Data Communication

Lars Rikard Rådstoga | 223786

2022-01-18

# Introduction

This document contains answers to the assignment “Data Communication” given as a compulsory assignment in the course “Industrial Information Technology” at the University of South-Eastern Norway – spring 2022.

Screenshot in Figure 1‑1: Figure 1‑1 shows a seemingly deterministic program which assigned the task “Describe communication directions”. The program scanned my computer for information and took my student information as input to generate the task. Repeated tests of the program with the same inputs resulted in the same task. Thus, the program is inductively postulated to be deterministic.

Graphical user interface, text, application, email

Description automatically generated

Figure 1‑1: Student and computer information

# System information

This chapter contains answers to tasks in chapter 1.3 from the assignment document.

## Interface protocols between motherboard and storage devices

My computer has two disk drives as seen in Figure 2‑1. They are sized roughly 1TB and 2TB as seen in Figure 2‑2 and Figure 2‑3. The disks are of types: SSD and SSHD (some sort of hybrid drive) respectively. The SSD is connected using the NVMe protocol as seen in Figure 2‑4 and the SSHD uses the SATA protocol seen in Figure 2‑5.

Text

Description automatically generated

Figure 2‑1 Disk drives on my computer.

Graphical user interface

Description automatically generated

Figure 2‑2 Details of my M.2 SSD.

Graphical user interface

Description automatically generated

Figure 2‑3 Details of my HDD.

Text

Description automatically generated

Figure 2‑4 Storage controllers

Text

Description automatically generated

Figure 2‑5 Standard interface for SATA communication

## Serial ports

Figure 2‑6 should reflect the amount of USB ports on the computer, but the list contains 11 ports, but the computer case only has 9 ports connected to the motherboard. Perhaps this means it could support more ports if the case supported it.

Text

Description automatically generated

Figure 2‑6 An array of USB drivers.

## Bluetooth devices

Figure 2‑7 shows the installed Bluetooth profiles [1] . Interestingly the connected audio devices have separate profiles for the audio streaming and the remote-control features of the device. E.g., the Momentum 3 headset uses the AVCRP profile to pause, skip and adjust volume, etc.

Text

Description automatically generated

Figure 2‑7 Bluetooth drivers.

# Network information

This chapter contains answers to tasks in chapter 1.4 from the assignment document.

## Describe communication directions

In a physical cable, communication can only occur in one direction at a time. A setup with communication set to always be in the same direction is called simplex. A setup which requires two-way communication will either require half-duplex or duplex. Duplex communication requires two cables or channels. Half-duplex allows for two-way communication with only one cable, using some sort of Primary/Replica protocol to avoid signals from interfering with each other.

## Number of network devices

Figure 3‑1 shows network drivers of the computer. These do not reflect the number of network devices.

Text

Description automatically generated with medium confidence

Figure 3‑1 Network drivers.

## Physical address

Figure 3‑2 shows that the physical address (MAC) of the network adapter is D4-5D-64-48-94-34.

Text

Description automatically generated

Figure 3‑2 Network details

## Logical address

Figure 3‑2 shows that the logical address of the computer on the network is 10.0.0.10. The logical address of the network is 10.0.0.0 [2].

## Why TCP/IP

It is likely to be a TCP/IP address because it is characteristic of a class A network class. 10.0.0.0 is the network address and 0.0.0.10 is the host address. The network class is of type A because the first octet of the network address is between 0 and 127 [2]. This network class uses the default subnet mask of 255.0.0.0. It is a dynamic address because the DHCP server is turned in the router settings shown in Figure 3‑3.

Graphical user interface, application

Description automatically generated

Figure 3‑3 Router settings

## Other network protocols installed

Figure 3‑4 and Figure 3‑5 show other installed network protocols.

Graphical user interface, text, application

Description automatically generated

Figure 3‑4 Installed network protocols (1/2)

Graphical user interface, text, application

Description automatically generated

Figure 3‑5 Installed network protocols (2/2)

## IPv6 support?

Yes, seemingly IPv6 is supported, but not activated/used. see Figure 3‑5.

## Wireless

This chapter contains answers to tasks in chapter 1.4.1 from the assignment document.

### Which network will you use?

The network that is first from the top is the one I would connect to as it has the best signal strength.

### Does it require a password?

It does not require a password as it is unencrypted.

