Technical Assignment for ELEC0032: Networking Systems 23/24

Author

Candidate Number: JGLR6

Taught by

Dr Anna Maria Mandalari
Gianluca Anselmi

An assignment submitted in fulfillment of requirements for the degree of
Bachelor of Arts and Science: Cultures & Engineering

Teaching department: Electronic and Electrical Engineering University College London
2023

Assignment:

You are being contracted as a consultant to advise a large technology company on investing in new technologies and protocols across the entire OSI and IoT stacks.

Contents

List of Figures

List of Tables

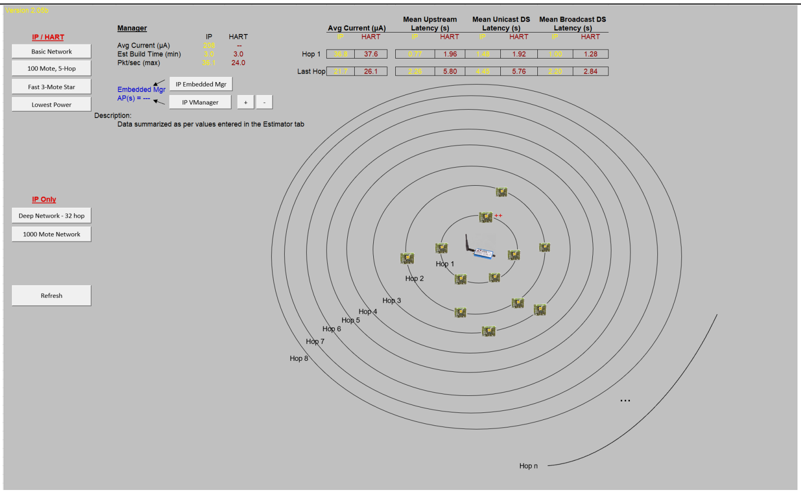
1 Part 1 - Medium Access Control

Contents

1.1 Performance estimator for 3 instances.

With the help of the performance estimator, generate 3 instances of an Internet of Things (IoT) scenario and comment on the rela- tionship between power and performance of the network. [40%]

Instance 1 - Urban Smart Lighting



Version 2.05b

IP / HART

Basic Network | 100 Mote, 5-Hop | Fast 3-Mote Star | Lowest Power

Manager

	IP	HART
Avg Current (µA)	208	--
Est Build Time (min)	3.0	3.0
Pkt/sec (max)	36.1	24.0

Embedded Mgr
AP(s) = ---

IP Embedded Mgr | IP VManager | + | -

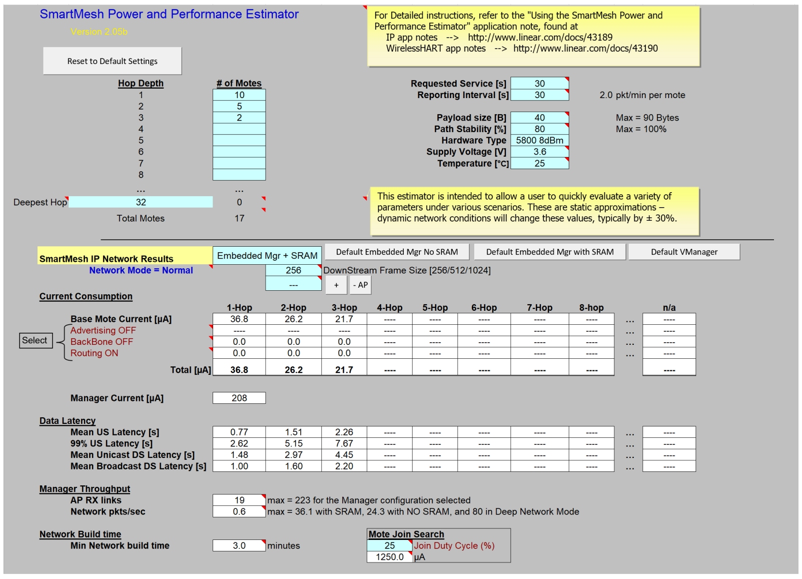
Description:
Data summarized as per values entered in the Estimator tab

	Avg Current (µA)		Mean Upstream Latency (s)		Mean Unicast DS Latency (s)		Mean Broadcast DS Latency (s)	
	IP	HART	IP	HART	IP	HART	IP	HART
Hop 1	36.8	37.6	0.77	1.96	1.48	1.92	1.00	1.28
Last Hop	21.7	26.1	2.26	5.80	4.45	5.76	2.20	2.84

IP Only

Deep Network - 32 hop | 1000 Mote Network

Refresh



SmartMesh Power and Performance Estimator

Version 2.05b

Reset to Default Settings

For Detailed instructions, refer to the "Using the SmartMesh Power and Performance Estimator" application note, found at
IP app notes --> http://www.linear.com/docs/43189
WirelessHART app notes --> http://www.linear.com/docs/43190

Hop Depth	# of Motes
1	10
2	5
3	2
4	
5	
6	
7	
8	
...	...
Deepest Hop 32	0
Total Motes	17

Requested Service [s]	30	
Reporting Interval [s]	30	2.0 pkt/min per mote
Payload size [B]	40	Max = 90 Bytes
Path Stability [%]	80	Max = 100%
Hardware Type	5800 8dBm	
Supply Voltage [V]	3.6	
Temperature [°C]	25	

This estimator is intended to allow a user to quickly evaluate a variety of parameters under various scenarios. These are static approximations – dynamic network conditions will change these values, typically by ± 30%.

SmartMesh IP Network Results | Embedded Mgr + SRAM | Default Embedded Mgr No SRAM | Default Embedded Mgr with SRAM | Default VManager

Network Mode = Normal

256 DownStream Frame Size [256/512/1024]

--- + - AP

Current Consumption

	1-Hop	2-Hop	3-Hop	4-Hop	5-Hop	6-Hop	7-Hop	8-hop		n/a
Base Mote Current [µA]	36.8	26.2	21.7	----	----	----	----	----	...	----
Select Advertising OFF	----	----	----	----	----	----	----	----	...	----
BackBone OFF	0.0	0.0	0.0	----	----	----	----	----	...	----
Routing ON	0.0	0.0	0.0	----	----	----	----	----	...	----
Total [µA]	**36.8**	**26.2**	**21.7**	**----**	**----**	**----**	**----**	**----**		**----**

Manager Current [µA] 208

Data Latency

Mean US Latency [s]	0.77	1.51	2.26	----	----	----	----	----	...	----
99% US Latency [s]	2.62	5.15	7.67	----	----	----	----	----	...	----
Mean Unicast DS Latency [s]	1.48	2.97	4.45	----	----	----	----	----	...	----
Mean Broadcast DS Latency [s]	1.00	1.60	2.20	----	----	----	----	----	...	----

Manager Throughput

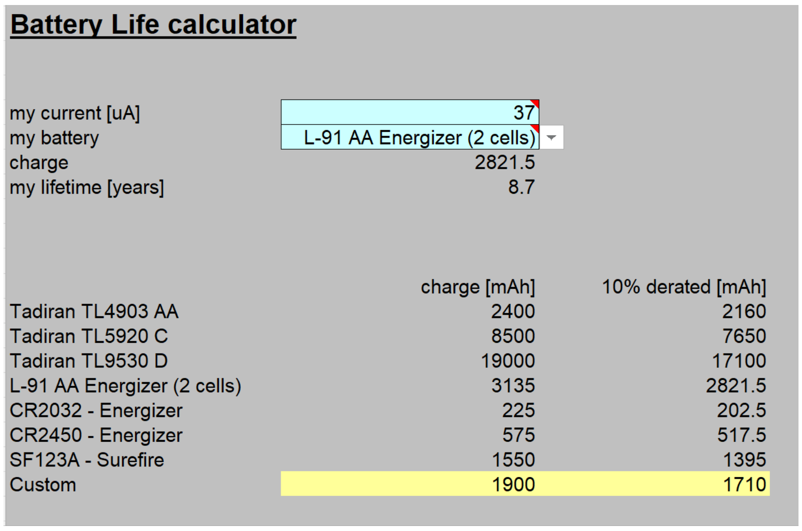
AP RX links	19	max = 223 for the Manager configuration selected
Network pkts/sec	0.6	max = 36.1 with SRAM, 24.3 with NO SRAM, and 80 in Deep Network Mode

Network Build time

Min Network build time 3.0 minutes

Mote Join Search

25	Join Duty Cycle (%)
1250.0	µA



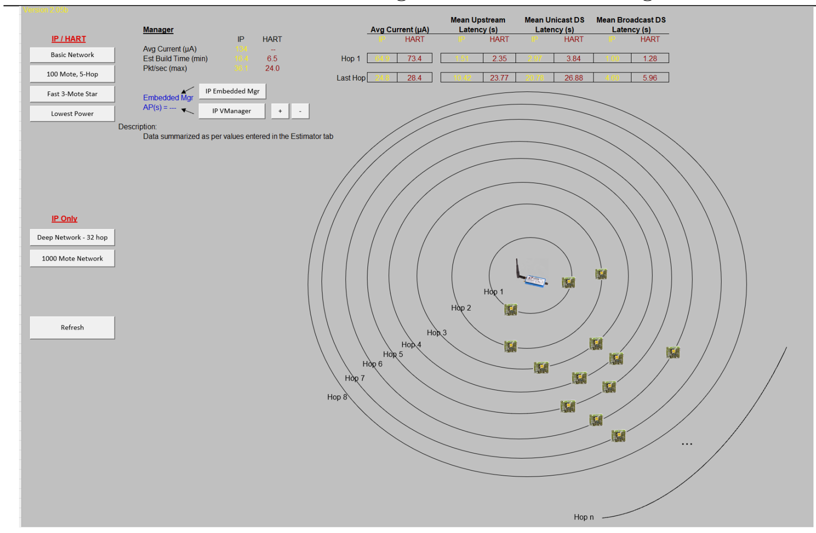
Battery Life calculator

my current [uA]	37
my battery	L-91 AA Energizer (2 cells)
charge	2821.5
my lifetime [years]	8.7

	charge [mAh]	10% derated [mAh]
Tadiran TL4903 AA	2400	2160
Tadiran TL5920 C	8500	7650
Tadiran TL9530 D	19000	17100
L-91 AA Energizer (2 cells)	3135	2821.5
CR2032 - Energizer	225	202.5
CR2450 - Energizer	575	517.5
SF123A - Surefire	1550	1395
Custom	1900	1710

Table 1.1: A network proposal for a couple of sensor-enabled IoT powered lamp posts. High density of nodes (lights), each operating at low transmission power with moderate reporting interval and payload size for status updates. Uses Zigbee because of short distances.

Instance 2 - IoT Agricultural Monitoring



Version 2.05b

IP / HART

Manager

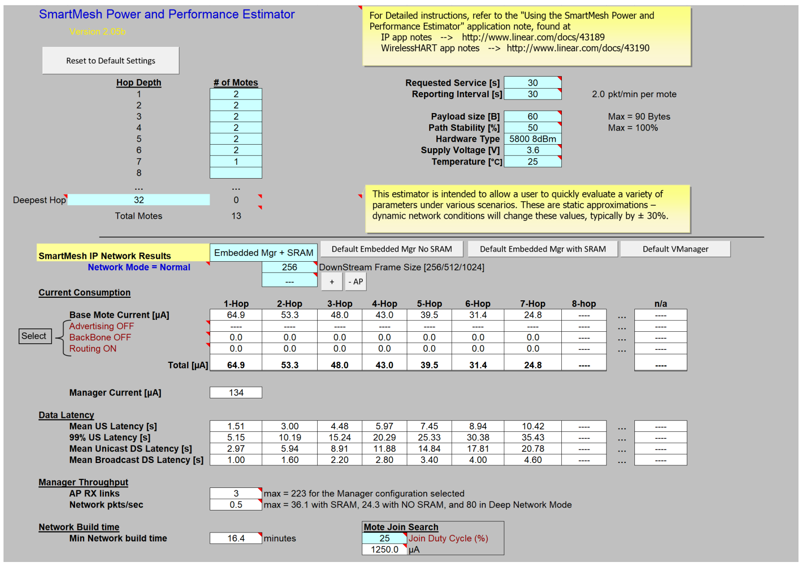
	IP	HART
Avg Current (µA)	134	--
Est Build Time (min)	16.4	6.5
Pkt/sec (max)	36.1	24.0

Embedded Mgr
AP(s) = ---

Description:
Data summarized as per values entered in the Estimator tab

	Avg Current (µA)		Mean Upstream Latency (s)		Mean Unicast DS Latency (s)		Mean Broadcast DS Latency (s)	
	IP	HART	IP	HART	IP	HART	IP	HART
Hop 1	64.9	73.4	1.51	2.35	2.97	3.84	1.00	1.28
Last Hop	24.8	28.4	10.42	23.77	20.78	26.88	4.60	5.96

IP Only



SmartMesh Power and Performance Estimator

Version 2.05b

For Detailed instructions, refer to the "Using the SmartMesh Power and Performance Estimator" application note, found at
IP app notes --> http://www.linear.com/docs/43189
WirelessHART app notes --> http://www.linear.com/docs/43190

Hop Depth	# of Motes
1	2
2	2
3	2
4	2
5	2
6	2
7	1
8	
...	...
Deepest Hop 32	0
Total Motes	13

Requested Service [s]	30	
Reporting Interval [s]	30	2.0 pkt/min per mote
Payload size [B]	60	Max = 90 Bytes
Path Stability [%]	50	Max = 100%
Hardware Type	5800 8dBm	
Supply Voltage [V]	3.6	
Temperature [°C]	25	

This estimator is intended to allow a user to quickly evaluate a variety of parameters under various scenarios. These are static approximations – dynamic network conditions will change these values, typically by ± 30%.

SmartMesh IP Network Results Embedded Mgr + SRAM
Network Mode = Normal
256 DownStream Frame Size [256/512/1024]

Reset to Default Settings
Default Embedded Mgr No SRAM
Default Embedded Mgr with SRAM
Default VManager
+
- AP
Select

Current Consumption

	1-Hop	2-Hop	3-Hop	4-Hop	5-Hop	6-Hop	7-Hop	8-hop		n/a
Base Mote Current [µA]	64.9	53.3	48.0	43.0	39.5	31.4	24.8	----	...	----
Advertising OFF	----	----	----	----	----	----	----	----	...	----
BackBone OFF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	----	...	----
Routing ON	0.0	0.0	0.0	0.0	0.0	0.0	0.0	----	...	----
Total [µA]	**64.9**	**53.3**	**48.0**	**43.0**	**39.5**	**31.4**	**24.8**	**----**		**----**

Manager Current [µA] 134

Data Latency

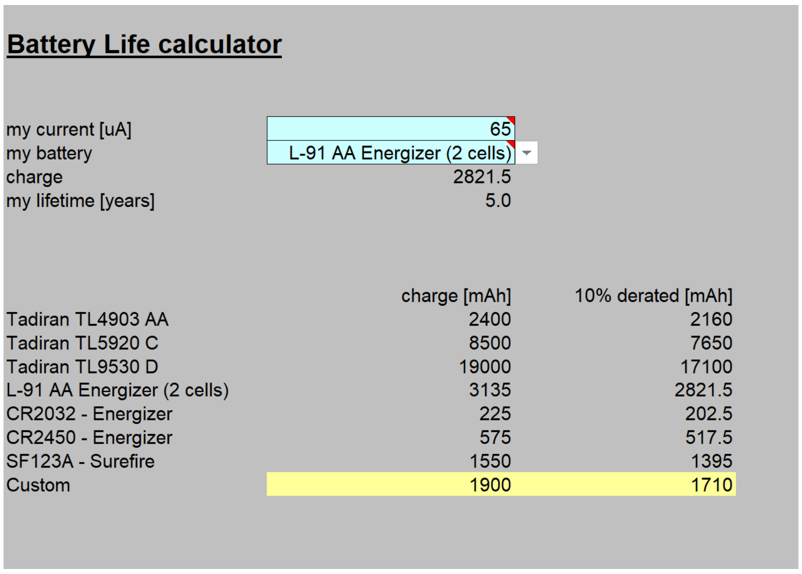
Mean US Latency [s]	1.51	3.00	4.48	5.97	7.45	8.94	10.42	----	...	----
99% US Latency [s]	5.15	10.19	15.24	20.29	25.33	30.38	35.43	----	...	----
Mean Unicast DS Latency [s]	2.97	5.94	8.91	11.88	14.84	17.81	20.78	----	...	----
Mean Broadcast DS Latency [s]	1.00	1.60	2.20	2.80	3.40	4.00	4.60	----	...	----

Manager Throughput

AP RX links	3	max = 223 for the Manager configuration selected
Network pkts/sec	0.5	max = 36.1 with SRAM, 24.3 with NO SRAM, and 80 in Deep Network Mode

Network Build time
Min Network build time 16.4 minutes

Mote Join Search
25 Join Duty Cycle (%)
1250.0 µA



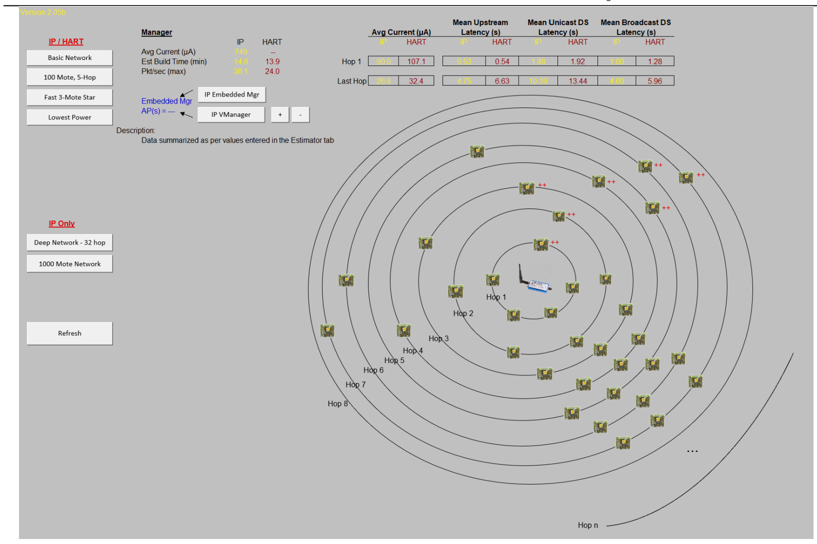
Battery Life calculator

my current [uA]	65
my battery	L-91 AA Energizer (2 cells)
charge	2821.5
my lifetime [years]	5.0

	charge [mAh]	10% derated [mAh]
Tadiran TL4903 AA	2400	2160
Tadiran TL5920 C	8500	7650
Tadiran TL9530 D	19000	17100
L-91 AA Energizer (2 cells)	3135	2821.5
CR2032 - Energizer	225	202.5
CR2450 - Energizer	575	517.5
SF123A - Surefire	1550	1395
Custom	1900	1710

Table 1.2: A network with nodes equipped with soil moisture and temperature monitors in an agricultural land. Sparse node distribution across a large area. Larger payload due to detailed sensor data and longer reporting intervals throughout the day.

