

Assignment 2 in IIA2017

Torstein Solheim Ølberg | 263054

February 4, 2024

Contents

1	Introduction	3
2	Results	3
2.1	System Information	3
2.2	Communication Directions	4
2.3	Network Protocols	5
2.3.1	Wireless	8
2.4	Network Test	9
2.5	Digitalization	12
2.6	Cloud Systems	14
3	Conclusion	14
3.1	This Computer	14
3.2	Network testing	14
3.3	DAQ Application	14
4	Appendix A	16
5	Appendix B	16

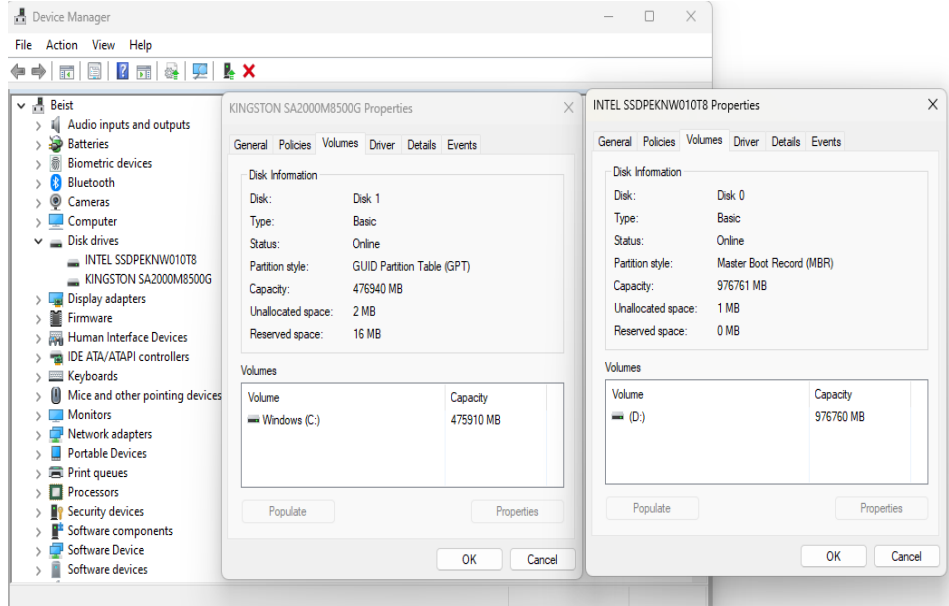


Figure 1: Screenshot containing info on the two SSD drives of this computer. The C: drive has about 500 GB of storage space and the D: drive double that amount.

1 Introduction

The industrial automatisaton profession is often times occupied with installing and configuring different devices which communicate with each other over a network. Examples of such can be single robots controlled by a computer or whole production lines controlled by set of computers and servers. By studying a computer, its network and communication capabilities, it is possible to learn a lot about how this is done and how best to do it. This report will study a Komplette Khameleon laptop bought in 2020 in Norway.

2 Results

2.1 System Information

From figure 1 we can see that this computer has two storage drives. The first one, C:, has a capacity of approximately 475 GB. From NotebookChecks benchmarking of drive, we know that the interface it uses is NVMe, which is a very fast protocol [1]. As for the D: drive, we see that it has a capacity of

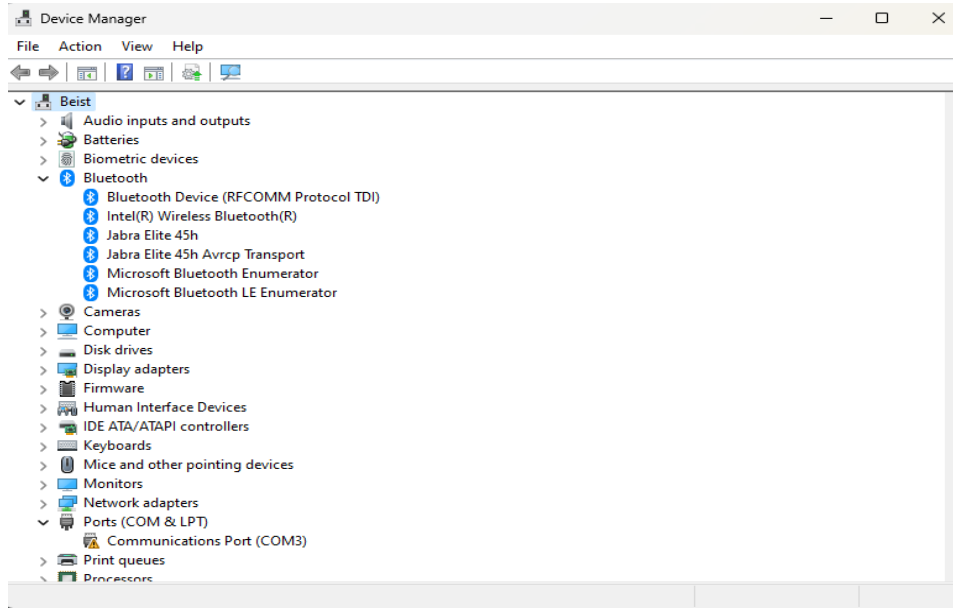


Figure 2: Screenshot containing info on the available Bluetooth devices for this computer. You can also see there is a fault with the Serial ports list.

