Measurement guide and report template

Investigating Ethernet and TCP/IP protocols

V04b (2016.02.17.)

Measurement assembled by: KOVÁCSHÁZY Tamás, BME MIT, 2008-2010.

Report template modified by: TÓTH Csaba, BME MIT, 2011.

Measurement tasks and report template modified by: HUSZERL Gábor, BME MIT, 2012-2013.

Report template modified by: NASZÁLY Gábor, BME MIT, 2014.

Report template translated to English by: NASZÁLY Gábor, BME MIT, 2014.

Report template modified by: NASZÁLY Gábor, BME MIT, 2016.

|  |  |
| --- | --- |
| Location of measurement: | **I.B141/IB142, place #:** |
| Date / time of measurement: |  |
| Students: | **name, group**  **name, group** |
| Name of this file: | **<file\_name>**  (ml4 \_<course><group>\_<number\_of\_measurement>.doc,  eg. **ml4\_ml4a01\_1.doc**) |
| Lecturer: | **...** |

**How to treat this document:**

* Fill in the bright yellow fields only! Do not forget to fill in the above header too!
* Replace the <<screenshot>> texts with the corresponding screenshot required!
* The tasks marked with \* are advanced level tasks. This means you can complete this measurement without them but if you want to get a mark of 5 to the exam measurement at the end of the term you have to be able to accomplish them (at the exam).
* Some kinds of tasks are optional. This means you do not have to accomplish them to pass this measurement but they contain useful additional information and you may find them interesting.
* If you have any feedback on the measurement feel free to tell it to the lecturer!

# Task: Ethernet and TCP/IP settings

**Aim of the task:** to get familiar with the basic Ethernet and TCP/IP settings by the help of command line tools **ipconfig** and **route print**.

## Network interfaces and their settings

Find out what kind of network interfaces the computer used has and what are their network settings!

Run the commands **ipconfig** then **ipconfig /all** in a Command Prompt window! Save the results to this report! (An efficient solution is to redirect the output of the commands to a file (like: **ipconfig > ip\_settings.txt**), then copy the content of the file into this report.)

|  |
| --- |
| **ipconfig** |
|  |

|  |
| --- |
| **ipconfig /all** |
|  |

Elaborate and compare the results of **ipconfig** and **ipconfig /all** then fill in the table bellow! (You do not have to care about tunnel adapters, they are for implementing IPv6 functions in IPv4 environment. We do not use them during the measurement.)

Network interface settings:

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | ***…*** | ***…*** | ***…*** |
| IP address |  |  |  |
| Subnet mask |  |  |  |
| Default gateway |  |  |  |
| DHCP enabled? |  |  |  |
| DNS servers |  |  |  |
| Connection specific DNS suffix |  |  |  |

Look at the other command line switches and parameters of **ipconfig** (**ipconfig /?**)!

(For example **/flushdns** will be needed during the measurement.)

|  |
| --- |
| **ipconfig /?** |
|  |

## IP address ranges

What are the valid addresses on the local subnet of the laboratory (both “from-to” notation and “subnet address/mask bit” notation)?

|  |  |
| --- | --- |
| **IP range:** |  |

## (Deleted task)

## DNS servers

|  |
| --- |
| Why does it useful to set more than one DNS server? |
|  |
| Why do we use IP addresses instead of domain names to set the DNS servers? |
|  |

## Internet routes

Run the command **route print**!

(Only the “IPv4 Route Table” portion is interesting now.)

|  |
| --- |
| **route print** |
|  |

|  |
| --- |
| What does a row represent in the result? What do the columns mean?[[1]](#footnote-1) |
|  |
| How does the TCP/IP protocol stack of the computer use the information stored in the routing table? |
|  |
| What are those **0.0.0.0** and **127.0.0.0** row? When / for what are they used? |
| **0.0.0.0:**  **127.0.0.0:** |
| And the **224.0.0.0** rows? |
| **224.0.0.0:** |

# Task: connection between Ethernet and IP addresses

**Aim of the task:** understanding the connection between Ethernet and IP addresses by the use of command line tool **ping**.

**Ping** is a simple network test tool that sends an ICMP Echo Request message (by default four times) to the network node passed as a command line parameter (either IP address or domain name). The node usually answers by an ICMP Echo Request (unless a network firewall blocks it).

During the measurement we use ARP (*Address Resolution Protocol*). This protocol assigns Ethernet addresses to IP addresses. The ARP implementations usually maintain a cache to minimize the number of ARP requests (for example on Windows XP this cache is valid for 10 minutes). The content of this cache can be displayed by the command line tool **arp -a**. The cache can also be deleted by issuing **arp -d \***! Deleting the cache requires administrative privileges. You can gain this privileges by starting a so called “elevated” Command Prompt (you can find a suitable start icon on the tray).