Instance 3 - Industrial IoT Factory



Version 2.05b

IP / HART

Basic Network | 100 Mote, 5-Hop | Fast 3-Mote Star | Lowest Power

Manager

	IP	HART
Avg Current (µA)	749	--
Est Build Time (min)	14.0	13.9
Pkt/sec (max)	36.1	24.0

Embedded Mgr
AP(s) = ---

IP Embedded Mgr | IP VManager | + | -

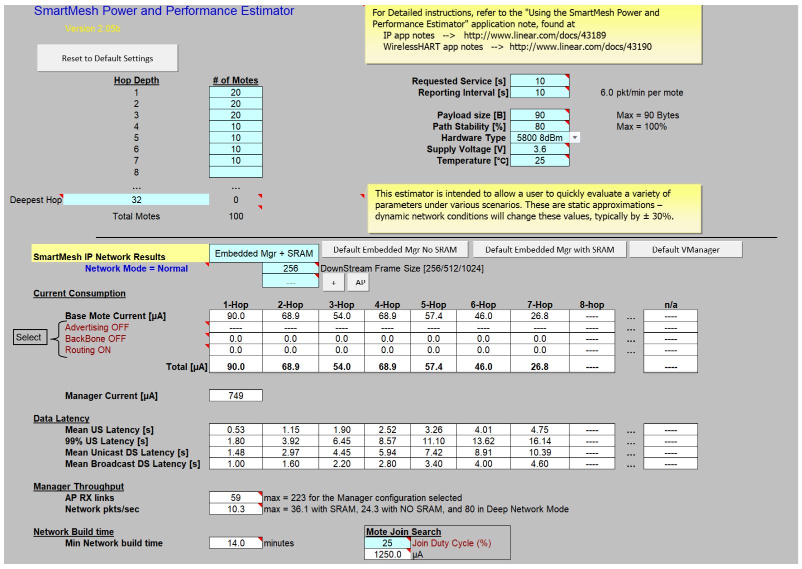
Description:
Data summarized as per values entered in the Estimator tab

	Avg Current (µA)		Mean Upstream Latency (s)		Mean Unicast DS Latency (s)		Mean Broadcast DS Latency (s)	
	IP	HART	IP	HART	IP	HART	IP	HART
Hop 1	90.0	107.1	0.53	0.54	1.48	1.92	1.00	1.28
Last Hop	26.8	32.4	4.75	6.63	10.39	13.44	4.60	5.96

IP Only

Deep Network - 32 hop | 1000 Mote Network

Refresh



SmartMesh Power and Performance Estimator

Version 2.05b

Reset to Default Settings

For Detailed instructions, refer to the "Using the SmartMesh Power and Performance Estimator" application note, found at
IP app notes --> http://www.linear.com/docs/43189
WirelessHART app notes --> http://www.linear.com/docs/43190

Hop Depth	# of Motes
1	20
2	20
3	20
4	10
5	10
6	10
7	10
8	
...	...
Deepest Hop 32	0
Total Motes	100

Requested Service [s]	10	
Reporting Interval [s]	10	6.0 pkt/min per mote
Payload size [B]	90	Max = 90 Bytes
Path Stability [%]	80	Max = 100%
Hardware Type	5800 8dBm	
Supply Voltage [V]	3.6	
Temperature [°C]	25	

This estimator is intended to allow a user to quickly evaluate a variety of parameters under various scenarios. These are static approximations – dynamic network conditions will change these values, typically by ± 30%.

SmartMesh IP Network Results | Embedded Mgr + SRAM | Default Embedded Mgr No SRAM | Default Embedded Mgr with SRAM | Default VManager

Network Mode = Normal

256 DownStream Frame Size [256/512/1024]

--- | + | AP

Current Consumption

	1-Hop	2-Hop	3-Hop	4-Hop	5-Hop	6-Hop	7-Hop	8-hop		n/a
Base Mote Current [µA]	90.0	68.9	54.0	68.9	57.4	46.0	26.8	----	...	----
Select Advertising OFF	----	----	----	----	----	----	----	----	...	----
BackBone OFF	0.0	0.0	0.0	0.0	0.0	0.0	0.0	----	...	----
Routing ON	0.0	0.0	0.0	0.0	0.0	0.0	0.0	----	...	----
Total [µA]	**90.0**	**68.9**	**54.0**	**68.9**	**57.4**	**46.0**	**26.8**	**----**		**----**

Manager Current [µA] 749

Data Latency

Mean US Latency [s]	0.53	1.15	1.90	2.52	3.26	4.01	4.75	----	...	----
99% US Latency [s]	1.80	3.92	6.45	8.57	11.10	13.62	16.14	----	...	----
Mean Unicast DS Latency [s]	1.48	2.97	4.45	5.94	7.42	8.91	10.39	----	...	----
Mean Broadcast DS Latency [s]	1.00	1.60	2.20	2.80	3.40	4.00	4.60	----	...	----

Manager Throughput

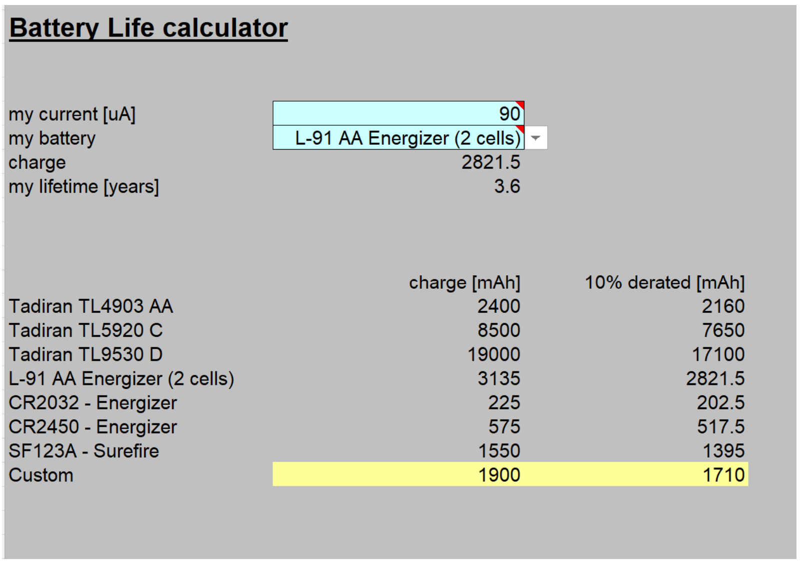
AP RX links	59	max = 223 for the Manager configuration selected
Network pkts/sec	10.3	max = 36.1 with SRAM, 24.3 with NO SRAM, and 80 in Deep Network Mode

Network Build time

Min Network build time 14.0 minutes

Mote Join Search

25	Join Duty Cycle (%)
1250.0	µA



Battery Life calculator

my current [uA]	90
my battery	L-91 AA Energizer (2 cells)
charge	2821.5
my lifetime [years]	3.6

	charge [mAh]	10% derated [mAh]
Tadiran TL4903 AA	2400	2160
Tadiran TL5920 C	8500	7650
Tadiran TL9530 D	19000	17100
L-91 AA Energizer (2 cells)	3135	2821.5
CR2032 - Energizer	225	202.5
CR2450 - Energizer	575	517.5
SF123A - Surefire	1550	1395
Custom	1900	1710

Table 1.3: A network for monitoring the health and efficiency of machinery in a smart IoT factory. Dense node distribution, high data throughput, short reporting intervals, and high transmission power for real-time monitoring.

1. The different parameters that characterize the network.
   - **Reporting interval**: how frequent a mote needs to send information. For example, for Instance 3, each mote will send a packet every 10 seconds because of the real-time monitoring needed for industrial IoT in factories. Comparing this to Instance 1 and 2 where higher reporting intervals are better because the nodes are equipped with sensors for either someone walking in front of the lamp-post (Instance 1) or for temperature of the soil declining by a certain level (Instance 2)
   - **Payload size**: the dimension of the packet that each mote sends. For Instances 1 and 2 the payload size is smaller because it is sensor based-data. Someone either activates the sensor or not. Payload for instance 2 is higher than for Instance 1 because data such as soil temperature and moisture might have to be transmitted. And Instance 3 has the highest payload because it would collect data from various machines and equipment in the factory, allowing manufacturers to identify trends and areas for improvement.
   - **Path stability**: reliability of the connection (e.g., 50% means that on average half of the transmissions fail). Typical path stability varies per environment but 80% is usual for indoor settings with moderate WiFi use which is why the scenario of Smart Urban Lighting and the Industrial IoT Factory (Instances 1 and 3) have 80% stability while the Agricultural IoT Monitoring in a possibly remote and outdoor area only has 50% stability because of parameters such as interference, collision with overlapping local networks and multi path fading. [1]
2. The latency experienced at different hops.
   - Latency is higher for packets being sent from motes that are far away because they have to go through multiple hops which is the case for the Instance 2 and 3. Urban scenario may exhibit low latency due to shorter distances, whereas agricultural monitoring might face higher latency due to long distances. Unlike Instance 1, where the area needed to be covered could be the length of one small park street, Instances 2 and 3 require larger areas such as acres of land or a couple km of factories. The more hops they have to go through, the more likely it is that collisions will happen because other packets are being sent forward at the same time. This leads to more latency for packets being sent from far away mops that need to hop multiple times in the MAC layer [2].
3. Battery vs number of hops.

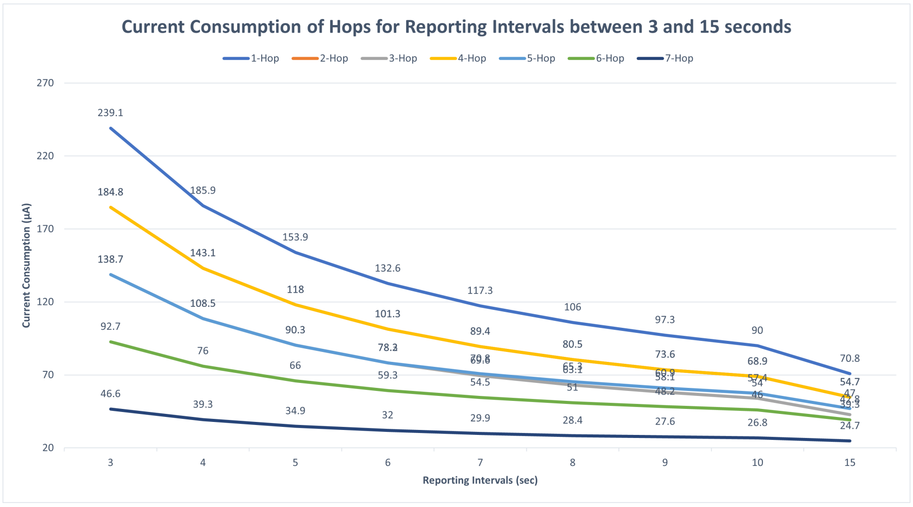


Figure 1.1: This figure was done with the provided Power and Performance Estimator Worksheet. 2-Hop is almost identical to 3-Hop and 5-Hop which is why the line underneath (Graph made from data in Tables 1.1 - 1.3).

- More hops typically mean more power consumption and reduced battery life. Additionally, sending more frequent information increases the data-rate at the trade-off for greater battery consumption, while longer packets leads to a gain in battery consumption at the trade-off of increased delay.
- Motes far away consume much less because they don't need to carry the packets forward like all the motes do that are closer to the gateway. Which is why in Instance 3, where there are the most amount of motes on most hops, the battery life for 1-Hop is the 1.4 years less than in Instance 2 and 5.1 years less than Instance 1.

4. Transmission frequency vs consumption.

   - Sending more frequent information increase the data-rate but at a large energy consumption price. The reporting interval for Instance 3 (IoT factory) is much lower than Instances 1 and 2. A 20 second difference in reporting intervals contributes to the highest current consumption out of all Instances. At a 10 second reporting interval it can be seen that 1-Hop current consumption is $90\mu A$. This is the only interval in which the current consumption is below 90 for all Hops which optimises the network's battery life while the machines functioning in the IoT Factory as seen in Fig 1.1 [2]

5. Latency vs consumption.

   - Motes far away consume less but at the price of higher latency. This can be seen very clearly in Instance 3 (IoT Factory) where even though there are less motes on the hops further away, the latency is much higher on the 7-Hop ($16.14s$) compared to 1-Hop

($1.80s$). While latency increases the further away the mote is, the current consumption however, decreases from 1-Hop ($90\mu A$) to 7-Hop ($26.8\mu A$). This is because the motes closest to the gateway have to process the information sent from all the previous motes which increases consumption but once it is processed it can be sent to the gateway quickly, therefore low latency. [1]

6. Network build time vs consumption.

   - Faster network setup might require more power. In critical applications like industrial IoT, quick network establishment is essential, even if it's power-intensive.
   - The protocol is scheduled, but the duty cycle has an effect at the beginning when the network is building up. This can be seen in Instance 3 (IoT Factory) where the minimum time to have all the motes join the network manager and be "operational" is 14 minutes. With higher duty cycle the network synchronizes faster, but at the price of higher consumption[3].

1.2 Slotted Aloha Python Simulation for the instances.

With the help of Python simulate Slotted Aloha for your instances. [40%] Analyze in detail (use plots and text for presenting your results):

1. **The impact on slotted Aloha's performance with different packet arrival rates.**

   Code available in Appendix A.1. Result seen in Figure 1.2

   The packet arrival rates range for industrial IoT factory applications was determined to be between 0.016 and 1 from literature. [4]

   It is clear that the most long-lasting efficiency for a window of 8 is for the packet arrival rates of 0.416 but that comes at the price of high volatility in slot efficiency making it less predictable. We recommend using a packet arrival rate of 0.616 for an instance such as agricultural monitoring where the total nr of nodes would be 13 or for the urban smart lighting for 17 nodes in total. A different window size is recommended for an instance of more than 100 nodes such as the industrial IoT factory. [5]

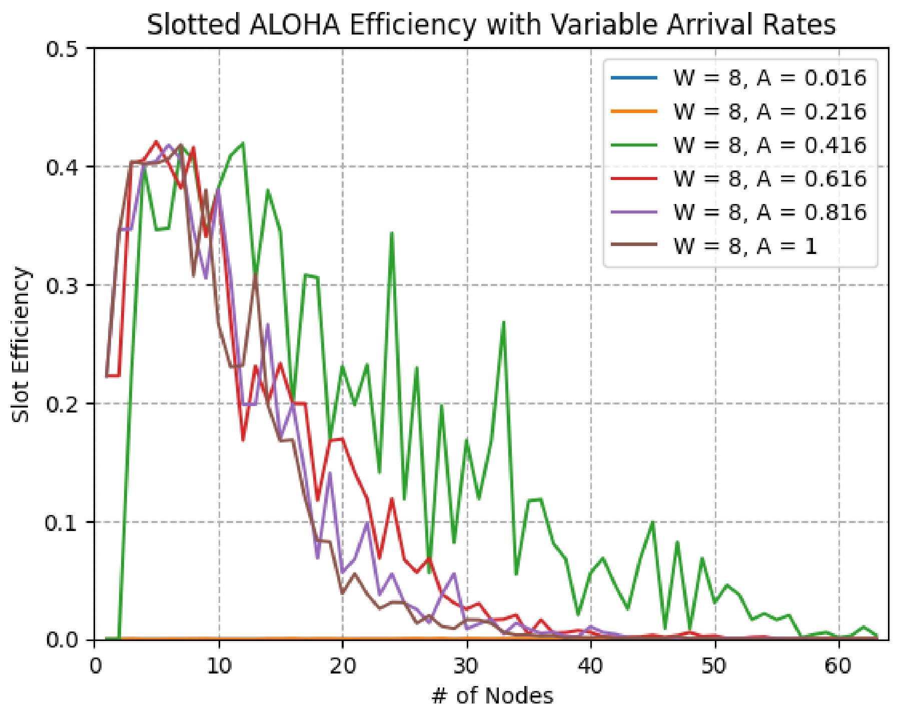


Figure 1.2: Graph of different packet arrival rates for a window size of 8 for Slotted ALOHA protocol.

2. **Adjust the slot size and assess how it affects the protocol's performance.**

   Code available in Appendix A.2. Result seen in Figure 1.3

   For the case of the packet arrival rate being 0.616 a couple of window sizes were tested in Fig 1.3. Because the number of nodes for the instance of the IoT factory is a total of 100 Motes, the window size with the highest slot efficiency over the longest period of time would have to be one of either 128 or 64. The change of slot sizes affects the protocol's performance because the smaller the slot size, the more efficient it would be for small number of nodes. Once a lot of packets are being sent through more than 50 nodes, larger slot sizes are better because they can be timed accordingly. More nodes also means higher latency because it would take longer for information to be sent down to the gateway.

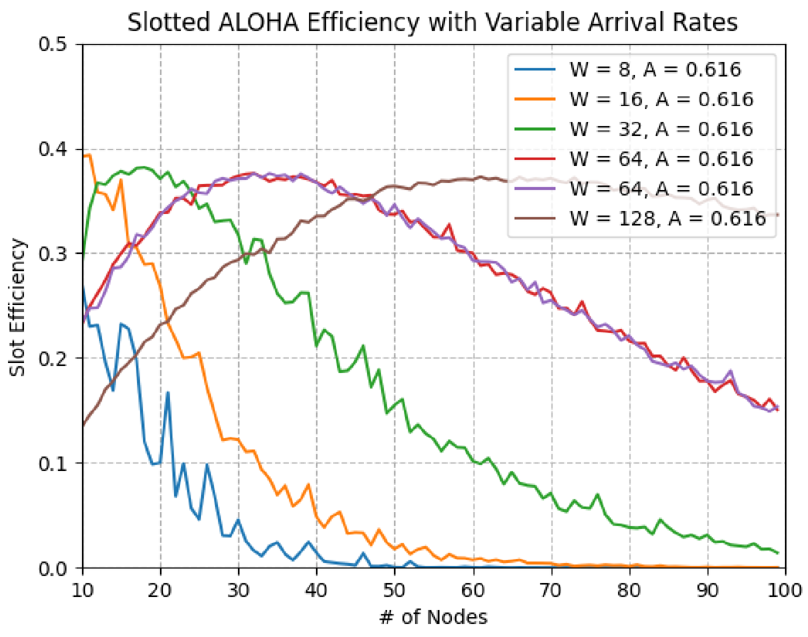


Figure 1.3: Graph of same packet arrival rate for different window size for Slotted ALOHA protocol for a halved slot size.

3. **Explore the consequence of collisions and discuss potential collision handling strategies.**

   - **Exponential Backoff**: After a collision, the device waits for a random amount of time before reattempting transmission, with the wait time increasing exponentially with each subsequent collision. This BACKOFF is a key part of the CSMA protocol and not Slotted-ALOHA.
   - **Hand-Shaking**: Techniques like CSMA/CA can be employed to reduce the likelihood of collisions by listening to the channel before transmitting. These might be handshake methods or carrier sensing but they come at the cost of higher energy consumption. A handshaking method would also solve the Hidden Node Problem.
   - **Controlling Packet Overhead**: Controlling packet overhead and overhearing can be used to reduce collisions. This is because it can reduce re-transmissions. This is a trade-off between energy efficiency and latency.

