SYS2 (NET) Laboratory 9 : IP, ICMP, ARP & MAC

These weekly lab scripts have been written as a self-study resource i.e. an activity to be worked on at your own pace, in your own time. The timetabled practical sessions are an opportunity to get advice and guidance. Therefore, you may not always finish each lab during the timetabled session, but do finish each lab's tasks in your own time. Hardware labs are open Mon-Fri, 9:00-17:00.

In this week's lab we shall finish off our investigation into the IP protocol and examine how hosts and routers communicate diagnostic and control information using the Internet Control Message Protocol (ICMP). Before starting this lab make sure you have watched video **Lecture 5**. We will then examine the protocols used in the lowest level of our protocol stack: layer 2, the Link layer and how these "link" into the final layer: layer 1, the Physical layer. In this lab we will limit our investigation of these layers to the Ethernet protocol as it's one of the most widely used. However, as with any of the layers we have investigated other protocols are available e.g. wifi, rather than hard-wired copper. In the Link layer we move away from logical addressing (IP) to physical addressing i.e. Media Access Control (MAC) addressing. The analogy I like to use to explain this is that a logical address is comparable to your home address, and the physical address is comparable to your name. Therefore, when you communicate at a distance you would use your home address to send a letter, but for local communications you would use a person's name i.e. when you are talking to them face-2-face. The final protocol that we shall look at is the Address Resolution Protocol (ARP), that allows a host to convert between IP and MAC addresses. At the end of this practical you will understand how :

- ICMP packets are used to carry control and status information across a network.
- MAC addresses are used in layer-2 switches to route Ethernet frames.
- hosts covert layer-3 IP address into layer-2 MAC addresses.
- data is communicated across Cat5 cables.

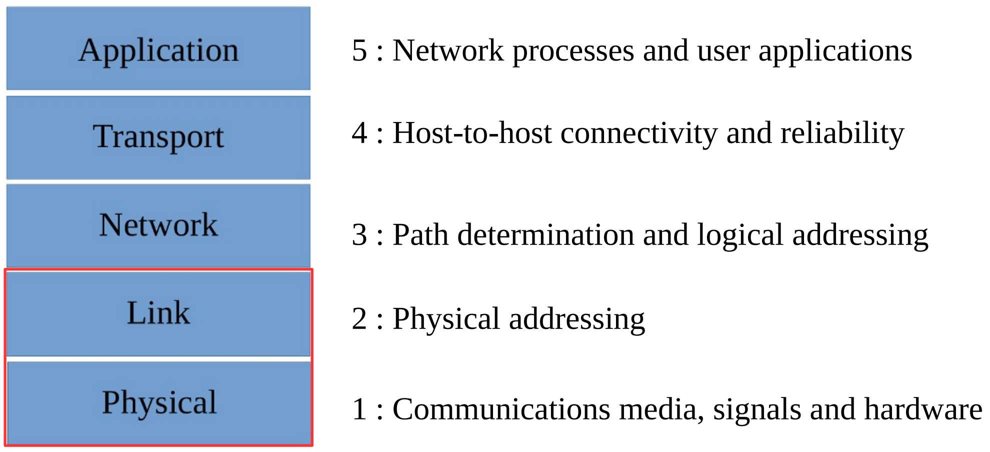

Application
5 : Network processes and user applications
Transport
4 : Host-to-host connectivity and reliability
Network
3 : Path determination and logical addressing
Link
2 : Physical addressing
Physical
1 : Communications media, signals and hardware

Figure 1 : The Internet protocol stack

Task 1

In previous labs we have looked at how the IP protocol can be used to transfer user data between hosts, but we did not looked at what happens when things go wrong i.e. how is diagnostic and control informations transferred between hosts and routers? A good example of this are the TTL exceeded packets we saw in practical 7, when a packet got stuck in a loop. To perform these tasks the ICMP protocol was developed alongside IP and is an integral part of its operation i.e. rather than a stand-alone protocol. Developed in 1981, RFC792 (https://datatracker.ietf.org/doc/html/rfc792). Like UDP this is a very lightweight protocol, as shown in figure 2, that uses the underlying functionality of IP to transfer data between hosts.

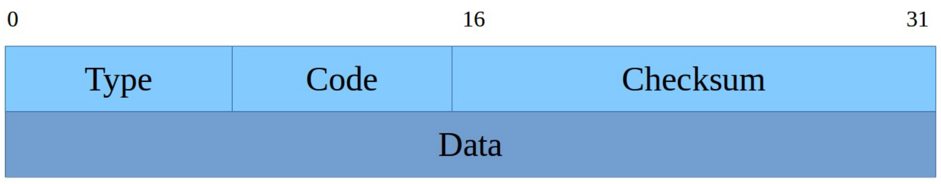
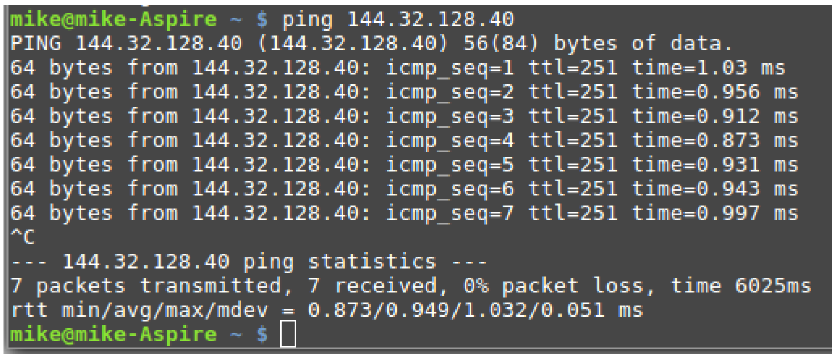


Figure 2 : ICMP header

Note, the data section can vary in size depending on the `type` field value. All network packets must be at least 64B, therefore, for short ICMP packets these will be padded with “random” characters to fill up the payload.

You will have already seen the ICMP protocol in action from lab 1 when using the ping and traceroute commands, as shown in figure 3 i.e. the `icmp_seq` number.



```
mike@mike-Aspire ~ $ ping 144.32.128.40
PING 144.32.128.40 (144.32.128.40) 56(84) bytes of data.
64 bytes from 144.32.128.40: icmp_seq=1 ttl=251 time=1.03 ms
64 bytes from 144.32.128.40: icmp_seq=2 ttl=251 time=0.956 ms
64 bytes from 144.32.128.40: icmp_seq=3 ttl=251 time=0.912 ms
64 bytes from 144.32.128.40: icmp_seq=4 ttl=251 time=0.873 ms
64 bytes from 144.32.128.40: icmp_seq=5 ttl=251 time=0.931 ms
64 bytes from 144.32.128.40: icmp_seq=6 ttl=251 time=0.943 ms
64 bytes from 144.32.128.40: icmp_seq=7 ttl=251 time=0.997 ms
^C
--- 144.32.128.40 ping statistics ---
7 packets transmitted, 7 received, 0% packet loss, time 6025ms
rtt min/avg/max/mdev = 0.873/0.949/1.032/0.051 ms
mike@mike-Aspire ~ $ 
```

Figure 3 : example ping command

However, to see what is happening under the bonnet we will need to switch to Wireshark. On the PC click on the start button and select the command prompt.

`-> Command Prompt`

then SSH into Pi-1 by entering the command below:

```
ssh pi@192.168.X.1                    (IP address will vary, X=Desk)
```

Next, open a VNC session on Pi-1, then within the VNC session start Wireshark, click on the menu icon and select :

Applications `-> Internet -> Wireshark`

This will open the Wireshark GUI. Select the network interface (NIC) to capture packets on, left click on (highlight) the Ethernet 1 (`eth1`) in the Interface list. In Wireshark enter the packet protocol filter :

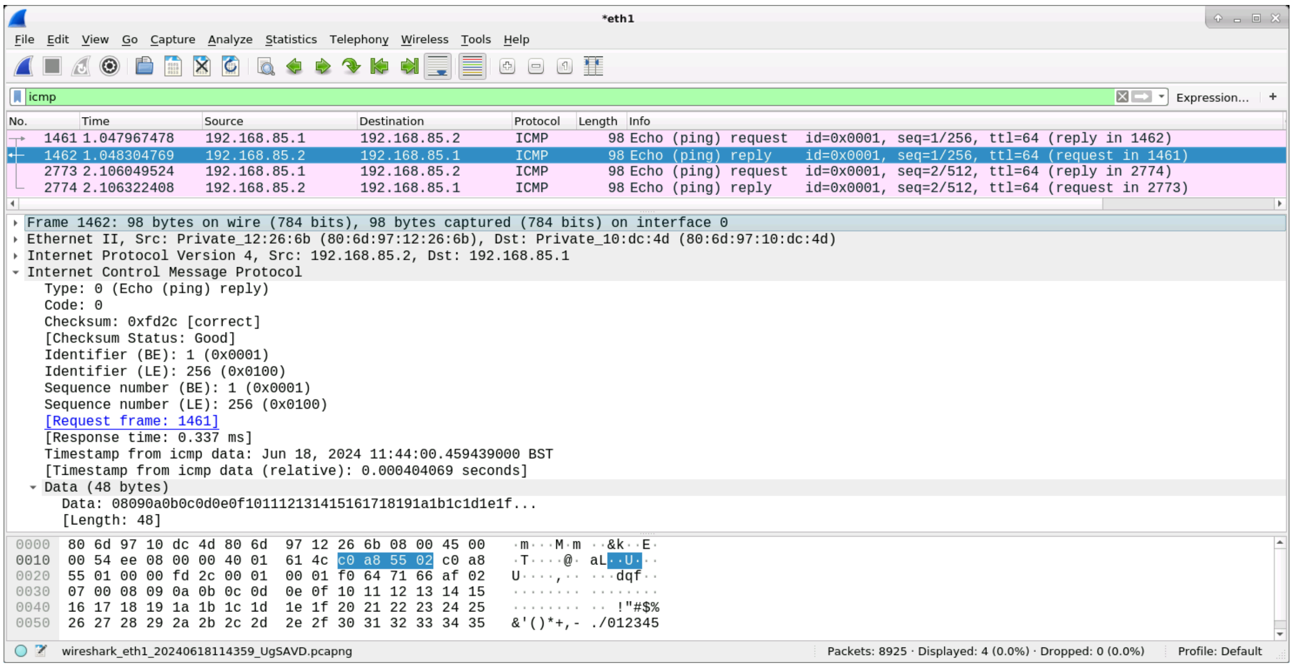
```
icmp
```

and click on the Apply button (this will remove most of the network “noise”). Next click on the Start icon on the top toolbar in Wireshark. Then in the Pi-1 SSH terminal enter the following command:

```
ping -c 2 192.168.X.2             (X=Box/Desk)
```

Note, the `-c 2` flag limits the ping command to 2 packets.