Notice: if for some kind of odd reason **arp -d \*** does not work, you can try issuing the following command to delete the ARP cache:

**netsh interface ip delete arpcache**

This command also requires administrative privileges: Command Prompt (run as admin).

Investigate the other parameters of **arp** (**arp /?**)!

During this task we use DNS (Domain Name System) too. This protocol assigns domain names to IP addresses (more on this later). Like with ARP to minimize the number of requests DNS implementations maintain a DNS cache (on Windows the entries are valid for 24 hours). This cache can be displayed by issuing the command **ipconfig /displaydns** and deleted with **ipconfig /flushdns**.

You have to **ping** five different target nodes. Each time you also have to capture and records the network traffic by the help of the application named **Wireshark**. By investigating the traffic you can answer the questions.

The nodes to be pinged:

* 1. default gateway,
  2. DHCP server,
  3. a computer outer to the local subnet of the laboratory (alaplab.mit.bme.hu),
  4. a domain name with no computer behind it (fantom.mit.bme.hu),
  5. the subnet broadcast.

Before every of the above mentioned measurements delete both the ARP and DNS caches! (This is required to observe fully the protocols in operation.) To accomplish this it is worth to create a batch script file! (A batch file is a simple text file having an extension of either **.bat** or **.cmd**. Every line of the file contains a command to run. It is also possible to pass command line parameters to the batch file. You can refer to these parameters with **%1**, **%2** and so on (for the first, the second parameter respectively…).

Measurement procedure:

1. Create a batch file the deletes the ARP and DNS caches then issue a **ping** command with its first parameter! (Do not call the batch file “ping”. Think over why would it be a bad idea…)
2. Start capturing the network traffic in Wireshark!
3. Using your batch script ping the network node in question!
4. Stop capturing the network traffic!
5. Put a screenshot of the Command Prompt window (in which you ran your batch file) to this report!
6. Create and use a suitable display filter to filter out any network traffic what is not relevant to the operation of ping (and the underlying ARP and DNS protocols)!
7. Put a screenshot of Wireshark to this report (if possible, the whole window, if not, just a relevant part of it)!
8. **Save the captured data for further analysis!**
9. Repeat the steps (2-8) for the other four nodes to be pinged!

## Creating the batch file

|  |
| --- |
| Create the required batch script file to delete the ARP and DNS cache then invoke **ping**!  Copy the content of the file bellow: |
|  |

## Pinging the default gateway

|  |
| --- |
| Ping: default gateway – the screenshot of cmd window: |
| <<screenshot>> |
| Ping: default gateway – screenshot of captured network traffic in Wireshark (without filter) |
| <<screenshot>> |

## Pinging the DHCP server

|  |
| --- |
| Ping: DHCP server – the screenshot of cmd window: |
| <<screenshot>> |
| Ping: DHCP server – screenshot of captured network traffic in Wireshark (with display filter) |
| <<screenshot>> |

## Pinging a computer outer to the local subnet of the laboratory

|  |
| --- |
| Ping: alaplab.mit.bme.hu – screenshot of cmd window: |
| <<screenshot>> |
| Ping: alaplab.mit.bme.hu – screenshot of captured network traffic in Wireshark (with display filter) |
| <<screenshot>> |

## Pinging a domain name with no computer behind it

|  |
| --- |
| Ping: fantom.mit.bme.hu – screenshot of cmd window: |
| <<screenshot>> |
| Ping: fantom.mit.bme.hu – screenshot of captured network traffic in Wireshark (with display filter) |
| <<screenshot>> |

## Pinging the subnet broadcast address

|  |
| --- |
| Ping: subnet broadcast address – screenshot of cmd window: |
| <<screenshot>> |
| Ping: subnet broadcast address – screenshot of captured network traffic in Wireshark (with display filter) |
| <<screenshot>> |

## Question on ARP and DNS requests

|  |  |  |
| --- | --- | --- |
| In the following cases are there ARP and DNS request(s)? If yes, for what IP address and domain name, if not, why not? | | |
|  | ARP request(s) | DNS request(s) |
| Default gateway: |  |  |
| DHCP server: |  |  |
| alaplab.mit.bme.hu: |  |  |
| **Explanation:** | | |

## Question on used Ethernet and IP addresses

|  |  |  |
| --- | --- | --- |
| What Ethernet and IP addresses were used during the **ping** command for the following cases (and why)? | | |
|  | Target Ethernet address | Target IP address |
| Default gateway |  |  |
| DHCP server |  |  |
| alaplab.mit.bme.hu |  |  |
| **Explanation:** | | |

