Technische Universität München
Lehrstuhl Informatik VIII
Prof. Dr.-Ing. Georg Carle
Lukas Schwaighofer, M.Sc.
Florian Wohlfart, M.Sc.

Master Course Computer Networks

Project: Mininet

Last Update: January 20, 2014

General Information

Your overall task in this project is to measure the impact of *Tail Loss Probe* (TLP), an algorithm for the Transmission Control Protocol that can improve performance under specific conditions. The project will guide you through the process of creating a framework for measuring the performance impact using the network emulator Mininet.
For the duration of the project you will be provided with a virtual machine at the Chair of Network Architectures and Services with Mininet set up. These virtual machines are running a recent Linux Kernel (v3.11 "Linux for Workgroups"), which supports activating and deactivating the TLP algorithm. If you prefer to work on your own hardware, please make sure that your programs work and produce the same results on our virtual machine. We will use the virtual machines for grading your work. Information on how to access the virtual machines will be available on the website of the course[1] once they are available.
One of the aims of this project is to familiarize you with tools used for network research. The `python` programming language will be used for the programs created throughout the project. Adhere to the `python` style guide PEP 8[2] for your programs and don't forget to comment your program generously – code quality is part of the grade. You can use the `pylint` tool to get a rating of the quality of your code and suggestions for improvements. Important `python` modules used in the projects are `dpkt`, `numpy` and `matplotlib`. You will also use other tools like `tcpdump`, `wireshark`, `netcat` and `iptables`. This project was designed for students with some experience in working with Linux and using a shell. We recommend that at least one team member has some previous experience, otherwise you will require additional work.
After the team selection process is over, you will get a shared folder with your teammate in the Subversion repository of the course. The final result of each task has to be committed into the group folder before the deadline. The state of the repository at that time will be used for grading the respective part.

1 Project Plan – November 12, 2013 [1 point]

The goal of creating a project plan is that your team has a common understanding how you want to handle the project. Read all the tasks of the project and discuss them conceptually. How do you plan to split the work among yourselves? How much time do you plan to dedicate to the project? Are there specific difficulties you already anticipate?

[1] http://www.net.in.tum.de/en/teaching/ws1314/vorlesungen/master-course-computer-networks/
[2] http://www.python.org/dev/peps/pep-0008/

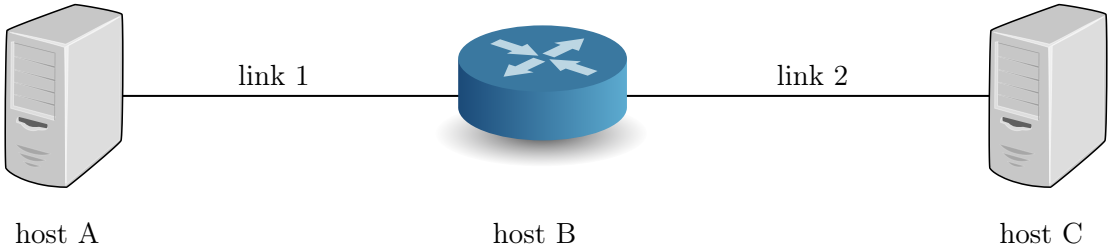

Figure 1: Mininet topology

configuration	link 1		link 2	
	bandwidth	latency	bandwidth	latency
fast	1 Mbit/s	5 ms	1 Mbit/s	5 ms
moderate	500 Kbit/s	10 ms	500 Kbit/s	10 ms
slow	100 Kbit/s	20 ms	100 Kbit/s	20 ms

Table 1: Topology configuration

We expect you, as a team, to exchange thoughts and think about the project. The output of this task should be about one page of text (or more if considered useful for your work), possibly including a Gantt chart to outline your schedule. The point will be given to you if this document shows that you have a clear conception of how you will proceed and a mutual understanding regarding the above questions.

2 A Mininet Topology – November 26, 2013 [2 points]

In this task you will create the framework for your experiments. After finishing you should have written a `python` program called `mininet_tlp_measurement.py` performing all these tasks (possibly importing other python files). The Introduction to Mininet[3] provides information about Mininet's `python` API. Document the usage of your `mininet_tlp_measurement.py` program in a `README` file.

a) Emulate the simple network topology shown in Figure 1 using Mininet. Mind the following notes:

- Host B is just a normal host configured to act as a router (do not forget to activate `ip_forward`)
- Use only hosts and links (no switches)
- Use IPv4 (otherwise there may be unforseen complications...)