You will now see some activity in the main window as Wireshark captures the packet associated with this ping command, as shown in figure 4. Finally, click on the Stop icon 🟥 on the top toolbar to stop capturing network traffic.


*eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
Expression...
+

```
No.    Time         Source          Destination     Protocol  Length  Info
  1461 1.047967478  192.168.85.1    192.168.85.2    ICMP         98 Echo (ping) request  id=0x0001, seq=1/256, ttl=64 (reply in 1462)
  1462 1.048304769  192.168.85.2    192.168.85.1    ICMP         98 Echo (ping) reply    id=0x0001, seq=1/256, ttl=64 (request in 1461)
  2773 2.106049524  192.168.85.1    192.168.85.2    ICMP         98 Echo (ping) request  id=0x0001, seq=2/512, ttl=64 (reply in 2774)
  2774 2.106322408  192.168.85.2    192.168.85.1    ICMP         98 Echo (ping) reply    id=0x0001, seq=2/512, ttl=64 (request in 2773)

▸ Frame 1462: 98 bytes on wire (784 bits), 98 bytes captured (784 bits) on interface 0
▸ Ethernet II, Src: Private_12:26:6b (80:6d:97:12:26:6b), Dst: Private_10:dc:4d (80:6d:97:10:dc:4d)
▸ Internet Protocol Version 4, Src: 192.168.85.2, Dst: 192.168.85.1
▾ Internet Control Message Protocol
    Type: 0 (Echo (ping) reply)
    Code: 0
    Checksum: 0xfd2c [correct]
    [Checksum Status: Good]
    Identifier (BE): 1 (0x0001)
    Identifier (LE): 256 (0x0100)
    Sequence number (BE): 1 (0x0001)
    Sequence number (LE): 256 (0x0100)
    [Request frame: 1461]
    [Response time: 0.337 ms]
    Timestamp from icmp data: Jun 18, 2024 11:44:00.459439000 BST
    [Timestamp from icmp data (relative): 0.000404069 seconds]
  ▾ Data (48 bytes)
      Data: 08090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f...
      [Length: 48]

0000  80 6d 97 10 dc 4d 80 6d  97 12 26 6b 08 00 45 00   ·m···M·m ··&k··E·
0010  00 54 ee 08 00 00 40 01  61 4c c0 a8 55 02 c0 a8   ·T····@· aL··U···
0020  55 01 00 00 fd 2c 00 01  00 01 f0 64 71 66 af 02   U····,·· ···dqf··
0030  07 00 08 09 0a 0b 0c 0d  0e 0f 10 11 12 13 14 15   ········ ········
0040  16 17 18 19 1a 1b 1c 1d  1e 1f 20 21 22 23 24 25   ········ ·· !"#$%
0050  26 27 28 29 2a 2b 2c 2d  2e 2f 30 31 32 33 34 35   &'()*+,- ./012345

wireshark_eth1_20240618114359_UgSAVD.pcapng    Packets: 8925 · Displayed: 4 (0.0%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 4 : ping packet capture

From the packet trace you can see that there are two sets of request/response packets. Pi-1 transmits an echo request ICMP packet, using a type field value of 8 to Pi-2. Pi-2 responds with an echo response ICMP packet, using a type field value of 0. For more information of the ICMP type field refer to:

https://en.wikipedia.org/wiki/Internet_Control_Message_Protocol#Control_messages

Note, for type 0 and 8 packets the code field shown in figure 2 is not used i.e. it is set to 0, as no additional parameters are needed.

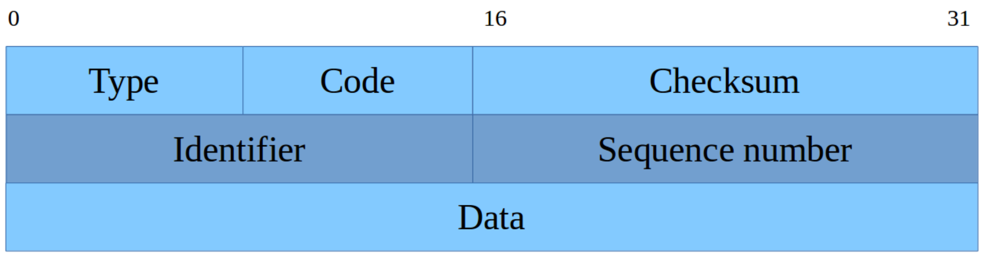


Figure 5 : ICMP packet format

Included within the ICMP echo request/response data section is a 16bit sequence number (incrementing count value) and identifier (fixed for that ping), as shown in figure 5. This allows the transmitting host to identify missing response packets. The remaining data section is padded with “random” characters to pad the packet to the required minimum size.

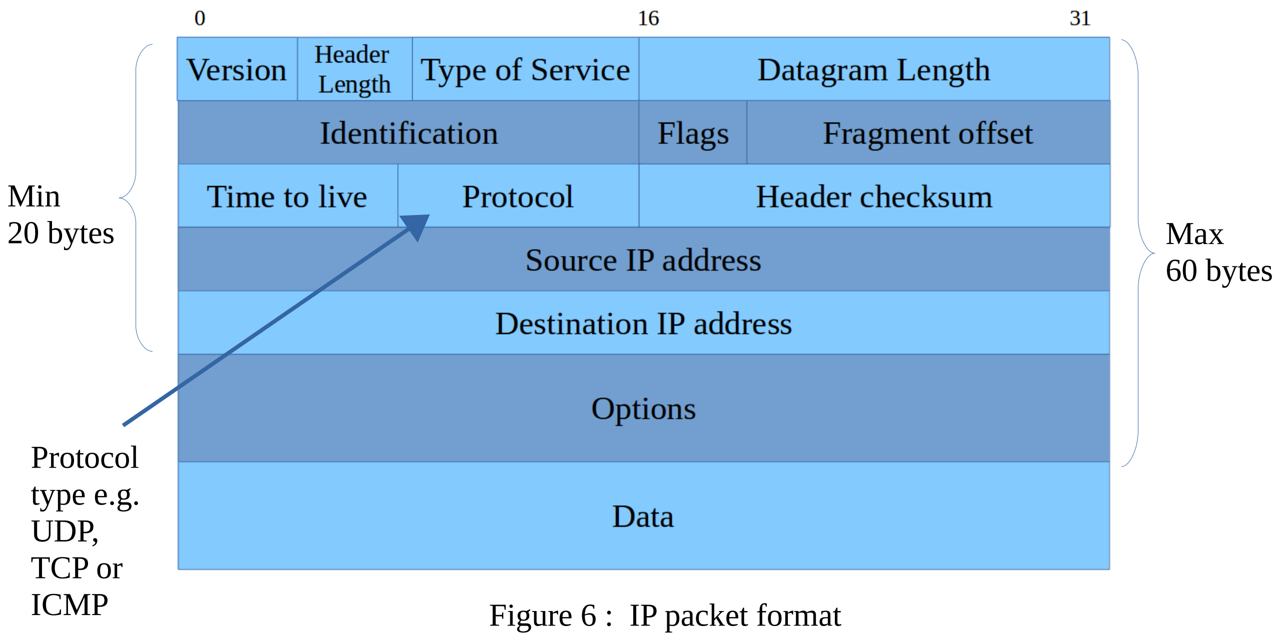


Figure 6 : IP packet format

The IP protocol (layer 3) source and destination IP addresses are used to route these packets between hosts. ICMP packets are identified within the IP datagram as ICMP packets by the protocol field shown in figure 6 i.e. set to the value 1. For more information on the IP protocol field refer to:

https://en.wikipedia.org/wiki/List_of_IP_protocol_numbers

Task : examine the ping packets captured in Wireshark, make sure you can identify the type, code and protocol fields?

Unlike TCP/UDP the ICMP protocol is used to transfer diagnostic and control data. Consider the scenario where a host pings an unreachable network. In these situations if a router detects that the destination network is invalid, or the destination host detects that the requested port or protocol is invalid, the transmitting host will receive an error message using ICMP.

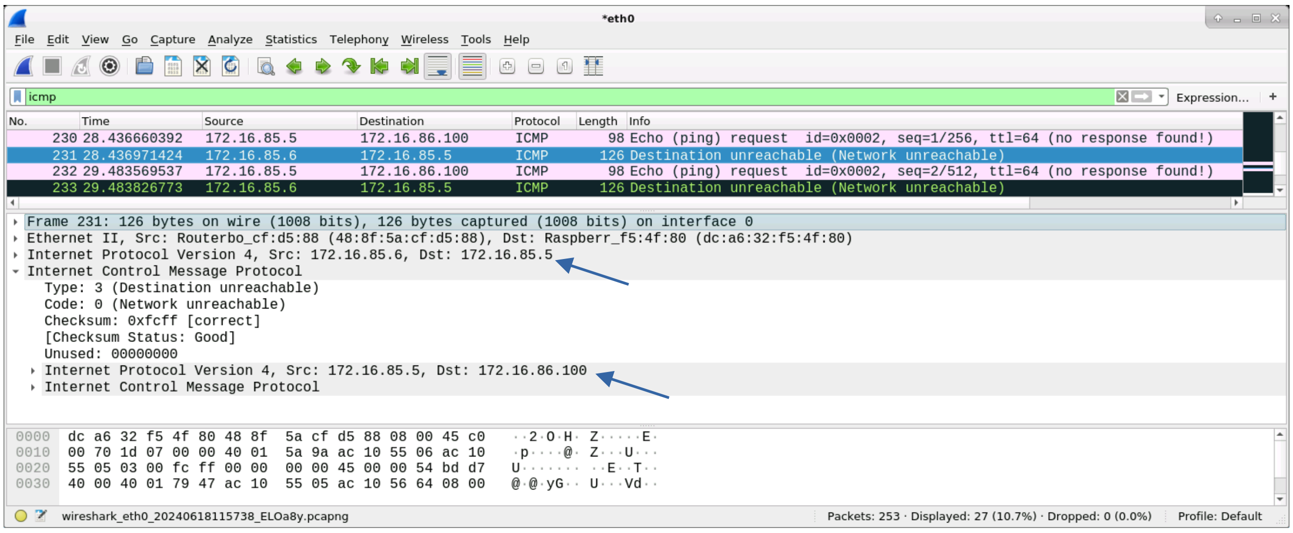
To see this is action restart Wireshark, left click on (highlight) the Ethernet 0 (`eth0`) icon in the Interface list. Next enter the packet protocol filter :

```
icmp
```

and click on the Apply button (this will remove most of the network “noise”). Next click on the Start icon on the top toolbar in Wireshark. Then in the Pi-1 SSH terminal enter the following command:

```
ping -c 2 172.16.X.100            (IP address will vary, X=Desk)
```

You will now see some activity in the main window as Wireshark captures the packet associated with this ping command. Finally, click on the Stop icon on the top toolbar to stop capturing network traffic. Examine the “Network unreachable” packets in Wireshark, an example packet capture is shown in figure 7.


*eth0
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
icmp
Expression…

```
No.     Time            Source          Destination     Protocol Length Info
    230 28.436660392    172.16.85.5     172.16.86.100   ICMP        98 Echo (ping) request  id=0x0002, seq=1/256, ttl=64 (no response found!)
    231 28.436971424    172.16.85.6     172.16.85.5     ICMP       126 Destination unreachable (Network unreachable)
    232 29.483569537    172.16.85.5     172.16.86.100   ICMP        98 Echo (ping) request  id=0x0002, seq=2/512, ttl=64 (no response found!)
    233 29.483826773    172.16.85.6     172.16.85.5     ICMP       126 Destination unreachable (Network unreachable)

▸ Frame 231: 126 bytes on wire (1008 bits), 126 bytes captured (1008 bits) on interface 0
▸ Ethernet II, Src: Routerbo_cf:d5:88 (48:8f:5a:cf:d5:88), Dst: Raspberr_f5:4f:80 (dc:a6:32:f5:4f:80)
▸ Internet Protocol Version 4, Src: 172.16.85.6, Dst: 172.16.85.5
▾ Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 0 (Network unreachable)
    Checksum: 0xfcff [correct]
    [Checksum Status: Good]
    Unused: 00000000
  ▸ Internet Protocol Version 4, Src: 172.16.85.5, Dst: 172.16.86.100
  ▸ Internet Control Message Protocol

0000  dc a6 32 f5 4f 80 48 8f  5a cf d5 88 08 00 45 c0   ··2·O·H· Z·····E·
0010  00 70 1d 07 00 00 40 01  5a 9a ac 10 55 06 ac 10   ·p····@· Z···U···
0020  55 05 03 00 fc ff 00 00  00 00 45 00 00 54 bd d7   U······· ··E··T··
0030  40 00 40 01 79 47 ac 10  55 05 ac 10 56 64 08 00   @·@·yG·· U···Vd··

wireshark_eth0_20240618115738_ELOa8y.pcapng    Packets: 253 · Displayed: 27 (10.7%) · Dropped: 0 (0.0%)    Profile: Default
```

Figure 7 : Network unreachable packet capture

Note, in addition to transferring “error” messages, ICMP is also used to transfer routing information i.e. a router can ask a host/router to update its routing table, to use an alternative / preferred route. As always the host/router does not have to listen :).

Task : why does there seem to be a second IP and ICMP section shown in figure 7? What is this used for? Using Wireshark can you capture an ICMP packets for the scenarios: Host unreachable, Network unreachable and Port unreachable?

Hints, compare the second IP and ICMP values to the previous ping packets. Generating the other scenarios is a very good test of your network core knowledge. There are number of different commands that could be used to generate the required packets, however, as always `telnet` is a useful tool :).

Task 2

In layer 3 (Network layer) packets are routed between hosts using the IP protocol. In layer 2 (Link layer) packets are transferred between hosts using their Media Access Control (MAC) addresses i.e. a physical address. This raises the question why do we need two ways of identifying a host i.e. each host will have an IP address and a MAC address?

Note, just to complicate things you could manually assign multiple IP addresses and MAC addresses to each NIC :).

The answer to this question is again linked to our protocol stack i.e. each layer does one thing. As previously discussed we try to decouple layers so that they can be replaced with other protocols. At layer 3 we need a protocol that can route packets between the network of networks that is the Internet. A protocol that can be used to structure, partition our network i.e. subnets. At layer 2 we need a protocol that can allow hosts to efficiently communicate on a local network (LAN). These are different problems that can be solved in different ways. Separating a host's logical (IP) and physical (MAC) addresses reduces design complexity and adds flexibility in how a network is organised and structured.

The Ethernet protocol uses a 48bit MAC address, typically represented as six, two-digit hexadecimal characters as shown in figure 8.

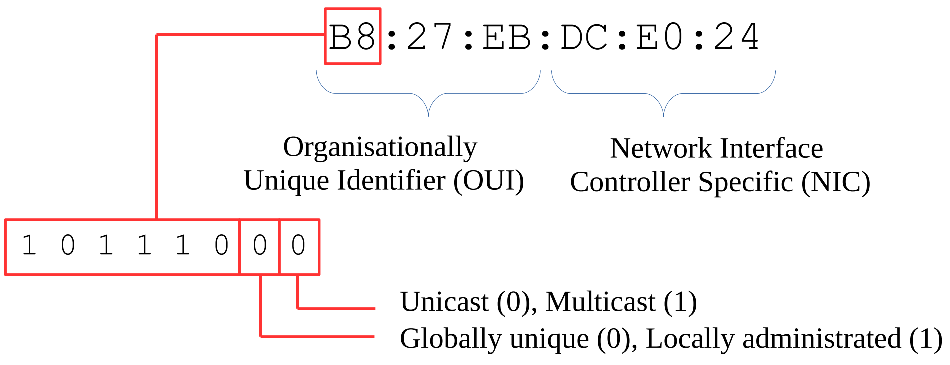

B8:27:EB:DC:E0:24
Organisationally Unique Identifier (OUI)
Network Interface Controller Specific (NIC)
1 0 1 1 1 0 0 0
Unicast (0), Multicast (1)
Globally unique (0), Locally administrated (1)

Figure 8 : example Raspberry Pi MAC address

Each NIC i.e. the physic9 ptal network hardware, has a unique MAC address set in hardware. This is the "burned-in" address or physical address, set during its manufacture. The MAC address is divided into two parts an organisational ID and a network hardware ID. MAC addresses are used by the family of IEEE 802 network protocols e.g. Ethernet (802.3), Wifi (802.11). For more information on these protocols refer to:

https://en.wikipedia.org/wiki/IEEE_802

The original Raspberry Pi organisational ID was `B8:27:EB`, so if you see a MAC address starting with these numbers there is a high probability that its associated host is one of the older Pis. However, there are now a lot of Raspberry Pi in the world so the Raspberry Pi foundation now has multiple organisational IDs. Also, MAC addresses can be spoofed, so its not a 100% guarantee :).