## Question on routes

|  |
| --- |
| How do these pinging measurements connect to the output of route print in the previous task? |
|  |

## Question on network equipment manufacturers

|  |  |
| --- | --- |
| From a given Ethernet address one can tell the manufacturer of the network equipment having that address. What are the manufacturers for the following cases? Or if it can not be figured out, why? | |
|  | Manufacturer |
| Computer used: |  |
| Default gateway: |  |
| DHCP server: |  |
| alaplab.mit.bme.hu |  |

## Question on ARP addressing methods

|  |
| --- |
| What kind of addressing (broadcast, unicast) used by ARP in the requests and responses (and why)? |
|  |

## Background traffic \*

|  |
| --- |
| What kind of other (let’s call it background) traffic can be observed on the network? What are the sources of this traffic? |
|  |
| What kind of protocol is the mainly background traffic STP? |
| STP: |
| What kind of protocol is the mainly background traffic LLDP? |
| LLDP: |
| What kind of protocol is the mainly background traffic SSDP? |
| SSDP: |

## Capturing traffic with capture filter \*

|  |
| --- |
| By observing the properties of the annoying background traffic SSDP and STP create a suitable capture filter to filter them out! |
| Ping: an arbitrary network node – the screenshot of cmd window: |
| <<screenshot>> |
| Ping: the same arbitrary network node – screenshot of the captured traffic in Wireshark (without display filter): |
| <<screenshot>> |

# Task: the operation of DNS

**Aim of this task:** get familiar with the fundamentals of DNS operation by the help of the command line tool **nslookup**.

During this measurement we query an IP address (or IP addresses) belonging to a domain name or query the domain name belonging to an IP address. Before running **nslookup** it is required to delete the ARP and DNS caches (just like in the previous measurement). To make it easier create a suitable batch script (**mynslookup.bat**)!

## Creating the batch file (mynslookup.bat)

|  |
| --- |
| Copy the content of the script what clears the caches then invokes **nslookup**: |
|  |

## Query the IP address belonging to a domain name (1)

|  |
| --- |
| Query the IP address of alaplab.mit.bme.hu:  **mynslookup alaplab.mit.bme.hu** |
| <<screenshot of cmd window>> |
| Screenshot of the observed traffic in Wireshark: |
| <<screenshot>> |

## Query the IP address belonging to a domain name (2)

|  |
| --- |
| Query the IP address of alaplab.mit.bme.hu exactly as bellow (with the tailing dot!):  **mynslookup alaplab.mit.bme.hu.** |
| <<screenshot of cmd window>> |
| Screenshot of the observed traffic in Wireshark: |
| <<screenshot>> |

## Query the IP address belonging to a domain name (compare)

|  |
| --- |
| What is the difference between the two cases (with and without the tailing dot)? |
|  |

## Reverse lookup: query the domain name of an IP address

|  |
| --- |
| Figure out the domain name of the machine having **152.66.252.1** as its IP address (reverse lookup)! |
| <<screenshot of cmd window>> |
| Screenshot of the observed traffic in Wireshark: |
| <<screenshot>> |

## Question on the operation of DNS

|  |
| --- |
| Answer the following questions by investigating the network traffic by Wireshark! |
| a) How does the TCP/IP protocol stack use the DNS suffix search list: **kl.bme.hu**? |
|  |
| b) What lower level protocols are used by DNS and how? |
|  |
| c) How are the IP addresses represented by DNS incase of reverse lookup? What is the order of the IP address fields and why? |
|  |
| d) What does the Time-To-Live parameter mean in the response? (**Warning**: this question is about the TTL in the DNS reply and not the TTL of an IP packet!) |
|  |
| e) What happens if the IP address belonging to a domain name changes? Who reaches the domain name on the new IP address and who tries to communicate with it on the old address? |
|  |

## Multiple IP addresses in DNS reply \*

|  |
| --- |
| Query the IP address of **microsoft.com** (without a leading “www”) with **nslookup**! |
| <<screenshot of cmd window>> |
| <<screenshot of Wireshark window>> |
| Repeat the same query! |
| <<screenshot of cmd window>> |
| <<screenshot of Wireshark window>> |
| Repeat the same query again! |
| <<screenshot of cmd window>> |
| <<screenshot of Wireshark window>> |
| Compare the results! What can be observed? For what kind of reasons is this behavior useful? |
|  |

# Task: tracing the route of IP packets

**The aim of this measurement:** tracing the route of the IP packets by the help of the command line tool **tracert**.

**tracert** is a network utility what tries to determine the midway routers between the computer running the tool and the network node specified as its first parameter. (In a UNIX environment this tool is called **traceroute** or **tracepath**. In fact these tools inspired **tracert**). Like with the previous measurements clear the ARP and DNS caches before running **tracert** by the help of a suitable batch file!