approximately 977 GB and from Mouser Electronics, we know that it uses a PCIe interface [2]. This also is a very fast interface, but is most often beaten by NVMe if NVMe is combined with PCIe.

The number of serial ports is not possible to see on this computer as there is a known error when trying to display this after updating to Windows 10. However, if I normally where to do this, I would have checked the Ports (COM & LPT) list, but as you can see in figure 2 there is something wrong with this list.

The Bluetooth communications devices are listed in figure 2

2.2 Communication Directions

There are three types of protocols for communication directions. First, there is simplex, where communication only goes in one direction. For this, you only need one communication channel. Then there is half-duplex, where communication can go both ways, but only one at a time. This protocol also only needs one channel. Finally, there is duplex, where communication can go both ways at the same time. This protocol needs two channels.

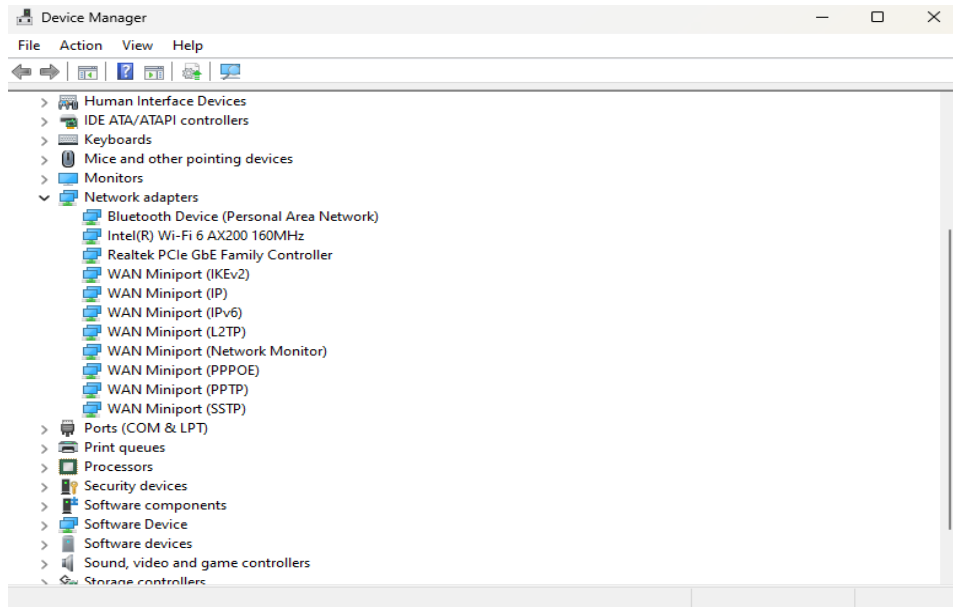


Figure 3: Screenshot containing info on the number and type of each network device on this computer.

2.3 Network Protocols

The number of network devices can be seen in 3. Here we can see there are eleven network devices.

The physical address of my active network is DC-FB-48-01-23-BD as seen in figure 4

The logical address is 192.168.0.10, also seen in figure 4

As the logical address starts with 192, we know that it is a class C IPv4 network with a subnet mask of 255.255.255.0 [3]. Since this address is assigned by a router and no specific adjustment has been done by the user, it is very likely to be a dynamically assigned address.

From figure 5, where netstat -s has been run to list stats of available network protocols, it is possible to see that the computer has both TCP and UDP installed. It is also possible to see that the computer supports both TCP IPv4 and IPv6.

