WirelessLab WS 2016/17

Homework 9: Transport over Wireless

Group 6

Gasper Kojek, Jens Klein

Terminology

- N6: Node 6
- N15: Node 15
- ST: Stepping Stone
- PC: Personal Computer running Linux or Mac OS X

Question 1: Setup

a)

already covered in previous tasks.

SSH Config File on PC for simpler access:

```
host wlab
  HostName wirelesslab.inet.tu-berlin.de
  IdentityFile ~/.ssh/id_rsa_wlab
  User group06

host node6
  HostName 172.17.3.106
  User root
  ProxyJump wlab

host node15
  HostName 172.17.3.115
  User root
  ProxyJump wlab
```

b)

for this task we need the already installed tools:

- iperf
- iw
- tcpdump
- nc

c)

State of `/etc/config/network` on N6

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'lan'
        option type 'bridge'
        option ifname 'eth0'
        option proto 'dhcp'

config interface 'wlan0'
        option ifname 'wlan0'
        option ipaddr '172.17.5.10'
        option netmask '255.255.255.0'
        option proto 'static'
        option dns '172.17.255.254'
```

No change required!

State of `/etc/config/network` on N15

```
config interface 'loopback'
        option ifname 'lo'
        option proto 'static'
        option ipaddr '127.0.0.1'
        option netmask '255.0.0.0'

config interface 'lan'
        option type 'bridge'
        option ifname 'eth0'
        option proto 'dhcp'

config interface 'wlan0'
        option ifname 'wlan0'
        option ipaddr '172.17.5.11'
        option netmask '255.255.255.0'
        option proto 'static'
        option dns '172.17.255.254'
```

No change required!

Updated wireless config `/etc/config/wireless` on N6 (AP)

```
config wifi-device 'radio0'
        option type 'mac80211'
        option channel '11'
        option hwmode '11g'
        option path 'pci0000:00/0000:00:0c.0'
        option disabled '0'

config wifi-iface 'default_radio0'
        option device 'radio0'
        option mode 'ap'
        option encryption 'none'
        option ssid 'group06_ap'
        option network 'wlan0'
        option ifname 'wlan0'
```

Apply changes on N6 with: `wifi`

Updated wireless config `/etc/config/wireless` on N15 (STA)

```
config wifi-device 'radio0'
        option channel '11'
        option hwmode '11g'
        option path 'pci0000:00/0000:00:0e.0'
        option disabled '0'
        option type 'mac80211'

config wifi-iface 'default_radio0'
        option device 'radio0'
        option mode 'sta'
        option ssid 'group06_ap'
        option ifname 'wlan0'
        option network 'wlan0'
```

Apply changes on N15 with: `wifi`

Check for connection with `ping`

From N15 to N6:

N15: `ping 172.17.5.10`

Output (head):

```
PING 172.17.5.10 (172.17.5.10): 56 data bytes
64 bytes from 172.17.5.10: seq=0 ttl=64 time=3.817 ms
```

N6 is reachable from N15.

From N6 to N15:

N6: `ping 172.17.5.11`

Output (head):

```
PING 172.17.5.11 (172.17.5.11): 56 data bytes
64 bytes from 172.17.5.11: seq=0 ttl=64 time=1.995 ms
```

N15 is reachable from N6.

From ST to N6:

ST: `ping 172.17.3.106` (ip address of wired interface)

Output (head):

```
PING 172.17.3.106 (172.17.3.106) 56(84) bytes of data.
64 bytes from 172.17.3.106: icmp_seq=1 ttl=64 time=0.787 ms
```

N6 is reachable from ST.

From ST to N15:

ST: `ping 172.17.3.115` (ip address of wired interface)

Output (head):

```
PING 172.17.3.115 (172.17.3.115) 56(84) bytes of data.
64 bytes from 172.17.3.115: icmp_seq=1 ttl=64 time=0.712 ms
```

N15 is reachable from ST.

d)

N6 has a monitor capable interface. we called it `wlan1` With the wireless configuration above we already configured it.

Now test if it still works:

N6: `tcpdump -i wlan1 -c 1`

- -i argument defines the interface on which we want to capture
- -c argument is the number of packets we want to trace

Output:

```
tcpdump: WARNING: wlan1: no IPv4 address assigned
tcpdump: verbose output suppressed, use -v or -vv for full protocol decode
listening on wlan1, link-type IEEE802_11_RADIO (802.11 plus radiotap header), capt
ure size 65535 bytes
11:50:26.198838 828674862us tsft 24.0 Mb/s 2412 MHz 11g -71dB signal [bit 29] Beac
on (TUB-intern) [18.0 24.0* 36.0 48.0 54.0 Mbit] ESS CH: 1
1 packet captured
118 packets received by filter
0 packets dropped by kernel
```

Monitoring works.

Question 2: TCP performance

a)

following commands are issued both on N6 and N15

monitor interface uses ath9k card, all other ath5k

Transmission power: 1 dBm

```
iw dev wlan0 set txpower fixed 100
```

Transmission rate: 6 Mbps

```
iw dev wlan0 set bitrates legacy-2.4 6.0
```

HW mode

```
uci show wireless.radio0.hwmode
```

Output: `wireless.radio0.hwmode='11g'` on both N6 and N15

Disable ANI

```
echo ani-off > /sys/kernel/debug/ieee80211/phy0/ath5k/ani
```

Test: `cat /sys/kernel/debug/ieee80211/phy0/ath5k/ani | grep "operating mode"`

Output: `operating mode: OFF`

Set Noise Immunity Level to 0

```
echo noise-low >/sys/kernel/debug/ieee80211/phy0/ath5k/ani
```

Test: `cat /sys/kernel/debug/ieee80211/phy0/ath5k/ani | grep "noise immunity"`

Output: `noise immunity level: 0`

Enable OFDM Weak Signal Detection

`echo ofdm-on > /sys/kernel/debug/ieee80211/phy0/ath5k/ani`

Test: `cat /sys/kernel/debug/ieee80211/phy0/ath5k/ani | grep "OFDM weak"`

Output: `OFDM weak signal detection: on`

RX and TX Antenna = 2 and disable diversity

`echo fixed-b > /sys/kernel/debug/ieee80211/phy0/ath5k/antenna`

Test:

```
echo clear >  /sys/kernel/debug/ieee80211/phy0/ath5k/antenna
cat /sys/kernel/debug/ieee80211/phy0/ath5k/antenna | grep "\[antenna"
```

Output:

```
[antenna 1] 0   0
[antenna 2] <some number gt 0>  <some number gt 0>
[antenna 3] 0   0
[antenna 4] 0   0
```

Only Antenna 2 sends and receives

b)

Template for a capture

Start nc to receive the trace:

ST: `nc -l -p 8080 > "trace-<tcp/udp>-<6/24/54>mbps-$(date +%s).cap"`

Start iperf server (tcp)

N6: `iperf -s`

Start iperf server (udp)

N6: `iperf -s -u`

Start tcpdump on monitor interface on STA

N6:
`tcpdump -i wlan1 -w- <udp/tcp> and ip src or dst 172.17.5.10 | nc 172.17.3.1 8080`

- -i defines the interface to capture

- -w- outputs to STDOUT
- udp/tcp filters only udp packets
- ip src or dst filters only packets coming or going from host x

Start sending packets with iperf

N15:

`iperf -c 172.17.5.10 -u -b <7/25/56>M -t 60` * -u for UDP, for TCP omit this flag * -b sets the bandwidth used (saturated channel) * -t 60 runs this for 60 seconds

Perform runs with these parameters:

- Transmission rate 6 Mbps, udp
- Transmission rate 24 Mbps, udp
- Transmission rate 54 Mbps, udp
- Transmission rate 6 Mbps, tcp
- Transmission rate 24 Mbps, tcp
- Transmission rate 54 Mbps, tcp

Capturing data

We tied all these commands together using two scripts, first one was using Expect, which we used to automatically connect to the nodes and the ST and execute the commands. Second one was just to run the first one with different settings for the 6 different one. The second script was then added to crontab on our personal RaspberryPi server and it ran every 15 minutes.

- Script 1 (Expect): `hw09.exp`
- Script 2: `hw09.sh`
- crontab entry: `*/15 * * * * /home/pi/WirelessLab/hw09/hw09.sh`

Overview of the captures

We also provided a script to display the capture times of the files in a more human readable format.

`timetable.py` output the following overview:

```
Ordered Overview of capture-time and settings
=======================================================
| 13:06:13 18.01.2017  | PROTO: udp | TX RATE:  6mbps |
| 13:07:31 18.01.2017  | PROTO: udp | TX RATE: 24mbps |
| 13:08:49 18.01.2017  | PROTO: udp | TX RATE: 54mbps |
| 13:10:07 18.01.2017  | PROTO: tcp | TX RATE:  6mbps |
| 13:11:25 18.01.2017  | PROTO: tcp | TX RATE: 24mbps |

( TRUNCATED )

| 20:32:53 18.01.2017  | PROTO: udp | TX RATE: 54mbps |
| 20:34:11 18.01.2017  | PROTO: tcp | TX RATE:  6mbps |
| 20:35:28 18.01.2017  | PROTO: tcp | TX RATE: 24mbps |
| 20:36:46 18.01.2017  | PROTO: tcp | TX RATE: 54mbps |
| 20:45:18 18.01.2017  | PROTO: udp | TX RATE:  6mbps |
| 20:46:35 18.01.2017  | PROTO: udp | TX RATE: 24mbps |
| 20:47:53 18.01.2017  | PROTO: udp | TX RATE: 54mbps |
| 20:49:11 18.01.2017  | PROTO: tcp | TX RATE:  6mbps |
-------------------------------------------------------