### Why do we not see any other networks?

Wireless networks do not have infinite practical reach, this can be one reason we don’t see other networks. But it is also possible host networks using hidden SSIDs, these won’t appear in the network list either.

### Assumptions about network owners

I assume that the computer that is scanning for the networks are in the same building as the “CM” network and the AB networks belong to a neighboring office or similar. I assume that the encryptions are weak (WEP) because I would honestly expect WPA2 in the windows XP era, but one of the networks explicitly states it uses WPA so I assume WPA2 would also show up in the interface if it was the case.

### My available networks

Networks available from my location shown in Figure 3‑6.

Graphical user interface

Description automatically generated

Figure 3‑6 My network list.

# Network testing

This chapter contains answers to tasks in chapter 1.5 from the assignment document.

## Pinging usn.no

We can see in Figure 4‑1 that the average ping response time to usn.no is 22ms.

Text

Description automatically generated

Figure 4‑1 A single ping with cmd.

## Continuous pinging

Continuous pinging seen in Figure 4‑2.

Text

Description automatically generated with medium confidence

Figure 4‑2 Continuous pinging with cmd.

The TCP/IP address of the remote node is 80.239.119.247 as seen in

Graphical user interface, application, table, Excel

Description automatically generated

Figure 4‑3 Wireshark: filtered to see the ping messages.

## Testing the ping application

Test shows that the ping from my pc to twitch.tv is 8ms as seen in Figure 4‑4. See Appendix A: Testing ping to a connected node for the source code.

Text

Description automatically generated

Figure 4‑4 Ping response for twitch.tv.

It is clearly possible to sniff the message sent with the ping as shown in Figure 4‑5.

Graphical user interface, application

Description automatically generated

Figure 4‑5 Sniffing the message in the ping with WireShark.

## Pseudo code continuous ping application

A ping application that takes 5 recent ping values and averages them:

Recent pings = Array of size 5

int i = 0;

while (true){

Recent pings[i%recent\_pings.length] = ping(website);

Print(average(recent pings));

i++;

sleep(a bit);

}

## Pseudo code of node checking ping application

A program that iterates local network nodes, stores and prints their response times:

Nodes = array of size 256

for(i = 0; i < nodes.length, i++){

pingObject = ping(10.0.0.i)

nodes[i] = pingObject

print(pingObject)

}

## Implementation of the node checker ping application

A program was created to ping all the local nodes on the network. The source code can be found in Appendix B: testing ping to all local nodes. A screenshot of the console outputs can be seen in Figure 4‑6. Though it does not seem to be able to get a response from all nodes. Figure 4‑7 shows devices visible for the router and Figure 4‑8 shows the ARP table of the network.

Text

Description automatically generated

Figure 4‑6 Outputs of the node checker application

Table

Description automatically generated

Figure 4‑7 All devices connected to the Router

A picture containing text

Description automatically generated

Figure 4‑8 Outputs of the "arp -a" cmd command

# Conclusion

We can gather information about our computers and our networks, but features in Windows change such as the device manager, it being because of a lack of maintenance or change of features. It seems information that the assignment asks for is not presented there in the way it might once was. We also try to ping all nodes on the local network, but not all are visible to the computer, which was perhaps not as expected.

# References

|  |  |
| --- | --- |
| [1] | "List of Bluetooth profiles," [Online]. Available: https://en.wikipedia.org/wiki/List\_of\_Bluetooth\_profiles#Audio/Video\_Remote\_Control\_Profile\_(AVRCP). [Accessed 29 01 2022]. |
| [2] | "Master/slave (technology)," 23 January 2022. [Online]. Available: https://en.wikipedia.org/wiki/Master/slave\_(technology). [Accessed 28 January 2022]. |
| [3] | "Understand TCP/IP addressing and subnetting basics," Microsoft, 23 09 2021. [Online]. Available: https://docs.microsoft.com/en-us/troubleshoot/windows-client/networking/tcpip-addressing-and-subnetting. [Accessed 28 01 2022]. |

# Appendices

Appendix A: Testing ping to a connected node

Appendix B: testing ping to all local nodes

# Appendix A: Testing ping to a connected node

*///////////////////////////////////////////////////////////////////////////////////////////*

*///*

*/// PingAppConsole: application testing the ping() connection to a node*

*///*

*/// Based on code form Microsoft MSDN about the ping() command*

*///*

*/// Version: 1.1: 28-JAN-22: LRR*

*///*

*///////////////////////////////////////////////////////////////////////////////////////////*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using System.Net;

using System.Net.NetworkInformation;

