CA169 Networks Assignment Two

Answer Sheets

STUDENT NAME:	Jake Grogan
STUDENT NUMBER:	16456346
PROJECT NUMBER:	2
MODULE CODE:	CA169
DEGREE: {CA|EC|CPSSD|ECSA]	CA
LECTURER:	Brian Stone

Declaration

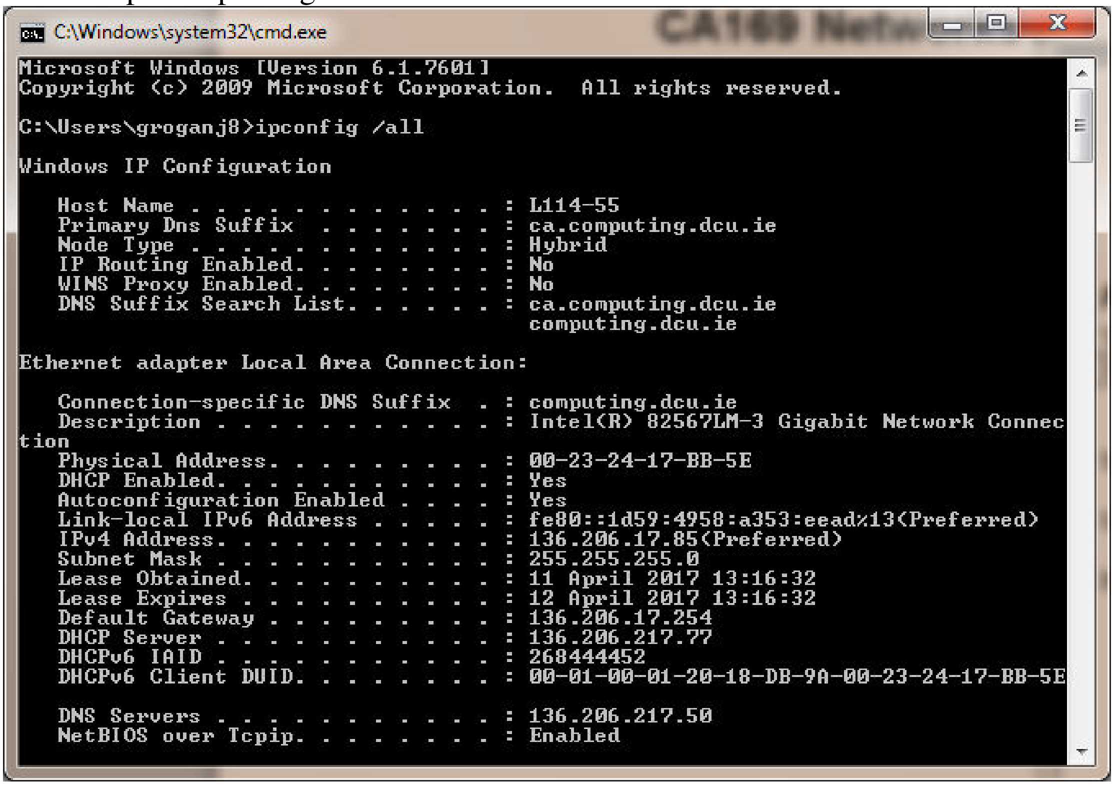
In submitting this project, I declare that the project material, which I now submit, is my own work. Any assistance received by way of borrowing from the work of others has been cited and acknowledged within the work. I make this declaration in the knowledge that a breach of the rules pertaining to project submission may carry serious consequences.