1.3 A multi-user scenario:

1. Consider a multi-user scenario with N = 10 sources transmitting based on the Slotted-Aloha protocol. Assume that each source has a packet generation rate of $\lambda = 10^3$ packets per second and packet length $(M = 10)^3$ bits, transmitting over a channel with transmission rate R. Given that a normalized traffic $L = 1$ is experienced, evaluate the transmission rate. What is the impact of a higher packet generation rate? [20%]

In Slotted-ALOHA protocol, the transmission rate is influenced by the number of active users and their packet generation rate. As the throughput S of the protocol is given by the formula $S = G \times e^{-G}$ where G is the average number of packets generated by the system.[6]

Given that there are $N = 10$ users, each generating $\lambda = 10^3$ packets per second, and assuming packet length of $M = 10^3$ bits, we need to calculate packet time T.

The normalised traffic $L = 1$ can be substituted in the equation

$$L = N \times \lambda \times T = 1$$

to solve for T: $10 \times 10^3 \times T = 1$

$$\therefore T = 1/10000$$

The formula $T = M/R$ can be rearranged to solve for $R = M/T$. [3]

$$R = 10^3 \div \frac{1}{10000}$$

$$\therefore R = 10,000,000 \text{ bits per second}$$

Increasing the packet generation rate (λ) while keeping other parameters constant will increase the normalized traffic L, leading to a higher likelihood of packet collisions in the Slotted-Aloha system. This will decrease the system efficiency and throughput. Because this is a contention-based protocol, it is scale-able but at the trade-off of more collisions. This is also ideal for discontinuous data and can therefore accommodate more users than channel resources.

Because this is Slotted-ALOHA, and not CSMA, the sources transmit without sensing the channel which means that it has a lower energy consumption at the trade-off of possibly more collisions. This is reasonable for our multi-user scenario because Slotted-ALOHA is efficient at low load.

2 Part 2 - Physical Layer

Contents

The company is looking to acquire a new wireless technology. You have been told that the inventors are using 16-QAM as a modulation scheme in the $2.4 * 10^7$ Hz band. It claims to offer bit rates of 100Mbit/s, 50Mbit/s, 25Mbit/s and 10Mbit/s at a BER = 10-4 and that the system will work up to 2km.

You have been asked to prepare a document evaluating the technology. The company has asked for the following issues to be considered in your document **maximum six pages**:

2.1 Create the QAM and OFDM objects and generate the transmit signal. [30%]

Given the bit rates $BR_1 = 10e+8$ bits/s, $BR_2 = 5e+7$ bits/s, $BR_3 = 2.5e+7$ bits/s, $BR_4 = 1e+7$ bits/s and the bandwidth = 2.4 Hz we can find the Symbol Rate (SR) with the formula $SR = \frac{BR}{log_2 M}$.

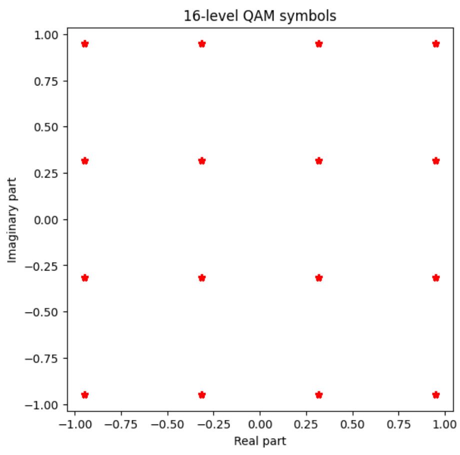
To get fft-size we need to calculate the subcarrier spacing with the formula:

$$\text{subcarrier spacing} = \frac{SR}{\text{nr subcarriers}}$$

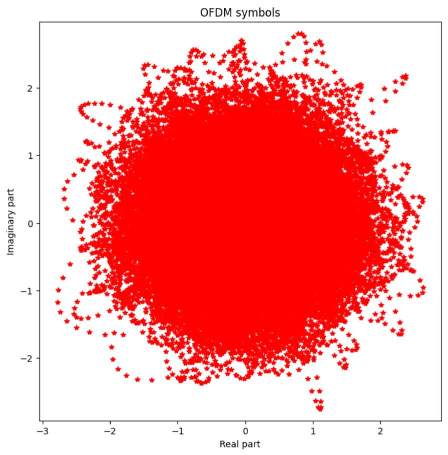
We then substitute the subcarrier spacing in the following formula to get multiple ftt-sizes:

$$\text{fft-size} = \frac{\text{bandwidth}}{\text{subcarrier spacing}}$$

We then see that $BR_4 = 10e7$ bits/s gets us to the fft-size = 8192 which then gives the $BER = 0.0325$. This is the closest to 10^{-4} than any other Bit Error Rates generated. ($BER_1 =$ ERROR because the ftt-size was 576 which is lower than the number of subcarriers $BER_2 = 0.04062$, $BER_3 = 0.06483$, $BER_4 = 0.03254$)



QAM-16 modulation constellation in a scatter plot



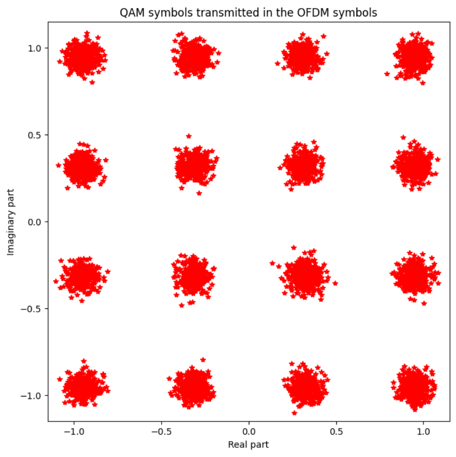
OFDM modulated data scatter plot

Table 2.1: 16-QAM and OFDM objects and the generated transmit signal

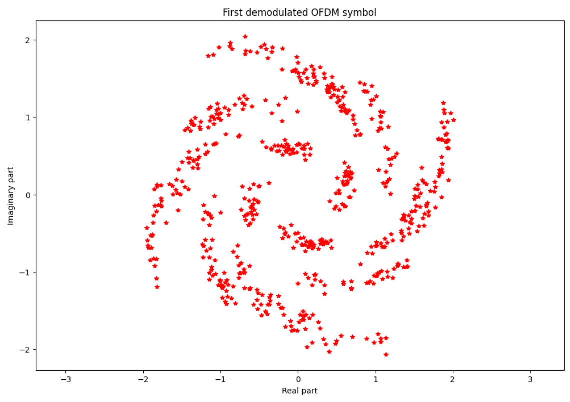
```
Average Bit Error Rate (30 runs): 0.01068
Average Bit Error Rate (60 runs): 0.01472
```

Code for average Bit Error Rates is available in Appendix B.1.

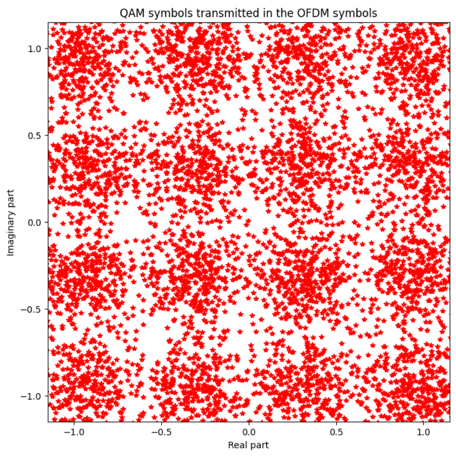
We found that increasing the `noise var` from $1e-1$ to $9e-1$ also increases the BER and SER and changes the preciseness of the signal as seen in Table 2.3 below.



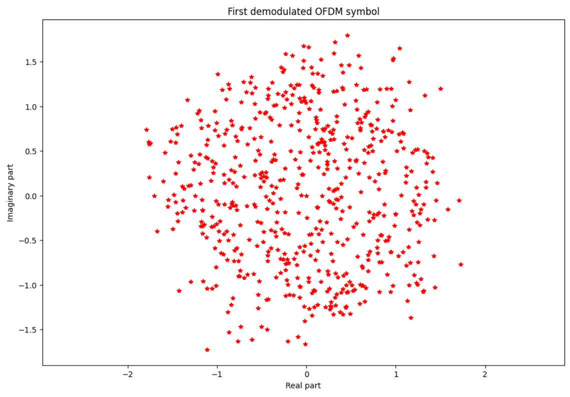
(a) 16-QAM symbols transmitted in the OFDM symbols with noise variation= $1e-1$, represented with its real and imaginary part



(b) First demodulated OFDM symbols with noise variation $= 1e-1$, represented with its real and imaginary part



(a) 16-QAM symbols transmitted in the OFDM symbols with noise variation $= 9e-1$, represented with its real and imaginary part



(b) First demodulated OFDM symbols with noise variation $= 9e-1$, represented with its real and imaginary part

Table 2.3: Characteristics of the impulse response channel before and after adding more noise for both the forst demodulated OFDM signal and the 16-QAM signal

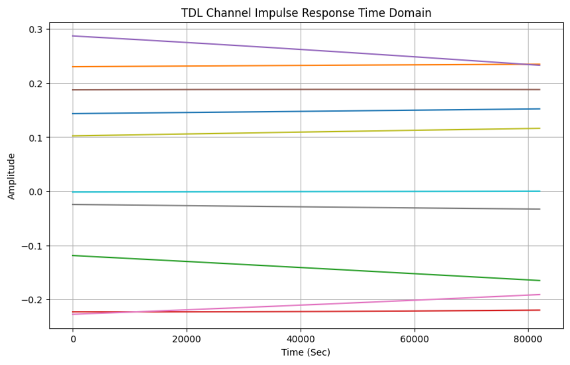
```
Symbol Error Rate WITHOUT noise: 0.03466

Symbol Error Rate WITH noise: 0.124
```

Code for Symbol Error Rates is available in Appendix B.2.

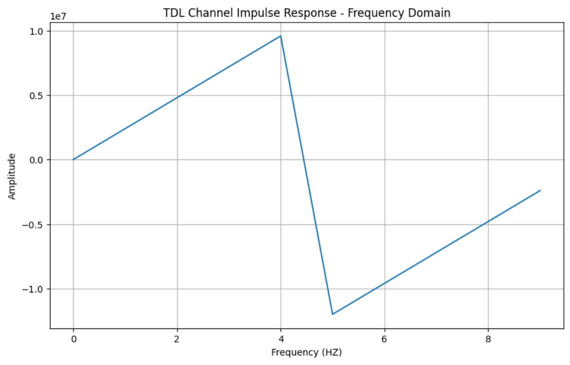
2.2 Characterize the impulse response of the channel. [30%]

The object `<pyphysim.channels.fading.TdlImpulseResponse` from `print(impulse_response)` is called TDL impulse response. The documentation of `pyphysim` library shows that the parameter `tap_values` from the `TdlImpulseResponse` class is an array. These values can be used to plot them against time and frequency to characterise the impulse response of the channel. Code available in Appendix B.3

Considering different times and different frequencies, the impulse response varies [7]. As seen in Table 2.4, the TDL Impulse Response based in time, varief only slightly, over time. When looking at the TDL Impulse Response based in frequency, the Amplitude increases up to 4(HZ), decreases drastically from 4(HZ) to 5(HZ) and then rises again with the same gradient to 9(HZ).



Impulse Response variation based in time



Impulse Response variation based in frequency

Table 2.4: Impulse Response Characteristics

It characterizes how the channel distorts or modifies a signal as it passes through and then tells the equalizer about this behaviour so that it can fix the Symbol Error Rate and reduce it to 0

Table 2.3 shows the characteristics of the impulse response channel before and after adding more noise. Code available in Appendix B.3

When selecting a modulation scheme we will always be trading off power efficiency against bandwidth efficiency. If the system is to be used in doors for example we might also need to consider a modulation scheme that provides immunity to Multipath Fading.

To avoid intersymbol interference for short bits on single carriers, we can use multi-carrier modulation. The short bits are taken through a Serial-to-Parallel Converter which outputs longer bits which would then be modulated on different frequency channels. Multipath allows for fading only to affect a small part of the signal and they main method is OFDM (Orthogonal Frequency

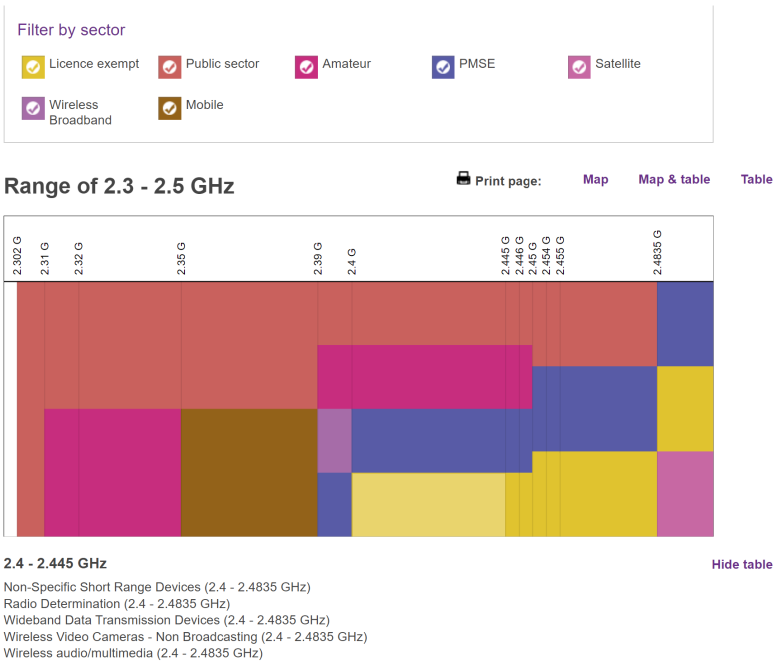
Division Multiplexing). This scheme that uses multiple channels that each carry a different data stream reduces the data rate on each carrier and gives better immunity to multipath. OFDM can achieve high data rates and robustness to fading by using multiple subcarriers, but it also requires more bandwidth, more power consumption, and more complexity than QAM.

2.3 Provide a short summary of the regulatory conditions in the UK associated with this frequency band. [20%]

The frequency band of 2.4GHz tends to be of low range and is characterised by technologies such as Bluetooth, ZigBee, ANT, 802.11b and 6LoWPAN [8]. This frequency band is heavily congested because it has all the Wireless LANs, Bluetooth, etc. which are used by microwaves, ovens, strip lighting, ring doorbells, baby monitors, Home Routers and HotSpot locations and for most wireless devices [9]. Wireless LANS, together with the 2.4GHz band are in the unlicensed spectrum band. The 2.4 GHz has been an unlicensed band since June 2003 in the UK and throughout Europe [10].

In the UK, Short Range Devices (SRD) do not need to be licensed unless they are interfering with radio communications services such as radars and microphones [11].

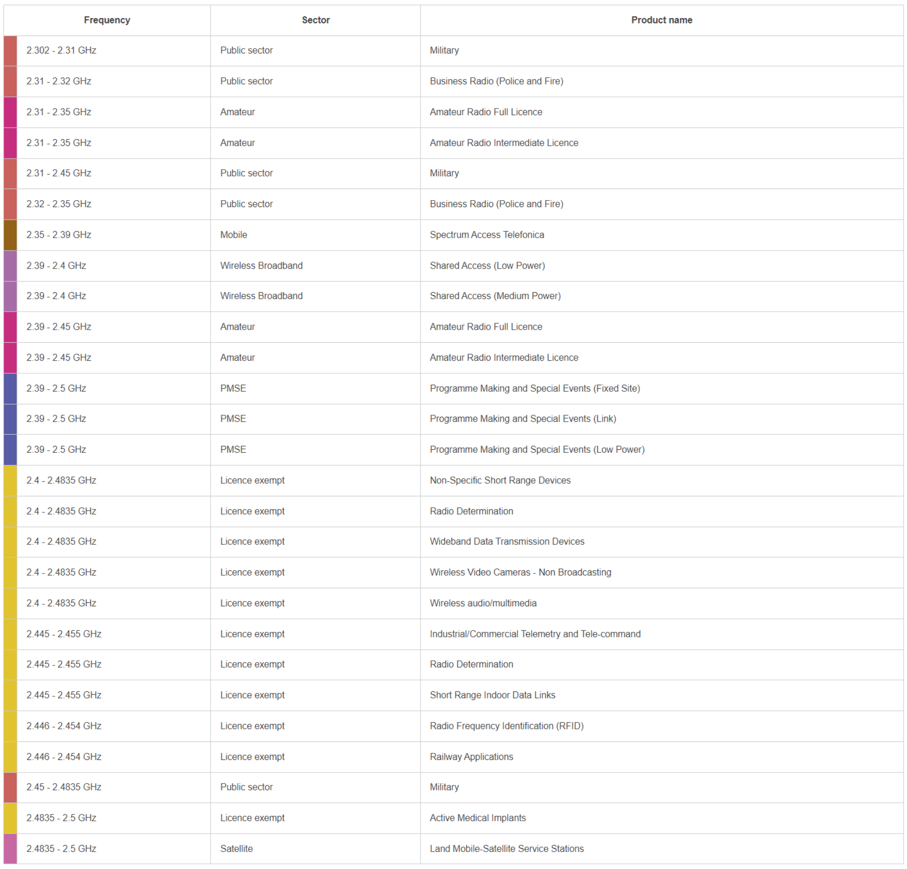
The low power 2.4GHz devices exempt from licensing are contained in Ofcom's Interface Requirement IR2030 as seen in Fig 2.2. The 2.4GHz band is also populated by the following products and sectors as seen in Fig 2.1 and Fig 2.2. The spectrum operating frequency range is 2412 – 2484MHz and the OFCOM /ETSI standard for UK Operating channels are between Channel 1 - 13 [12].



2.4 - 2.445 GHz

Non-Specific Short Range Devices (2.4 - 2.4835 GHz)
Radio Determination (2.4 - 2.4835 GHz)
Wideband Data Transmission Devices (2.4 - 2.4835 GHz)
Wireless Video Cameras - Non Broadcasting (2.4 - 2.4835 GHz)
Wireless audio/multimedia (2.4 - 2.4835 GHz)

Figure 2.1: OFCOM's UK Frequency Allocation Table (UKFAT). Last Updated 29th March 2022 [13]



	Frequency	Sector	Product name
🟥	2.302 - 2.31 GHz	Public sector	Military
🟥	2.31 - 2.32 GHz	Public sector	Business Radio (Police and Fire)
🟪	2.31 - 2.35 GHz	Amateur	Amateur Radio Full Licence
🟪	2.31 - 2.35 GHz	Amateur	Amateur Radio Intermediate Licence
🟥	2.31 - 2.45 GHz	Public sector	Military
🟥	2.32 - 2.35 GHz	Public sector	Business Radio (Police and Fire)
🟫	2.35 - 2.39 GHz	Mobile	Spectrum Access Telefonica
🟪	2.39 - 2.4 GHz	Wireless Broadband	Shared Access (Low Power)
🟪	2.39 - 2.4 GHz	Wireless Broadband	Shared Access (Medium Power)
🟪	2.39 - 2.45 GHz	Amateur	Amateur Radio Full Licence
🟪	2.39 - 2.45 GHz	Amateur	Amateur Radio Intermediate Licence
🟦	2.39 - 2.5 GHz	PMSE	Programme Making and Special Events (Fixed Site)
🟦	2.39 - 2.5 GHz	PMSE	Programme Making and Special Events (Link)
🟦	2.39 - 2.5 GHz	PMSE	Programme Making and Special Events (Low Power)
🟨	2.4 - 2.4835 GHz	Licence exempt	Non-Specific Short Range Devices
🟨	2.4 - 2.4835 GHz	Licence exempt	Radio Determination
🟨	2.4 - 2.4835 GHz	Licence exempt	Wideband Data Transmission Devices
🟨	2.4 - 2.4835 GHz	Licence exempt	Wireless Video Cameras - Non Broadcasting
🟨	2.4 - 2.4835 GHz	Licence exempt	Wireless audio/multimedia
🟨	2.445 - 2.455 GHz	Licence exempt	Industrial/Commercial Telemetry and Tele-command
🟨	2.445 - 2.455 GHz	Licence exempt	Radio Determination
🟨	2.445 - 2.455 GHz	Licence exempt	Short Range Indoor Data Links
🟨	2.446 - 2.454 GHz	Licence exempt	Radio Frequency Identification (RFID)
🟨	2.446 - 2.454 GHz	Licence exempt	Railway Applications
🟥	2.45 - 2.4835 GHz	Public sector	Military
🟨	2.4835 - 2.5 GHz	Licence exempt	Active Medical Implants
🟪	2.4835 - 2.5 GHz	Satellite	Land Mobile-Satellite Service Stations

Figure 2.2: OFCOM's UK Frequency Allocation Table (UKFAT). [13]

2.4 Provide a short summary of competitor technologies. [20%]

A wireless standard for IoT with a high range of 2km in the low frequency band of 2.4GHz that has bit rates of 100kbit/s, 50kbit/s, 1Mbit/s and 5Mbit/s is hard to find. The best strategy is to consider which one of these metrics the company values the most. Given our case of 1Mbit/s causing the lowest Bit Error Rate, WiFi 802.11g could be the best solution for this case. The major modulation scheme for WiFi is OFDM with each carrier, carrying a 16 level QAM with a time division duplex (TDD). The maximum network bandwidth is 54 Mbps in the 2.4 GHz bands, however, it only operates in short distances [14].