Task : using `ifconfig` and `ipconfig` can you identify the MAC addresses of the NICs on Pi-2 and the PC?

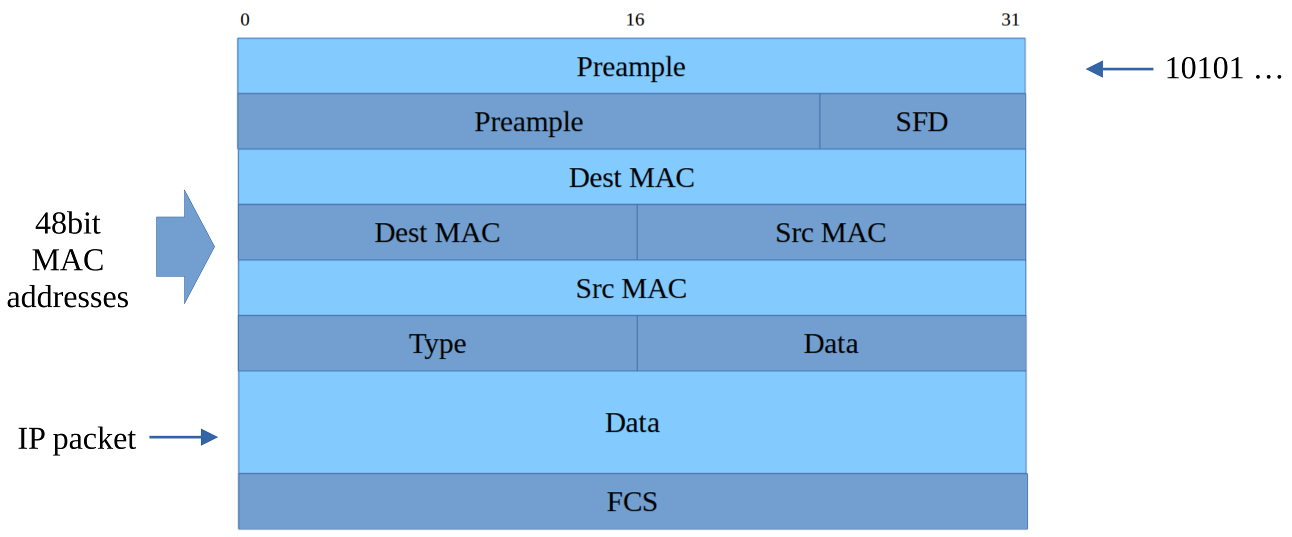


Figure 9 : Ethernet frame

Task 3

As discussed in the previous section when using the Ethernet protocol i.e. layer 2, communication between hosts is implemented using MAC addresses, as shown in figure 9. At this level IP addresses are not used. This raises the question how does the network switch know were to send these Ethernet frames? The short answer is: when it is first powered up its doesn't. However, as hosts send packets across this local network (broadcast domain) the switch learns what host is connected to what port.

Note, unfortunately the term `port` is reused here to describe the physical network sockets on a switch or router i.e. the holes the network cables plug into. In this context we are not talking about UDP / TCP ports :).

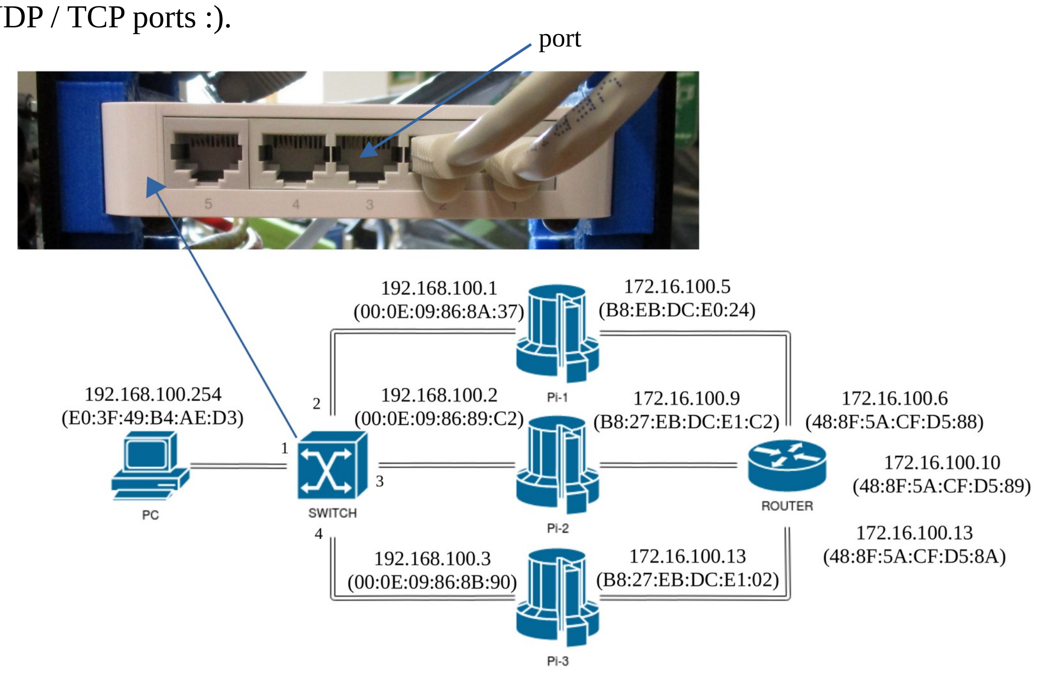
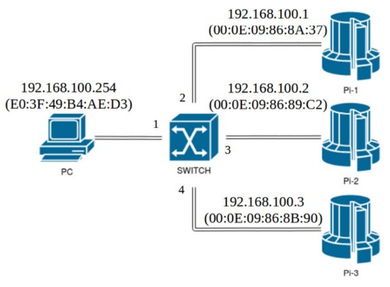

port
192.168.100.1 (00:0E:09:86:8A:37)
172.16.100.5 (B8:EB:DC:E0:24)
192.168.100.254 (E0:3F:49:B4:AE:D3)
192.168.100.2 (00:0E:09:86:89:C2)
172.16.100.9 (B8:27:EB:DC:E1:C2)
172.16.100.6 (48:8F:5A:CF:D5:88)
172.16.100.10 (48:8F:5A:CF:D5:89)
172.16.100.13 (48:8F:5A:CF:D5:8A)
192.168.100.3 (00:0E:09:86:8B:90)
172.16.100.13 (B8:27:EB:DC:E1:02)
PC
SWITCH
ROUTER
PI-1
PI-2
PI-3
1
2
3
4

Figure 10 : Network switch, hardware (top), example network (bottom)

Consider our Pi system shown in figures 10. The network switch contains a block of Contents Addressable Memory (CAM). This memory is not searched using an address i.e. sequentially, as with SRAM. Rather CAM is associative memory that can be searched in parallel using a key. Stored in this memory is the Source Address Table (SAT), mapping observed MAC addresses to a port (socket) on the switch.

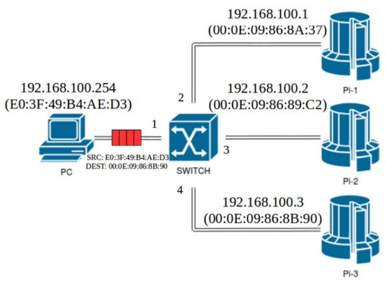
Note, comparing SRAM and CAM to python data structures, SRAM is an array or list. CAM is a dictionary implemented in hardware. Each CAM location has a built-in hardware comparator allowing searches to be performed in parallel, in a constant time i.e. one memory transaction. To state the obvious CAM is therefore a lot more expensive than SRAM, so it tends to be a lot smaller in size. The SAT is also sometimes called a MAC address table.

To start with we will assume that each host already knows the MAC address of the destination host it wishes to communicate with. However, the network switches SAT is empty (figure 11a). The host transmits a frame (figure 11b). This is buffered in the switch, its source MAC address is searched for in the SAT, resulting in a miss, so the PC's MAC address and port are added to the table. Next, the destination MAC address is searched for, resulting in a miss. Therefore, this frame is broadcasted on all other ports i.e. ports 2,3,4 and 5 (figure 11c), but not the source port i.e. port 1.



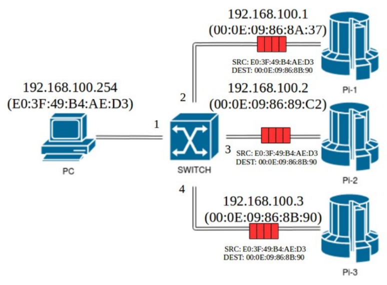
SAT	
X	
X	
X	
X	
X	

(a)



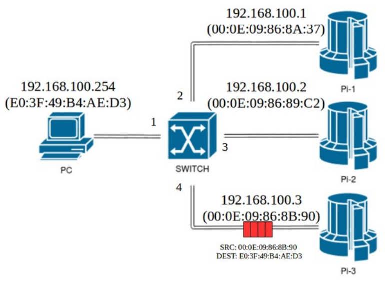
SAT	
1	E0:3F:49:B4:AE:D3
X	
X	
X	
X	

(b)



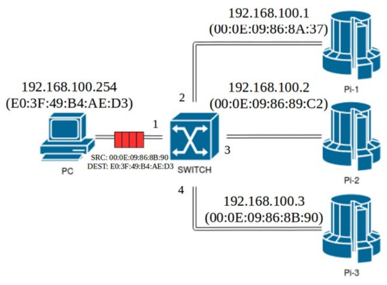
SAT	
1	E0:3F:49:B4:AE:D3

(c)



SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
X	
X	
X	

(d)



SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
X	
X	
X	

(e)

Figure 11 : updating SAT

Hosts connected to these ports process this packet. Pi-3 identifies its MAC and in this example responds to Pi-3 e.g. the PC is starting a TCP connection (figure 11d). The switch receives this frame from Pi-3 and buffers it. Again, the source MAC address is searched for in the SAT, resulting in a miss, so Pi-3's MAC address and port are added to the table. Next, the destination MAC address is searched for, resulting in a hit. Therefore, rather than broadcasting the response packet to all other ports it is forwarded to port 1 i.e. the PC (figure 11e).

Note, the SAT is typically stored in CAM, therefore, data can be stored in any CAM location i.e. the MAC address is used as the search key when accessing this memory, returning the Port number that the associated host is connected to. The physical address / location in the CAM memory is not important i.e. it does not need to be sequential memory locations as shown in figure 11.

A switch can learn what host is connected to what port. However, in a typical application each host does not know the MAC address of the host it wishes to communicate with i.e. it only knows its own IP and MAC addresses. Therefore, before a network packet can be transferred the transmitting host has to find the destination MAC address. To do this we use the Address Resolution Protocol (ARP) (RFC 826: https://tools.ietf.org/html/rfc826). This protocol was developed in 1982 and is used in conjunction with the Ethernet protocol to allow a host to map a destination IP address to its MAC address.

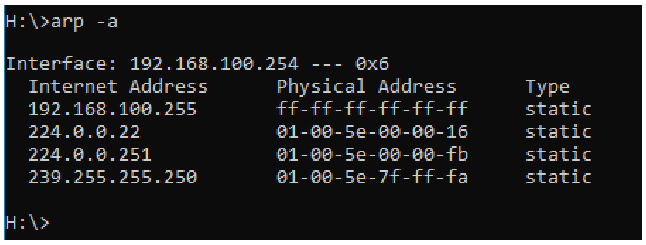
Each host maintains an ARP table, listing known IP addresses and there associated MAC addresses. To view the PCs ARP table, click on the start button and select the command prompt.