Part 1: DHCP traffic

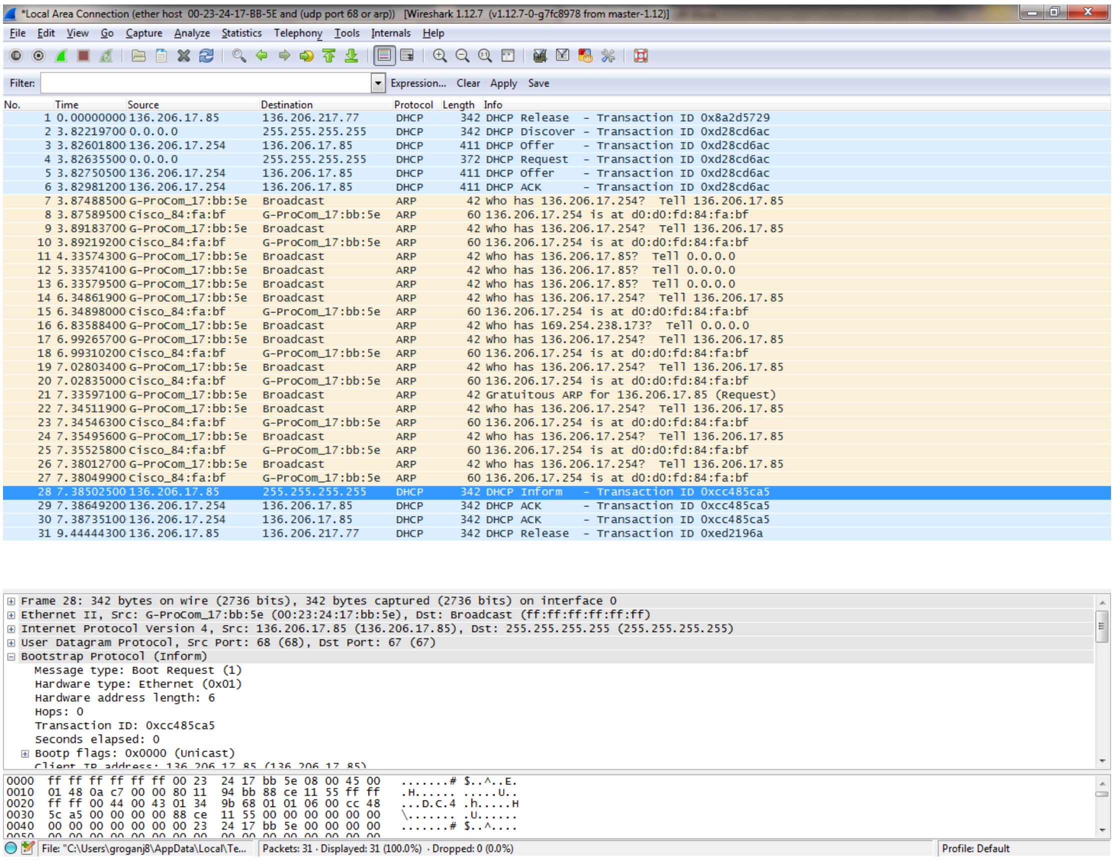
Your IP & MAC address for this experiment (use ipconfig)

IP = 136.206.17.85	MAC = 00-23-24-17-BB-5E

Screen capture: ipconfig information **cmd** window



Screen capture of Wireshark with DHCP and all ARP packets shown.



Packet numbers relevant to the DHCP interaction:

a. DHCP DISCOVER – Packet 2
b. DHCP OFFER – Packets 3, 5
c. DHCP Request – Packet 4
d. DHCP Acknowledgement – Packets 6, 29, 30
e. DHCP Release – Packets 1, 31
f. All ARP packets used – Packets 7 to 27

Function of each packet

a. DHCP DISCOVER

Packet 2:

This is a broadcast packet sent out across the network from the host machine looking for a DHCP server in order to obtain an IP address. It contains information including its MAC address and hostname.

b. DHCP OFFER

Packets 3, 5:

This is a message sent in reply to a DHCP Discover packet. It is from the DHCP server offering an IP address lease to the machine that sent the DHCP Discover packet.

c. DHCP Request

Packet 4:
This packet is broadcasted by the client in reply to the DHCP Offer packet. It is a packet accepting the offer of the IP address from the DHCP server.

d. DHCP Acknowledgement

Packets 6, 29, 30:
These are the packets sent by the DHCP server which acknowledges the request of the client machine. It also contains the length of the lease renewal.

e. DHCP Release

Packets 1, 31:
This is a packet sent from the client to the DHCP server which releases its IP address so that the DHCP server can lease it out to other clients.