## Creating the required batch file (mytracert.bat)

|  |
| --- |
| Copy the content of the batch script here: |
|  |

## Tracing the route to a domestic server

|  |
| --- |
| Trace the route of IP packets by **tracert** to the server **www.bme.hu**! |
| <<cmd screenshot>> |
| <<Wireshark screenshot>> |
| Investigate the captured network traffic: |
|  |

## Tracing the route to a server abroad (1)

|  |
| --- |
| Trace the route of IP packets by **tracert** to a server abroad, preferably overseas (great number of midway routers)! (For example: slashdot.org) |
| <<cmd screenshot>> |
| <<Wireshark screenshot>> |
| Investigate the captured network traffic: |
|  |

## Tracing the route to a server abroad (2)

|  |
| --- |
| Trace the route of IP packets by **tracert** to a server abroad, preferably overseas (great number of midway routers)! (For example: cnn.com) |
| <<cmd screenshot>> |
| <<Wireshark screenshot>> |
| Investigate the captured network traffic: |
|  |

## The operation of tracert

|  |
| --- |
| How does the command line tool **tracert** works? How does it explore the midway routers? What kind of lower level protocols does it use and how? |
|  |

## Tracing to an offline network node \*

|  |
| --- |
| What happens if the network node accessed by **tracert** is offline (or the firewall configured on it blocks the answers)? Produce this phenomenon! |
| <<cmd screenshot>> |
| <<Wireshark screenshot>> |
| **Explanation:** |

# Task: investigating TCP data flows by the help of netstat

**Aim of this task:** investigating TCP data flows by the help of command line tool **netstat** and Wireshark.

In this measurement we investigate the network traffic during downloading a web page by running netstat at predefined times. (Web browsers cache the downloaded web pages. The operation of this cache and HTTP will be observed in the next laboratory occasion. At this point we need to make the browser to download the whole content of the web page by deleting its cache.)

Measurement process:

* Run the command **netstat -a**! Save the output to a file (like: **netstat\_pre.txt**)!
* Start a web browser and empty its cache!
* Delete the ARP and DNS caches!
* Start capturing the network traffic in Wireshark!
* Download the web page **www.bme.hu**!
* Run the command **netstat -a** again! Store its output to a file (like: **netstat\_web.txt**)!
* Exit the browser!
* Run the command **netstat -a** once more! Save its output to a file (like: **netstat\_post.txt**)!
* Stop capturing network traffic in Wireshark!

## Phase “pre”

|  |
| --- |
| << content of netstat\_pre.txt>> |

## Phase “web”

|  |
| --- |
| << content of netstat\_web.txt>> |
| <<Wireshark screenshot>> |

## Phase “post”

|  |
| --- |
| << content of netstat\_post.txt>> |

## Identifying the protocols used during web page download

|  |
| --- |
| What kind of protocol does the browser use during the download process and for what reason? |
|  |

## Figuring out the number of TCP flows

|  |
| --- |
| By investigating the network traffic captured by Wireshark (using filters) and the output of **netstat** figure out the number of parallel TCP flows opened by the browser during the download of **www.bme.hu**! |
|  |

## Closing connections

|  |
| --- |
| When and how the TCP connections close?  Create a suitable display filter to find the closing of the connection and investigate also the output of **netstat**! |
|  |

## The properties of a TCP flow

|  |
| --- |
| Make a filter to show only IP packets belonging to a shorter (~10-15 messages) TCP flow (right click on a packet / Conversation Filter / TCP). |
| <<Wireshark screenshot>> |
| Are the packets in order? How can someone figure it out? |
|  |
| Investigate in the TCP flow the procedure of message sending (Statistics / Flow Graph with TCP flow selected)! What are those *Sequence number* and *Acknowledgment number* fields? |
| <<Wireshark screenshot>> |
| Explanation: |

## Bandwidth requirements of network protocols

|  |
| --- |
| Make diagrams about bandwidth (byte/s) and packet rate (packet/s) of some often used protocols! Explain the diagrams! |
| <<Wireshark screenshot>> |
| Explanation: |
| <<Wireshark screenshot>> |
| Explanation: |

## Identifying local network services \*

|  |
| --- |
| What TCP and UDP ports are open on the local machine (according to the output of **netstat**) and what are the services offered at those ports? |
|  |

1. More on routing on Windows platforms: TCP/IP Fundamentals for Microsoft Windows, Chapter 5 IP Routing, **http://technet.microsoft.com/en-us/library/bb727001.aspx#EEAA** [↑](#footnote-ref-1)