If the channel is noisy and fading, OFDM may be a better choice than QAM, because it can cope with channel variations and use different modulation levels for each subcarrier. If the channel is stable and clear, QAM may be a better choice than OFDM [15]. OFDM has Good performance under delay spread conditions, good bandwidth efficiency and it is easier to equalise when compared to single carrier signals. Some disadvantages are poor performance under Doppler spread conditions (a characteristic of fast moving applications with time-varying channels) and a large Peak-to-average power ratio (PAPR). This causes poor performance under non-linear distortion which is the same as in QAM.

Another wireless technology such as ZigBee for example, is a mesh local area network (LAN) protocol which is in the 2.4GHz band. However, the bit rate is below 250kbits/s and the system would only work for up to 291m. Bluetooth LE is similar but with a higher data rate of 1Mb/s and a slightly lower range of 77m [16].

If a 2.4GHz frequency and higher range are important, then LoRa 2.4GHz might be an appropriate solution. It covers a range up to 1km on the 2.4GHz at the cost of a smaller bit rate than the standard LoRa variant (254kbit/s). However, the LoRa physical layer takes advantage of a Chirp Spreading Spectrum (CSS) modulation rather than the preferred Quadrature Amplitude Modulation (QAM) scheme [17].

3
Part 3 - Network and Transport

Contents

In this part, you are going to produce a design for an IP network of a commercial ISP in the UK. The ISP has about 30% of the market of Russian commercial residential broadband uniformly around the country and provides to its users a 100 Mbit/s connection. You should include:

3.1 The backbone topology of the network with realistic IP addresses and OSPF weights. [25%]

3.1.1 Topology

Assuming that the ISP operates in UK and in Russia, we propose designing the backbone topology as a hierarchical design with a core layer, distribution layer, and access layer as seen in Figure 3.2 [18].

1. **The core layer** will consist of high-speed routers that will provide connectivity between different regions. Because we have a large ISP network, the core layer will involve routing between different Autonomous System (AS) [19]. Destinations outside are learned through Borger Gateway Protocol (BGP) and are disseminated inside an AS through iBGP protocol [20].

2. **The distribution layer** will consist of routers that will connect to the core layer and provide connectivity to the access layer. This would also facilitate efficient data flow and enable network services like load balancing and security. These intra-domain routing protocols run inside each AS through and aggregate data before it is transported to the core layer.

3. **The access layer** will consist of switches that will connect to end-users and provide access to the Internet. These intra-domain routing protocols run inside each AS and destinations in the same AS use Open Shortest Path First Protocol (OPSF) led by the Dijkstra's algorithm [21]. This layer can be tailored to the specific needs of different user groups.

According to UK Government guidance, the IP network of a commercial ISP should have installed a resilient high speed internet link of 10Gbps. A bearer with the required maximum bandwidth should also be used, one that is flexible enough to support actual use. Smaller hubs could use a 1Gbps bearer, but this will reduce future expansion options [22].

The internet link must have:

1. 2 bearers (fibre links installed into the building).

2. 2 customer premises equipment (CPE) devices that connect the bearers to the customer network, using a first hop redundancy protocol (FHRP) such as Virtual Router Redundancy Protocol (VRRP) to provide failover capability.

3. routing protocols such as border gateway protocol (BGP) and open shortest path first (OSPF) to learn the default route depending on ISP capability and service options.

4. available public routed IP space (IPv4 and IPv6).

5. a switch that allows the internet link to connect to multiple devices.

3.1.2 IP addresses

We could assign the following IP addresses to the different layers for example:

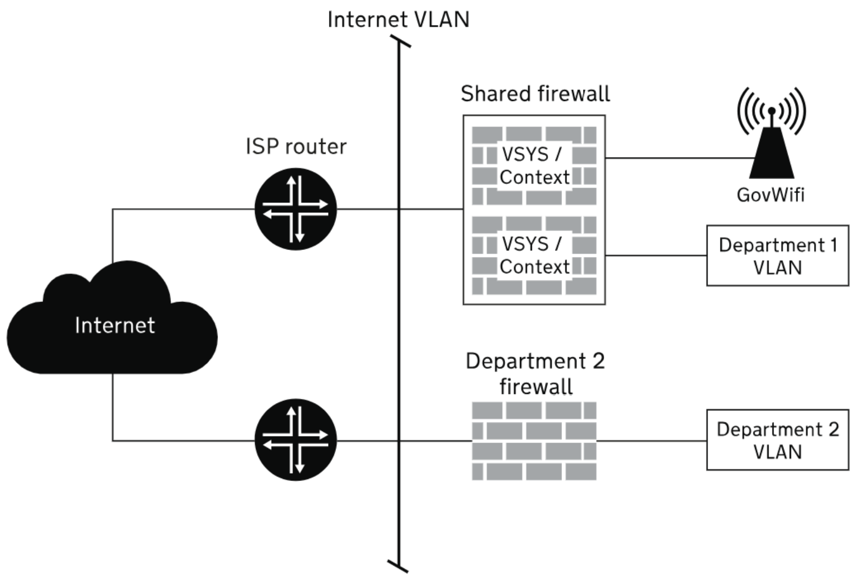


Figure 3.1: Diagram of network with an example of two ISP routers. [22]

1. Core layer: 10.0.0.0/8
2. Distribution/Regional layer: 172.16.0.0/12
3. Access layer: 192.168.0.0/18

These are the suggested IP addresses but they need to be requested officially through the RIPE NNC database [23].

We are suggesting an IPv4 address for the access layer with a subnet mask of 18 bits for network identification (netid) and the remaining 14 bits for hosting addresses within that network (hostid).

A larger subnet of /12 subnet is commonly used for organisations' internal networks. It allows for easier management of a large number of devices without the need for excessive subnetting. It facilitates efficient routing by summarizing multiple smaller networks (from the access layer) into a single, larger network. which is why we can use it to aggregate data on the distribution layer before sending it across Autonomous Systems [24].

Lastly, because the existing network infrastructure (distribution and access layers) predominantly uses IPv4, it may be practical for the core layer to also use IPv4 for compatibility and ease of integration. However, considering the exhaustion IPv4 address and the advanced features of IPv6, it's wise to plan for IPv6 implementation. This can be done gradually, using dual-stack configurations where devices run both IPv4 and IPv6 simultaneously. IPv6 introduces Quality of

Service (QoS) support, removes Checksum, reducing processing time in routers at each hop. IPv6 is justified by the need to manage a vast and varied customer base effectively.

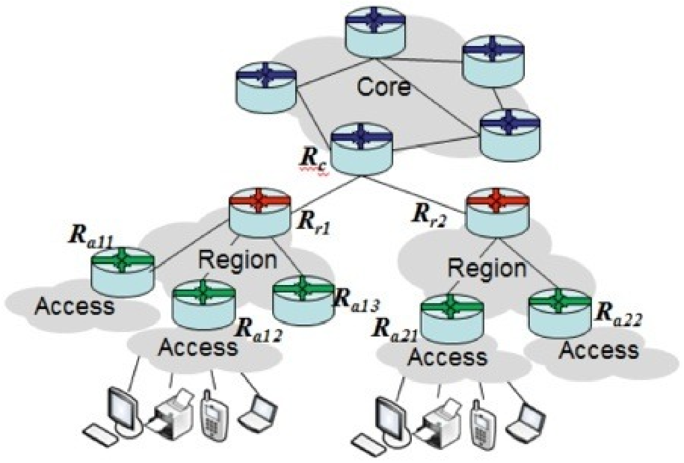


Figure 3.2: Diagram of router topology between Access, Distribution and Core Layers. [25]

Private IP network are connected to the Internet via a Network Address Translation (NAT) device. This is also feature of office routers that allows multiple computers in the building to communicate with the outer world which is essential for our commercial ISP with a 30% share of it's market in Russian residential broadband [26].

A study by BSRIA showed that there is a need for reliable numbers for connected devices in commercial buildings, with one example is the emergence of new Wi-Fi access points (Wi-Fi 6, 802.11ax) that can handle multiple devices using several protocols such as Bluetooth and Zigbee for IoT stacks.

BSRIA estimates the number of connected (wireless) devices in commercial buildings to be 150 – 200 million in 2019 worldwide [27].

"The majority of devices today are IP and are linked in subnetworks with a common backbone or connected via V-LANs, which enable centralised monitoring and control." Shown in Figure 3.2'

Based on the population densities in Russia, we recommend installing a router for every reagion that the Core Layer of th UK ISP should have access to. This Includes: Moscow, St Petersburg and Rostov-on-Don as seen in Figure 3.3.

We therefore recommend installing a router for every region in the Core Layer such as London, Manchetser, Birmingham, Edinburgh, Moscow, and St Petersburg, Rostov-on-Don. Those would be connected to the main infrastructure of the UK ISP and have the following OSPF weights attributed from the list of available suppliers in the UK [28].

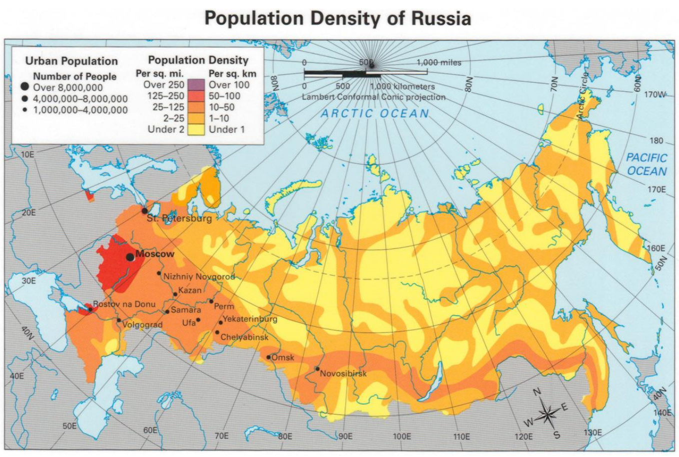


Figure 3.3: Population Density in Russia

3.1.3 OSPF Weights

The weights are inversely proportional to the bandwidth of the links and proportional to the latency. We'll assign higher weights to links farther from the Russian nodes to route more traffic through Moscow and St. Petersburg to make up 30% of ISP connected to the Russian commercial residential market. We can use OSPF as the routing protocol for the backbone network and assign the following OSPF weights to the different links:

1. Core layer links: 10
   (a) For example London - Moscow, London-Birmingham, London - Manchester, London - Edinburgh, London - St. Petersburg, Moscow - St. Petersburg, Moscow - Rostov-on-Don.
2. Distribution layer links: 20
   (a) For example Birmingham - Edinburgh, Manchester - Birmingham, Manchester - Edinburgh, Moscow - St Petersburg, Moscow - Rostov-on-Don.
3. Access layer links: 30
   (a) For example 200 users to 4 Wi-Fi 6, 802.11ax.

The given weights are an example and would be adjusted based on actual network performance metrics and traffic patterns. The priority is to ensure that a significant portion of the traffic is routed through Moscow and St Petersburg to cater to the 30% Russian market share. This design aims to balance the traffic load efficiently while prioritizing the Russian segment of the network.

Regular monitoring and adjustments would be necessary to optimize performance as traffic patterns evolve.

3.2 Proposed capacity of each link taking into account population in each region/city. [25%]

To determine the capacity of each link, we need to take into account the population in each region/city. Assuming that the ISP has 30% of the market share in Russia. Based on this, we can estimate the number of users in Russia and calculate the required link capacity. For Russia's population of 143,666,931 in 2014, 30% market share would mean 43,100,079 users. For a 100 Mbit connection, the total required capacity for that Russia would be 4,310,007 Gbit/s.

Based on this calculation we can propose the following link capacities based on the population in each city:

1. London - Moscow: 649,443.3 Gbit/s
2. Manchester - Moscow: 376,500 Gbit/s
3. Birmingham - Moscow: 394,500 Gbit/s
4. Edinburgh - Moscow: 375,900 Gbit/s
5. London - St Petersburg: 448,443.3 Gbit/s
6. Manchester - St Petersburg: 177,000 Gbit/s
7. Birmingham - St Petersburg: 196,500 Gbit/s
8. Edinburgh - St Petersburg: 178,950 Gbit/s
9. London - Manchester: 305,500 Gbit/s
10. London - Birmingham: 309,000 Gbit/s
11. London - Edinburgh: 304,900 Gbit/s
12. Manchester - Birmingham: 171,000 Gbit/s
13. Manchester - Edinburgh: 171,950 Gbit/s
14. Birmingham - Edinburgh: 171,450 Gbit/s

[29]

These link capacities should be sufficient to handle the traffic from the estimated number of users in each region/city. However, the ISP should monitor the network traffic and upgrade the link capacities if necessary. Given that the BT sees UK network traffic peak at 28 Terabits per second during their busiest hour, the stated links are likely to be too high. With a ridiculous 4,471.586 Tb/s in total, strategies such as statistical multiplexing and data compression have to be used for lower link capacities [30]

If we only consider the changes need to be made to the UK ISP to account for a 30% share of the Russian market, we can propose the following link capacities that would support the network at the necessary quality and provide 100 Mbits/s broadband for each user in Russia:

1. Core layer links: 100 Gbit/s
2. Distribution layer links: 10 Gbit/s
3. Access layer links: 1 Gbit/s

3.3 Wireshark

Using Wireshark imagine you are capturing a portion of the traffic in one of the routers in your network. Use Python to analyze the performances of the network and the factors that influence its functionality. [50%]

After capturing a portion of the traffic on one laptop (not in the router) we assumed that it represents a portion of the network. After accessing multiple websites we then saved the data into a *.pcapng* file and used the following *tshark* command to extract the data into a *.csv* file with an additional column for ACK Numbers. Code available in Appendix C.1

```
tshark -r PATH_PCAP_FILE -T fields -e frame.number -e frame.time_epoch -e
    frame.len -e frame.protocols -e eth.src -e eth.dst -e ip.src -e
    ip.dst -e ip.proto -e ip.len -e ip.id -e tcp.srcport -e tcp.dstport
    -e udp.srcport -e udp.dstport -e tcp.flags -e dns.qry.name -e
    dns.resp.name -e http.request.method -e tcp.ack -E header=n -E
    separator=, -E quote=d -E occurrence=f > ACK.csv
```

File available for download here:

```
https://drive.google.com/file/d/1WaUmVLKlCMb41QBkGN9ppjPRc6l5XoOZ/view?usp=sharing
```

3.3.1 Analysing the distribution of traffic (TCP vs. UDP)

These protocols have been the backbone of the Internet since the 80s. We have seen limited adoption of new protocols because of the rigidity, or ossification, of the internet's architecture. This is a growing concern because the Internet's historical changes have been slow and gradual [31]. `WhytheInternetonlyjustworksFile`. Therefore it is important to analyse TCP and UDP usage because they massively influence the network's functionality.

As seen in Figure 3.4 the majority of packets were sent using the TCP protocol (82.1% compared to a 17.9%). TCP (Transmission Control Prtocol) is mostly used for general web browsing, emailing, and for streaming pre-recorded content on sites like Netflix. TCP is connection-orientated, flow controlled and accumulates ACKs (Acknowledgements). UDP (User Datagram Potocol) is a connectionless protocol, with no handshaking between UDP sender and receiver. Unlike TCP, UDP has no congestion control, and can therefore function when the network service is compromised and exchange segments quickly although sometimes out of order. UDP is mostly used for streaming multimedia apps, online gaming or DNS. For our case, this is a good balance between UDP and TCP as TCP might be a bit slower but more reliable.

3.3.2 Analysisng TCP Fast Retransmissions, TCP Duplicate ACKS, and ACK RTT

In order to analyse the functionality of the TCP protocol which is the majority of the network packet transfers, we have to look at the nr of duplicate ACKs and how many Fast Retransmissions there were. Duplicate ACKs occur when the same ACK number is seen and it is lower than the last byte of data sent by the sender. If the receiver detects a gap in the sequence numbers, it will generate a duplicate ACK for each subsequent packet it receives on that connection, until the missing packet is successfully received (retransmitted). This is a clear indication of dropped/missing packets [32]. TCP Fast Retransmission occurs when the sender retransmits a packet before the expiration of the acknowledgement timer. Senders receive some packets which sequence number are bigger than the acknowledged packets. Senders should Fast Retransmit upon receipt of 3 duplicate

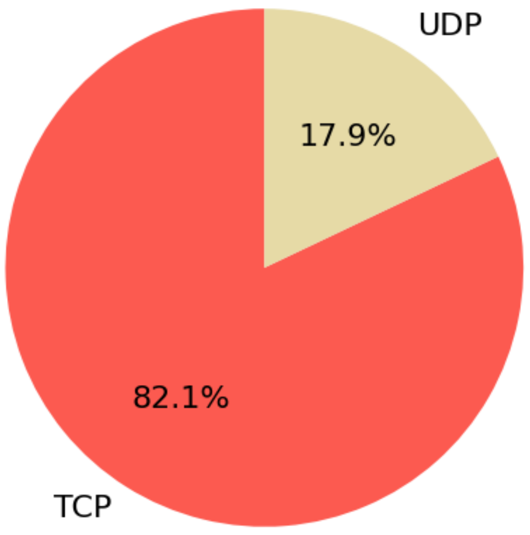


Figure 3.4: Pie Chart of the percentage of number of transmissions made over UDP and TCP protocols in the Transport layer. Code available in Appendix C.1

ACKs. It is likely that the segment is lost and therefore it shouldn't wait for timeout [33]. Lastly, we also accounted for ACK Round-trip time (RTT).