f. ARP

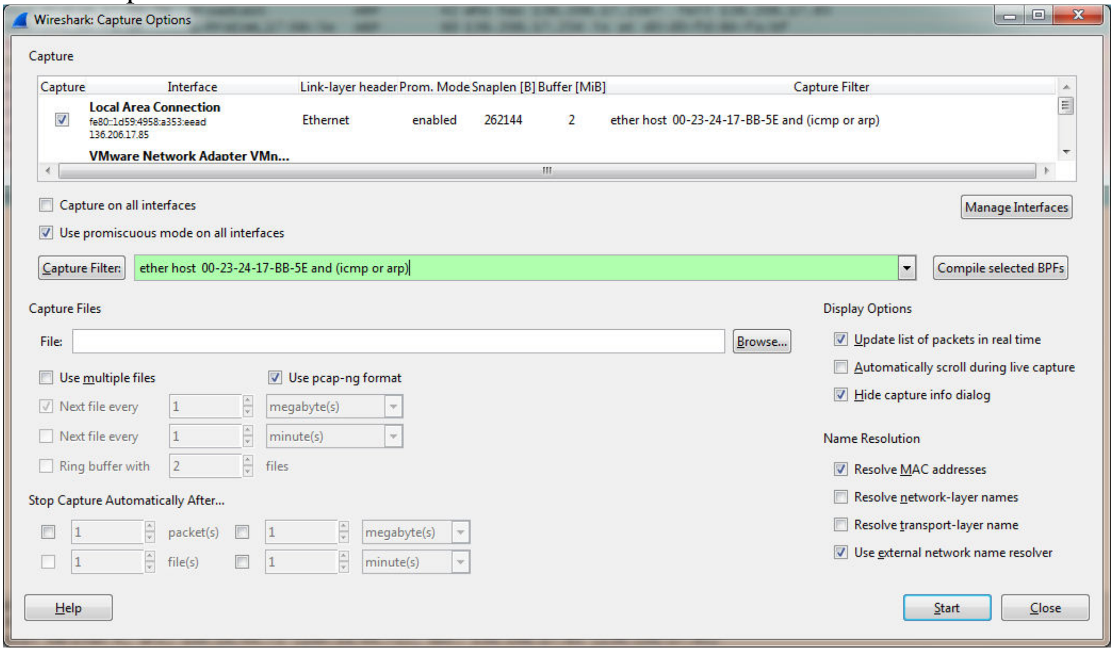
Packets 7 to 27:
In this capture there are ARP Requests and Replies. The ARP Requests are sent from the client over the network as a broadcast message (MAC: FF:FF:FF:FF:FF:FF). It contains the IP address of the machine it wishes to talk to. If the machine that the client is looking for is on the network and its IP address is the same as the one contained in the ARP Request, it sends back an ARP Reply containing its MAC address. Now both machines can communicate.
There is also a Gratuitous ARP packet in packet 21 which is sent from the client, out across the network, telling other hosts on the network about its new IP/MAC mapping.

Part 2: `ping` traffic

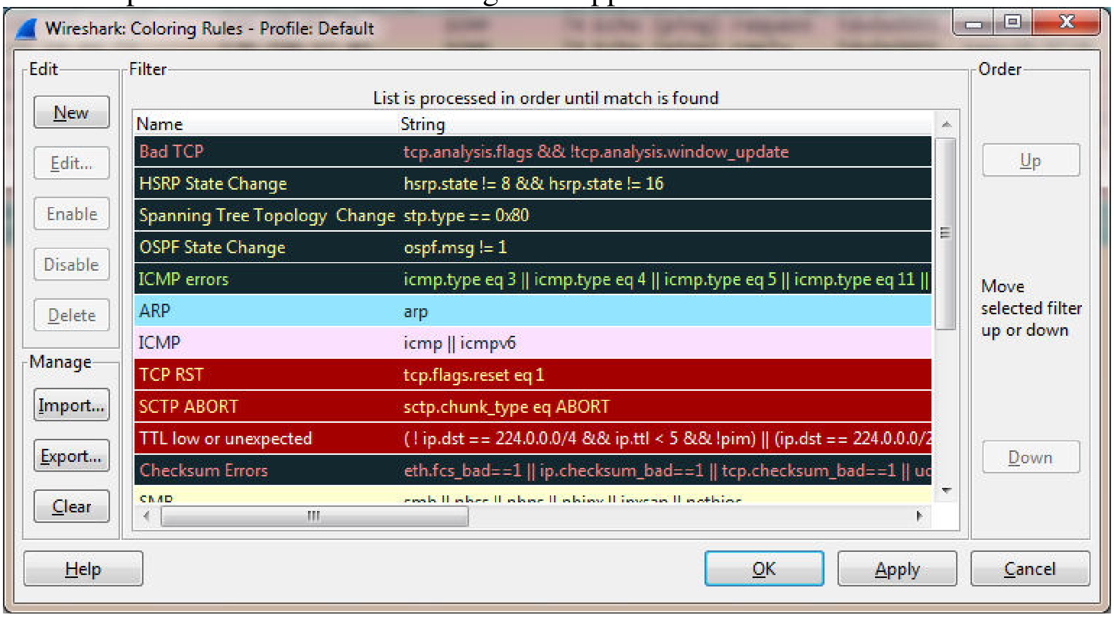
Your IP & MAC address for this experiment (use ipconfig)