Your setup should be created and destroyed by calling the newly created `mininet_tlp_measurement.py` program. The link configuration from Table 1 needs to be selectable using a command line parameter. **Hint**: use the `argparse` module for convenient parameter handling.

b) Extend the framework to start a TCP transfer between the hosts A and C using `nc6` (which is already installed on your virtual machine, see the `man` page for details). The TCP-transfer should be similar to a HTTP transfer: Host A should act as the server (serving data) and host C should serve as a client. The client C connects to the server A in order to request the data (use the `--rev-transfer` option for `nc6`). Use the special Linux-device `/dev/zero` as data source. You have to support the transfer sizes listed in Table 2, which need to be accessible using a command line parameter when

[3] https://github.com/mininet/mininet/wiki/Introduction-to-Mininet

configuration	TCP segments
short	64
medium	128
long	256

Table 2: Transfer size

calling `mininet_tlp_measurement.py`. Assume a Maximum Transmission Unit (MTU) of 1500 to calculate the number of bytes to transfer.
Note: Only consider the TCP-segments transferring the data for your computation.

c) Extend your framework to record the network traffic on the interface of host B connected to host A. Launch `tcpdump` (limiting the capture size to 68 bytes using the `-s` option) on host B to perform the capturing. The destination file is given to `mininet_tlp_measurement.py` as a command line argument.

Recommendation: At this point use `wireshark` on a generated pcap-file to verify your setup works as expected.

3 Causing Tail Loss – December 17, 2013 [2 points]

If the last segments of a TCP transmission are lost, we refer to this as "tail loss". In this scenario, the receiver won't send any duplicate acknowledgements because no additional TCP segments arrive. Without the TLP algorithm, TCP will wait until the retransmission timeout (RTO) timer has expired before resending segments. This leads to a quite long idle period. Information about the TLP algorithm is available as an Internet-Draft[4].
Before we can evaluate the effectiveness of this algorithm we need to be able to cause tail loss in our Mininet setup.

a) Write the `python`-program `drop_tail.py` using the `nfqueue` python bindings (see Examples[5]). The program may assume that the amount of data in each TCP segment of one TCP flow is constant. This program is responsible for creating the tail loss by attaching itself to a netfilter queue and decide which packets are sent and which are dropped. We will take care about sending packets to the queue in the next subtask.
`drop_tail.py` is started with at least the following two mandatory arguments:

1. Payload size of the TCP transmission in bytes
2. Number of TCP segments to drop at the end of the transfer

b) In this subtask the newly created `drop_tail.py` program is integrated into your framework (started using `mininet_tlp_measurement.py`). First we have to send the data packets into a netfilter queue. This can be done using the following `iptables` command on host B:

```
iptables -A FORWARD -i <in-if> -o <out-if> -p tcp -j NFQUEUE --queue-num 0
```

<in-if> and <out-if> need to be replaced with the ingoing and outgoing interface, such that the packets holding the data (not the acknowledgements) are sent to the queue. Then the `drop_tail.py` program is started on host B. Setting the number of dropped TCP segments must be possible using

[4] https://tools.ietf.org/html/draft-dukkipati-tcpm-tcp-loss-probe-01
[5] https://www.wzdftpd.net/redmine/projects/nfqueue-bindings/wiki/Examples

a command line parameter for `mininet_tlp_measurement.py` and handed to `drop_tail.py`. The payload size is already known and should also be given to `drop_tail.py`. Implementing the proper handling of the parameters is up to you.

Recommendation: At this point the generated packet captures of your framework should contain tail drops – use `wireshark` to verify this.

4 Evaluating Tail Loss Probe – January 26, 2014 [3 points]

The TLP algorithm is available since Linux 3.10 and also enabled by default. It can be controlled using the **sysctl** interface. Use the values 2 and 3 for the `tcp_early_retrans`[6] setting to disable and enable TLP respectively (i.e. delayed early retransmit should never be disabled).
Note: This setting can only be changed system-wide and is unavailable on the Mininet virtual hosts.

a) Write your own analysis tool `tcp_analysis.py` for the recorded pcap files using `python` and its `dpkt` module. The tool needs to be able to extract the following times (in milliseconds) from the pcap files for each TCP transfer:

- **Completion time**: The time elapsed between first and last observed packet of the TCP connection (remember that TCP connections are bidirectional).
- **Retransmission time**: The time elapsed between the first lost and the first retransmitted data packet. Note that this isn't the same packet when TLP is activated and more than one packet was dropped. If no retransmission occurred the value should be 0.

  Because of the short-term change of the definition for the retransmission time, the original definition (i.e. the time until the first dropped packet is retransmitted) is also acceptable. If you use this definition you have to mention that in the final assessment.

For each analyzed TCP flow output a text line containing first the completion time and then the retransmission time separated by a tabulator on standard output.

b) Automate measurements and analysis to get sufficient data for each combination of the configurations listed in the Tables 1 and 2, and with a tail loss of 1, 2, 4 and 8 TCP segments. Evidently you also need to do each measurement with both TLP enabled and disabled, so in total you have to measure $3^2 \cdot 4 \cdot 2 = 72$ different configurations.
For each measurement performed save both the completion time and the time until the retransmission of the first lost data segment (the output of `tcp_analysis.py`). Write the results of all repeated measurements of the same configuration into one file (i.e. you should have 72 files). Each file needs to hold **at least** ten measurements. Try to generate significantly more measurements if you are well in time.

c) Use `python matplotlib` (`pylab` module) to visualize the data collected in your measurements as box-and-whisker plots. For each of the nine combinations of configurations from Tables 1 and 2 you have to create two plots. A schematic example, what the plots should look like, is shown in Figure 2. The different tail loss length is to be on the x-axis. The boxes for the measurements without TLP should be slightly shifted to the left, for those with TLP enabled they should be shifted to the right. For the first plot, the completion time is used as the data. For the second, the retransmission time. In total, this should produce 18 plots.