Code available in Appendix C.3 To get this data we proceeded to do another Wireshark capture for 150second where we accessed Russian websites during the capture and we used the following filters to analyse the TCP protocol:

```
    tcp.analysis.fast_retransmission
    tcp.analysis.duplicate_ack
    tcp.analysis.ack_rtt
```

Then we created a dataframe from the copied numbers and produced the results seen above.

During 150 seconds of web browsing, there were:

```
TCP Fast Retransmissions: 16
TCP Duplicate ACKs: 48
Average ACK Round-trip time (RTT): 105.8993710691824
Total nr of Packets: 63506
```

It is clear from the analysis that a small number of packets were lost and had to be re-transmitted as seen for example, at the 70s - 80s markers in Figure 3.5. Compared to the complete

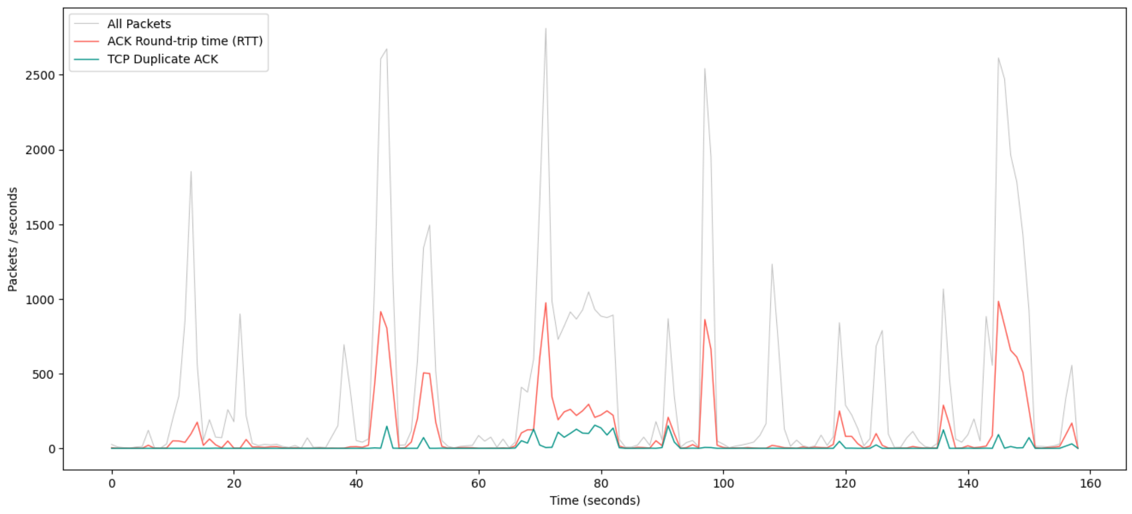


Figure 3.5: TCP Analysis from 150 seconds of captured data in Wireshark. Code available in Appendix C.3

traffic of all packets, there was a small amount of retransmissions and therefore not a lot of lost segments.

The Fast Retransmission and Duplicate ACKs figures make sense because for every 3 duplicate ACKs, there was one fast retransmission. This is typical for TCP protocols and shows that the network's TCP congestion control functions normally.

Average ACK RTT could have been useful if Wireshark calculated it correctly. For a future analysis the filter `tcp.analysis.ack_rtt` should not be used for calculating ACK RTT. We therefore calculated the RTT separately [34]. Code available in Appendix C.2

```
Average RTT: 0.009181104617253065
```

For our scenario this network functionality is considered standard and can be implemented for the proposed topology. However, the assumptions made have to be taken into consideration. This was not an analysis of a router, it was from a laptop and only tracked the client side. For specific bottleneck links between the source and destination TCP CUBIC could be used. This increases TCP's sending rate until packet loss occurs at some router's output and that is useful because we can focus on the congested bottleneck link and understand the congestion.

4
Part 4 - Data Analytics

Contents

Consider an AI-assisted scenario centred around predictive maintenance for water pumps. In this context, you've been given access to a dataset (sensor.csv) containing information from 52 distinct sensors, along with timestamps and the water pump's status.

4.1 Data Visualization. [20%]

4.1.1 Create visualisations illustrating the variation of each sensor's value over time.

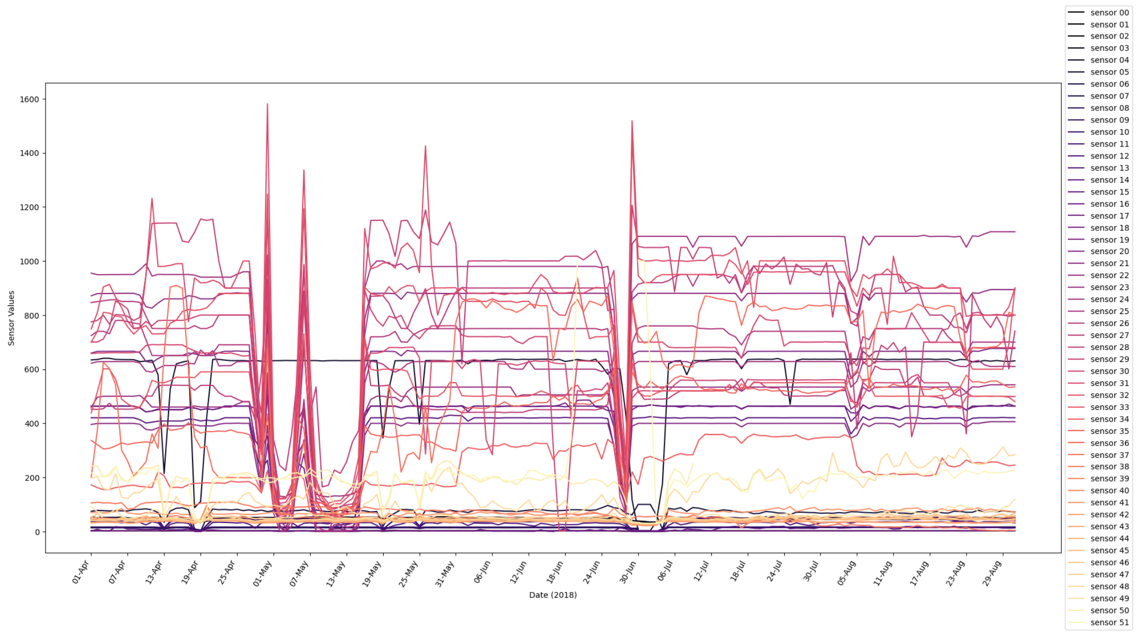


Figure 4.1: Each sensor's mean daily average value over time. Code in Appendix D.1

4.1.2 Generate a count plot displaying the quantity of the unique labels of the machine status. What insights can you derive from the histogram?

From Figure 4.2 we can see that only 7 water pumps were labelled with the status broken, and 14477 were recovering. Out of all the 220320 recordings, 93.4% were functioning normally, which shows that a good part of all readings were water pump sensors that were registered as functioning normally.

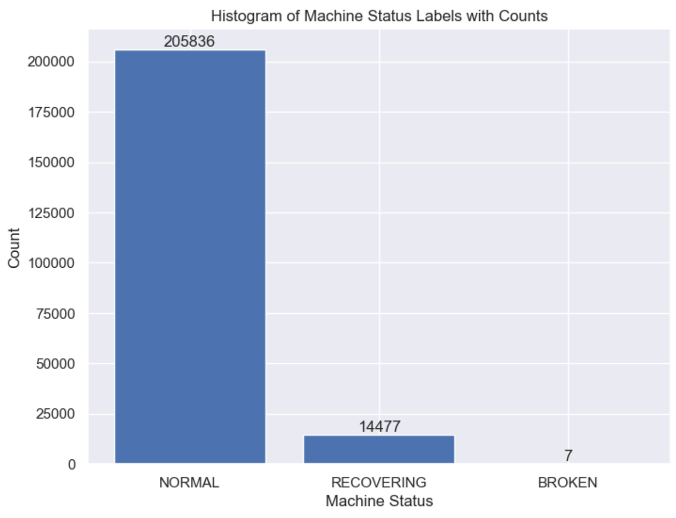


Figure 4.2: Quantity of unique labels of the machine status. Code in Appendix D.2

4.2 Data Exploration: [40%]

4.2.1 Plot the Pearson correlation of the data with a correlation coefficient greater than 0.9. What insights we can derive based on the produced results and task A.a? Is it possible to group any of the sensor data together? If yes, could you provide an example of such a group?

The results from Figure 4.1, 4.3, 4.4 show that a big chunk of sensors have a strong positive correlation with other sensors. This could mean that multiple sensors could have been placed on one water pump such and `sensor 19` and `sensor 20`. It could be possible to group sensor data together such as sensors having a strong positive Pearson Correlation Coefficient of 0.96. Figure 4.4 would show that would be the sensors where the correlation coefficients are red-toned. We would however suggest against that grouping because it could also be because there are many water pumps in one area which causes the same reading fo a cluster of consecutive clusters [35].

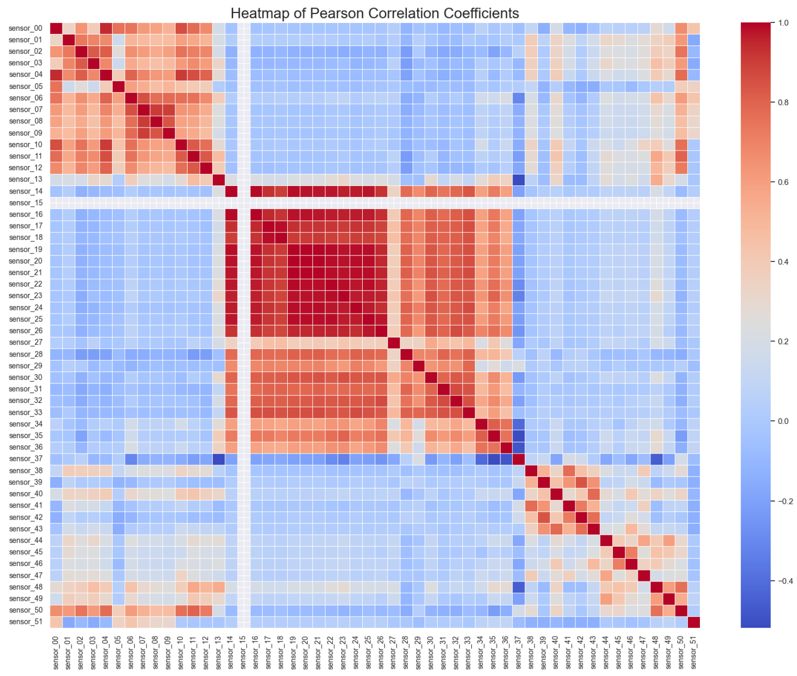


Figure 4.3: Overall Pearson Correlation. Code in Appendix D.3

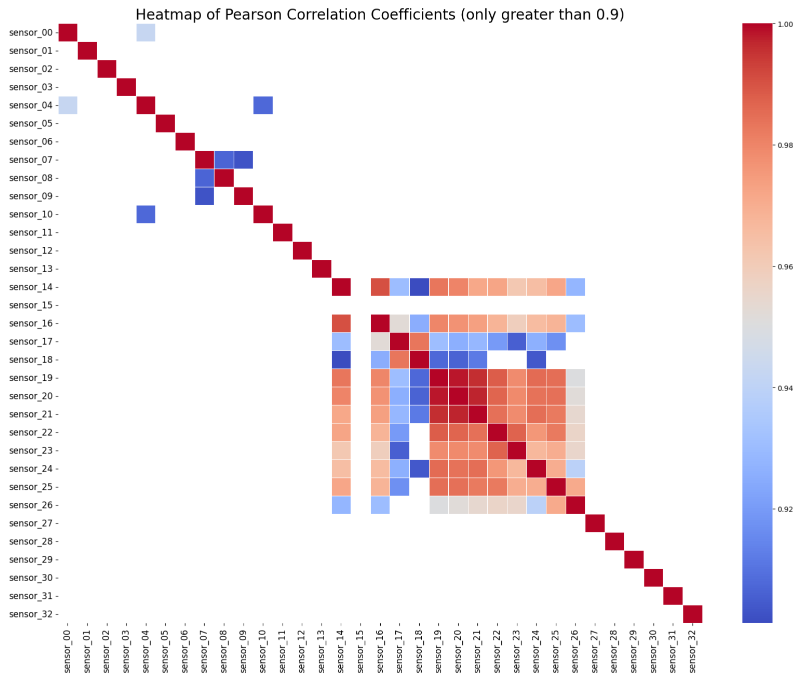
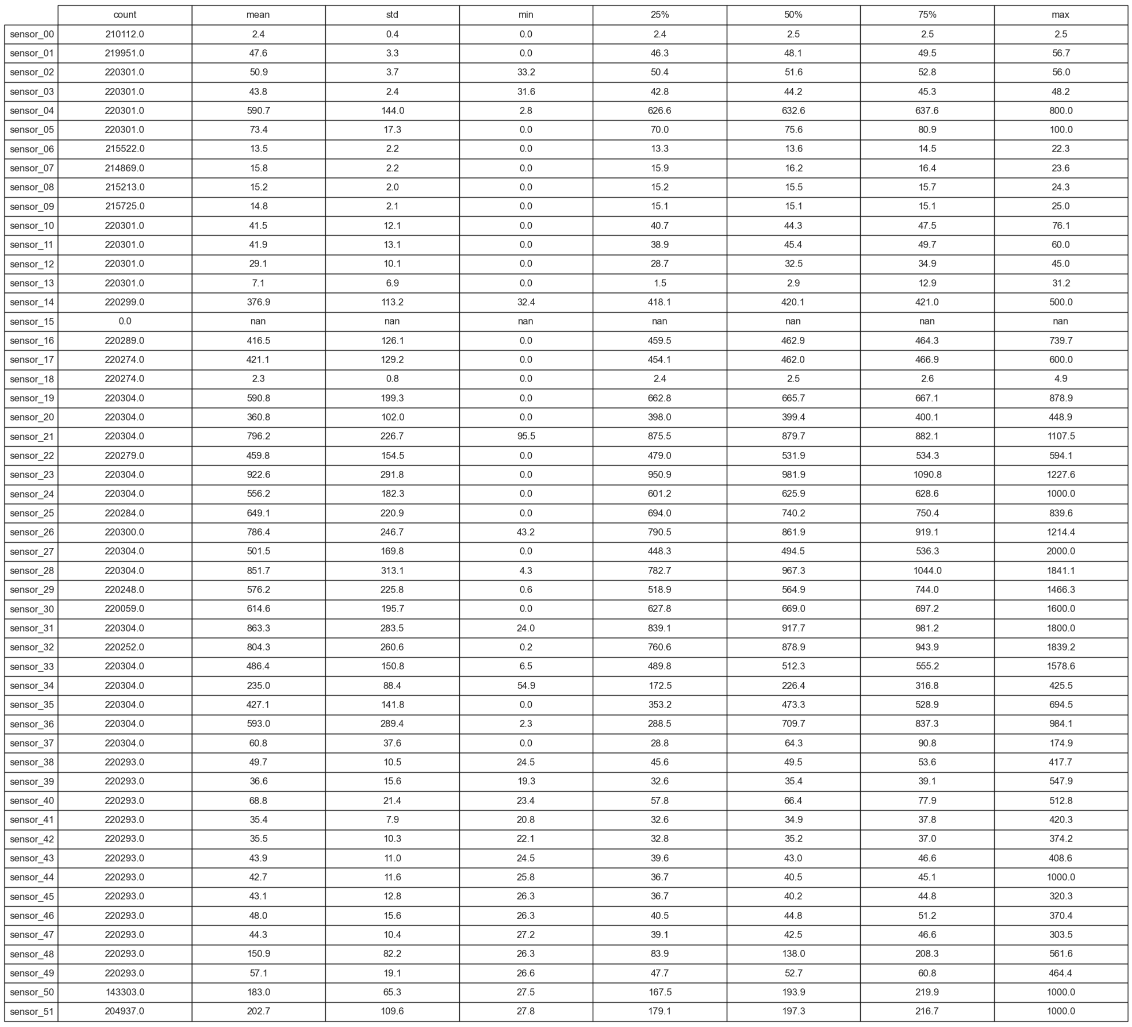


Figure 4.4: Pearson Correlation greater than 0.9. Code in Appendix D.4



	count	mean	std	min	25%	50%	75%	max
sensor_00	210112.0	2.4	0.4	0.0	2.4	2.5	2.5	2.5
sensor_01	219951.0	47.6	3.3	0.0	46.3	48.1	49.5	56.7
sensor_02	220301.0	50.9	3.7	33.2	50.4	51.6	52.8	56.0
sensor_03	220301.0	43.8	2.4	31.6	42.8	44.2	45.3	48.2
sensor_04	220301.0	590.7	144.0	2.8	626.6	632.6	637.6	800.0
sensor_05	220301.0	73.4	17.3	0.0	70.0	75.6	80.9	100.0
sensor_06	215522.0	13.5	2.2	0.0	13.3	13.6	14.5	22.3
sensor_07	214869.0	15.8	2.2	0.0	15.9	16.2	16.4	23.6
sensor_08	215213.0	15.2	2.0	0.0	15.2	15.5	15.7	24.3
sensor_09	215725.0	14.8	2.1	0.0	15.1	15.1	15.1	25.0
sensor_10	220301.0	41.5	12.1	0.0	40.7	44.3	47.5	76.1
sensor_11	220301.0	41.9	13.1	0.0	38.9	45.4	49.7	60.0
sensor_12	220301.0	29.1	10.1	0.0	28.7	32.5	34.9	45.0
sensor_13	220301.0	7.1	6.9	0.0	1.5	2.9	12.9	31.2
sensor_14	220299.0	376.9	113.2	32.4	418.1	420.1	421.0	500.0
sensor_15	0.0	nan	nan	nan	nan	nan	nan	nan
sensor_16	220289.0	416.5	126.1	0.0	459.5	462.9	464.3	739.7
sensor_17	220274.0	421.1	129.2	0.0	454.1	462.0	466.9	600.0
sensor_18	220274.0	2.3	0.8	0.0	2.4	2.5	2.6	4.9
sensor_19	220304.0	590.8	199.3	0.0	662.8	665.7	667.1	878.9
sensor_20	220304.0	360.8	102.0	0.0	398.0	399.4	400.1	448.9
sensor_21	220304.0	796.2	226.7	95.5	875.5	879.7	882.1	1107.5
sensor_22	220279.0	459.8	154.5	0.0	479.0	531.9	534.3	594.1
sensor_23	220304.0	922.6	291.8	0.0	950.9	981.9	1090.8	1227.6
sensor_24	220304.0	556.2	182.3	0.0	601.2	625.9	628.6	1000.0
sensor_25	220284.0	649.1	220.9	0.0	694.0	740.2	750.4	839.6
sensor_26	220300.0	786.4	246.7	43.2	790.5	861.9	919.1	1214.4
sensor_27	220304.0	501.5	169.8	0.0	448.3	494.5	536.3	2000.0
sensor_28	220304.0	851.7	313.1	4.3	782.7	967.3	1044.0	1841.1
sensor_29	220248.0	576.2	225.8	0.6	518.9	564.9	744.0	1466.3
sensor_30	220059.0	614.6	195.7	0.0	627.8	669.0	697.2	1600.0
sensor_31	220304.0	863.3	283.5	24.0	839.1	917.7	981.2	1800.0
sensor_32	220252.0	804.3	260.6	0.2	760.6	878.9	943.9	1839.2
sensor_33	220304.0	486.4	150.8	6.5	489.8	512.3	555.2	1578.6
sensor_34	220304.0	235.0	88.4	54.9	172.5	226.4	316.8	425.5
sensor_35	220304.0	427.1	141.8	0.0	353.2	473.3	528.9	694.5
sensor_36	220304.0	593.0	289.4	2.3	288.5	709.7	837.3	984.1
sensor_37	220304.0	60.8	37.6	0.0	28.8	64.3	90.8	174.9
sensor_38	220293.0	49.7	10.5	24.5	45.6	49.5	53.6	417.7
sensor_39	220293.0	36.6	15.6	19.3	32.6	35.4	39.1	547.9
sensor_40	220293.0	68.8	21.4	23.4	57.8	66.4	77.9	512.8
sensor_41	220293.0	35.4	7.9	20.8	32.6	34.9	37.8	420.3
sensor_42	220293.0	35.5	10.3	22.1	32.8	35.2	37.0	374.2
sensor_43	220293.0	43.9	11.0	24.5	39.6	43.0	46.6	408.6
sensor_44	220293.0	42.7	11.6	25.8	36.7	40.5	45.1	1000.0
sensor_45	220293.0	43.1	12.8	26.3	36.7	40.2	44.8	320.3
sensor_46	220293.0	48.0	15.6	26.3	40.5	44.8	51.2	370.4
sensor_47	220293.0	44.3	10.4	27.2	39.1	42.5	46.6	303.5
sensor_48	220293.0	150.9	82.2	26.3	83.9	138.0	208.3	561.6
sensor_49	220293.0	57.1	19.1	26.6	47.7	52.7	60.8	464.4
sensor_50	143303.0	183.0	65.3	27.5	167.5	193.9	219.9	1000.0
sensor_51	204937.0	202.7	109.6	27.8	179.1	197.3	216.7	1000.0

Figure 4.5: Descriptive statistics, for each sensor. Code in Appendix D.5

4.2.2 Produce a table containing descriptive statistics, summarizing the central tendency, dispersion and shape of a dataset's distribution, for the sensor data.

