Networking Lab 2 – Introduction to Wireshark

The goal of this lab is to understand and get used to the usage of Wireshark, a network analyzer software, or sniffer. Wireshark is program that allows the user (with the proper rights) to capture all packets that are received and sent by a network adapter, and to visualize them using advanced features. Thanks to the knowledge of protocols themselves, Wireshark can correctly decode almost all protocols headers, showing very detailed (and often too detailed) protocol information.

1. Wireshark filters

Filters are useful to limit the number of packets exposed to the user. When capturing packets in operative network, a network interface typically receives several hundreds of packets per seconds. Thus, it is necessary to select and filter those packets the network analyst is interested into.

Wireshark offers two types of filters

- **Capture filters**: packets that do not pass the filter are discarded and not captured at all
- **Display filters**: captured packets that pass the filter are visualized. Others are only temporarily hidden.

Capture filters are useful to limit the amount of packets Wireshark will capture. Wireshark uses the `libpcap` language syntax for capture filters. This is explained in the `tcpdump` man page and we will not use them in the course.

Display filters allow you to concentrate on the packets you are interested in, while hiding the currently uninteresting ones. They allow you to select packets by protocol, by the presence of a field, by the values of fields, or by a comparison between fields, and a lot more.

To specify a filter, simply enter it in the filter box. Experiment with it, using the GUI.

2. Simple capture session

1. Configure the testbed as usual.
2. Start capturing packets on `H2` BEFORE sending any packet
3. From `H1`, start sending `ping` messages to `H2` and from `H2` start sending ping messages to `H3`
4. Stop capturing after about 60s, and stop the ping commands

Get confident with the Wireshark GUI, experimenting with the various menu voices that the program offers.

1. Define the display filter that allows to see all PDUs involving `H2` and `H3` only.
2. Set a coloring rule so that `ICMP echo requests` and `ICMP echo reply messages` are highlighted using different background colors
3. Consider a `ICMP-req` message, and check ALL protocols headers you can identify
   a. At the Ethernet layer
   b. At the IP layer
   c. At the ICMP layer
   d. At the application layer
4. **Is there any field that is missing**? **What are the reasons to have some headers**? What could happen if such a header were missing? For example, look at the ICMP header.

- How many bits are used for the sequence number?
- Why you need an identifier fields?
- How is the RTT computed? Why?

Consider `ARP-req messages` and `ARP-rep messages` and check the Ethernet headers.

- What is missing? Recall, an Ethernet payload MUST be larger or equal than 46B (and shorter than 1500B)

3. Connectivity check – basic ping

Using the ping command, check if it is possible to reach other hosts in your LAN. Run the command

```
H1: ping H2 -c 4
PING H2 (H2) X(Y) bytes of data.
Z bytes from H2: icmp_seq=1 ttl=64 time=18.6 ms
Z bytes from H2: icmp_seq=2 ttl=64 time=0.127 ms
Z bytes from H2: icmp_seq=3 ttl=64 time=0.127 ms
Z bytes from H2: icmp_seq=4 ttl=64 time=0.125 ms

--- H2 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.125/4.758/18.656/8.024 ms
```

1) What are the numbers **`X,Y,Z`** ?
2) How can the ping program report the `icmp_seq`? Why there is such a header? What could happen if such header were not present?
3) What is the **`ttl`** value? Is it the sender or receiver TTL? Who decides to use 64?
4) What does it change if you change the size of the requests using the `-s<size>` option? Which is the minimum size that allows ping to measure and report the "`time`" field? Why this is happening? If you were the programmer working on the implementation of a `ping` command, how would you implement the "`time`" measure?
5) Look at the time measured for the first and other requests. Why are so different?
6) What happens if `H1` tries to ping a host that is not active but belonging to your subnet, e.g., `H1: ping H4`? Is there any packet that is sent on the LAN (check the leds on the switch)? Which packets are those?
7) What happens if `H1` tries to ping a host which is not active and does NOT belong to your subnet, e.g., `HH1`? Is there any packet that is sent on the LAN (check the leds on the switch)? Which packets are those?
   `H1: ping UNKOWN1`
8) Check IP fragmentation. Send large messages, and eventually using the –M option, observe how IP packets and fragments are built. How can the receiver reassemble the packet from received fragments? How can eventually reorder them? How can it detect that all fragments have been received? In which order fragments are sent by the sender? Describe the possible algorithm the receiver implements.

3. Arp table management

Using the `arp` command, check how the system handles the ARP tables.

Suggestion: to clean up arp tables, put the interface `DOWN` and then `UP` again

1) Check the status of the ARP table before and after issuing a ping command. If hosts `H1` pings hosts `H2`, which hosts update their ARP tables?
2) Did host `H3` update it? Why?
3) What happens if `H1` tries to ping a host that is not active but belonging to your subnet, e.g., `H4`? How does the arp table of `H1` change?
4) For how long and entry remains into the arp table? Suggestion: use the command `date` to track time and keep checking the arp entry. The PC can run simple programs, like `while true; do arp; date; sleep 1; done`
5) Is there any difference considering “`incomplete`” entries?

4. Arp requests and link failure

Delete all entries from the arp tables of `H2` and `H1`. Start pinging `H2` from `H1`. After about 90s, simulate a link failure by disconnecting the cable from the switch. Observe the arp messages exchanged between `H2` and `H1` using Wireshark

1) What happens during the first 90s when everything is working?
2) What happens if the failure lasts less than 5s? Check the arp tables of H1 and H2
3) What happens if the failure lasts more than 30s? Check the arp tables of H1 and H2
4) What happens as soon as the link is reestablished at the physical layer? CHECK THE RTT measurements of ICMP messages after the link goes up again

5. Connectivity check – Advanced ping

In this section we try to understand what happens in some strange configurations

1) What happens when a host pings the broadcast address?
2) What happens when a host pings the networking address?

Duplicate addresses: Configure your LAN so that there are two hosts with the same IP address, i.e., `H1`, `H1`’, `H2`.

1) What happens when `H2` pings `H1`? Check the arp tables of `H1`, `H1`’, and `H2`
2) What happens when `H1` and `H1`’ starts pinging `H2` at the same time? Check the arp tables.

Wrong Netmask: Configure your LAN so that `H1` sees `H2` as belonging to his subnet, but `H2` sees `H1` as NOT belonging to his subnet. Suggestion – use different Netmask for `H1` and `H2`

1) What happens when `H1` pings `H2`? Which packets is `H1` sending? Which packets is `H2` sending? How do arp tables of `H1` and `H2` change?
2) What happens when `H2` pings `H1`? Which packets is `H1` sending? Which packets is `H2` sending? How do arp tables of `H1` and `H2` change?

Wrong Netmask and conflict with broadcast address Configure your LAN so that

- `H1` has address 172.16.0.127 and Netmask 255.255.255.0
- `H2` has address 172.16.0.1 and Netmask 255.255.255.128

1) What happens when `H1` pings `H2`? Which packets is `H1` sending? Which packets is `H2` sending? How do arp tables of `H1` and `H2` change?
2) What happens when `H2` pings `H1`? Which packets is `H1` sending? Which packets is `H2` sending? How do arp tables of `H1` and `H2` change?