136.206.17.85	00-23-24-17-BB-5E

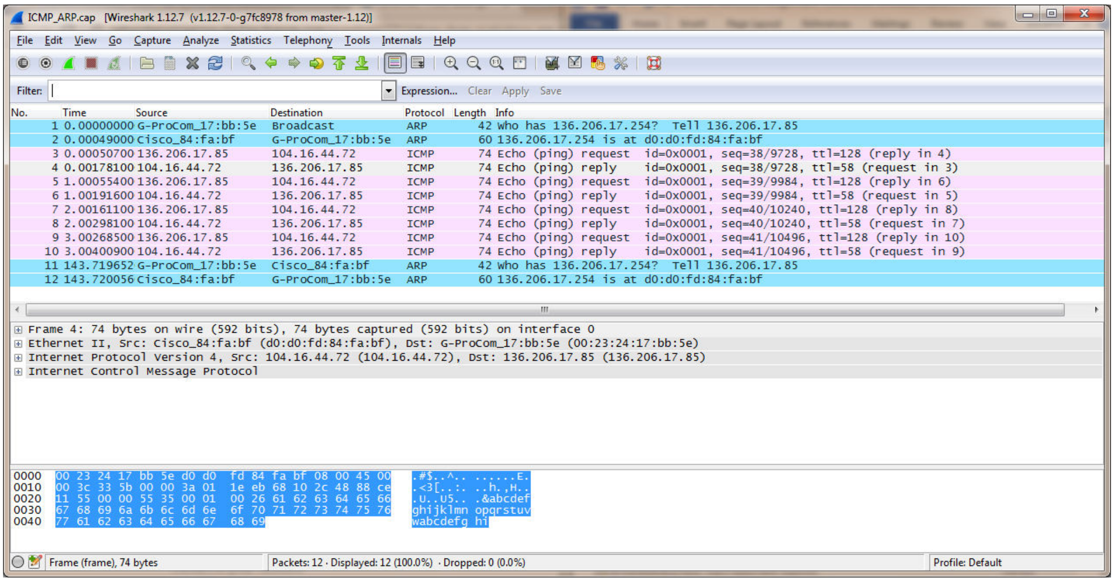
Screen capture of Wireshark filter utilised.



Screen capture of Wireshark colouring rules applied



Screen capture of Wireshark packet trace showing all relevant ping generated traffic, including ARP and ICMP traffic.



Packet numbers relevant to the experiment:

Packets 1 to 10 are relevant for this experiment

Explanation for each packet

For this experiment I pinged www.rte.ie after clearing the ARP cache.

- Function, why packet is generated and data contained in within it.

Packet 1: This is an ARP Request of length 42 bytes which is broadcasted across the network by the host machine asking which machine on the network has IP address 136.206.17.254, in this case, the router. The host machine needs the MAC address of 136.206.17.254 in order to communicate with it. It contains the host machines IP address, MAC address and the IP address of the target machine it wishes to communicate with.

Packet 2: This is the ARP Reply sent by 136.206.17.254 (Router), after receiving the ARP Request. When a machines IP address matches the IP address contained in the ARP Request, it sends back an ARP Reply containing its MAC address. Now the host machine and target machine can communicate. This process was necessary in order for the host machine to send the ICMP packets to www.rte.ie which follow.

Packet 3: This is the first ICMP Echo Request. It was sent when I pinged www.rte.ie . It is 74 bytes long. Its data payload is 32 bytes. ICMP is a basic way of checking if one machine can communicate with another. Inside the packet there is the Ethernet header which contains the destination address which is where the packet needs to go next, in this case, the router. Encapsulated by the Ethernet header is the IP header. This contains the IP address of www.rte.ie (the final destination) and inside of that, the ICMP header which contains information including the ICMP packet type indicating it is an Echo request and the data payload. It also has a TTL (time to live) which is used detect a packet which is caught in some form of routing loop. Each time the packet reaches a router its TTL is decremented. When it reaches zero it is discarded. Indicating it could not reach the target.

Packet 4: This is the ICMP Echo Reply. It was sent from 104.16.44.72 (RTE). It was sent in reply to Packet 3. Within the ICMP header is the type indicating it is an Echo Reply, identifier and sequence fields which can determine which stop replies and requests getting mixed up and its data payload. When the host machine receives an ICMP reply it now knows it can communicate with the target(RTE).