4.2.3 Compute the duration, in terms of the number of days, for which the data was collected.

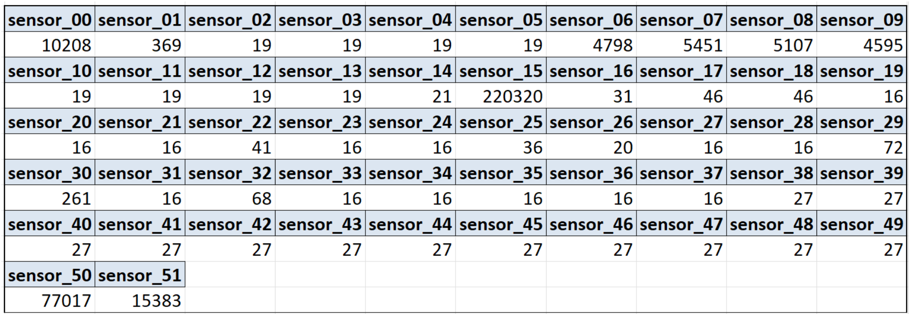
Code in Appendix D.6

```
Number of days: 152
```

4.3 Data Pre-processing: [40%]

4.3.1 Identify and count the number of null values per attribute, then remove entries with null values.

Null values for rows and columns were removed with code seen in Appendix D.7. Count of null values can be seen in Figure 4.6



sensor_00	sensor_01	sensor_02	sensor_03	sensor_04	sensor_05	sensor_06	sensor_07	sensor_08	sensor_09
10208	369	19	19	19	19	4798	5451	5107	4595
sensor_10	sensor_11	sensor_12	sensor_13	sensor_14	sensor_15	sensor_16	sensor_17	sensor_18	sensor_19
19	19	19	19	21	220320	31	46	46	16
sensor_20	sensor_21	sensor_22	sensor_23	sensor_24	sensor_25	sensor_26	sensor_27	sensor_28	sensor_29
16	16	41	16	16	36	20	16	16	72
sensor_30	sensor_31	sensor_32	sensor_33	sensor_34	sensor_35	sensor_36	sensor_37	sensor_38	sensor_39
261	16	68	16	16	16	16	16	27	27
sensor_40	sensor_41	sensor_42	sensor_43	sensor_44	sensor_45	sensor_46	sensor_47	sensor_48	sensor_49
27	27	27	27	27	27	27	27	27	27
sensor_50	sensor_51								
77017	15383								

Figure 4.6: Count of Null Values for each sensor. Code in Appendix D.7

4.3.2 Identify and count any duplicated entries and remove them from the dataset.

Code in Appendix D.8

```
    No duplicate rows based on timestamp
```

4.3.3 Encode the data in the machine status column.

"NORMAL" was encoded to 0, "BROKEN" to 1 and "RECOVERING" to 2. Code below is part of Appendix D.9

```
# encoding words to categorical number
cleaned_df['machine_status'] = cleaned_df['machine_status'].replace('NORMAL',0)
cleaned_df['machine_status'] = cleaned_df['machine_status'].replace('BROKEN',1)
cleaned_df['machine_status'] =
    cleaned_df['machine_status'].replace('RECOVERING',2)

#Print data frame to check if words were encoded
```

```
cleaned_df
```

4.3.4 Determine the data types of the sensor data, and normalise the relevant input features.

Data types were determined with code in Appendix D.10 The command `cleaned_df.dtypes` was used to get the types of all data.

```
Unnamed: 0          int64
timestamp          object
sensor_00         float64
sensor_01         float64
sensor_02         float64
sensor_03         float64
sensor_04         float64
sensor_05         float64
sensor_06         float64
sensor_07         float64
sensor_08         float64
sensor_09         float64
sensor_10         float64
sensor_11         float64
sensor_12         float64
sensor_13         float64
sensor_14         float64
sensor_16         float64
sensor_17         float64
sensor_18         float64
sensor_19         float64
sensor_20         float64
sensor_21         float64
sensor_22         float64
sensor_23         float64
...
sensor_49         float64
sensor_50         float64
```

```
sensor_51          float64
machine_status       int64
dtype: object
```

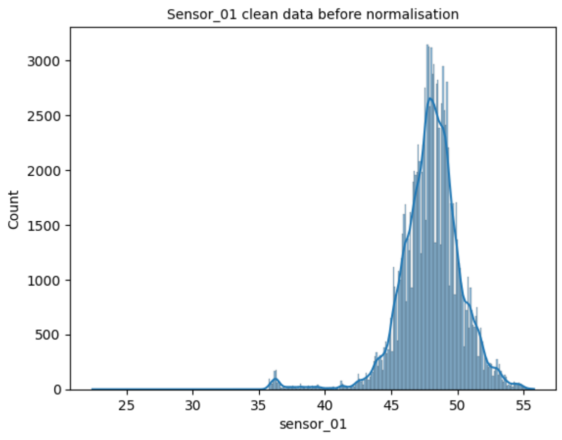


Figure 4.7: Sensor 1 before min-max normalisation. Code in Appendix D.10

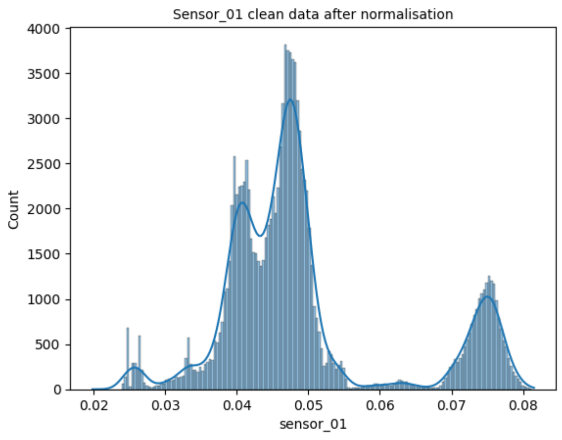


Figure 4.8: Sensor 1 after min-max normalisation. Code in Appendix D.10

Figures 4.7 and 4.8. show how the normalisation affected the data of one sensor. The min max normalisation was applied to every sensor because unlike Z-score, Min-Max normalisation guarantees all features will have the exact same scale. This come at the price of Min-Max normalisation not being able to handle outliers well. We want the exact same scale for every sensor to be able to then compare the water pumps, based on location of the sensors and variance [36].

A Code Appendix Part 1

A.1 The impact on slotted Aloha‘s performance with different packet arrival rates.

```
import random
import pandas as pd
import matplotlib.pyplot as plt



TSLOTS = 100000 # Define the total number of time slots for simulation
SLOT_SIZE = 1  # Assuming a fixed slot size for simplicity

class classNode:
    def __init__(self, ttl, arrival_rate): # Time-to-live for a packet in the
        node
        self.ttl = ttl
        self.queue = 0  # Queue to store incoming packets
        self.arrival_rate = arrival_rate

    def tick(self):  # Simulate a time slot passing and decrease ttl
        self.ttl -= 1
        if random.random() < self.arrival_rate: # Simulate packet arrival
```

```
            self.queue += 1


def main():
    random.seed()
    for window_size in [8]: # One window size
        for arrival_rate in [0.016, 0.216, 0.416, 0.616, 0.816, 1]:  #
           Different packet arrival rates
            Nlist, selist, throughput_list = [], [], []

            for N in range(10, 100):
                snode = [classNode(random.randrange(0, window_size),
                    arrival_rate) for _ in range(N)]
                successful_slots = 0

                for slot in range(TSLOTS): # Simulate each time slot
                    transmitted_nodes = []
                    for i in range(N):
                        if snode[i].queue > 0 and not snode[i].ttl:
                            transmitted_nodes.append(i)
                            snode[i].queue -= 1
                            snode[i].ttl = random.randrange(0, window_size)
                        else:
                            snode[i].tick()

                    if len(transmitted_nodes) == 1:
                        successful_slots += 1

                    if len(transmitted_nodes) > 1:
                        for j in transmitted_nodes:
                            snode[j].ttl = random.randrange(0, window_size)

                slot_efficiency = successful_slots / float(TSLOTS)
                throughput = successful_slots * SLOT_SIZE / float(TSLOTS)

                print(f"Window: {window_size}, Arrival Rate: {arrival_rate}, N
                    = {N}: Efficiency: {slot_efficiency:.3f}, Throughput:
                    {throughput:.3f}")
                Nlist.append(N)
                selist.append(slot_efficiency)
                throughput_list.append(throughput)

            # Plotting results
            plt.plot(Nlist, selist, label=f"W = {window_size}, A = {arrival_rate}")

    plt.xlabel("# of Nodes")
    plt.ylabel("Slot Efficiency")
    plt.legend(loc='upper right')
    plt.axis([0, 64, 0, 0.5])
    plt.title('Slotted ALOHA Efficiency with Variable Arrival Rates')
    plt.grid(linestyle='--')
    plt.show()

if __name__ == "__main__":
    main()
```

A.2 Adjusted slot size

```
import random
import pandas as pd
import matplotlib.pyplot as plt


TSLOTS = 100000 # Define the total number of time slots for simulation
SLOT_SIZE = 1  # Assuming a fixed slot size for simplicity

class classNode:
    def __init__(self, ttl, arrival_rate): # Time-to-live for a packet in the node
        self.ttl = ttl
        self.queue = 0  # Queue to store incoming packets
        self.arrival_rate = arrival_rate

    def tick(self):  # Simulate a time slot passing and decrease ttl
        self.ttl -= 1
        if random.random() < self.arrival_rate: # Simulate packet arrival
            self.queue += 1

def main():
    random.seed()
    for window_size in [8, 16, 32, 64, 128]: # Different window sizes
        for arrival_rate in [0.816]:  # Same packet arrival rates
            Nlist, selist, throughput_list = [], [], []

            for N in range(1, 64):
                snode = [classNode(random.randrange(0, window_size),
                    arrival_rate) for _ in range(N)]
                successful_slots = 0

                for slot in range(TSLOTS): # Simulate each time slot
                    transmitted_nodes = []
                    for i in range(N):
                        if snode[i].queue > 0 and not snode[i].ttl:
                            transmitted_nodes.append(i)
                            snode[i].queue -= 1
                            snode[i].ttl = random.randrange(0, window_size)
                        else:
                            snode[i].tick()

                    if len(transmitted_nodes) == 1:
                        successful_slots += 1

                    if len(transmitted_nodes) > 1:
                        for j in transmitted_nodes:
                            snode[j].ttl = random.randrange(0, window_size)

                slot_efficiency = successful_slots / float(TSLOTS)
                throughput = successful_slots * SLOT_SIZE / float(TSLOTS)
                print(f"Window: {window_size}, Arrival Rate: {arrival_rate}, N = {N}: Efficiency: {slot_efficiency:.3f}, Throughput: {throughput:.3f}")

                Nlist.append(N)
                selist.append(slot_efficiency)
                throughput_list.append(throughput)

            # Plotting results
            plt.plot(Nlist, selist, label=f"W = {window_size}, A = {arrival_rate}")

    plt.xlabel("# of Nodes")
    plt.ylabel("Slot Efficiency")
    plt.legend(loc='upper right')
    plt.axis([0, 64, 0, 0.5])
    plt.title('Slotted ALOHA Efficiency with Variable Arrival Rates')
    plt.grid(linestyle='--')
    plt.show()

if __name__ == "__main__":
    main()
```

B
Code Appendix Part 2

B.1 Average Bit Error Rate

```
%matplotlib inline
!pip install pyphysim

import math
import sys

import numpy as np
from matplotlib import pyplot as plt

# PyPhySim is a Python library for simulating the physical layer of digital
    communications
from pyphysim import channels
from pyphysim.channels.fading import COST259_RAx, TdlChannel
from pyphysim.channels.fading_generators import JakesSampleGenerator
from pyphysim.modulators import OFDM, PSK, QAM
from pyphysim.modulators.ofdm import OfdmOneTapEqualizer
from pyphysim.util.misc import randn_c
from pyphysim.util.conversion import linear2dB

#Create list for all 60 BER
BER_list = []
```

```
for _ in range(60):
 # Loop through 100 times

  M = 16  # Size of the modulation constelation

  # Bandwidth that we can use for transmitting
  bandwidth = 2.4e7   # in Hz

  # Parameters for simulating a real channel
  noise_var = 1e-1 # Noise that we will have in our channel
  Fd = 10  # Doppler frequency (in Hz)
  Ts = 1. / bandwidth  # Sampling interval

  #Bit Rate and Symbol Rate
  BR= 10e6
  SR = BR / np.log2(M)
  num_used_subcarriers = 600
  subcarrierspacing= SR / num_used_subcarriers
  fft_size = int(bandwidth / subcarrierspacing)
  fft_size = 2**int (np.ceil(np.log2(fft_size))) # Ensure it's a power of 2
  # cp_size= int(fft_size / 10) # Adjust according to your requirements

  num_ofdm_symbols = 10
  num_symbols = num_ofdm_symbols * num_used_subcarriers

  cp_size = 10  # Size of the OFDM cyclic interval (in samples)

  qam = QAM(M)
  ofdm = OFDM(fft_size, cp_size, num_used_subcarriers)

  # Generate some random data to modulate with QAM
  data = np.random.randint(0, M, num_symbols)

  # Modulate the data
  qam_symbols = qam.modulate(data)
```

```
# OFDM Modulate the QAM symbols
ofdm_symbols = ofdm.modulate(qam_symbols)
jakesObj = JakesSampleGenerator(Fd, Ts, L=20)

# Creates the tapped delay line (TDL) channel model
tdlchannel = TdlChannel(jakesObj, COST259_RAx)

# Transmit the ofdm modulated signal through the TDL channel
received_ofdm_symbols = tdlchannel.corrupt_data(ofdm_symbols)

# Add random white noise
received_ofdm_symbols += math.sqrt(noise_var) * randn_c(
    received_ofdm_symbols.size)

# OFDM Demodulate received data (the last samples corresponding
# only to channel memory are removed)
ofdm_demodulated_data = ofdm.demodulate(
    received_ofdm_symbols[0:ofdm_symbols.size])

# Reshape the demodulated data to make it easy to extract samples for each
    individual OFDM symbol
ofdm_demodulated_data = np.reshape(ofdm_demodulated_data,
                                   [-1, num_used_subcarriers],
                                   order='C')

received_ofdm_symbol1 = ofdm_demodulated_data[0]

ofdm_equalizer = OfdmOneTapEqualizer(ofdm)

# Impulse response of the channel
# The impulse response of a channel represents how a communication channel
    or system responds to an impulse or a delta function input.
# In other words, it characterizes how the channel distorts or modifies a
    signal as it passes through.
impulse_response = tdlchannel.get_last_impulse_response()
```

```
  equalized_ofdm_demodulated_data = ofdm_equalizer.equalize_data(
      ofdm_demodulated_data, impulse_response)

  received_data = qam.demodulate(equalized_ofdm_demodulated_data)
  ser = 1 - np.sum(data == received_data) / data.size
  # print("Symbol Error Rate: {0}".format(ser))

  BER = ser / np.log2(M)
  BER_list.append(BER)
# print("Bit Error Rate: {0}".format(BER))

  sum(BER_list)/60
```

