate? (e.g. burst control)
- …
Traffic Generation Tools

$$$$$, Hardware, high quality
(Ixia, Spirent,..)

$$ Software/hardware based, medium quality
(OSNT, MoonGen,..)

Commodity, Software, low quality
(TCPReply,.. )
TCP Replay

- Free, software-based
- Replays network traffic stored in pcap files
  - Not just TCP
  - (not just pcap)
- Included in Linux
- Packets are sent according to pcap file timestamps
Software based traffic generators

- Traditional tools (e.g., D-ITG, trafgen):
  - Rely on the interface provided by the kernel for packet IO
- Modern tools (e.g., MoonGen, pktgen, zsend):
  - Use special frameworks which bypass the network stack of an OS
  - Optimized for high speed and low latency
  - Cost: compatibility and support for high-level features
Measuring what happened.

- Recall: Active measurement (ping/iperf) &
  Passive measurement (tcpdump/intercept)

Passive measurement means we control the clocks, and we see what really happened (at least in one particular place)
How to capture traffic?

- When did the packet arrive?
  - A hard question!
- Can part / all of the packet be captured?
- How many packets can be captured?
- What is the maximal rate of packets that can be captured?
- ...
What is the time?

- Free running clocks, e.g.,
  - CPU’s time stamp counter (TSC)
  - NIC’s on board oscillator
  - Clocks drift!

- Synchronization signals, e.g.,
  - 1 PPS (pulse-per-second)

- Synchronization protocols, e.g.,
  - Network Time Protocol (NTP) – milliseconds accuracy
  - Precision Time Protocol (PTP) – microseconds accuracy (nanoseconds, depending on deployment)
## Timestamping

- **At the port – highest accuracy**
  - If you want to measure *the network*
- **At the NIC – less accurate**
  - Buffering, clock domain crossing etc.
- **At the OS**
  - Exhibits PCIe effects, scheduling dependencies
- **At the Application – least accurate**
  - Unless you are interested in the user's perspective – then it’s the *only* place
Traffic Capture

$$$$$, Hardware, high quality
(Ixia, Spirent,..)

$$ Software/hardware based, medium quality
(DAG, OSNT, NIC based,..)

Commodity, Software, low quality
(tcpdump, tshark, wireshark,.. )
tcpdump (libpcap)

- Software only
- libpcap (historically tcpdump)
- Other applications: tshark, wireshark…
- Captures data and <does stuff> including write stuff to a file
- Uses the pcap format (and others…)
- Timestamp comes from the Linux network stack (default: kernel clock)
PCAP Files

- PCAP – Packet CAPture
- libpcap file format
- Commonly used for packet capture/generation
- Format:

<table>
<thead>
<tr>
<th>Global Header</th>
<th>Packet Header</th>
<th>Packet Data</th>
<th>Packet Header</th>
<th>Packet Data</th>
<th>Packet Header</th>
<th>Packer Data</th>
</tr>
</thead>
</table>

- Global header: magic number, version, timezone, max length of packet, L2 type, etc.
- PCAP Packet header:

<table>
<thead>
<tr>
<th>ts_sec</th>
<th>ts_usec</th>
<th>incl_len</th>
<th>orig_len</th>
</tr>
</thead>
</table>

Networking and Systems Measurements (L50)
PCAP Files – a one slide outline

- PCAP – Packet CAPture
- libpcap file format
- Commonly used for packet capture/generation
- Format:

<table>
<thead>
<tr>
<th>Global Header</th>
<th>Packet Header</th>
<th>Packet Data</th>
<th>Packet Header</th>
<th>Packet Data</th>
<th>Packet Header</th>
<th>Packer Data</th>
</tr>
</thead>
</table>

- Global header: magic number, version, timezone, max length of packet, L2 type, etc.
- PCAP Packet header:

<table>
<thead>
<tr>
<th>ts_sec</th>
<th>ts_usec</th>
<th>incl_len</th>
<th>orig_len</th>
</tr>
</thead>
</table>

Networking and Systems Measurements (L50)
Packet Capture

Common example:

- $ sudo tcpdump -i en0 -tt -nn host www.cl.cam.ac.uk

tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 65535 bytes
1507838714.207271 IP 192.168.1.107.50650 > 128.232.0.20.80: Flags [S], seq 3761395339, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 256908862 ecr 0,sackOK,eol], length 0
1507838714.207736 IP 192.168.1.107.50651 > 128.232.0.20.80: Flags [S], seq 527865303, win 65535, options [mss 1460,nop,wscale 5,nop,nop,TS val 256908862 ecr 0,sackOK,eol], length 0

...
Where do I trace?

- Sometimes on the interface of a host (eg ‘eth0’)
  - `Tcpdump -i en1` # this will spew entries to the console one line per packet approximately
  - `-tt -nn` # useful options long form timestamps & numbers not names

- Interception using “Tap”
  (think wire-tapping)
Endace (DAG)

- DAG - Data Acquisition and Generation
- A commercial data capture card
- Packet capture at line rate
- Timestamping in the hardware (at the port)
- Nanosecond resolution
- Clock synchronization possible
- Will be used in the labs
erf. binary dec
.....0001 232ps,
.....0010 466ps,
.....0011 698ps,
.....0100 931ps,
.....0101 1163ps,
.....0110 1397ps
.....0111 1629ps
.....1000 1862ps

Why 232ps?
erf = extensible record format
• Measuring (latency/change) between locations
  ➢ Common time base: ntp? Ptp? GPS?