```
-> Command Prompt
```

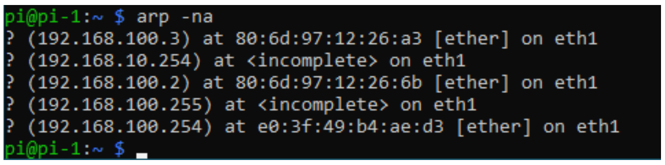
then enter the command below:

```
arp -a
```

This will display the PC's current ARP table as shown in figure 12, static entries being defined by the OS and dynamic entries discovered by the ARP protocol. The same command can also be used on the Pi, but with the addition of a `-n` parameter to specify IP addresses rather than resolved names.



```
H:\>arp -a

Interface: 192.168.100.254 --- 0x6
  Internet Address      Physical Address      Type
  192.168.100.255       ff-ff-ff-ff-ff-ff     static
  224.0.0.22            01-00-5e-00-00-16     static
  224.0.0.251           01-00-5e-00-00-fb     static
  239.255.255.250       01-00-5e-7f-ff-fa     static

H:\>
```



```
pi@pi-1:~ $ arp -na
? (192.168.100.3) at 80:6d:97:12:26:a3 [ether] on eth1
? (192.168.10.254) at <incomplete> on eth1
? (192.168.100.2) at 80:6d:97:12:26:6b [ether] on eth1
? (192.168.100.255) at <incomplete> on eth1
? (192.168.100.254) at e0:3f:49:b4:ae:d3 [ether] on eth1
pi@pi-1:~ $
```

Figure 12 : ARP tables, PC (left), Pi-1 (right)

Task : examine the ARP tables on Pi-1, Pi-2 and Pi-3. Are they the same? If not why?

Hints, what generates this ARP table, what is static and what is dynamic?

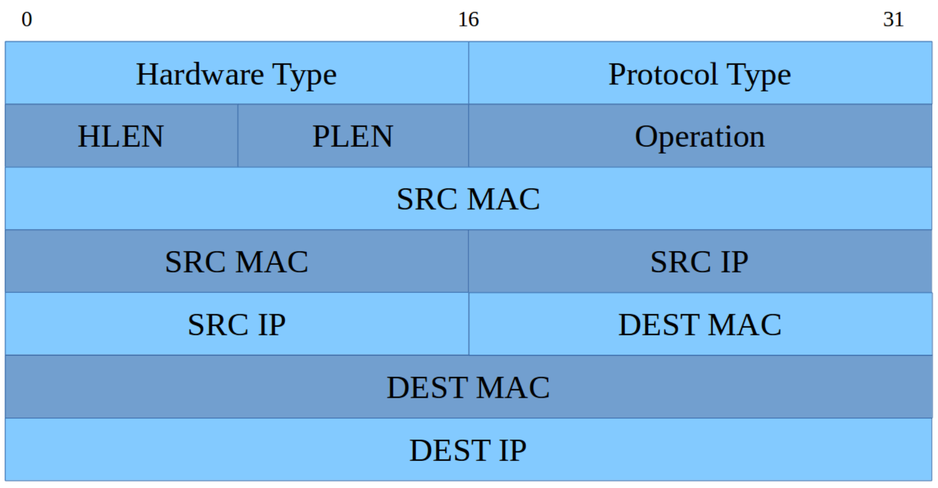
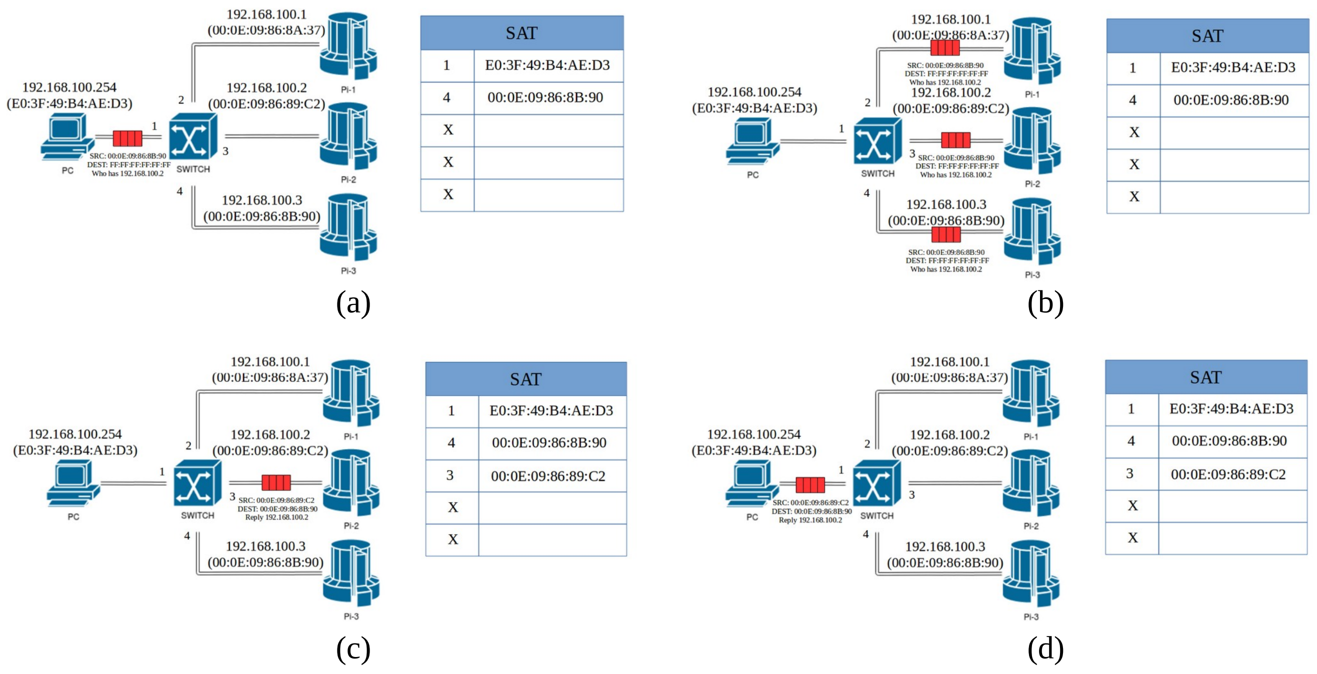


Figure 13 : ARP header

The ARP header is shown in figure 13. As the ARP protocol is used by a number of different hardware systems and protocol types, these are specified by the first two fields i.e. hardware type 1 is Ethernet and protocol type 0x0800 is IPv4. The operation field indicates if this is a Request (1) or Reply (2) packet. For more information please refer to RFC 826:

https://tools.ietf.org/html/rfc826.

When a host wishes to send a network packet e.g. TCP/UDP, it first refers to its local ARP table. If it finds the destination IP address it will use the associated MAC address in the Ethernet frame. If it does not, it will use the ARP protocol to discover this MAC address, as shown in figure 14.



(a)

SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
X	
X	
X	

(b)

SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
X	
X	
X	

(c)

SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
3	00:0E:09:86:89:C2
X	
X	

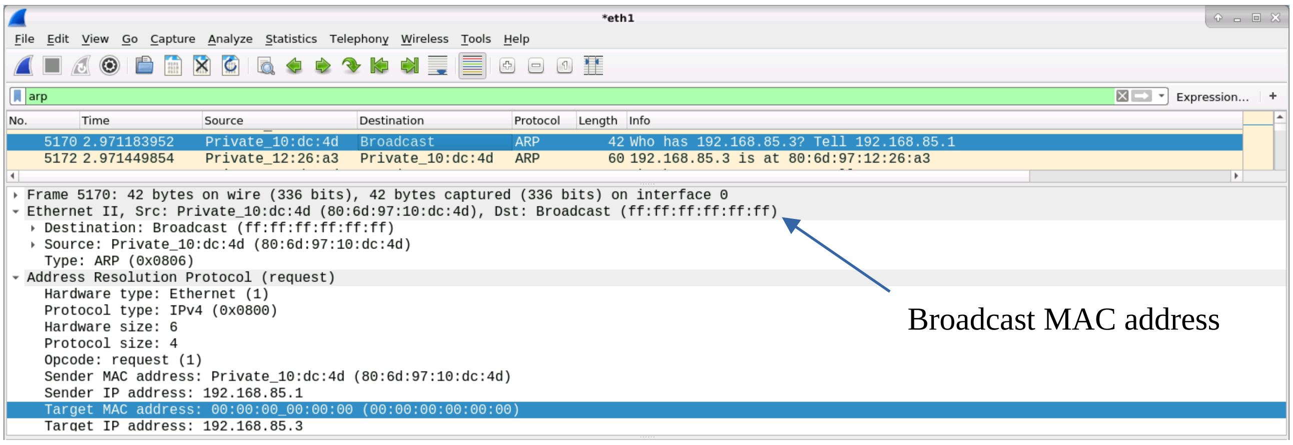
(d)

SAT	
1	E0:3F:49:B4:AE:D3
4	00:0E:09:86:8B:90
3	00:0E:09:86:89:C2
X	
X	

Figure 14 : ARP broadcast

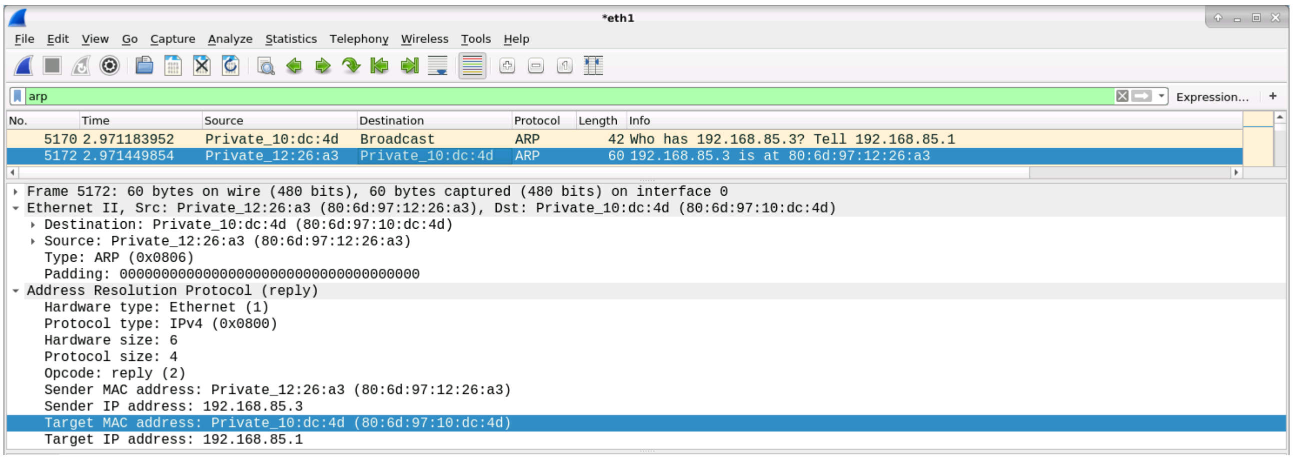
The Ethernet protocol defines a broadcast MAC address: `FF:FF:FF:FF:FF:FF`. All NICs receiving an ARP packet with this address will process this packet, allowing the receiving host to examine the destination (target) IP address field. In this example, this matches Pi-2's IP address. Therefore, Pi-2 will respond, replacing the broadcast and destination MAC address fields with the correct information.

Note, don't confuse this MAC broadcast address with the local IP broadcast address 255.255.255.255, they operate on different layers, they are completely separate addresses.


*eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Expression…
Broadcast MAC address

```
No.  Time          Source             Destination        Protocol Length Info
5170 2.971183952   Private_10:dc:4d   Broadcast          ARP      42 Who has 192.168.85.3? Tell 192.168.85.1
5172 2.971449854   Private_12:26:a3   Private_10:dc:4d   ARP      60 192.168.85.3 is at 80:6d:97:12:26:a3
```

```
▸ Frame 5170: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▾ Ethernet II, Src: Private_10:dc:4d (80:6d:97:10:dc:4d), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: Private_10:dc:4d (80:6d:97:10:dc:4d)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: Private_10:dc:4d (80:6d:97:10:dc:4d)
    Sender IP address: 192.168.85.1
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 192.168.85.3
```


*eth1
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
arp
Expression…

```
No.  Time          Source             Destination        Protocol Length Info
5170 2.971183952   Private_10:dc:4d   Broadcast          ARP      42 Who has 192.168.85.3? Tell 192.168.85.1
5172 2.971449854   Private_12:26:a3   Private_10:dc:4d   ARP      60 192.168.85.3 is at 80:6d:97:12:26:a3
```

```
▸ Frame 5172: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▾ Ethernet II, Src: Private_12:26:a3 (80:6d:97:12:26:a3), Dst: Private_10:dc:4d (80:6d:97:10:dc:4d)
  ▸ Destination: Private_10:dc:4d (80:6d:97:10:dc:4d)
  ▸ Source: Private_12:26:a3 (80:6d:97:12:26:a3)
    Type: ARP (0x0806)
    Padding: 000000000000000000000000000000000000
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: Private_12:26:a3 (80:6d:97:12:26:a3)
    Sender IP address: 192.168.85.3
    Target MAC address: Private_10:dc:4d (80:6d:97:10:dc:4d)
    Target IP address: 192.168.85.1
```

Figure 15 : ARP request (top), response (bottom)

When the transmitting host receives this updated packet it adds this IP and MAC address to its ARP table and is now able to transmit the TCP segment. To see this in action open the Wireshark GUI. Left click on (highlight) the Ethernet 1 (`eth1`) icon in the Interface list. Next enter the packet protocol filter :

```
arp
```

and click on the Apply button (this will remove most of the network "noise"). Next click on the Start icon on the top toolbar in Wireshark. Then in the Pi-1 SSH terminal enter the following command:

```
sudo arp -d 192.168.X.3       (X=Box/Desk)
```

This will delete the ARP table entry for Pi-3 `eth1`. However, you have no control over the different services running on Pi-1, so this address may be added back very quickly by a helpful process :). This isn't a big problem, you will still capture this ARP request, but assuming that this entry is deleted we can regenerate it by entering the following ping command:

```
ping -c 5 192.168.X.3          (X=Box/Desk)
```

You will now see some activity in the main window as Wireshark captures the packet associated with this ping command. Finally, click on the Stop icon ■ on the top toolbar to stop capturing network traffic. As we have deleted the ARP table entry for Pi-3, Pi-1 will send out an ARP request to obtain Pi-3's MAC address before it can transmit the ICMP ping. Check the ARP table on Pi-1 to confirm this has been updated, an example trace is shown in figure 15.

<u>Task</u> : delete the ARP table entries for other devices on the Raspberry Pi base system's network e.g. entries associated with the Mikrotik router, PC or other Raspberry Pis. Then generate ARP requests using their associated IP addresses. Capture these updates in Wireshark and examine the IP and MAC addresses used, do you understand how these addresses are used i.e. IP and MAC?

Task 4

In the previous lab we saw how routing between different networks is performed at layer 3 i.e. the network layer. However, as we have seen in the previous tasks Ethernet frames are transferred across the local network using layer 2 MAC addresses. Therefore, what IP and MAC addresses will be used when we route packets across our networks? Consider the example in figure 16, what IP and MAC addresses will be used when Pi-1 pings Ping-2 using its 172 address?

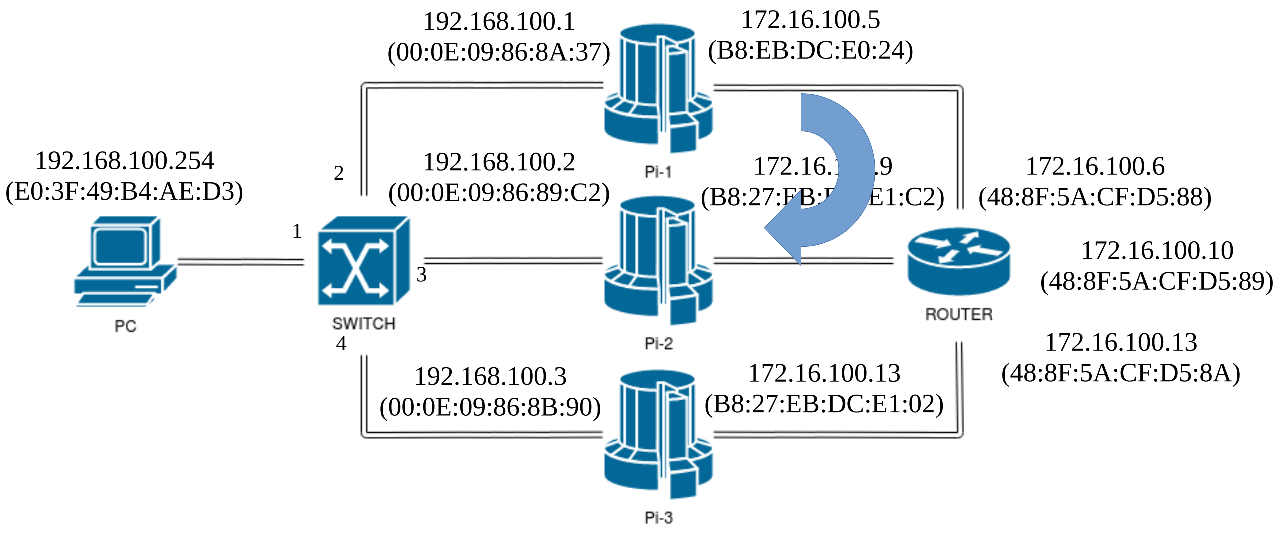


Figure 16 : How are ping packets routed from Pi-1 to Pi-2?

To help answer this question we can capture this packet trace. Close all open SSH terminals and VNC sessions. On the PC click on the start button and select the command prompt.

```
-> Command Prompt
```

then SSH into Pi-1 by entering the command below:

```
ssh pi@192.168.X.1                (X=Desk/Box)
```

Open a VNC session on Pi-1. Then within the VNC session start Wireshark, click on the menu icon and select :

```
Applications  -> Internet -> Wireshark
```

This will open the Wireshark GUI. Select the network interface (NIC) to capture packets on, left click on (highlight) the Ethernet 0 (`eth0`) in the Interface list and enter the packet protocol filter :

```
arp or icmp
```

and click on the Apply button. Next, click on the Start icon on the top toolbar in Wireshark. Then in the Pi-1 SSH terminal enter the following command:

```
sudo ip -s neigh flush all
```

This will remove all ARP entries from Pi-1's ARP table. You can then check Pi-1's ARP table using the command:

```
arp -n
```

Remember that this lookup is a race against time, as any process / service running on Pi-1 that needs to communicate across the `eth0` network will trigger ARP requests and update this table :). However, what you should see is that all ARP entries are removed except the one you are communicating across i.e. the SSH and VNC connections from the PC.

From Pi-1 ping Pi-2's `eth0` NIC using the ping command :

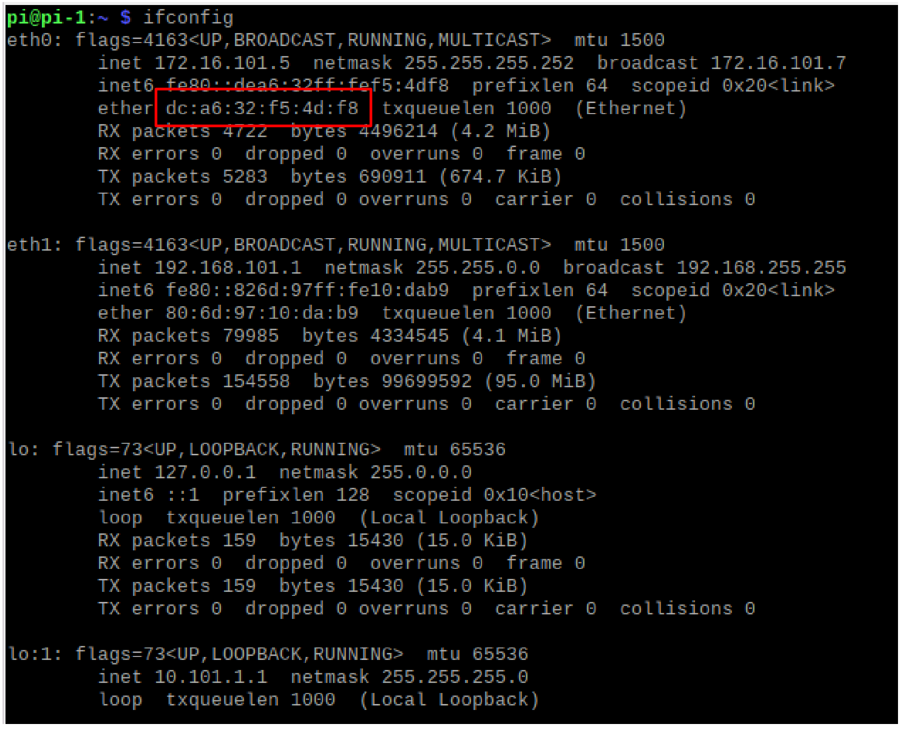
```
ping -c 2 172.16.X.9                    (X = Desk)
```

Finally, click on the Stop icon on the top toolbar to stop capturing network traffic.

Task : what MAC addresses are now present in Pi-1's ARP table? Do you understand why these entries are present and perhaps why some you thought should be there are not?

Hint, what does Pi-1 need to know to send these ping packets? Have a second look at figure 16, can you see why Pi-1 does not need to know Pi-2's MAC address?

To understand what MAC addresses have been used we need to identify what MAC addresses have been assigned to each NIC. For the Raspberry Pi this information can be identified using the `ifconfig` command as shown in figure 17.



```
pi@pi-1:~ $ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 172.16.101.5  netmask 255.255.255.252  broadcast 172.16.101.7
        inet6 fe80::dea6:32ff:fef5:4df8  prefixlen 64  scopeid 0x20<link>
        ether dc:a6:32:f5:4d:f8  txqueuelen 1000  (Ethernet)
        RX packets 4722  bytes 4496214 (4.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 5283  bytes 690911 (674.7 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

eth1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.101.1  netmask 255.255.0.0  broadcast 192.168.255.255
        inet6 fe80::826d:97ff:fe10:dab9  prefixlen 64  scopeid 0x20<link>
        ether 80:6d:97:10:da:b9  txqueuelen 1000  (Ethernet)
        RX packets 79985  bytes 4334545 (4.1 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 154558  bytes 99699592 (95.0 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (Local Loopback)
        RX packets 159  bytes 15430 (15.0 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 159  bytes 15430 (15.0 KiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

lo:1: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 10.101.1.1  netmask 255.255.255.0
        loop  txqueuelen 1000  (Local Loopback)
```

Figure 17 : `ifconfig` command

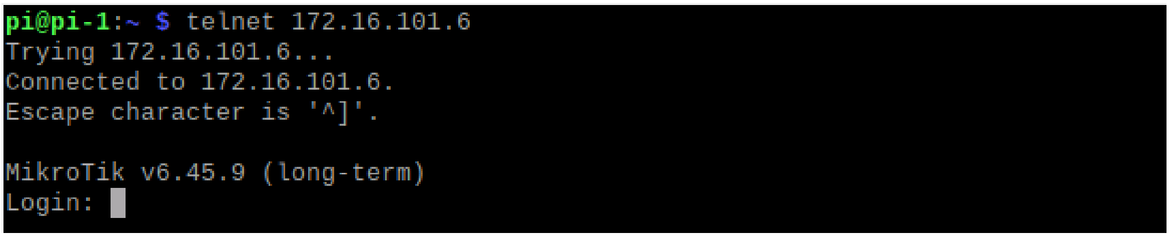
<u>Task</u> : identify the `eth0` MAC addresses assigned to Pi-1 and Pi-2.

To identify the MAC addresses assigned to each port of the MikroTik router we will need to telnet into the router. In the Pi-1 SSH terminal enter the command:

```
telnet 172.16.X.6             (X = Desk/Box)
```

This will produce the login prompt shown in figure 18, login using the username and password below:

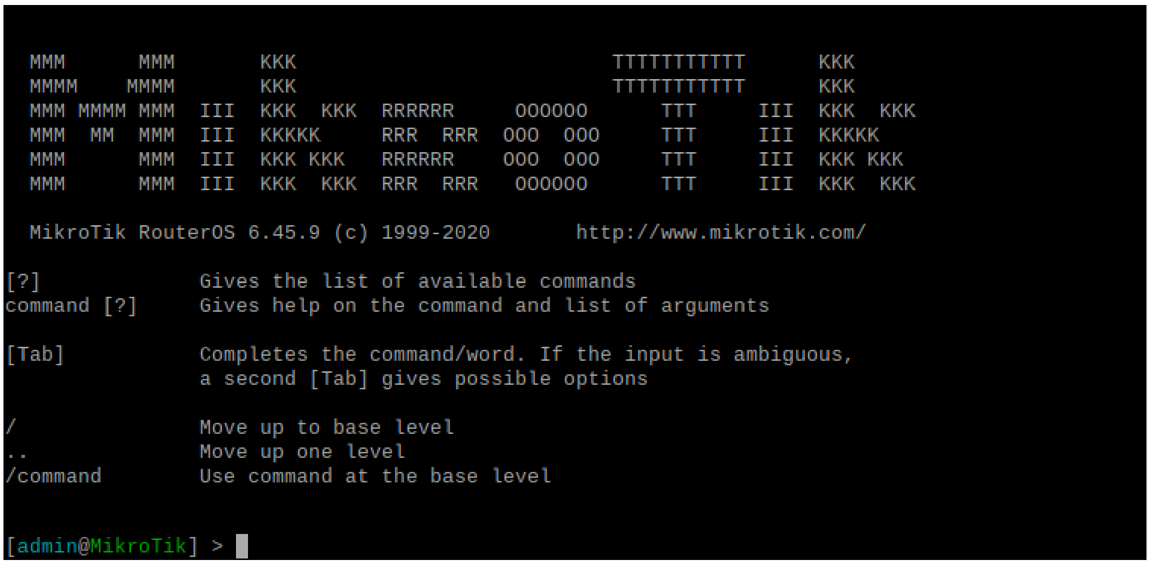
```
user name : admin
password : 12345
```



```
pi@pi-1:~ $ telnet 172.16.101.6
Trying 172.16.101.6...
Connected to 172.16.101.6.
Escape character is '^]'.

MikroTik v6.45.9 (long-term)
Login:
```

Figure 18 : MikroTik login prompt

If your login is successful the MikroTik welcome banner will be displayed, listing helpful commands, as shown in figure 24. Remember this is <u>not</u> a terminal you can not use `cd` or `ls` etc, this is the router's command line interface (CLI), they may look the same, but they are very different. In a router CLI you can only use the commands listed at each level e.g. use the "`?`" command to list the commands in the current level or "folder". Commands are displayed in **purple**, folders in **blue**. To change folder enter the folder name and press enter. To move back to the previous folder enter "`..`", to return to the root folder enter "`/`".



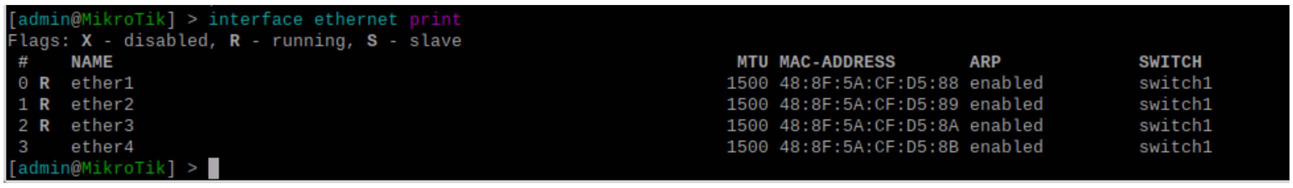
```
  MMM      MMM       KKK                          TTTTTTTTTTT      KKK
  MMMM    MMMM       KKK                          TTTTTTTTTTT      KKK
  MMM MMMM MMM  III  KKK  KKK  RRRRRR     OOOOOO      TTT     III  KKK  KKK
  MMM  MM  MMM  III  KKKKK     RRR  RRR  OOO  OOO     TTT     III  KKKKK
  MMM      MMM  III  KKK KKK   RRRRRR    OOO  OOO     TTT     III  KKK KKK
  MMM      MMM  III  KKK  KKK  RRR  RRR   OOOOOO      TTT     III  KKK  KKK

  MikroTik RouterOS 6.45.9 (c) 1999-2020       http://www.mikrotik.com/

[?]             Gives the list of available commands
command [?]     Gives help on the command and list of arguments

[Tab]           Completes the command/word. If the input is ambiguous,
                a second [Tab] gives possible options

/               Move up to base level
..              Move up one level
/command        Use command at the base level


[admin@MikroTik] >
```

Figure 19 : MikroTik login banner

At the prompt enter the command:

```
interface ethernet print
```

This will list the MAC addresses assigned to each port on the router, as shown in figure 20.



```
[admin@MikroTik] > interface ethernet print
Flags: X - disabled, R - running, S - slave
 #    NAME                                                   MTU MAC-ADDRESS       ARP             SWITCH
 0 R  ether1                                                1500 48:8F:5A:CF:D5:88 enabled         switch1
 1 R  ether2                                                1500 48:8F:5A:CF:D5:89 enabled         switch1
 2 R  ether3                                                1500 48:8F:5A:CF:D5:8A enabled         switch1
 3    ether4                                                1500 48:8F:5A:CF:D5:8B enabled         switch1
[admin@MikroTik] >
```

Figure 20 : MikroTik port MAC addresses

To exit the MikroTik command line interface enter the command:

```
quit
```

Task : from the previous Wireshark capture examine the ARP and ICMP packets generated by the ping command. Examine the MAC addresses used in the Ethernet frame headers. Do you understand what MAC and IP addresses have been used and why?

Hints, can you spot any MAC addresses in these packets that are associated with the Mikrotik router, or the Pis?

Task 5

The final layer in the Internet protocol stack is layer 1, the Physical layer. This layer defines the low level hardware used to transport the previous protocols. Wired Ethernet is a serial asynchronous protocol i.e. data is transmitted a bit at a time, with no clock (to show when each data bit is valid). Therefore, the receiving host has to determine the transmitting hosts clock speed i.e. its bit rate, how fast it is sending data e.g. 10/100/1000 Mbps. To do this the transmitting and receiving NICs need to synchronise their clocks. This is done by examining the pre-ample, a 7 byte block of alternating 1s and 0s i.e. 10101010, as shown in figures 21.

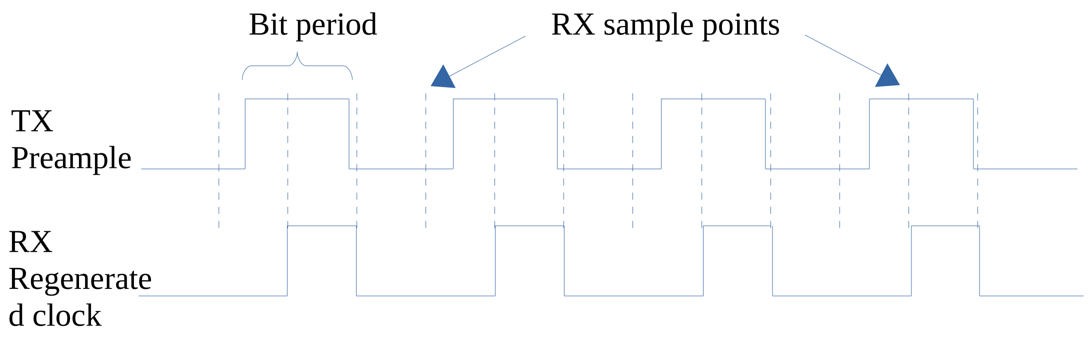


Figure 21 : preamble

By sampling this serial bit stream the receiving host can regenerate the transmitting clock i.e. calculate its frequency, such that it can determine when to capture each transmitted data bit.

Note, to regenerate a more accurate RX clock you would need more sample points than shown in figure 21. For more information on this topic refer to:

https://en.wikipedia.org/wiki/Nyquist%E2%80%93Shannon_sampling_theorem

To signal the end of the pre-ample a 1 byte Start of Frame Delimiter (SFD) bit pattern is used i.e. 10101011. Following this are the 48bit destination and source MAC addresses, and the encapsulated data (payload). The type of payload is identified by the TYPE field e.g. for IPv4 the type code is 0x08000. For more information on these different type codes refer to:

https://en.wikipedia.org/wiki/EtherType

At the end of the Ethernet frame there is a 32bit Frame Check Sequence (FCS), a Cyclic Redundancy Check (CRC) code that provides error detection for the complete Ethernet frame i.e. header + payload. A CRC produces a single checksum value for a block of data i.e. a CRC can not be used to check individual bytes, but they can detect if a Single Event Upset (SEU) has occurred within a larger block of data. This checksum is generated by XORing each block of data with a specific polynomial (key). Figure 22 shows a simple CRC calculation in action i.e. shows the steps involved in the TX and RX hosts.

Note, to simplify this example the data block is only six nibbles: 1011 0110 0000 0001 1111 1001 using the key 1001 i.e. a CRC-4 key. The Ethernet protocol uses a CRC-32 key. For more details on this technique refer to:

https://en.wikipedia.org/wiki/Cyclic_redundancy_check

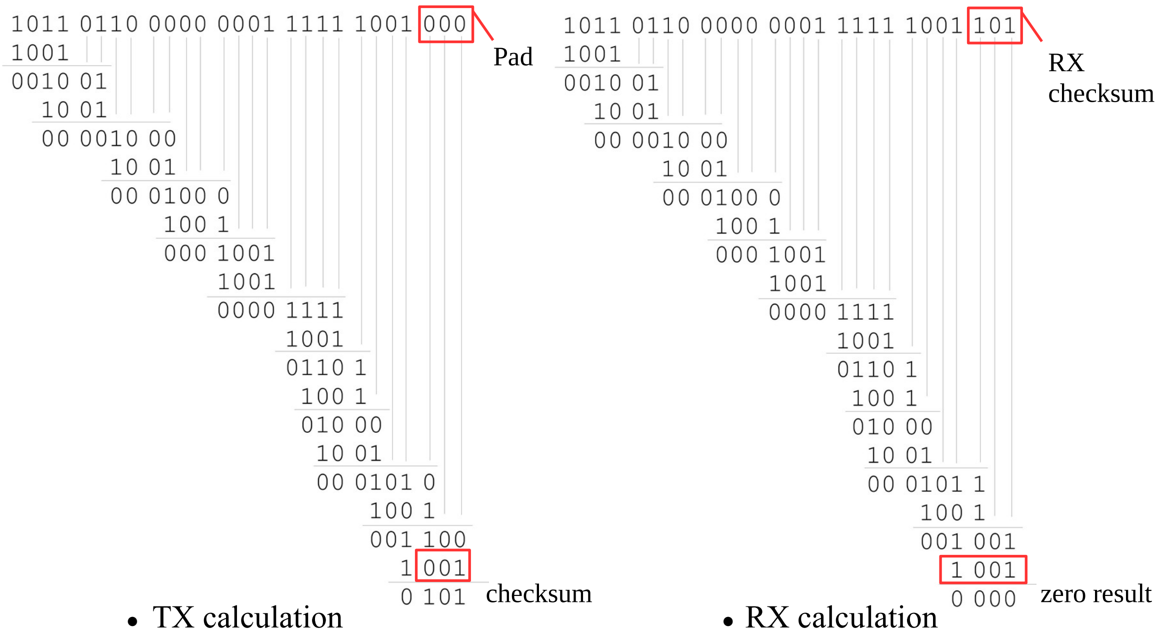


Figure 22 : CRC TX / RX calculations with no errors

To calculate the CRC, first pad onto the end of the data block length(key)-1 of “0” bits i.e. for a 4bit key this will be “000”. During the calculation leading zeros within the data block are skipped, the bit-wise XOR of the key and the first 1XXX data word is performed (X=don't care bits). The result of this calculation is then padded with the next data bits to produce the next 1XXX value. This process is repeated until all the data bits are used. The final 3-bit value is the CRC-4 checksum. This is appended to the TX data and transmitted to the RX host. The RX unit then repeats this XOR process. If the data value has been transmitted correctly the final result will be zero, otherwise an error (SEU) has occurred i.e. the data packet has been corrupted. Corrupted packets will be discarded by the receiving host.

Task : work through the calculation in figure 22 , make sure you can perform these calculations.

Note, remember there are also ones-complement addition checksums in the UDP, TCP and IP headers in addition to this CRC.

Task : during its transmission the data shown in figure 22 is corrupted to : 1001 0110 0000 0001 1111 1001. Rerun the CRC process performed by the RX host. Is this SEU detected?

Task 6

Finally we need to consider the actual wires. For Ethernet this is typically Category 5 cable (Cat 5) as shown in figure 23. In this example the plastic outer coating removed, so that you can see the individual wires. For more information on this cable refer to :

https://en.wikipedia.org/wiki/Category_5_cable

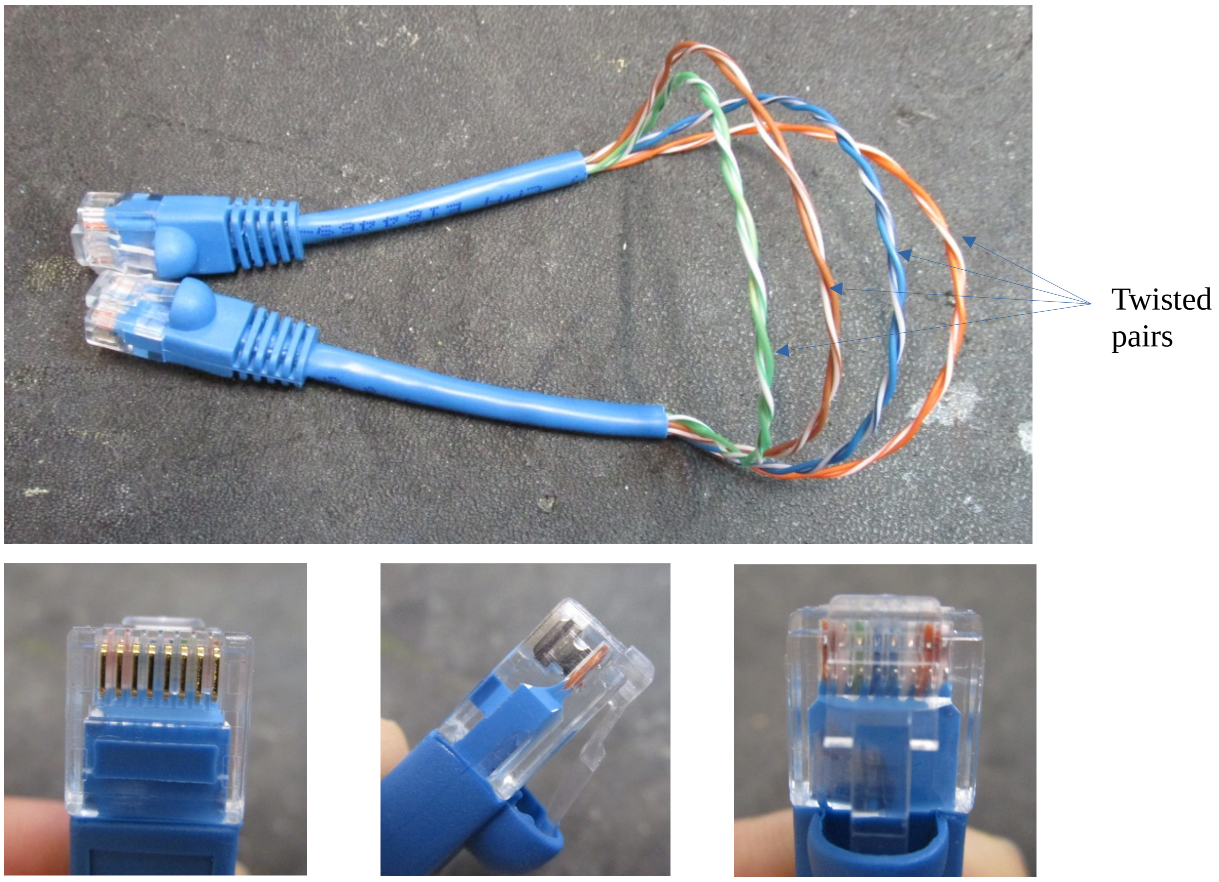


Figure 23 : Category 5 cable (Cat 5) (top), RJ45 connector (bottom)

Cat5 cable is constructed using four twisted pairs of cables i.e. 8 wires. However, the number of wires actually used to transmit data is dependent on the Ethernet standard used by the NIC. Common examples are listed in figure 24.

Name	IEEE Standard	Speed	Cable
10BASE-T	802.3i	10Mb/s	Cat 3
100BASE-TX	802.3u	100Mb/s	Cat 5
1000BASE-SX	802.3z	1Gb/s	Multi-Mode Fibre (MMF)
1000BASE-T	802.3ab	1Gb/s	Cat 5e
10GBASE-SR	802.3.ae	10Gb/s	Multi-Mode Fibre (MMF)
10GBASE-T	802.3an	10Gb/s	Cat 6A / Cat 7
40GBASE-T	802.3bq	40Gb/s	Cat 8
100GBASE-SR4	802.3bm	100Gb/s	Laser MMF

Figure 24 : Ethernet standards

These cables are terminated using 8 pin RJ45 connectors (Registered Jack). The network switch and router used in this Raspberry Pi system supports the IEEE802.3u standard, commonly referred to as 100BASE-TX i.e. 100Mb/s, Twisted pair, Fast Ethernet. This standard uses two pairs of wires to transfer data connected to socket pins 1,2,3 and 6, as shown in figure 25. One pair to transmit (TX) data and one pair to receive (RX) data i.e. the connection is fully duplex i.e. it can transmit and receive data at the same time.

To reduce costs these cables do not use electrical shielding i.e. are not wrapped in a conducting material to block electromagnetic interference. Instead they rely on differential signalling to remove electrical interference. Therefore, this type of cable is also called Unshielded Twisted Pair (UTP). More expensive cables e.g. Cat-6 or Cat-8 cables, contain additional shielding to reduce pickup from external noise sources or cross talk from internal wires.

Task : using the web page below, can you identify why the switch shown in figure 25 can not support the 1000BASE-T Ethernet standard?

https://en.wikipedia.org/wiki/Ethernet_over_twisted_pair

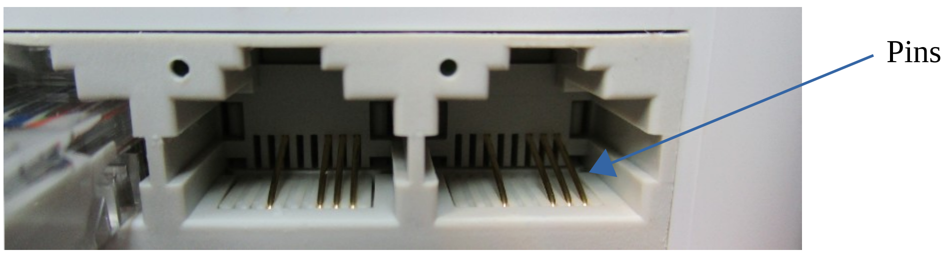


Figure 25 : RJ45 socket (port)

Optional Task : you are not truly a network engineer until you have made your own Cat5 cable. If you would like to crimp your own Ethernet cable, crimping tools, connectors and cable are available in the lab.

Appendix A : Lab server MAC and IP addresses

Compute-0

```
inet 172.16.200.2  netmask 255.255.255.0  broadcast 172.16.200.255
inet6 fe80::ba27:ebff:fecf:2c6a  prefixlen 64  scopeid 0x20<link>
ether b8:27:eb:cf:2c:6a  txqueuelen 1000  (Ethernet)
```

Compute-1

```
inet 172.16.200.3  netmask 255.255.255.0  broadcast 172.16.200.255
inet6 fe80::ba27:ebff:fe1e:5b9c  prefixlen 64  scopeid 0x20<link>
ether b8:27:eb:1e:5b:9c  txqueuelen 1000  (Ethernet)
```

Compute-2

```
inet 172.16.201.2  netmask 255.255.255.0  broadcast 172.16.201.255
inet6 fe80::ba27:ebff:fe0c:e1c2  prefixlen 64  scopeid 0x20<link>
ether b8:27:eb:0c:e1:c2  txqueuelen 1000  (Ethernet)
```

Compute-3

```
inet 172.16.201.3  netmask 255.255.255.0  broadcast 172.16.201.255
inet6 fe80::ba27:ebff:feeb:c0a  prefixlen 64  scopeid 0x20<link>
ether b8:27:eb:eb:0c:0a  txqueuelen 1000  (Ethernet)
```

Mail-server

```
inet 192.168.100.2  netmask 255.255.0.0  broadcast 192.168.255.255
inet6 fe80::ba27:ebff:fedc:e024  prefixlen 64  scopeid 0x20<link>
ether b8:27:eb:dc:e0:24  txqueuelen 1000  (Ethernet)

inet 172.16.200.1  netmask 255.255.255.0  broadcast 172.16.200.255
inet6 fe80::826d:97ff:fe12:27c6  prefixlen 64  scopeid 0x20<link>
ether 80:6d:97:12:27:c6  txqueuelen 1000  (Ethernet)

inet 172.16.201.1  netmask 255.255.255.0  broadcast 172.16.201.255
inet6 fe80::826d:97ff:fe10:dc38  prefixlen 64  scopeid 0x20<link>
ether 80:6d:97:10:dc:38  txqueuelen 1000  (Ethernet)
```