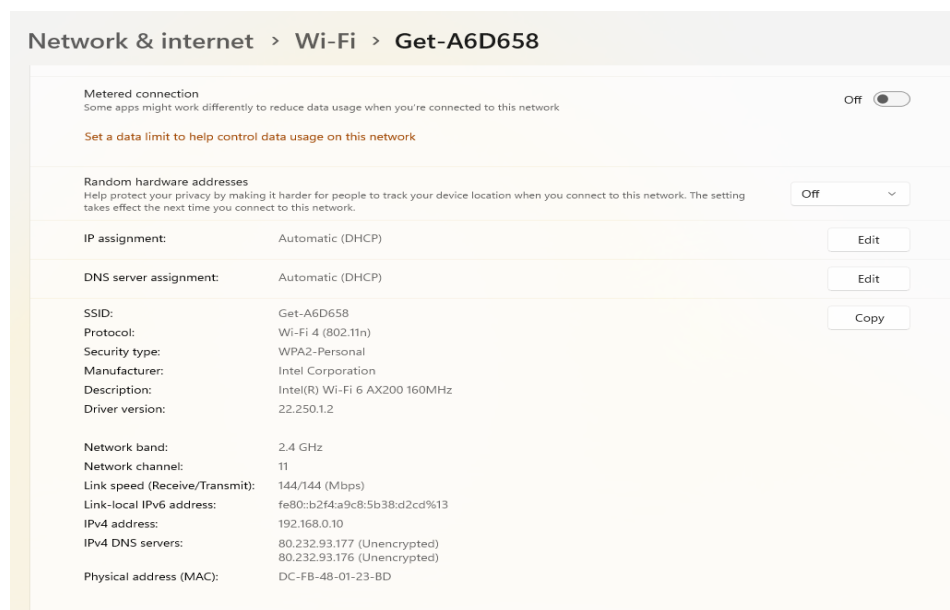


Figure 4: Screenshot containing info on the current active network of this computer. This is a wireless network named Get-A6D658, with a physical and logical address displayed.

```
TCP Statistics for IPv4

Active Opens           = 1591
Passive Opens          = 417
Failed Connection Attempts = 39
Reset Connections      = 201
Current Connections    = 97
Segments Received      = 146599
Segments Sent          = 170685
Segments Retransmitted = 624

TCP Statistics for IPv6

Active Opens           = 33
Passive Opens          = 4
Failed Connection Attempts = 29
Reset Connections      = 0
Current Connections    = 8
Segments Received      = 182
Segments Sent          = 130
Segments Retransmitted = 52

UDP Statistics for IPv4

Datagrams Received      = 147946
No Ports                = 1608
Receive Errors          = 0
Datagrams Sent          = 25524

UDP Statistics for IPv6

Datagrams Received      = 669
No Ports                = 0
Receive Errors          = 0
Datagrams Sent          = 288
```

Figure 5: Screenshot containing info on which protocols have been installed on this computer. We see that TCP IPv4 and IPv6 as well as UDP IPv4 and IPv6. has been installed.

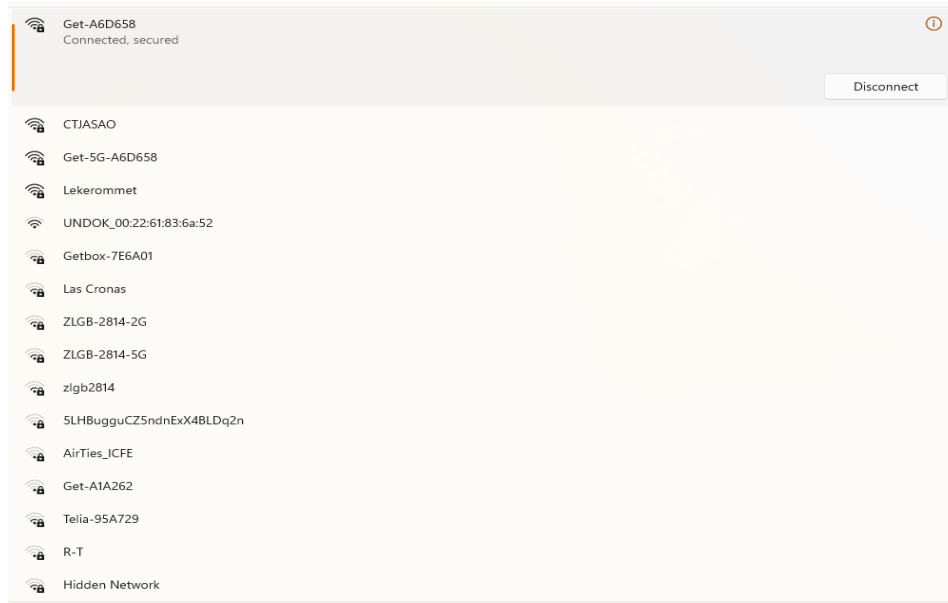


Figure 6: Screenshot containing info on the available networks in this computers location.