[6] https://www.kernel.org/doc/Documentation/networking/ip-sysctl.txt

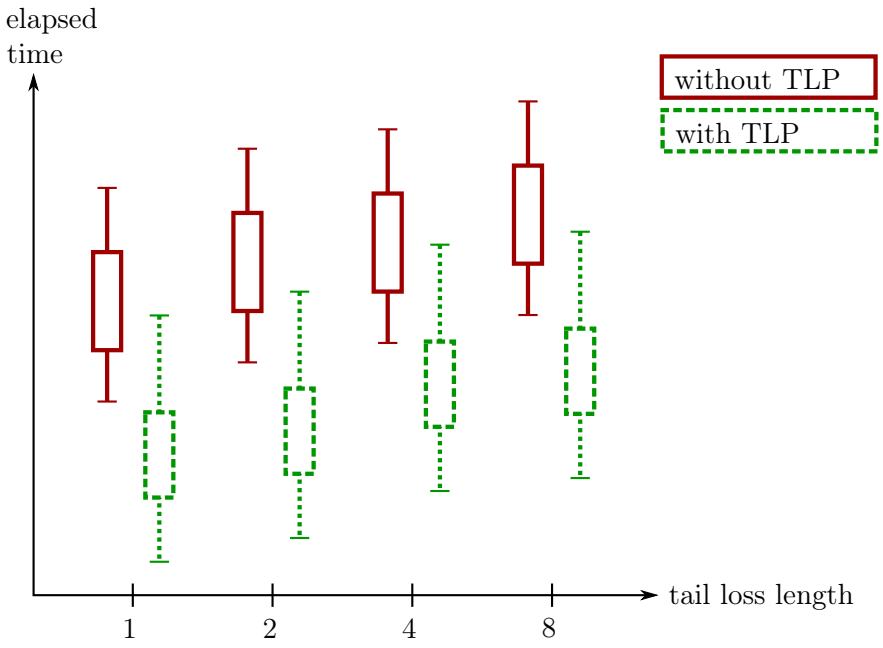

Figure 2: Box-and-whisker plot example

5 Final Assessment – February 4, 2014 [2 points]

Create a report about the project.

a) The first part of the report should be about your results and your findings. Specifically you should include some of your plots and (try to) explain the observed behavior. The scope of this part is about two pages of text (excluding the plots).

b) The second part is about reflecting your work and giving feedback. Specifically you should address the following questions:

- How much time did you invest into the project?
- Was it too easy/too difficult/just right?
- What did you like/dislike about the project?
- What would you do differently now?
- What would you tell people who still had to do this assignment?

Known Problems and Workarounds

This section explains how to deal with some issues that may arise in conjunction with this project and the provided Mininet virtual machine.

TCP segments are larger than allowed by MTU
We recommend not using the hierarchical token bucket (`use_htb=True`) when adding the links. Using the token bucket filter instead (`use_tbf=True`) solves the issue. Alternatively you can disable segmentation offloading using `ethtool`.

Mininet stops working
Our VMs are currently affected by a known Mininet problem published in the "Errata" section of the release notes[7]. If Mininet has stopped working for you and cleaning up using `sudo mn -c` doesn't help check if `/var/log/syslog` contains lines similar to:
`unregister_netdevice: waiting for lo to become free. Usage count = X`
If similar lines are present you have to reboot your VM (execute `sudo shutdown -r now`).

There is currently no solution for this problem. We are working on changing the setup to avoid triggering the bug. Meanwhile, the following suggestions seem to avoid the problem most of the time:

- Use the `-p` flag for `tcpdump` (disables promiscuous mode which is not needed)
- Wait at least 1 second after you have created your topology and before sending any data (use the `sleep` function in python's `time` module)
- After the transfer has finished stop all running processes (e.g. `tcpdump`) on all Mininet hosts and wait 1 second again before stopping Mininet.

If you found anything else that helps please let us know.

The nfqueue callback function is not working
Because the signature of the callback function is currently changing, the recommended definition (functional with both the old and the new signature) is as follows:

```
def callback(handle, handle_new=None):
    """Your docstring here"""
    if handle_new is not None:
        handle = handle_new
    # work with handle here
    # (...)
```

[7] https://github.com/mininet/mininet/wiki/Mininet-2.1.0-Release-Notes