190 captures in total
```

The full output is saved in `timetable_overview.txt`

c)

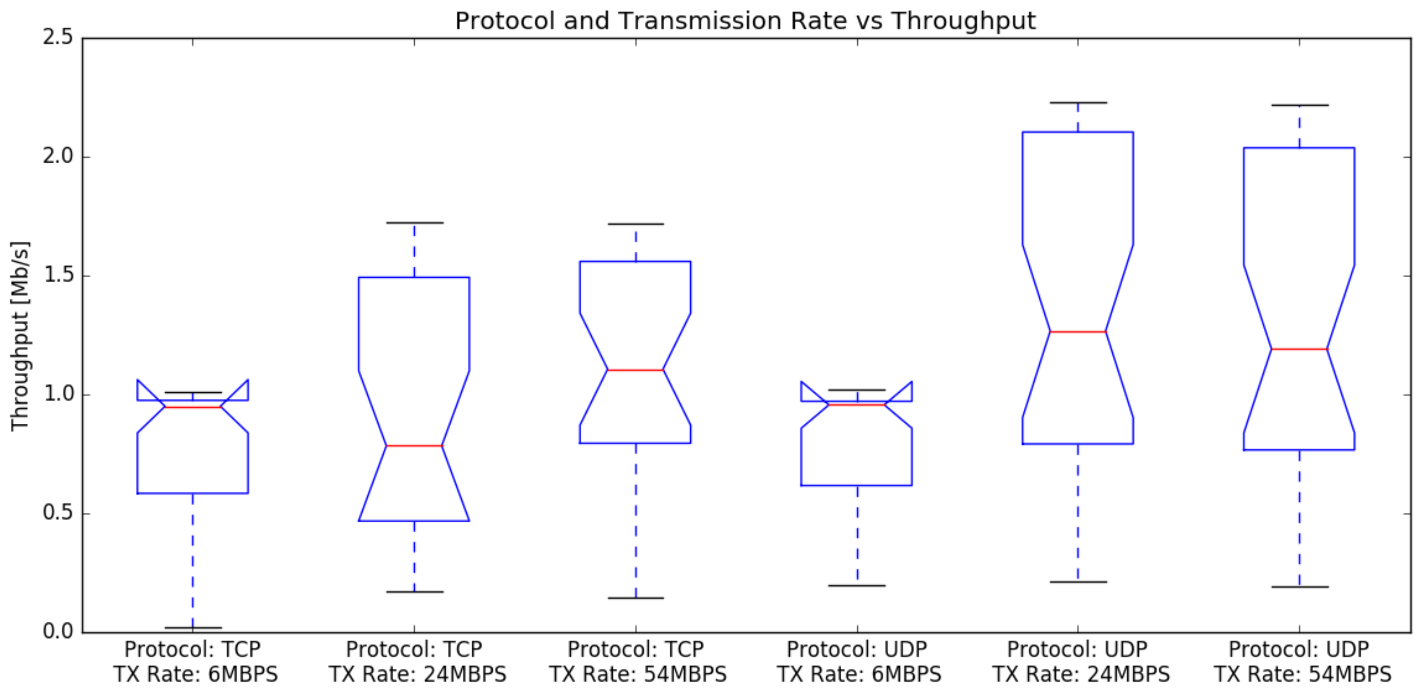
Part 1: TCP and UDP throughput

Data extraction:

As our iperf output was not always complete, we had get the throughput by calculating it with the traces. Therefore we jused the conversation statistic of tshark that gives us received bytes and frames. With the knowledge of the overhead size introduced by the given protocol we could compute the throughput that way. Because we captured on the receiver we are can assure to count only packets that weren't lost. Retransmission occured really rarely that we can neglect the impact on the throughput.

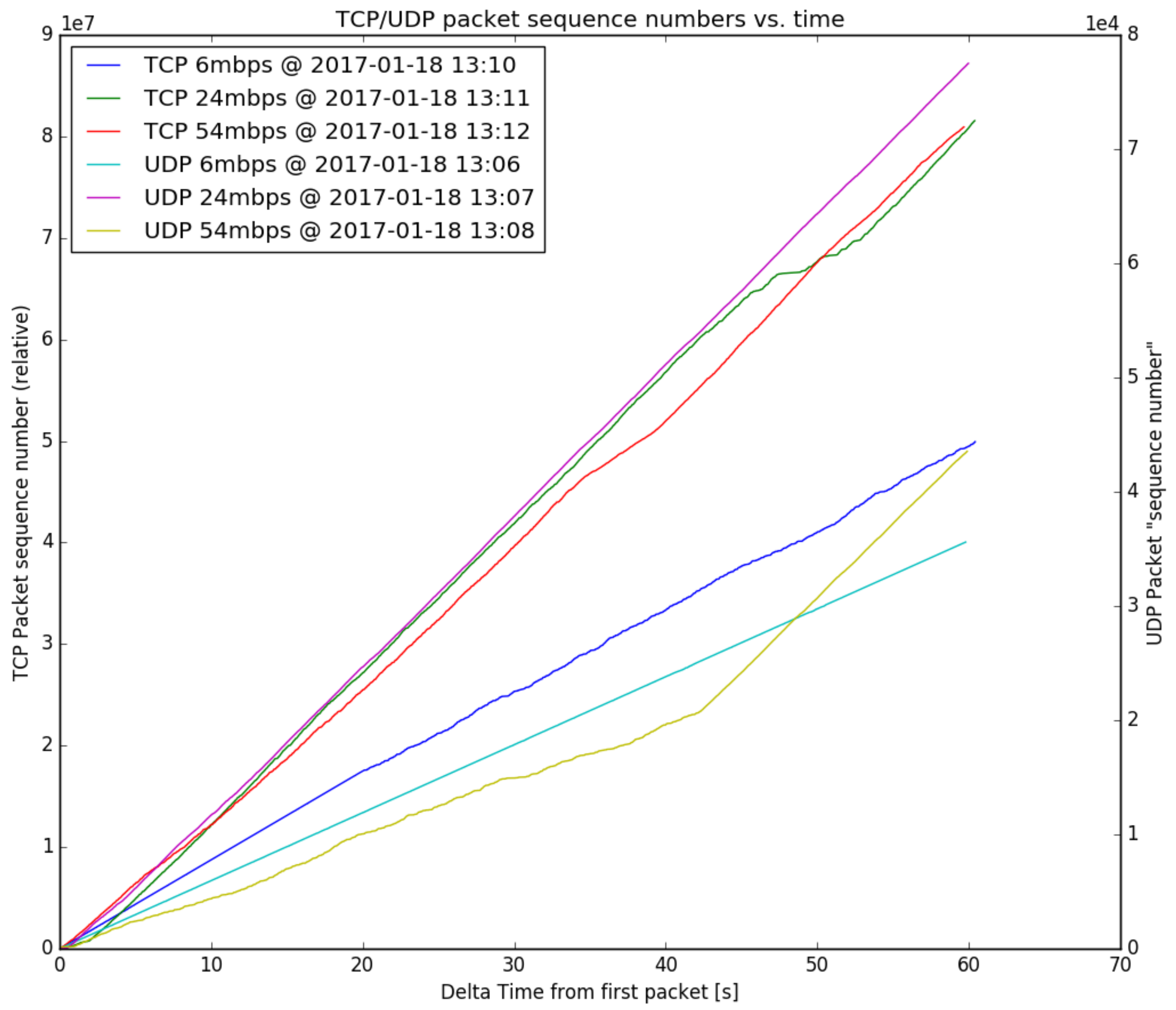
The script for computing and plotting is called `analyse_capture_throughput.py`

The plot



The first three boxplots are TCP packets and the remaining ones are UDP packets. We can see that the TP (Throughput) on 6 Mbps has almost exactly the same characteristic in both protocols. The difference gets bigger with increasing the transmission rate. TCP performs worse in terms of TP than UDP. One possible reason is the additional overhead introduced by TCP to the medium. But we can see that TCP has slightly more steady TP during the whole capture time.