2.3.1 Wireless

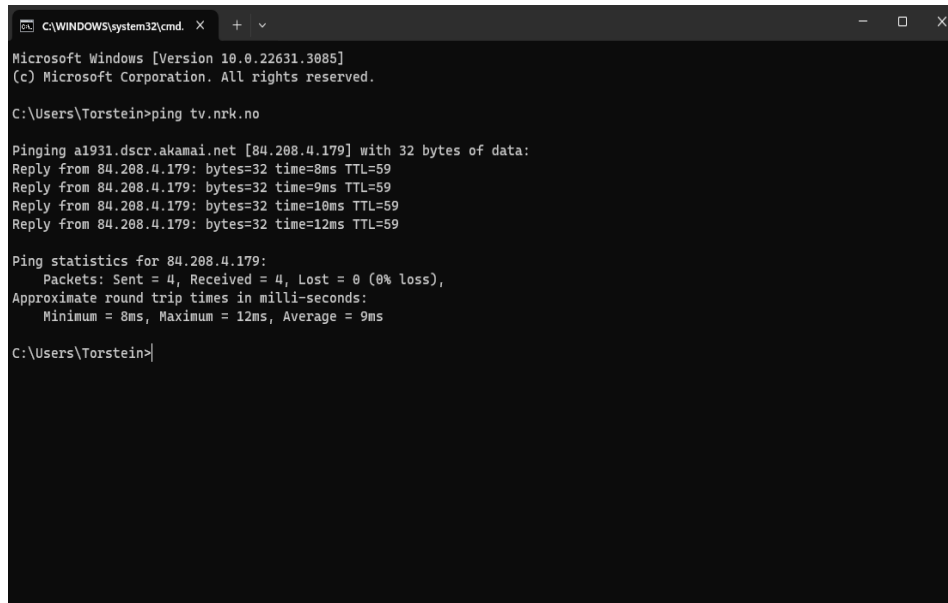
If I had to choose between the three networks listed in figure 2.1 in [4]. I would choose the top one as it has a greater signal strength.

The authentication to this network is unknown, so likely it needs some sort of password to access.

The reason we do not see any other networks than the LAN is because we connect to a WAN via the LAN.

The first network is a direct connection to another device. It is likely privately owned, possibly by someone named Thamm, and has no security. It also seems to be quite far away, or disrupted by a lot of walls, as it has a very faint signal strength. The other networks all have some sort of access security and are general networks. CM is the closest one, with strong signal, The other four are all quite faint, and likely controlled by the same owner which likes science fiction.

As for the networks available at the location of this computer, they can be seen in figure 6.



```
C:\WINDOWS\system32\cmd. X + -
Microsoft Windows [Version 10.0.22631.3085]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Torstein>ping tv.nrk.no

Pinging al931.dscr.akamai.net [84.208.4.179] with 32 bytes of data:
Reply from 84.208.4.179: bytes=32 time=8ms TTL=59
Reply from 84.208.4.179: bytes=32 time=9ms TTL=59
Reply from 84.208.4.179: bytes=32 time=10ms TTL=59
Reply from 84.208.4.179: bytes=32 time=12ms TTL=59

Ping statistics for 84.208.4.179:
    Packets: Sent = 4, Received = 4, Lost = 0 (0% loss),
    Approximate round trip times in milli-seconds:
        Minimum = 8ms, Maximum = 12ms, Average = 9ms

C:\Users\Torstein>
```

Figure 7: Screenshot of a ping of tv.nrk.no. The ping was done four times, all of them successful, and generally quite quick.

2.4 Network Test

Starting with pinging `tv.nrk.no`. In figure 7 we can see an attempt at pinging this website four times and all of them where successful. The ping was generally quick, with a minimum time of 8ms and a max time of 12ms. If we try to ping it continuously, and trace the messages with WireShark, we can see from figure 8 that IPv4 is used and that the logical address of the website is 84.208.4.179. A ping application can easily be made in C# to help perform pings and to control what data is sent. One such application has been created by Nils-Olav Skeie and can be seen in the appendix [4]. Changing the data and host of the original application to tv.nrk.no and the string CanIWatchPoirot? we can run the application and see that we get a response. A run of this application can be seen in figure 9 where we see that the website has been pinged, it has the same address as earlier, and managed a round trip in 6ms. A known problem with the communication performed in circumstances as we have just done are that the protection of the information transmitted with TCP/IP is not that good. Another application can easily gain access to the info transmitted, as seen in figure 10. The data packages are sent in plain text and anyone whom intercepts the data can read it.