In Packets 5, 7 and 9 the ICMP Echo Request process is repeated.

The Packets 6, 8 and 10 are the ICMP Echo Replies from the above Requests.

The Echo Request and Reply process is repeated so we can get some statistics on the connection. We can find out the Round Trip times (time it takes from sending a request to receiving a reply) and amount of packet loss.

Part 3:

Your IP & MAC address for this experiment (use ipconfig)

IP = 136.206.17.80	MAC = 34-17-EB-B8-9C-18

For this experiment I connected to www.python.org

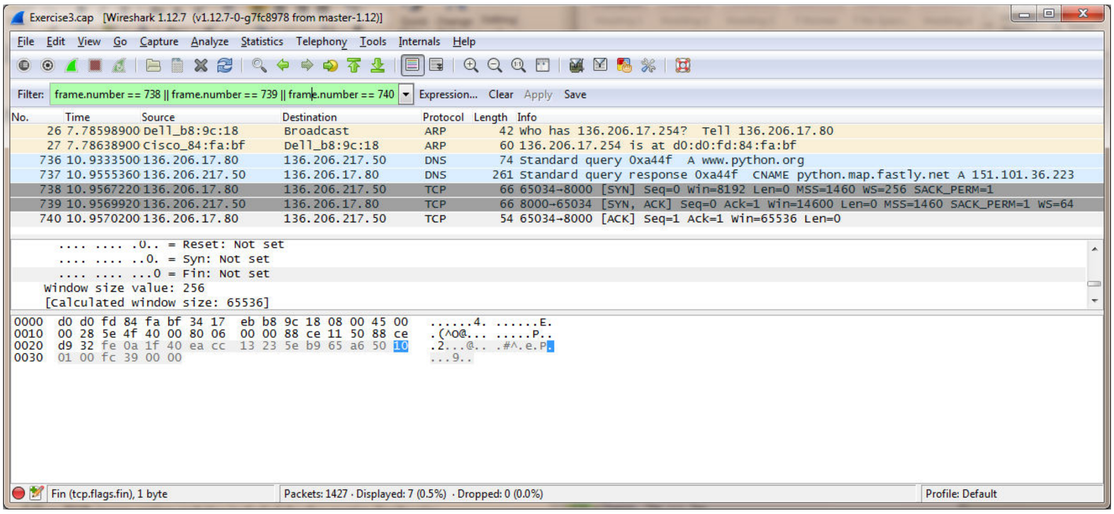
Filter to show only traffic concerning the test machine

Filter	tcp.stream eq 8 or dns contains “python”

Explain how you found the start of the interaction between your PC and the website.

> I found the interaction by starting the capture on Wireshark, loading the webpage then stopping the capture. I then entered the following display filter: “(*dns contains "python”or tcp) and eth.addr eq 34-17-EB-B8-9C-18*” this brought me to the packets where I could see the DNS queries for the website I connected to. Underneath these was the 3 way handshake and then the stream of data from the site. Then I chose the follow TCP stream option from the 3 way handshake and also included the DNS queries for the site, thus giving all traffic concerning the test machine.

Wireshark window showing the start of the interaction (should show ARP, DNS and TCP 3-way handshake)



Write down the numbers of the packets with the 3-way handshake.
Explain what is happening with these 3 packets.

> Packet 738 [SYN]: The SYN packet is sent by the host machine to the server. The purpose of this message is to ask the server if it is open for new connections.
> This packet contains a sequence number, j, which in this case is 0.
>
> Packet 739 [SYN/ACK]: When the server receives the SYN packet (Packet 738) it responds with a SYN/ACK packet indicating the host machine that sent