*//*

namespace PingAppConsole

{

    class Program

    {

*/// <summary>*

*/// //////////////////////////////////////////////////////////////////////////////*

*static* void Main(string[] args)

*///*

*/// Purpose: the main function in the application handling the ping()communication*

*///*

*/// Version: 1.1: 28-JAN-22: LRR*

*/// </summary>*

        {

            string host, data;

            byte[] buffer;

            int timeout;

            Ping pingSender = new Ping();

            PingOptions options = new PingOptions();

*// Use the default Ttl value which is 128,*

*// but change the fragmentation behavior.*

            options.DontFragment = true;

*// Create a buffer of 32 bytes of data to be transmitted.*

            data = "UniversityCollegeofSouthEastNorw";

            buffer = Encoding.ASCII.GetBytes(data);

            timeout = 120;

*// Name or address of node to access*

            host = "www.twitch.tv";

            PingReply reply = pingSender.Send(host, timeout, buffer, options);

            if (reply.Status == IPStatus.Success)

            {

                Console.WriteLine(" Ping communication status for {0}:", host);

                Console.WriteLine(" ------------------------------------------");

                Console.WriteLine(" Address: {0}", reply.Address.ToString());

                Console.WriteLine(" RoundTrip time (mSec): {0}", reply.RoundtripTime);

                Console.WriteLine(" Time to live: {0}", reply.Options.Ttl);

                Console.WriteLine(" Don't fragment: {0}", reply.Options.DontFragment);

                Console.WriteLine(" Buffer size: {0}", reply.Buffer.Length);

                Console.WriteLine(" ------------------------------------------");

            }

            else

            {

                Console.WriteLine(" Error connecting to network address/name {0}", host);

            }

            Console.WriteLine(" Press CR or Enter to Quit the application");

            Console.ReadLine();

        }

    }

}

*///*

*////////////////////////////////////////////////////////////////////////////////////////*

# Appendix B: testing ping to all local nodes

*///////////////////////////////////////////////////////////////////////////////////////////*

*///*

*/// PingNodes: application testing the ping() connection with all nodes*

*///*

*/// Based on code form Microsoft MSDN about the ping() command*

*///*

*/// Version: 1.1: 28-JAN-22: LRR*

*///*

*///////////////////////////////////////////////////////////////////////////////////////////*

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Threading;

using System.Net;

using System.Net.NetworkInformation;

namespace PingAppConsole

{

    class Program

    {

*/// <summary>*

*/// //////////////////////////////////////////////////////////////////////////////*

*static* void Main(string[] args)

*///*

*/// Purpose: the main function in the application handling the ping() communication*

*///*

*/// Version: 1.1: 28-JAN-22: LRR*

*/// </summary>*

        {

            string host, data;

            byte[] buffer;

            int timeout;

            Ping pingSender = new Ping();

            PingOptions options = new PingOptions();

*// Use the default Ttl value which is 128,*

*// but change the fragmentation behavior.*

            options.DontFragment = true;

*// Create a buffer of 32 bytes of data to be transmitted.*

            data = "NotASecretMessage";

            buffer = Encoding.ASCII.GetBytes(data);

            timeout = 120;

            var replies = new List<PingReply> { };

            for (int i = 0; i < 256; i++)

            {

*// Name or address of node to access*

                host = $"10.0.0.{i}";

                PingReply reply = pingSender.Send(host, timeout, buffer, options);

                if (reply.Status == IPStatus.Success)

                {

                    Console.WriteLine(" Ping communication status for {0}:", host);

                    Console.WriteLine(" ------------------------------------------");

                    Console.WriteLine(" Address: {0}", reply.Address.ToString());

                    Console.WriteLine(" RoundTrip time (mSec): {0}", reply.RoundtripTime);

                    Console.WriteLine(" Time to live: {0}", reply.Options.Ttl);

                    Console.WriteLine(" Don't fragment: {0}", reply.Options.DontFragment);

                    Console.WriteLine(" Buffer size: {0}", reply.Buffer.Length);

                    Console.WriteLine(" ------------------------------------------");

                }

            }

            Console.WriteLine(" Press Enter to Quit the application");

            Console.ReadLine();

        }

    }

}

*///*

*////////////////////////////////////////////////////////////////////////////////////////*