282	51.460990	192.168.8.10	84.208.4.179	ICMP	74 Echo (ping) request	id=0x0000, seq=62/15616, ttl=128 (reply in 283)	
283	51.467190	84.208.4.179	192.168.8.10	ICMP	74 Echo (ping) reply	id=0x0000, seq=62/15616, ttl=59 (request in 282)	
284	51.464853	192.168.8.10	84.208.4.179	ICMP	74 Echo (ping) request	id=0x0000, seq=62/15672, ttl=128 (reply in 285)	
285	52.473511	84.208.4.179	192.168.8.10	ICMP	74 Echo (ping) reply	id=0x0000, seq=62/15672, ttl=59 (request in 284)	
286	53.489576	192.168.8.10	84.208.4.179	ICMP	74 Echo (ping) request	id=0x0000, seq=63/16128, ttl=128 (reply in 287)	
287	53.477587	84.208.4.179	192.168.8.10	ICMP	74 Echo (ping) reply	id=0x0000, seq=63/16128, ttl=59 (request in 286)	
288	54.475868	192.168.8.10	84.208.4.179	ICMP	74 Echo (ping) request	id=0x0000, seq=64/16384, ttl=128 (reply in 289)	
289	54.485224	84.208.4.179	192.168.8.10	ICMP	74 Echo (ping) reply	id=0x0000, seq=64/16384, ttl=59 (request in 288)	

> Frame 282: 74 bytes on wire (592 bits), 74 bytes captured (592 bits) on interface \Device\NPF_{5E75C4D4-F657-419C-9950-AA0315C3485},	0000	a0 39 ee a6 86 5c dc f9 40 01 23 bd 00 00 45 00	9 ... \. H 8 . E
> Ethernet II, Src: Intel_RL123:bd (dc:f8:40:81:23:bd), Dst: SagecomProa_a6:86:5c (a6:39:ee:a6:a6:5c)	0010	00 3c 0e c9 00 00 00 01 00 00 c9 00 00 0a 34 40	< T
> Internet Protocol Version 4, Src: 192.168.8.10, Dst: 84.208.4.179	0020	04 03 00 00 64 1a 00 01 00 3d 62 62 64 65 66	...M...-abcdef
> Internet Control Message Protocol	0030	67 68 69 6a 6b 6c 6d 6e 6f 70 71 72 73 74 75 76	ghi jklm opqrstuv
	0040	77 61 62 63 64 65 66 67 68 69	wxyzdefg hi

Figure 8: Screenshot of WireShark displaying info on a ping request of the website tv.nrk.no. We can see the logical address of both the requester and the website

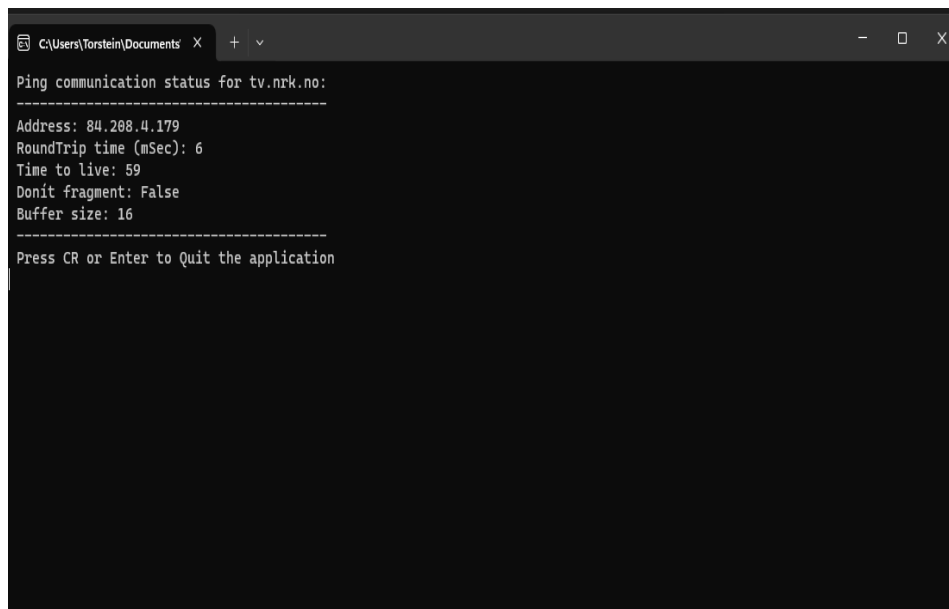


Figure 9: Screenshot of a run of the PingApp application. The name, address and trip time has been displayed.