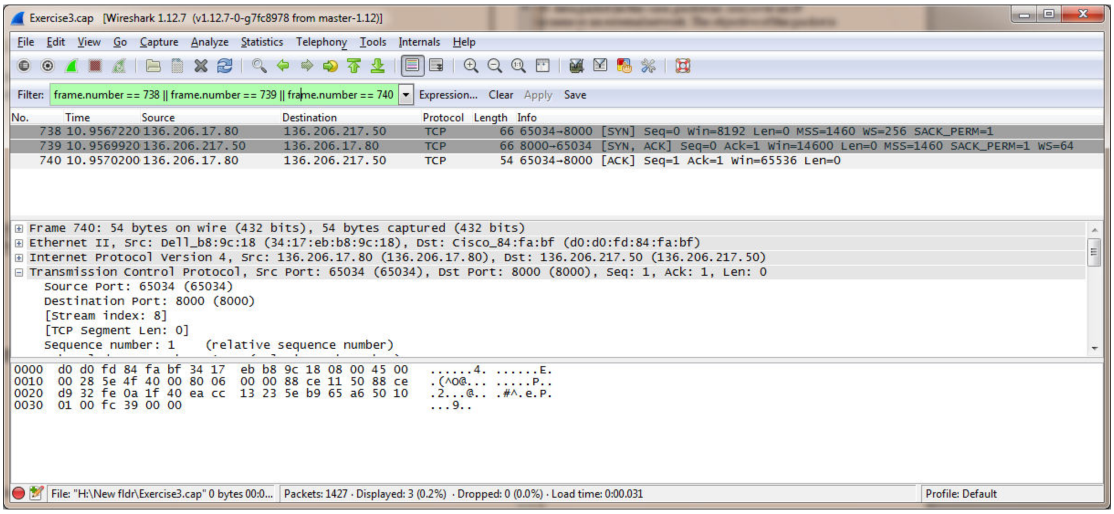
the SYN can connect to it. This packet contains a sequence number k, in this case and an ack number which is J + 1 which corresponds with SYN packet.

Packet 740 [ACK]: When the host machine receives the SYN/ACK packet (Packet 739) it responds with an ACK packet. Now a connection has been established. The ack number on this packet is K + 1 which corresponds with the previous SYN/ACK packet.

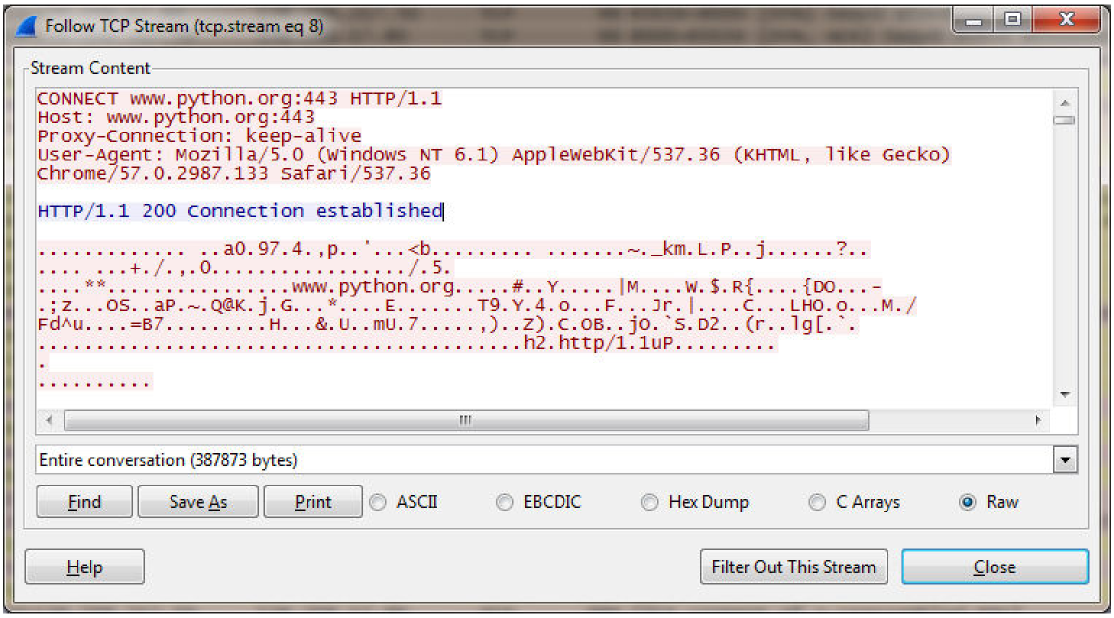
Write down a filter to show only these three-way-handshake packets

Filter	frame.number == 738 || frame.number == 739 || frame.number == 740

Wireshark window for the 3-way-handshake

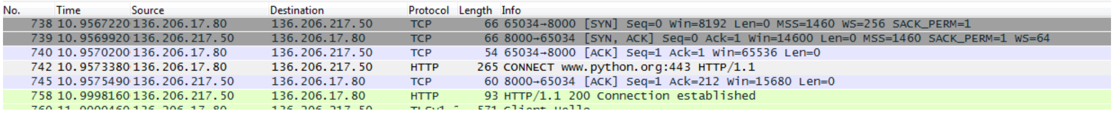


Show the **<u>Follow TCP Stream</u>** window here.