B.2 QAM and OFDM with and without noise

```
        %matplotlib inline
!pip install pyphysim
import math
import sys

import numpy as np
from matplotlib import pyplot as plt

# PyPhySim is a Python library for simulating the physical layer of digital
    communications
from pyphysim import channels
from pyphysim.channels.fading import COST259_RAx, TdlChannel
from pyphysim.channels.fading_generators import JakesSampleGenerator
from pyphysim.modulators import OFDM, PSK, QAM
from pyphysim.modulators.ofdm import OfdmOneTapEqualizer
from pyphysim.util.misc import randn_c
from pyphysim.util.conversion import linear2dB

################################################################################
# We want a 16-QAM, so 16 QAM symbols
# One symbol will contain four bits
M = 16  # Size of the modulation constelation

#Bandwidth that we can use for transmitting
bandwidth = 2.4e7 # in Hz

# Parameters for simulating a real channel
noise_var = 1e-1 # Noise that we will have in our channel
Fd = 10 # Doppler frequency (in Hz)
Ts =  1. / bandwidth # Sampling interval

BR= 10e6

SR = BR / np.log2(M)

# OFDM parameters
# fft_size = 1048 # Size of the Fast Fourier Transform
# The choice of the FFT size is based on various factors, including the
    available bandwidth, the need for subcarrier spacing,
# and the desired trade-off between spectral efficiency and resistance to
    frequency-selective fading. Smaller FFT sizes offer
#better frequency resolution but may be more susceptible to multipath
    interference, while larger FFT sizes provide greater robustness
# to channel conditions but may have a lower spectral efficiency.
num_used_subcarriers = 600

# The choice of the number of subcarriers depends on the specific use case,
    available spectrum, and the trade-offs between spectral
# efficiency, resilience to channel impairments, and system complexity.
    Smaller numbers of subcarriers may provide simplicity but may
# not utilize the available bandwidth efficiently. Larger numbers of
    subcarriers can provide higher data rates but may require more
# processing and introduce challenges in handling frequency-selective fading
subcarrierspacing= SR / num_used_subcarriers
fft_size = int(bandwidth / subcarrierspacing)
fft_size = 2**int (np.ceil(np.log2(fft_size))) # Ensure it's a power of 2
#cp_size= int(fft_size / 10) # Adjust according to your requirements
```

```
print (fft_size)
# OFDM Symbols (unit of data transmission)
num_ofdm_symbols = 10

# Number of QAM symbols that will be generated
num_symbols = num_ofdm_symbols * num_used_subcarriers

cp_size = 10 # Size of the OFDM cyclic interval (in samples)
print(cp_size)
# The CP (Cyclic Prefix)is a copy of the last part of an OFDM symbol, which is
    prefixed to the beginning of the symbol.
# It helps in mitigating mulipath interference, syncronization and
    Inter-Symbol Interference

################################################################################
# Creates the required objects
qam = QAM(M)
ofdm = OFDM(fft_size, cp_size, num_used_subcarriers)

# Generate some random data to modulate with QAM
data = np.random.randint(0, M, num_symbols)

# Modulate the data
qam_symbols = qam.modulate(data)

# OFDM Modulate the QAM symbols
ofdm_symbols = ofdm.modulate(qam_symbols)

################################### PLOT
    #######################################
fig, ax = plt.subplots(figsize=(6, 6))
ax.plot(np.real(qam_symbols), np.imag(qam_symbols), 'r*')
ax.set_title('QAM symbols')
ax.axis("equal")
```

```
################################### PLOT
    #######################################
fig2, ax2 = plt.subplots(figsize=(8, 8))
ax2.plot(np.real(ofdm_symbols), np.imag(ofdm_symbols), 'r*')
ax2.axis('equal')


################################################################################
# Create a jakes object with 20 rays. This is the fading model that controls
    how the channel vary in time.
# In wireless communications, the Jakes model is a mathematical model that
    simulates the behavior of a multipath fading channel.
# It's often used in simulations to mimic the effects of signal propagation in
    a real-world environment, where signals can
# experience fading due to reflections, diffraction, and scattering. The 20
    rays represent individual signal paths.
# This will be passed to the TDL channel object.
jakesObj = JakesSampleGenerator(Fd, Ts, L=20)

# Creates the tapped delay line (TDL) channel model
tdlchannel = TdlChannel(jakesObj, COST259_RAx)

# Transmit the ofdm modulated signal through the TDL channel
received_ofdm_symbols = tdlchannel.corrupt_data(ofdm_symbols)

# Add random white noise
received_ofdm_symbols += math.sqrt(noise_var) * randn_c(
    received_ofdm_symbols.size)

# OFDM Demodulate received data (the last samples corresponding
# only to channel memory are removed)
ofdm_demodulated_data = ofdm.demodulate(
    received_ofdm_symbols[0:ofdm_symbols.size])

# Reshape the demodulated data to make it easy to extract samples for each
    individual OFDM symbol
ofdm_demodulated_data = np.reshape(ofdm_demodulated_data,
                                    [-1, num_used_subcarriers],
                                    order='C')

received_ofdm_symbol1 = ofdm_demodulated_data[0]

################################### PLOT
    #######################################
fig3, ax3 = plt.subplots(figsize=(12, 8))

ax3.plot(np.real(received_ofdm_symbol1), np.imag(received_ofdm_symbol1), 'r*')
ax3.set_title('First demodulated OFDM symbol')
ax3.axis('equal')



##############################  EQUALIZER
    ####################################
ofdm_equalizer = OfdmOneTapEqualizer(ofdm)

# Impulse response of the channel
# The impulse response of a channel represents how a communication channel or
    system responds to an impulse or a delta function input.
# In other words, it characterizes how the channel distorts or modifies a
    signal as it passes through.
impulse_response = tdlchannel.get_last_impulse_response()



equalized_ofdm_demodulated_data = ofdm_equalizer.equalize_data(
    ofdm_demodulated_data, impulse_response)



#################################### PLOT
    #######################################
plt.figure(figsize=(8, 8))
plt.plot(np.real(equalized_ofdm_demodulated_data),
         np.imag(equalized_ofdm_demodulated_data), 'r*')
plt.title('QAM symbols transmitted in the OFDM symbols')
plt.xlim([-1.15, 1.15])
plt.ylim([-1.15, 1.15])
plt.xlabel('Real part')
plt.ylabel('Imaginary part')
plt.show()

############### Symbol Error Rate and Bit Error Rate ##########################
received_data = qam.demodulate(equalized_ofdm_demodulated_data)
ser = 1 - np.sum(data == received_data) / data.size
print("Symbol Error Rate: {0}".format(ser))

BER = ser / np.log2(M)
print("Bit Error Rate: {0}".format(BER))
```

B.3 Impulse Response Characteristics based in Time and Frequency

```
ofdm_equalizer = OfdmOneTapEqualizer(ofdm)

# Impulse response of the channel
# The impulse response of a channel represents how a communication channel or
    system responds to an impulse or a delta function input.
# In other words, it characterizes how the channel distorts or modifies a
    signal as it passes through.
impulse_response = tdlchannel.get_last_impulse_response()


equalized_ofdm_demodulated_data = ofdm_equalizer.equalize_data(
    ofdm_demodulated_data, impulse_response)

print(impulse_response)

################################ Time based
    ####################################
#Assuming impulse_response is an instance of TdlImpulseResponse
values_time = impulse_response.tap_values_sparse
```

```
#################################### PLOT
    ######################################
plt.figure(figsize=(10,6))
plt.plot(values_time.T)
plt.title('TDL Channel Impulse Response Time Domain')
plt.xlabel('Time (Sec)')
plt.ylabel('Amplitude')
plt.grid(True)
plt.show



################################ Frequency based
    ################################
values_frequency = impulse_response.tap_values_sparse
print(impulse_response)
fft_result = np.fft.fft(impulse_response, axis=0)
# fft_result = np.ravel(impulse_response, axis=0)
frequencies = np.fft.fftfreq(len(values_frequency),
    d=impulse_response.cannel.profile.Ts)

#################################### PLOT
    ######################################
plt.figure(figsize=(10,6))
plt.plot(frequencies.T)
plt.title('TDL Channel Impulse Response Frequency Domain')
plt.xlabel('Time (Sec)')
plt.ylabel('Frequency (Hz)')
plt.grid(True)
plt.show
```

C Code Appendix Part 3

C.1 TCP vs UDP

```
# Importing libraries
import numpy as np
import pandas as pd
from datetime import datetime
import sys
import matplotlib.pyplot as plt
import seaborn as sns
from io import StringIO



# Reading data
traffic = pd.read_csv('/content/ACK.csv',sep='\t')

yicam = pd.read_csv('/content/ACK.csv', delimiter=",") #, names = columns)
yicam

# Setting the column names, and user IP address
dev_ip = "10.97.252.163"
columns =
    ["frame_number","frame_time_epoch","frame_len","frame_protocols","eth_src","eth_dst","ip_sr
    "ACK"]
```

```
df = pd.read_csv('/content/ACK.csv', delimiter=",", names = columns,
    skiprows=[0])
df




################################################################################
# Comparing UDP vs TCP protocols
################################################################################

# check for when tcp_srcport & tcp_dstport are NOT NAN and ACK is NAN
df[
    (~df['tcp_srcport'].isna()) &
    (~df['tcp_dstport'].isna()) &
    (df['ACK'].isna())
    ].shape[0]

# check for when udp_srcport & udp_dstport are NOT NAN and ACK is NOT NAN
df[
    (~df['udp_srcport'].isna()) &
    (~df['udp_dstport'].isna()) &
    (~df['ACK'].isna())
    ].shape[0]

tcp_srcport_num = df.shape[0] - df['tcp_srcport'].isna().sum()
tcp_dstport_num = df.shape[0] - df['tcp_dstport'].isna().sum()

# tcp_srcport_num == tcp_dstport_num
pct_tcp = round((tcp_dstport_num)/df.shape[0]*100,2)
udp_srcport_num = df.shape[0] - df['udp_srcport'].isna().sum()
udp_dstport_num = df.shape[0] - df['udp_dstport'].isna().sum()

# udp_srcport_num == udp_dstport_num
pct_udp = round((udp_dstport_num)/df.shape[0]*100,2)

#################################### PLOT
    ######################################
```

C

```
# Data for plotting
labels = ['TCP', 'UDP']
sizes = [pct_tcp, pct_udp]

colors = ['#FC5A50', '#E6DAA6']  # Professional color palette (blue and orange)
textprops = {"fontsize": 16}


# Create the pie chart
plt.figure(figsize=(8, 6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90, colors=colors,
    textprops=textprops)
plt.title("Percentage of UDP & TCP Protocols of the MAC Layer(%)")
plt.show()


################################################################################
# Measuring Acknowledgments
################################################################################

# Reload the data
data_path = '/content/ACK.csv'
data = pd.read_csv(data_path)

# Access the last column as a Series
last_column = data.iloc[:, -1]

# Initialize the dictionaries
consecutive_starts = {}
consecutive_lengths = {}

# Initialize counters
count = 0
current_number = None

# Count consecutive repetitions
for i in range(len(last_column)):
    if i == 0 or last_column.iloc[i] == current_number:
        count += 1
        current_number = last_column.iloc[i]
    else:
        if count > 1:  # Only consider sequences of length > 1
            consecutive_starts[current_number] =
                consecutive_starts.get(current_number, 0) + 1
            consecutive_lengths[current_number] =
                consecutive_lengths.get(current_number, []) + [count]
        count = 1
        current_number = last_column.iloc[i]

# Check the last element
if count > 1:
    consecutive_starts[current_number] =
        consecutive_starts.get(current_number, 0) + 1
    consecutive_lengths[current_number] =
        consecutive_lengths.get(current_number, []) + [count]

# The 'consecutive_starts' dictionary contains the count of how many times
    each ACK number starts a consecutive sequence.
# The 'consecutive_lengths' dictionary contains the lengths of these sequences
    for each ACK number.
# For example, ACK: 1.0 has 26 sequences which can be seen in the consecutive
    lengths [2, 3, 3, 3, 12, 3, 3, 2, 3, 2, 3, 3, 2, 3, 3, 2, 3, 3, 2, 2, 2,
    2, 3, 3, 3, 4].
print("Consecutive Starts:", consecutive_starts)
print("Consecutive Lengths:", consecutive_lengths)

# Filter the dictionary to include only  ACK numbers with consecutive starts >
    10
filtered_consecutive_starts = {num: starts for num, starts in
    consecutive_starts.items() if starts > 10}

# Extracting the ACK numbers and their respective starts for plotting
filtered_numbers = list(filtered_consecutive_starts.keys())
filtered_starts = list(filtered_consecutive_starts.values())
```

```
##################################### PLOT
    ########################################
plt.figure(figsize=(10, 6))
ax = sns.barplot(x=filtered_numbers, y=filtered_starts, palette='Set2')

# Add values above bars
for bar in ax.patches:
    height = bar.get_height()
    ax.text(bar.get_x() + bar.get_width() / 2., height + 1, int(height),
        ha='center')



plt.xlabel('ACK Number')
plt.ylabel('Count of Consecutive Sequence Starts')
plt.title('Consecutive Sequence Starts Greater than 10 for each ACK Number')
plt.xticks(rotation=0)  # Rotate labels for readability
plt.show()
```

C.2 Average RTT

```
################################################################################
# Calculating RTT
################################################################################

# Load a small sample of the data to understand its structure
file_path = '/content/ACK.csv'

# Reload the data with correct headers
full_data = pd.read_csv(file_path)

# Displaying the first few rows of the reloaded data for verification
full_data.head()

# Column names as provided by the user
```

```
columns = ["frame_number", "frame_time_epoch", "frame_len", "frame_protocols",
    "eth_src", "eth_dst",
           "ip_src", "ip_dst", "ip_proto", "ip_len", "ip_id", "tcp_srcport",
               "tcp_dstport",
           "udp_srcport", "udp_dstport", "tcp_flags", "dns_qry_name",
               "dns_resp_name",
           "http_request_method", "ACK"]

# Reload the data with the specified column names and skipping the first row
network_data = pd.read_csv(file_path, names=columns, skiprows=1)

# Displaying the first few rows of the data for verification
network_data.head()

# # Filter for TCP packets (ip_proto = 6)
# tcp_packets = network_data[network_data['ip_proto'] == 6]



# Filter for TCP packets (ip_proto = 6) and create a copy
tcp_packets = network_data[network_data['ip_proto'] == 6].copy()

# Inspect the first few rows of the filtered TCP packets
tcp_packets.head()

# Function to check if ACK flag is set in tcp_flags
def is_ack_flag_set(tcp_flag_value):
    try:
        # Convert hex string to integer and check if the ACK bit (0x10) is set
        return int(tcp_flag_value, 16) & 0x10 == 0x10
    except:
        # In case of invalid format or NaN values
        return False

# Apply the function to identify ACK packets
tcp_packets['is_ack'] = tcp_packets['tcp_flags'].apply(is_ack_flag_set)
```

```
# Separate ACK packets and non-ACK packets
ack_packets = tcp_packets[tcp_packets['is_ack']]
non_ack_packets = tcp_packets[~tcp_packets['is_ack']]

# Display the first few rows of ACK packets and non-ACK packets for
    verification
ack_packets.head(), non_ack_packets.head()



# Function to calculate round-trip time (RTT)
def calculate_rtt(ack_packets):
    # Sort the packets by frame number
    sorted_ack_packets = ack_packets.sort_values(by='frame_number')

    # Initialize a list to store RTT values
    rtt_values = []

    # Iterate over the sorted packets to calculate RTT
    for i in range(len(sorted_ack_packets) - 1):
        current_packet = sorted_ack_packets.iloc[i]
        next_packet = sorted_ack_packets.iloc[i + 1]

        # Calculate the time difference between consecutive ACK packets
        time_diff = next_packet['frame_time_epoch'] -
            current_packet['frame_time_epoch']
        rtt_values.append(time_diff)

    return rtt_values

# Calculate RTT values for ACK packets
rtt_values = calculate_rtt(ack_packets)

# Compute average RTT
average_rtt = np.mean(rtt_values) if rtt_values else np.nan

# Display the average RTT
print("Average RTT:", average_rtt)
```

C.3 TCP Analysis

```
################################################################################
# TCP Analysis
################################################################################

# Data from Wireshark taken as a string
data_str2 = """
0,4,26,0,0
1,0,9,0,0
2,0,4,0,0
3,0,1,0,0
4,2,9,0,0
5,0,12,0,0
6,21,121,0,0
7,0,4,0,0
8,0,1,0,0
9,2,30,0,0
10,50,197,0,0
11,49,350,0,0
12,40,857,0,0
13,98,1853,0,0
14,175,562,0,0
15,20,53,0,0
16,63,192,0,0
17,24,75,0,1
18,3,71,0,0
19,49,259,0,0
20,1,178,0,0
21,1,900,0,0
22,59,218,0,0
23,9,33,0,0
24,9,19,0,0
```

```
25,6,26,0,0
26,10,23,0,0
27,11,27,0,0
28,3,12,0,0
29,1,6,0,0
30,4,19,0,0
31,0,0,0,0
32,0,70,0,0
33,0,6,0,0
34,2,8,0,0
35,0,5,0,0
36,1,79,0,0
37,0,152,0,0
38,1,694,0,0
39,10,395,0,0
40,11,52,0,0
41,7,41,0,0
42,21,63,0,0
43,425,1085,0,3
44,915,2607,0,1
45,805,2674,1,148
46,399,1125,0,1
47,1,21,0,0
48,4,23,0,0
49,42,116,0,0
50,198,585,0,0
51,505,1342,1,72
52,501,1494,0,0
53,174,514,0,0
54,13,52,0,1
55,1,13,0,0
56,1,3,0,0
57,5,13,0,0
58,5,17,0,0
59,4,20,0,0
60,0,86,0,0
61,0,48,0,0
```

```
62,1,75,0,0
63,0,7,0,0
64,3,62,0,0
65,0,4,0,0
66,17,41,0,2
67,103,409,1,51
68,124,376,0,35
69,124,596,1,128
70,606,1657,0,22
71,974,2811,0,5
72,344,985,0,7
73,191,729,2,108
74,244,819,1,74
75,261,913,0,100
76,220,865,1,128
77,251,926,1,102
78,295,1047,1,100
79,207,929,0,155
80,223,885,1,137
81,251,875,0,89
82,221,892,2,136
83,16,61,0,4
84,2,8,0,0
85,0,8,0,0
86,7,22,0,0
87,5,75,0,0
88,2,20,0,1
89,51,179,0,0
90,18,57,0,5
91,208,868,0,151
92,98,350,2,42
93,0,7,0,0
94,5,39,0,0
95,25,52,0,1
96,4,10,0,0
97,862,2542,0,6
98,663,1952,0,5
```

```
99,21,49,0,0
100,6,28,0,0
101,1,4,0,0
102,3,15,0,0
103,2,22,0,1
104,5,31,0,0
105,2,42,0,0
106,0,87,0,0
107,0,165,0,0
108,20,1233,0,0
109,13,713,0,0
110,4,129,0,0
111,3,14,0,0
112,0,55,0,0
113,9,18,0,0
114,2,7,0,0
115,8,16,0,0
116,5,88,0,0
117,4,20,0,0
118,26,77,0,0
119,250,841,1,47
120,80,288,0,1
121,80,224,0,1
122,31,133,0,0
123,2,17,0,0
124,15,65,0,0
125,99,684,1,23
126,20,789,0,0
127,2,95,0,0
128,0,6,0,0
129,2,7,0,0
130,2,70,0,0
131,12,113,0,0
132,5,47,0,0
133,2,12,0,0
134,2,4,0,0
135,2,33,0,0
```

```
136,289,1067,1,124
137,157,466,0,0
138,0,62,0,0
139,2,41,0,0
140,17,91,0,1
141,4,197,0,0
142,8,49,0,0
143,15,883,0,1
144,85,555,0,0
145,984,2613,3,92
146,822,2472,0,1
147,656,1963,0,12
148,611,1783,0,3
149,509,1429,0,4
150,259,920,0,72
151,5,13,0,0
152,0,13,0,0
153,5,10,0,0
154,7,16,0,0
155,13,29,0,0
156,90,332,0,15
157,169,556,0,30
158,0,2,0,0
"""



# Create a DataFrame from the data string
df = pd.read_csv(StringIO(data_str2), header=None)

# Save the DataFrame to a CSV file
df.to_csv('output.csv2', index=True)



#Defining file
file_path = '/content/output.csv2'

# Reload the data with correct headers
```

```
full_data = pd.read_csv(file_path)

df = pd.DataFrame(full_data)

#Assigning new column names
new_columns = {'0':'Id',
               '1': 'ACK Round-trip time (RTT)',
               '2': 'All Packets',
               '3': 'TCP Fast Retransmission',
               '4': 'TCP Duplicate ACK'}

df.rename(columns=new_columns, inplace=True)

df

# Extract the data
interval = df['Id']
tcp_analysis_ack_rtt = df['ACK Round-trip time (RTT)']
packets = df['All Packets']
tcp_analysis_fast_retransmission = df['TCP Fast Retransmission']
tcp_analysis_duplicate_ack = df['TCP Duplicate ACK']

#################################### PLOT
    ######################################
plt.figure(figsize=(16,7))

# plot lines
plt.plot(interval, packets, color='silver', linewidth=0.7,  label = "All
    Packets")
plt.plot(interval, tcp_analysis_ack_rtt, color='#FC5A50', linewidth=1,
    label="ACK Round-trip time (RTT)")
plt.plot(interval, tcp_analysis_duplicate_ack, color='#029386', linewidth=1,
    label = "TCP Duplicate ACK")
plt.xlabel("Time (seconds)")
plt.ylabel("Packets / seconds")
# #plt.legend(loc='lower right', ncol=1) #
plt.legend()
```

```
################################# Counting
    #####################################
# TCP Fast Retransmission count
count_fast_re = (df['TCP Fast Retransmission'] != 0).sum()
print("TCP Fast Retransmission count:", count_fast_re)

# Counting Duplicate ACKs
count_duplicate_acks = (df['TCP Duplicate ACK'] != 0).sum()
print("TCP Duplicate ACK count:", count_duplicate_acks)

# ACK_Round-trip time_(RTT) average
ack_mean = df["ACK Round-trip time (RTT)"].mean()
print("Average ACK Round-trip time (RTT):", ack_mean)

# TCP Fast Retransmission count
total_packets= (df['All Packets']).sum()
print("Total nr of Packets:", total_packets)
df
```