• Where do I measure?
  ➢ Nic?
  ➢ When the packet turns into useful work?

• Measuring inside the system (tracing a system)
NTP

- Designed for Internet-scale synchronization
  - E.g., email sent time < email received time
  - Milliseconds scale – emphasises frequency not phase

- A hierarchical system

- Using a few reference clocks

- Typically:
  - Host polls a few servers
  - Compensates for RTT and time offset
  - NTPv4 – RFC5905
PTP

- IEEE standard 1588 (v2 – 1588-2008)
- Designed for local systems
  - Microsecond level accuracy or better
- Uses a hierarchical master-slave architecture for clock distribution
  - Grandmaster – root timing reference clock
  - Boundary clock – has multiple network connections, can synchronize different segments
  - Ordinary clock – has a single network connection (can be master or slave)
- (And many more details)
PTP – clock synchronization

Timestamps known by slave:

Slave

Master

$t_1$, $t_2$, $t_3$, $t_4$

$t_2$, $t_1$, $t_2$

$t_1$, $t_2$, $t_3$

$t_2$, $t_1$

Slave

Master

$t_{master-slave}$

Mean propagation time $= (t_{master-slave} + t_{slave-master})/2$

$Offset = (t_{master-slave} - t_{slave-master})/2$
Using NIC

- Either implement PTP-derived timestamps or just timestamp the packets
  - sometimes in hardware
  - most times… not…
  - Not all NICs support time stamping
- Result: captured packets include a timestamp
- If PTP is used, end hosts are synchronized
- Else – free running counter
- Taken from *Understanding Software Dynamics* R. Sites
Figure 6: A typical user workflow in the general-purpose Dapper user interface.

- Taken from https://static.googleusercontent.com/media/research.google.com/en//archive/papers/dapper-2010-1.pdf
Capturing to disk…..

- Most (physical) disk systems can not capture 10Gb/s of data
- Capture takes resources!

- Format wars…. PCAP vs PCAP-ng vs others

- Binary representations / digital representations
What makes high-speed capture hard?

- Disk bandwidth
- Host bandwidth (memory, CPU, PCIe)
- Data management
- Lousy OS and software APIs
  - Byte primitives are dreadful when you want information on events, packets, & transactions…
  - A lot of effort has been invested into reinventing ring-buffers (circular buffers) to accelerate network interface cards.
  - Performance networking was done for capture first….
What makes high-speed capture work (better)?

- NVMe Disks
- Big machines, latest interfaces
- Collect metadata (version OS/system/hw/DNS)
- Bypass the OS
  - Older dedicated capture cards (e.g., Endace) pioneered kernel bypass capture
  - Any modern NIC 10Gb/s uses tricks that are useful for capture too
Measuring – Do’s and Don’t

- Make sure that you capture correctly
  - Disk, PCIe/DMA and other bottlenecks
- Make sure that your measurement does not affect the results
  - E.g., separate the capture unit from the device under test
- Understand what you are measuring
  - E.g. single host, application-to-application, network device etc.
- Make sure your measurement system does not affect the results
perf (not to be confused with iperf)

- So far we discussed *performance*
- What about *events*?
- Perf is a Linux profiler tool
- Allows us to instrument CPU performance counters, tracepoints and probes (kernel, user)
perf

- list – find events
- stat – count events
- record – write event data to a file
- report – browse summary
- script – event dump for post processing
Perf - example

```
:~/ssh$ perf stat ps
  PID  TTY          TIME       CMD
8747 pts/2    00:00:00   bash
11667 pts/2    00:00:00   perf
11670 pts/2    00:00:00   ps

Performance counter stats for 'ps':

12.745507 task-clock (msec)   # 0.929 CPUs utilized
    4 context-switches        # 0.314 K/sec
    0 cpu-migrations          # 0.000 K/sec
   140 page-faults            # 0.011 M/sec
  32,322,489 cycles          # 2.536 GHz              (40.80%)
<not supported> stalled-cycles-backend
<not supported> stalled-cycles-frontend
  27,644,922 instructions     # 0.86 insns per cycle  (68.86%)
  5,133,583 branches         # 402.776 M/sec        (68.92%)
 157,503 branch-misses       # 3.07% of all branches (94.06%)

0.013726555 seconds time elapsed
```

the tool scales the count based on total time enabled vs time running
Flame Graphs: an example of clever visualization

- Parsing traces is like finding a needle in a haystack
- Flame graphs - Visualise the outputs of profiling tools
  - E.g., using perf, dtrace
- Easy to understand
- Open source
  - https://github.com/brendangregg/FlameGraph
  - Brendan Gregg has several other useful performance-related tools
Flame Graphs

- Width is relative to “how much time spent running on the CPU”
- Top-down shows ancestry
- Not good for idles – so don’t try to use for profiling network events!
- Different types of flame graphs
  - E.g. CPU, memory, differential
Conclusion

- There are many .... So so many .... tools 
  each is shaped by its heritage

- Select carefully (understand the limitations)

- Consider and collect metadata – always

- How will you find/process/interpret/visualize your data?