Your notes on…

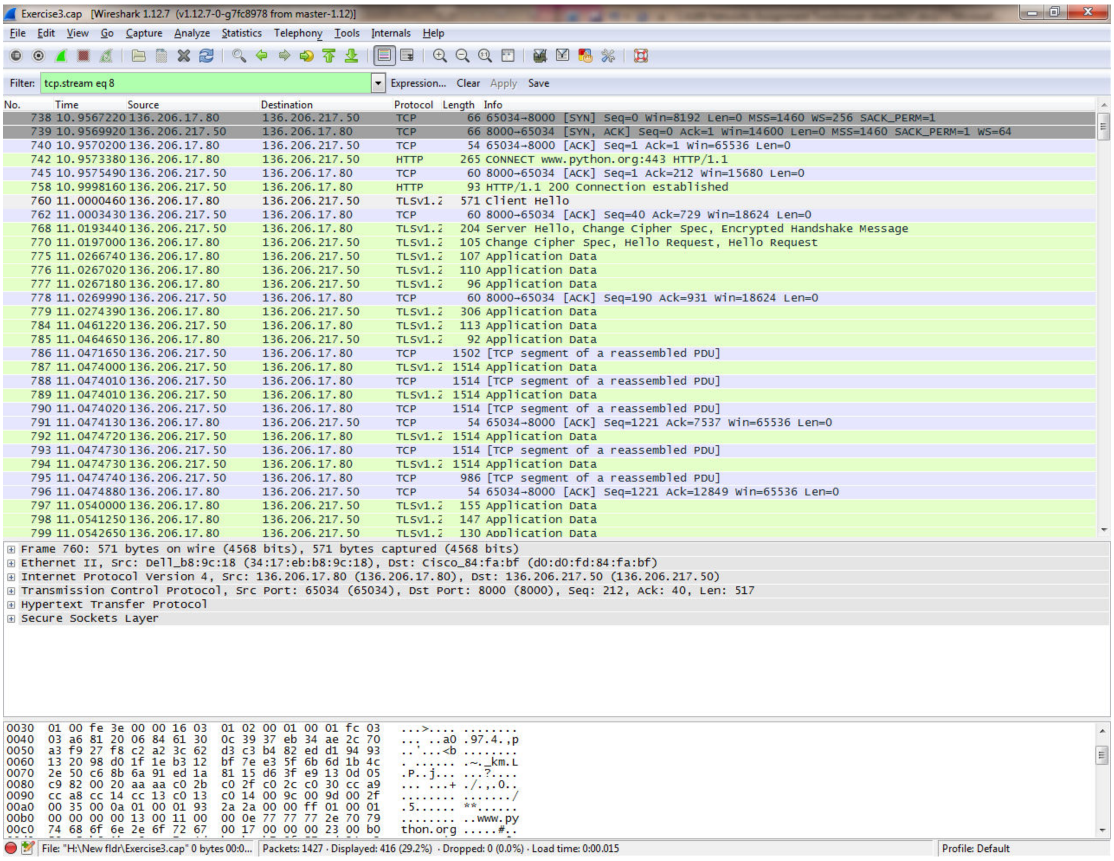
a. The GET requests made



No.	Time	Source	Destination	Protocol	Length	Info
738	10.9567220	136.206.17.80	136.206.217.50	TCP	66	65034→8000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
739	10.9569920	136.206.217.50	136.206.17.80	TCP	66	8000→65034 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=64
740	10.9570200	136.206.17.80	136.206.217.50	TCP	54	65034→8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0
742	10.9573380	136.206.17.80	136.206.217.50	HTTP	265	CONNECT www.python.org:443 HTTP/1.1
745	10.9575490	136.206.217.50	136.206.17.80	TCP	60	8000→65034 [ACK] Seq=1 Ack=212 Win=15680 Len=0
758	10.9998160	136.206.217.50	136.206.17.80	HTTP	93	HTTP/1.1 200 Connection established

I found a "CONNECT" request but no "GET". I looked into this and found out that this causes the proxy server to establish a HTTP tunnel between the two endpoints (my machine and the webserver) which allows my machine and the webserver to talk directly without the proxy being able to decrypt the bytes. It is called end-to-end encryption and utilised in https://. However it still requests the webpage from the webserver.