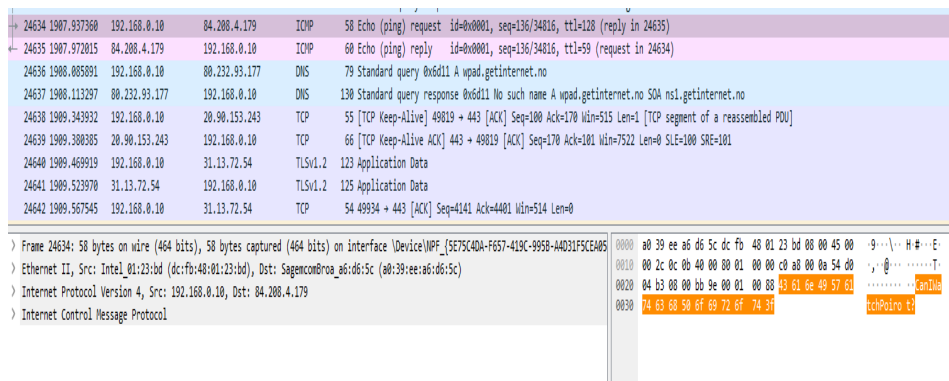


Figure 10: Screenshot of run of the application PingApp traced in Wire-Shark. We can see the data sendt in plaintext.

The application could be extended by adding a way to ping multiple times in a row and perform an averaging of the results. This could be done by adding something like this code to the program.

```
def average_pinger(int nr_pings):
{
    string data = "message"
    string host = "website.com"
    float total_time = 0
    for i from 0 to nr_pings:
    {
        reply = Ping(host, Encrypt(data))
        total_time += reply.RoundtripTime
    }
    WriteToConsole("The average time for a ping was " + total_time /
        nr_pings + "seconds")
}
```

A tool for pinging all the nodes in a network segment is also usefull. This could be implemented by code like the on showed underneath.

```
def ping_segment(string network_prefix):
{
    string data = "message"
    for i from 0 to 255:
    {
        string host = network_prefix + i
        reply = Ping(host, Encrypt(data))
        WriteToConsole("Pinged " + host + ": Status = " + reply.status)
    }
}
```

This code loops through all possible addresses for nodes on the network segment and pings all. Then it displayes if the ping was answered or not. An implementation of this code in the PingAppcan be seen in the appendix.

2.5 Digitalization

When choosing to digitilaze a system, it could be better to choose a plain text format rather than a database. An application taking sensor values an logging them to a text file would be useful. One such application was created by Torstein Solheim Ølberg [5]. A version of this application, using different samplingtime and semicolon as separator instead of comma, can be viewed

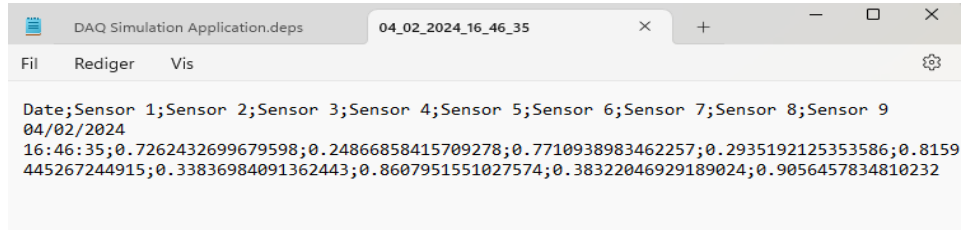


Figure 11: Screenshot of data stored by the DAQ simulation application.

in the appendix.

A flowchart, documenting the buildup of the application can be seen in figure 12.

This application could, be further developed by adding a low-pass filter between the sensors and the logging, such that any noise or problematic data points could be removed. This would be implemented by allowing the sensors to return values outside the expected range and checking the values before saving them to the sample list. Another possible addition would be to add support for other types of files to save to. The csv file is a text file where data is recorded in plain text, separated by commas, or in our case semicolons as seen in figure 11. This makes it a simple file to record to, but it is not very flexible. Another type of data storage file is XML. In this file format, data is stored within tags, making for a much more customizable way of categorising the data stored. This format also allows for data to be stored in a way that is readable for people, but since there are tags, it is much more easy for computers to extract the info in a useful way. A third option is the json file. Json is also a format readable by humans, but instead of using tags like XML, it uses objects. Data stored in objects are easy to group together, but adding information to it can be more difficult. It is the grouping ability of Json which makes it a very good pick for this applications choice. All of these file formats are however not optimal when it comes to security. This is because all of them store data in a human readable format. For the use of simulating any random sensor, it is not important to protect the data as the risk is not very high for the data to be misused. However, if the data stored is important it would be smart to use a file format where the data is encrypted. If the fileformat encrypted in a standard way, a real persistent threat could however decrypt it using a program, so if the data is really important, decryption using a key only authorised people know would be a smart choice.

2.6 Cloud Systems

IAAS or Infrastructure As A Service is a cloud service model where servers, networks and infrastructure is offered on-demand to customers, where they can create their own applications without needing to worry about physical space or maintenance crew. SAAS, or Software As A Service however is cloudbased access to fully operational software for applications.

