COP532 – Internet Protocol Design Report

Table of Contents

[1 – Abstract 3](#_Toc503189007)

[2 – Design 4](#_Toc503189008)

[2a – Core design 4](#_Toc503189009)

[2b – Reliability 4](#_Toc503189010)

[2c – Segmentation 5](#_Toc503189011)

[2d – Routing and Forwarding 6](#_Toc503189012)

[3 – Implementation 7](#_Toc503189013)

[3a – Reliability 7](#_Toc503189014)

[3b – Segmentation 7](#_Toc503189015)

[3c – Routing and Forwarding 7](#_Toc503189016)

[4 – Reflection on Coursework 8](#_Toc503189017)

# 1 – Abstract

This report will go into detail about the design decisions made for our project, and reflect on what the group might have done better if given the opportunity to redo the coursework. The report will also look into the design of the system, and the protocols themselves, why decisions on the protocols were taken and how that affected the system that the group produced.

# 2 – Design

## 2a – Core design

The core design focused around having a spine for the main program, and then creating instances of each method (layer of the protocol) in the spine to utilise, meaning on a programming standpoint, it was clear and concise, but it was also usable in the sense that it would not take much effort to alter the program. This was useful as it allowed for easy documentation and commenting of the program. With this, it meant that other developers could edit the code without being confused by a program that had no defined borders for each layer.

The alternative was to not have any clear and defined separation between the different protocol layers, which would have been easier to implement, but significantly harder to document and edit, when that became necessary. The decision to use a ‘spine’ approach was made very early on, to give as much time as possible to set out a good idea of the shape the program would take.

## 2b – Reliability

The reliability design was focused around the protocol design agreed on by both groups. In this design, it was decided that it and segmentation would effectively work as one layer as their jobs were not vastly different enough to warrant their own protocols. In this, the protocol design agreed (along with segmentation) used 4 bytes of the 100 total bytes per packet. Within reliability specifically, there was the ‘checksum’ byte (which calculated an integer to confirm the integrity of the message), the ‘acknowledgement’ bit (which would notify the receiver whether the incoming packet was simply an acknowledgement packet or a normal data packet), the ‘type’ nibble (which specified the real type of packet incoming, whether it be a file, a normal message, or routing information), and the ‘length’ byte (which specified the length of the data before padding takes place, which helps to combat corruption).

When a host receives a data packet, it will read the packet and take the necessary information from the header to confirm to the sender that the packet has been received correctly, and without corruption.

The information to be used will firstly be the checksum, which it will compare to the checksum it calculates itself. If they are different, then it can be confirmed that there has been corruption in transmission. Otherwise, the data is viewed as uncorrupted, and the packet accepted. If a corruption has been detected, then the packet will be dropped and no acknowledgement will be sent, meaning that the sender will need to resend the packet.

When a packet is viewed to be intact and legitimate, an acknowledgement will be sent to the sender to inform them that the packet has been received and not to worry about resending any packets, both hosts can then move on to other communications. If an acknowledge is not received within 200ms, then the sender will resend the packet, up to five attempts. If no acknowledgement is received within 200ms of the fifth packet, then the packet is dropped and the sender will cease to resend it. This may result in the receiver getting an incomplete message, but it is necessary as it was deemed that five attempts at 200ms each was clearly enough for a host to respond to a packet.

The length of the data is necessary and included as it shows the receiver (if the full 94 bytes is not utilised by the sender) where the sender’s message ends, and the padding from ICNS begins. The receiver can then remove the padding where the sender states the message ends, and the message can be viewed in full with no padding.

An acknowledgement packet will have the relevant message ID and sequence number included, along with the ‘acknowledgement’ bit being changed to ‘1’, so the receiver knows that the packet is an acknowledgement packet and not a regular packet. With this system, the packet will have all of the important information from the receiver to the sender, for the sender to know that their packet has been taken and received correctly.

## 2c – Segmentation

The segmentation layer (which is grouped with reliability, as stated previously) utilises the remaining bits of the 4 bytes of the reliability and segmentation layer, this includes the ‘end’ bit (which signifies the end of the message, if there has been more than one packet), the message ID’ byte (which allows for the sender to have more than one message in flight at once, meaning if a host sends more than one packet for a message, both can be received and reassembled by the receiver without mixing up the packets from the different messages), and the ‘sequence number’ byte (which signifies the place a packet must be positioned when unpacking to form the message back in order).

When a host sends a message that will take more than the maximum 94 bytes, it will be segmented (all packets of the message will have the same message ID, which will be taken from a list when not in use), and the sequence number for each packet will notify the receiver in which order to place the packets to reassemble the message correctly and in order.

The ‘end’ bit will be ‘0’ up until the last packet is created and sent, at which point it will be altered to ‘1’, which tells the receiver that the packet is the final packet of a message, meaning that any other packets after this will almost certainly be duplicates. If the bit is corrupted and therefore flipped to its opposite, this will be spotted in the checksum calculation and the packet dropped with no acknowledgement being sent by the receiver, so the sender will be forced to automatically resend the packet after 200ms.

## 2d – Routing and Forwarding

In the Routing and Forwarding layer, there are two headers segments, simply ‘source’ (the source address) and ‘destination’ (the destination address).

The routing is performed in a ‘hop-by-hop’ basis, meaning that the packet will be sent simply by a single jump all the way to the destination. This should make the transmission more optimal as each host will know where to send the packet given the destination address. With the ‘hop-by-hop’ system, the checksum is calculated and checked at every host, so if the packet is corrupted, it will be dropped before it gets to the destination, meaning the sender will resend the packet when no acknowledgement is received.

For example, host 1 (H1) may be connected to H2, and H2 to H3, and H2 also to H4. If H1 wants to send a packet to H4, H2 will know that given H4 is one of its neighbours, it can route it directly there. However, if H4 was connected to H3 instead, H2 would not know about H4, and so would send it to H3, who would then be able to forward the packet to its correct destination. This system relies on the entire network working properly together, and each host having up-to-date Forwarding and Lookup tables.

In the program, there are two tables dedicated for routing and forwarding. The ‘Lookup Table’ (which contains the neighbour name and the corresponding IP address), and the ‘Forwarding Table’ (which tells the host where to send a packet to get to another host). With this system, it makes the design of the protocol flexible and easily expandable as more hosts can be added at will and simply included in the Forwarding and Lookup tables, simply doing this will then fully include the new host in the network and it will be able to partake in discussions on the program.

# 3 – Implementation

## 3a – Reliability

## 3b – Segmentation

## 3c – Routing and Forwarding

# 4 – Reflection on Coursework