D
Code Appendix Part 4

D.1 Variation of sensors over time

```
#Importing libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm

#Reading the data
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
df= df.iloc[:,1:-1]

#plot average hourly
test_df = df.copy()                              # make a copy of df
test_df['timestamp'] = pd.to_datetime(test_df['timestamp']) # convert
    timestamp to datetime format
test_df.set_index('timestamp', inplace = True)

hourly_df = test_df.resample('H').mean()         # get hourly mean (so each row
    is per hour)
hourly_df.reset_index(inplace=True)
```

```
daily_df = test_df.resample('D').mean()        # get daily mean (so each row
    is per day)
daily_df.reset_index('timestamp', inplace=True)

daily_df.columns = daily_df.columns.str.replace('_', ' ')
daily_df.columns = daily_df.columns.str.replace('', '')

################################## PLOT ###################################

# Choose a colormap (e.g., 'viridis', 'plasma', 'inferno', 'magma', 'cividis')
colormap = cm.magma

# Generate colors from the colormap
num_lines = len(daily_df.columns) - 1  # Number of lines to plot (excluding
    the timestamp column)
colors = [colormap(i / num_lines) for i in range(num_lines)]

plt.figure(figsize=(20, 11))

# Iterate over the columns to plot each one with a color from the colormap
for idx, column in enumerate(daily_df.columns[1:]):  # Skip the 'timestamp'
    column
    plt.plot(daily_df['timestamp'], daily_df[column], label=column,
        color=colors[idx])

selected_dates = daily_df['timestamp'][::6]
formatted_dates = selected_dates.dt.strftime("%d-%b")

plt.xticks(selected_dates, formatted_dates, rotation=60, ha='right')
plt.xlabel("Date (2018)")
plt.ylabel("Sensor Values")

plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

D.2 Unique Labels Count

```
#Reading the data
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
df

#Looking at the type of data
df['machine_status'].dtype
df.dtypes

#Isolating the last column
status = df. iloc[:,-1:]

# Counting the occurrences of each machine status
status_counts = status['machine_status'].value_counts()

# Plotting the histogram with count labels on each bar
plt.figure(figsize=(8, 6))
bars = plt.bar(status_counts.index, status_counts.values)


# Adding count labels above each bar
for bar in bars:
    yval = bar.get_height()
    plt.text(bar.get_x() + bar.get_width()/2, yval, int(yval), va='bottom',
        ha='center')

################################## PLOT ###################################
plt.title('Histogram of Machine Status Labels with Counts')
plt.xlabel('Machine Status')
plt.ylabel('Count')
plt.xticks(rotation=0)
plt.show()
```

D.3 General Pearson Correlation

```
#Reading the data
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
df

# Calculate the Pearson correlation matrix
correlation_matrix = df.corr()



# Find pairs with a correlation greater than 0.9 (excluding self-correlation)
correlated_pairs = [(i, j) for i in correlation_matrix.columns for j in
    correlation_matrix.columns
                    if (i != j) and (abs(correlation_matrix.loc[i, j]) > 0.9)]

correlated_pairs, len(correlated_pairs)

################################## PLOT ###################################
# Setting up the matplotlib figure
plt.figure(figsize=(20, 15))

# Draw the heatmap
sns.heatmap(correlation_matrix, annot=False, cmap='coolwarm', linewidths=.5)

# Adding title and labels
plt.title('Heatmap of Pearson Correlation Coefficients', fontsize=20)
plt.xticks(fontsize=10)
plt.yticks(fontsize=10)

# Show the plot
plt.show()
```

D.4 Pearson Correlation for 0.9

```
# Load the dataset
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
data= df.iloc[:,2:-20]

# Display the first few rows of the dataset to understand its structure
data.head()



# Calculate the Pearson correlation matrix
correlation_matrix = data.corr()

# Find pairs with a correlation greater than 0.9 (excluding self-correlation)
correlated_pairs = [(i, j) for i in correlation_matrix.columns for j in
    correlation_matrix.columns
                    if (i != j) and (abs(correlation_matrix.loc[i, j]) > 0.9)]

# Print the correlated pairs and their count
print(correlated_pairs, len(correlated_pairs))

################################## PLOT ###################################
# Setting up the matplotlib figure
plt.figure(figsize=(20, 15))

# Modifying the correlation matrix to only show values above 0.9
# Set values below 0.9 to NaN
filtered_corr_matrix = correlation_matrix.where(np.abs(correlation_matrix) >
    0.9, np.nan)

# Draw the heatmap with the filtered matrix
sns.heatmap(filtered_corr_matrix, annot=False, cmap='coolwarm', linewidths=.5)

# Adding title and labels
```

```
plt.title('Heatmap of Pearson Correlation Coefficients (only greater than
    0.9)', fontsize=20)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Show the plot
plt.show()
```

D.5 Descriptive Statistics

```
import pandas as pd
import matplotlib.pyplot as plt



# Load the dataset
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
sensor_data= df.iloc[:,2:-1]

# sensor_data.head()

# Computing descriptive statistics
descriptive_stats = sensor_data.describe()

# Displaying the descriptive statistics
descriptive_stats.T
descriptive_stats.T.head()

# #Export to Excel
# descriptive_stats.to_excel('excel sum stats.xlsx')

# Function to format numbers to one decimal place
def format_to_1dp(x):
    return '{:.1f}'.format(x)
```

```

# Applying the formatting to the summary statistics dataframe
formatted_summary = descriptive_stats.applymap(format_to_1dp)



# Creating a figure to hold the table
fig, ax = plt.subplots(figsize=(12, 4))  # Adjust the size as needed
ax.axis('tight')
ax.axis('off')

# Creating the table
table = ax.table(cellText = formatted_summary.T.values,
                 colLabels = formatted_summary.T.columns,
                 rowLabels = formatted_summary.T.index,
                 cellLoc='center',
                 loc='center')

table.auto_set_font_size(True)
table.set_fontsize(12)
table.scale(2, 2)

# Adjust layout
# plt.tight_layout()



plt.show()
```

D.6 Number of days

```
#Importing libraries
import pandas as pd
from datetime import datetime

#Reading the data
```

```
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)

dates = pd.to_datetime(df['timestamp']).apply(lambda x: x.date())

# Create two datetime objects
date1 = datetime(2018, 4 ,1)
date2 = datetime(2018, 8, 31)

# Calculate the number of days between the two dates
delta = date2 - date1
num_days = delta.days

print(num_days)
```

D.7 Counting null values and removing them

```
import pandas as pd
# Created a test dataframe where all null values are dropped
#Reading the data
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)

df = df.drop('sensor_15', axis=1)          # Remove sensor_15 column because
    all values are null
df.iloc[:,14:19]                           # check sensor 15 is gone
# Exporting to Excel
# df.T.to_excel('Count of null values .xlsx')

test_df = df.dropna()                      # Drop all other rows that have
    null values
test_df.isnull().sum(axis=1)
```

```
test_df                                           # Checks most values are still
    there (119103 rows)

#Checks the number of null values in each row for test_df, converts it to a
    list and checks the sum of the lits.
sum(test_df.isnull().sum(axis=1).tolist())
```

D.8 Counting duplicate entries

```
import pandas as pd

#Reading the data
file_path = r'C:\Users\Ema\Documents\UCL\UCL 3\Networking
    Systems\Data\sensor.csv'
df = pd.read_csv(file_path)
df = df.drop('sensor_15', axis=1)
cleaned_df = df.dropna()
drop_duplicate_df = cleaned_df.drop_duplicates(subset=['timestamp'])

if cleaned_df.shape[0] == drop_duplicate_df.shape[0]:
    print("No duplicate rows based on timestamp")
```

D.9 Encoding the data in the machine status column

```
# encoding words to categorical number
cleaned_df['machine_status'] = cleaned_df['machine_status'].replace('NORMAL',0)
cleaned_df['machine_status'] = cleaned_df['machine_status'].replace('BROKEN',1)
cleaned_df['machine_status'] =
    cleaned_df['machine_status'].replace('RECOVERING',2)


cleaned_df
```

D.10 Data types and normalisation

```
import seaborn as sns
import matplotlib.pyplot as plt

#Printing data types
cleaned_df.dtypes

#Plotting sensor 1 before normalisation
sns.histplot(cleaned_df['sensor_01'], kde=True)
plt.title('Sensor_01 clean data before normalisation', fontsize=10)
plt.show()

#defining the min-max normalisation function
def min_max_normalize_row(row):
    min_value = row.min()
    max_value = row.max()
    return (row - min_value) / (max_value - min_value)

#Checking normalisation worked
cleaned_df.iloc[:,2:-1]

#Creating normalised dataset for every row
normalised_df = cleaned_df.iloc[:,2:-1].apply(min_max_normalize_row, axis=1)
normalised_df
################################## PLOT ###################################
sns.histplot(normalised_df['sensor_01'], kde=True)
plt.title('Sensor_01 clean data after normalisation', fontsize=10)
plt.show()
```

Bibliography

[1] A. F. Molisch, *Wireless communications*, 2nd ed. Chichester, West Sussex, U.K: Wiley : IEEE, 2011, 827 pp., OCLC: ocn613645390, ISBN: 978-0-470-74187-0 978-0-470-74186-3. (visited on 11/10/2023).

[2] B. Sklar, *Digital communications: fundamentals and applications*, 3rd ed. Hoboken: Pearson Education, Inc, 2020, ISBN: 978-0-13-458856-8. (visited on 11/11/2023).

[3] D. P. Bertsekas, R. G. Gallager, and R. Gallager, *Data networks* (Prentice-Hall International editions), 2. ed. Englewood Cliffs, NJ: Prentice-Hall International, 1992, 556 pp., ISBN: 978-0-13-200916-4 978-0-13-201674-2. [Online]. Available: `https://web.mit.edu/dimitrib/www/Multiaccess_Data_Nets.pdf` (visited on 11/15/2023).

[4] A. A. Kumar S., K. Ovsthus, and L. M. Kristensen., "An industrial perspective on wireless sensor networks — a survey of requirements, protocols, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 16, no. 3, pp. 1391–1412, 2014, ISSN: 1553-877X. DOI: `10.1109/SURV.2014.012114.00058`. [Online]. Available: `http://ieeexplore.ieee.org/document/6728782/` (visited on 11/11/2023).

[5] J. Gao, M. Li, W. Zhuang, Xuemin, Shen, and X. Li, "MAC for machine type communications in industrial IoT – part II: Scheduling and numerical results," 2020, Publisher: arXiv Version Number: 1. DOI: `10.48550/ARXIV.2011.11139`. [Online]. Available: `https://arxiv.org/abs/2011.11139` (visited on 11/11/2023).

[6] B. A. Forouzan and S. C. Fegan, *Data communications and networking* (McGraw-Hill Forouzan networking series), 4th ed. New York: McGraw-Hill Higher Education, 2007, 1134 pp., OCLC: ocm62878618, ISBN: 978-0-07-325032-8 978-0-07-296775-3. (visited on 12/06/2023).

[7] mtsokol. "Numpy/numpy/fft/pocketfft." (2023), [Online]. Available: `https://github.com/numpy/numpy/blob/main/numpy/fft/_pocketfft.py` (visited on 12/10/2023).

[8] Postscapes. "Internet of thinks protocols." (2020), [Online]. Available: `https://www.postscapes.com/internet-of-things-protocols/` (visited on 12/10/2023).

[9] LinkLabs. “Complete list of IoT network protocols.” (2016), [Online]. Available: `https://www.link-labs.com/blog/complete-list-iot-network-protocols` (visited on 12/12/2023).

[10] M. Distribution. “Wireless glossary.” (2023), [Online]. Available: `https://www.msdist.co.uk/support/articles/general-wireless/wireless-glossary` (visited on 12/12/2023).

[11] Ofcom. “Licensed short-range devices.” (2023), [Online]. Available: `https://www.ofcom.org.uk/manage-your-licence/radiocommunication-licences/licensed-short-range` (visited on 12/12/2023).

[12] M. Distribution. “Spectrum.” (2023), [Online]. Available: `https://www.msdist.co.uk/support/spectrum#:~:text=The%2024GHz%20spectrum%20is%20a,%2F752%2FEU%20band%20No` (visited on 12/12/2023).

[13] Ofcom. “UK frequency allocation table (UKFAT).” (2023), [Online]. Available: `https://static.ofcom.org.uk/static/spectrum/fat.html` (visited on 12/12/2023).

[14] C. Express. “802.11g.” (2023), [Online]. Available: `https://www.comms-express.com/infozone/article/802-11g-wi-fi/#:~:text=802.11g%20was%20developed%20by,in%20the%202.4%20GHz%20bands` (visited on 12/12/2023).

[15] L. Learning. “How do you compare the performance of OFDM with other modulation schemes such as QAM and PSK?” (2023), [Online]. Available: `https://www.linkedin.com/advice/1/how-do-you-compare-performance-ofdm` (visited on 12/12/2023).

[16] LinkLabs. “ZigBee vs bluetooth.” (2015), [Online]. Available: `https://www.link-labs.com/blog/zigbee-vs-bluetooth` (visited on 12/12/2023).

[17] R. Falanji, M. Heusse, and A. Duda, “Range and capacity of LoRa 2.4 GHz,” in *Mobile and Ubiquitous Systems: Computing, Networking and Services*, S. Longfei and P. Bodhi, Eds., vol. 492, Series Title: Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, Cham: Springer Nature Switzerland, 2023, pp. 403–421, ISBN: 978-3-031-34775-7 978-3-031-34776-4. DOI: `10.1007/978-3-031-34776-4_21`. [Online]. Available: `https://link.springer.com/10.1007/978-3-031-34776-4_21` (visited on 12/17/2023).

[18] RFC. “Internet protocol darpa internet program protocol specification.” (1981), [Online]. Available: `https://www.rfc-editor.org/rfc/rfc791` (visited on 12/14/2023).

[19] I. A. N. Authority. “Autonomous system (AS) numbers.” (2023), [Online]. Available: `https://www.iana.org/assignments/as-numbers/as-numbers.xhtml` (visited on 12/14/2023).

[20] IEFT. "Address allocation for private internets." (1918), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1918 (visited on 12/14/2023).

[21] Brilliant. "Dijkstra's shortest path algorithm." (2023), [Online]. Available: https://brilliant.org/wiki/dijkstras-short-path-finder/ (visited on 12/15/2023).

[22] GOV.UK. "How to install network infrastructure in shared buildings." (2018), [Online]. Available: https://www.gov.uk/guidance/how-to-install-network-infrastructure-in-shared-buildings (visited on 12/15/2023).

[23] R. NCC. "Request a /24 allocation via the waiting list." (2023), [Online]. Available: https://www.ripe.net/manage-ips-and-asns/ipv4/request-a-24-allocation-via-the-waiting-list (visited on 12/14/2023).

[24] RFC. "A borger gateway protocol (BGP -4)." (2006), [Online]. Available: https://datatracker.ietf.org/doc/html/rfc4271 (visited on 12/14/2023).

[25] I. Society. "IPv6 address planning: Guidelines for IPv6." (2013), [Online]. Available: https://www.internetsociety.org/resources/deploy360/2013/ipv6-address-planning-guidelines-for-ipv6-address-allocation (visited on 12/14/2023).

[26] A. Pashamokhtari, N. Okui, M. Nakahara, A. Kubota, G. Batista, and H. H. Gharakheili, *Quantifying and managing impacts of concept drifts on IoT traffic inference in residential ISP networks*, Jan. 30, 2023. arXiv: 2301.06695[cs]. [Online]. Available: http://arxiv.org/abs/2301.06695 (visited on 12/17/2023).

[27] BSRIA. "BSRIA's latest study shows uptake of convergence and IoT in commercial buildings." (2020), [Online]. Available: https://www.bsria.com/uk/news/article/according_to_bsria_there_is_an_uptake_of_convergence_and_iot_in_commercial_buildings/ (visited on 12/14/2023).

[28] C. C. Service. "How to install network infrastructure in shared buildings." (2023), [Online]. Available: https://www.crowncommercial.gov.uk/agreements/RM6116#:~:text=Lot%203a%3A%20IoT%20and%20Smart%20Cities%20(Smart%20shared%20and%20connected%20spaces) (visited on 12/15/2023).

[29] Ofcom. "IR 2030 UK interface requirements 2030." (2023), [Online]. Available: https://www.ofcom.org.uk/__data/assets/pdf_file/0028/84970/ir-2030.pdf (visited on 12/12/2023).

[30] I. Review. “Broadband ISP BT sees UK network traffic peak at 28tbps.” (2022), [Online]. Available: `https://www.ispreview.co.uk/index.php/2022/07/broadband-isp-bt-sees-uk-network-traffic-peak-at-28tbps.html` (visited on 12/15/2023).

[31] M. Handley, “Why the internet only just works,” *BT Technology Journal*, vol. 24, no. 3, pp. 119–129, Jul. 2006, ISSN: 1358-3948, 1573-1995. DOI: `10.1007/s10550-006-0084-z`. [Online]. Available: `http://link.springer.com/10.1007/s10550-006-0084-z` (visited on 11/17/2023).

[32] Wireshark. “Duplicate packets.” (2023), [Online]. Available: `https://wiki.wireshark.org/DuplicatePackets` (visited on 12/16/2023).

[33] Wireshark. “TCP_analyze_sequence_numbers.” (2023), [Online]. Available: `https://wiki.wireshark.org/TCP_Analyze_Sequence_Numbers` (visited on 12/16/2023).

[34] Wireshark. “RTT graph showing values higher than tcp.analysis.ack_rtt.” (2023), [Online]. Available: `https://osqa-ask.wireshark.org/questions/38607/rtt-graph-showing-values-higher-than-tcpanalysisack_rtt/` (visited on 12/16/2023).

[35] K. P. Murphy, *Machine learning: a probabilistic perspective* (Adaptive computation and machine learning series). Cambridge, MA: MIT Press, 2012, 1067 pp., ISBN: 978-0-262-01802-9. [Online]. Available: `https://mitpress.mit.edu/9780262018029/machine-learning/` (visited on 12/16/2023).

[36] J. M. Johnson and T. M. Khoshgoftaar, “Survey on deep learning with class imbalance,” *Journal of Big Data*, vol. 6, no. 1, p. 27, Dec. 2019, ISSN: 2196-1115. DOI: `10.1186/s40537-019-0192-5`. [Online]. Available: `https://journalofbigdata.springeropen.com/articles/10.1186/s40537-019-0192-5` (visited on 12/17/2023).