The DAQ Simulation Application can be extended to also work with a cloud based system. As to not force the sensors to be physically plugged into the computer running such an application, it is possible to transfer the data to the application through the use of an API, an Application Programming Interface. If this cloud system sends the data to the application, it would also be useful to store the data, once processed by the application, into a cloud stored file. This can be done in multiple ways, for example by using a different application which continuously backs up files stored on the computer, or by using the DAQ Application to save the file to an online server. Here it would be simple to use SAAS like Google Drive Dropbox, which allows for directly saving files to their online servers.

3 Conclusion

3.1 This Computer

From this analysis, this computer has a wide set of very quick communication methods, both internally and over internet. There are multiple network protocols installed, it is possible to display this information as well as use it to communicate.

3.2 Network testing

The active network this computer is on is a LAN, connected to the internet via TCP IPv4, where a ping attempt to a website uses a short roundabout time. It is also possible to create a simple application which can be used to ping different nodes. This does however display a security risk in that information sent via the protocol used is sent in plain text and can thus be intercepted and read by anyone.

3.3 DAQ Application

A simple Data acquisition application can be created and used to acquire data from different sensors and store them to a csv file or other format. This

application could also be extended to include the possibility of reading from and writing to a cloud based service.

Sections	Weighting (%)	Your evaluation (%)	Review (%)	Comments
System Information	2.5	15		Finding out the problem with Ports took a lot of work.
Network Information	20	35		
Network Testing	35	25		
Digitalization	30	10		I reused a lot of code from a different project.
Cloud Systems	10	12.5		
Conclusion	2.5	2.5		
Sum	100			

References

- [1] K. Hinum. “Kingston sa2000m8500g.” (2023), [Online]. Available: <https://www.notebookcheck.net/Kingston-SA2000M8500G-SSD-Benchmarks.588591.0.html>. (accessed: 02.02.2024).
- [2] M. Electronics. “Intel ssdpeknw010t8.” (2023), [Online]. Available: <https://no.mouser.com/ProductDetail/Intel/SSDPEKNW010T8?qs=Cb2nCFKsA8rCEtIiJugDg%3D%3D>. (accessed: 02.02.2024).
- [3] M. Outposts. “The five ipv4 classes - quick reference.” (2024), [Online]. Available: <https://www.meridianoutpost.com/resources/articles/IP-classes.php>. (accessed: 04.02.2024).
- [4] N.-O. Skeie. “Data communication and digitalization assignment.” (2023), [Online]. Available: <https://inst-fs-dub-prod.inscloudgate.net>. (accessed: 01.02.2024).

- [5] T. S. Ølberg. “Iia1319/assignment 1.” (2024), [Online]. Available: <https://github.com/MrTorstein/IIA1319/tree/main/Assignment%201>. (accessed: 04.02.2024).

4 Appendix A

Here you can find the documentation for the program.

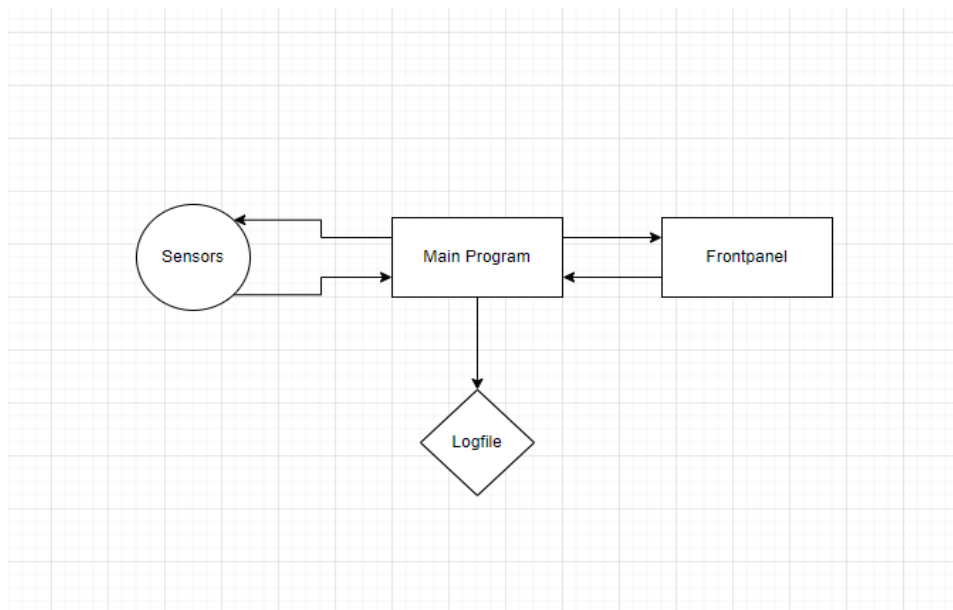


Figure 12: Flowchart documentation of the DAQ simulation application.