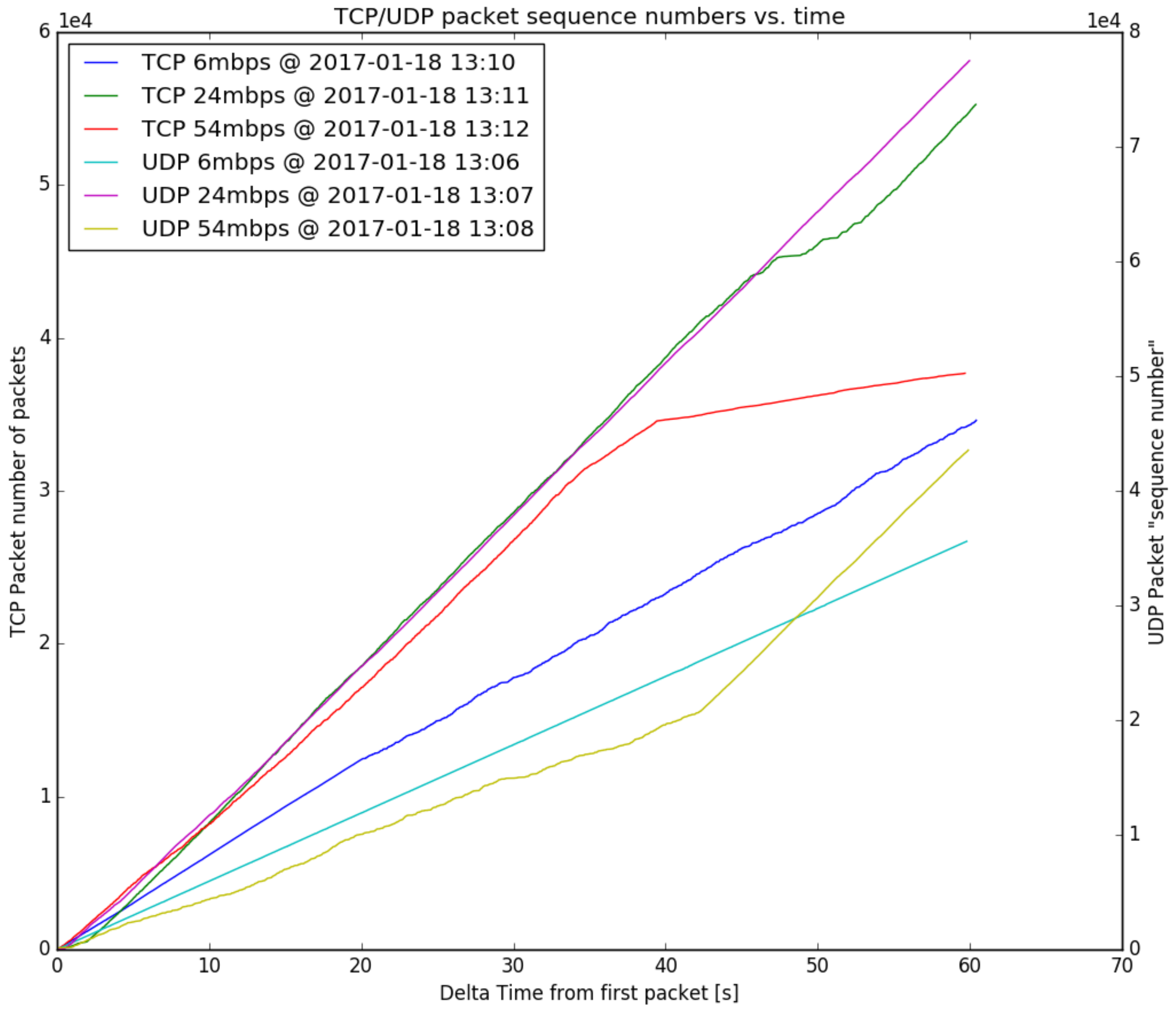
Part 2: Plot packet sequence number versus time



TCP generates its packet sequence numbers usually with a incremental random generator, whereas UDP does not support packet numbers natively. `iperf` , which we used to generate traffic in our experiments actually includes its own packet numbers in the UDP data payload, which has no connection to the UDP protocol as such, except that it is used for delivering the packet. `iperf` makes its packet numbers strictly sequential, with the difference of 1.

Because of that, we believe that this comparison holds no useful value, as TCP packet sequence numbers in our test increment on average for 1400 between packets on 6 and 24 Mbps and 2000 when 54 Mbps was used. Because of the random nature of increments in TCP we cannot know if any changes in slope were because of slower traffic (lower throughput) or because of the random generator.

If we instead look at a graph where we for TCP plot the converted sequence number (if current sequence number is the same as last, increment last by one and plot it like that), which basically means that we see how packets were delivered, we can see a much different picture, which is in our opinion much better suited for comparison of TCP and UDP.



Here we can see that at 6 Mbps TCP delivered more packets in the same time, suggesting a higher throughput. At 24 Mbps, UDP delivered slightly more packets, but we can see that packet delivery speed of TCP has fallen in the last 15s, which suggests that the channel was busier or TCP congestion control kicked in (perhaps somebody else started running their test). At 54 Mbps TCP again delivered more packets, but here the UDP test had a lower packet delivery speed in the first 40s than the last 20, whereas with TCP its packet delivery speed has fallen off in the last 20s. If we were to project the better packet delivery speed of UDP from the start, UDP would deliver more packets.

We expected that UDP will deliver its packets faster, because TCP also implements its own acknowledgements, which take time to transmit. When we checked that in wireshark, we saw that TCP acknowledgements accounted for around 35% of TCP packets (but not 35% traffic, as they are much smaller than data packets). UDP does not implement acknowledgements, because it also does not have the mechanism to detect correct delivery of packets. That is mostly confirmed by our tests, where UDP delivers its packets faster (it also has a bigger payload in the same packet length).