5 Appendix B

Here you can find the code files not made by Visual Studio itself.

```

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
///
/// PingAppConSile: application testing the ping() connection to a
/// node
///
/// Based on code form Microsoft MSDN about the ping() command
///
/// Version: 1.0: 8-JAN-17: NOS
  
```



```

///
////////////////////////////////////
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;
using Microsoft.VisualBasic;
//
namespace PingAppConsole
{
    class Program
    {
        /// <summary>
        ///
        ///
        static void Main(string[] args)
        /// Purpose: the main function in the application handling
        the
        /// ping()communication
        /// Version: 1.0: 8-JAN-17: NOS
        /// </summary>
        {
            string host, data;
            byte[] buffer;
            int timeout;
            Ping pingSender = new();
            PingOptions options = new()
            {
                // Use the default Ttl value which is 128,
                // but change the fragmentation behavior.
                DontFragment = true
            };

            // Create a buffer of 32 bytes of data to be transmitted.
            data = "CanIWatchPoirot?";
            buffer = Encoding.ASCII.GetBytes(data);
            timeout = 120;
            // Name or address of node to access
            host = "tv.nrk.no";
            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(" Dont 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);
}

PingSegment("192.168.0.");

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

```

```

// Description: Function used to ping all nodes in a network
                segment
// Author: Torstein Solheim Olberg
// Date: 04/02-2024
public static void PingSegment(string network_prefix)
{
    Ping pingSender = new();
    PingOptions options = new() {DontFragment = true};
    string data = "message";

    for (int i = 0; i < 255; i++)

```

```

        {
            string host = network_prefix + i.ToString();
            PingReply reply = pingSender.Send(host, 120,
                Encoding.ASCII.GetBytes(data), options);
            Console.WriteLine("Pinged " + host + ": Status = " +
                reply.Status);
        }
    }
}

```

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SensorApp
{
    // Purpose: Simulate a sensor which can give values between 0
    //           and 1
    // Version: 04/02-24
    // Author: Torstein Solheim Olberg
    public class Sensor(int sensor_number)
    {
        readonly Random random_sensor_value = new(sensor_number);

        // Purpose: Return a new value between 0 and 1.
        // Version: 04/02-24
        // Author: Torstein Solheim Olberg
        public double GetNewValue()
        {
            return random_sensor_value.NextDouble();
        }
    }

    // Purpose: Dataclass for storing sensor data of 9 sensors
    // Version: 04/02-24
    // Author: Torstein Solheim Olberg
    public class Sensordata : IEnumerable<double>
    {
        public string? Date { get; set; }
    }
}

```

```

public double _sensor_0 { get; set; }
    public double _sensor_1 { get; set; }
    public double _sensor_2 { get; set; }
    public double _sensor_3 { get; set; }
    public double _sensor_4 { get; set; }
    public double _sensor_5 { get; set; }
    public double _sensor_6 { get; set; }
    public double _sensor_7 { get; set; }
    public double _sensor_8 { get; set; }

    // Purpose: returning sensors values in order
    // Version: 04/02-24
    // Author: Torstein Solheim Olberg
    public IEnumerator<double> GetEnumerator()
    {
        yield return _sensor_0;
        yield return _sensor_1;
        yield return _sensor_2;
        yield return _sensor_3;
        yield return _sensor_4;
        yield return _sensor_5;
        yield return _sensor_6;
        yield return _sensor_7;
        yield return _sensor_8;
    }

    // Purpose: Making the dataclass iterable
    // Version: 30/01-24
    // Author: Torstein Solheim Olberg
    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }

    // Purpose: Making the data class indexable
    // Version: 04/02-24
    // Author: Torstein Solheim Olberg
    public double this[int index]
    {
        get
        {
            return index switch
            {
                0 => _sensor_0,
                1 => _sensor_1,
            }
        }
    }

```

```

        2 => _sensor_2,
        3 => _sensor_3,
        4 => _sensor_4,
        5 => _sensor_5,
        6 => _sensor_6,
        7 => _sensor_7,
        8 => _sensor_8,
        _ => 0,
    };
}
set
{
    switch (index)
    {
        case 0:
            _sensor_0 = value;
            break;
        case 1:
            _sensor_1 = value;
            break;
        case 2:
            _sensor_2 = value;
            break;
        case 3:
            _sensor_3 = value;
            break;
        case 4:
            _sensor_4 = value;
            break;
        case 5:
            _sensor_5 = value;
            break;
        case 6:
            _sensor_6 = value;
            break;
        case 7:
            _sensor_7 = value;
            break;
        case 8:
            _sensor_8 = value;
            break;
        default:
            break;
    }
}

```

```
}  
  }  
}
```