b. The responses from the server



No.	Time	Source	Destination	Protocol	Length	Info
738	10.9567220	136.206.17.80	136.206.217.50	TCP	66	65034→8000 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
739	10.9569920	136.206.217.50	136.206.17.80	TCP	66	8000→65034 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1460 SACK_PERM=1 WS=64
740	10.9570200	136.206.17.80	136.206.217.50	TCP	54	65034→8000 [ACK] Seq=1 Ack=1 Win=65536 Len=0
742	10.9573380	136.206.17.80	136.206.217.50	HTTP	265	CONNECT www.python.org:443 HTTP/1.1
745	10.9575490	136.206.217.50	136.206.17.80	TCP	60	8000→65034 [ACK] Seq=1 Ack=212 Win=15680 Len=0
758	10.9998160	136.206.217.50	136.206.17.80	HTTP	93	HTTP/1.1 200 Connection established
760	11.0000460	136.206.17.80	136.206.217.50	TLSv1.2	571	Client Hello
762	11.0003430	136.206.217.50	136.206.17.80	TCP	60	8000→65034 [ACK] Seq=40 Ack=729 Win=18624 Len=0
768	11.0193440	136.206.217.50	136.206.17.80	TLSv1.2	204	Server Hello, Change Cipher Spec, Encrypted Handshake Message
770	11.0197000	136.206.17.80	136.206.217.50	TLSv1.2	105	Change Cipher Spec, Hello Request, Hello Request
775	11.0266740	136.206.17.80	136.206.217.50	TLSv1.2	107	Application Data
776	11.0267020	136.206.17.80	136.206.217.50	TLSv1.2	110	Application Data
777	11.0267180	136.206.17.80	136.206.217.50	TLSv1.2	96	Application Data
778	11.0269990	136.206.217.50	136.206.17.80	TCP	60	8000→65034 [ACK] Seq=190 Ack=931 Win=18624 Len=0
779	11.0274390	136.206.17.80	136.206.217.50	TLSv1.2	306	Application Data
784	11.0461220	136.206.217.50	136.206.17.80	TLSv1.2	113	Application Data
785	11.0464650	136.206.17.80	136.206.217.50	TLSv1.2	92	Application Data
786	11.0471650	136.206.217.50	136.206.17.80	TCP	1502	[TCP segment of a reassembled PDU]
787	11.0474000	136.206.217.50	136.206.17.80	TLSv1.2	1514	Application Data
788	11.0474010	136.206.217.50	136.206.17.80	TCP	1514	[TCP segment of a reassembled PDU]
789	11.0474010	136.206.217.50	136.206.17.80	TLSv1.2	1514	Application Data
790	11.0474020	136.206.217.50	136.206.17.80	TCP	1514	[TCP segment of a reassembled PDU]
791	11.0474130	136.206.17.80	136.206.217.50	TCP	54	65034→8000 [ACK] Seq=1221 Ack=7537 Win=65536 Len=0
792	11.0474720	136.206.217.50	136.206.17.80	TLSv1.2	1514	Application Data
793	11.0474730	136.206.217.50	136.206.17.80	TCP	1514	[TCP segment of a reassembled PDU]
794	11.0474730	136.206.217.50	136.206.17.80	TLSv1.2	1514	Application Data
795	11.0474740	136.206.217.50	136.206.17.80	TCP	986	[TCP segment of a reassembled PDU]
796	11.0474880	136.206.17.80	136.206.217.50	TCP	54	65034→8000 [ACK] Seq=1221 Ack=12849 Win=65536 Len=0
797	11.0540000	136.206.17.80	136.206.217.50	TLSv1.2	155	Application Data
798	11.0541250	136.206.17.80	136.206.217.50	TLSv1.2	147	Application Data
799	11.0542650	136.206.17.80	136.206.217.50	TLSv1.2	130	Application Data

Frame 760: 571 bytes on wire (4568 bits), 571 bytes captured (4568 bits)
Ethernet II, Src: Dell_b8:9c:18 (34:17:eb:b8:9c:18), Dst: Cisco_84:fa:bf (d0:d0:fd:84:fa:bf)
Internet Protocol Version 4, Src: 136.206.17.80 (136.206.17.80), Dst: 136.206.217.50 (136.206.217.50)
Transmission Control Protocol, Src Port: 65034 (65034), Dst Port: 8000 (8000), Seq: 212, Ack: 40, Len: 517
Hypertext Transfer Protocol
Secure Sockets Layer

Packet 758: The initially response of the server is with a HTTP /1.1 200 Connection established.

Packets 768 and 770: These are the TSL handshakes when the client and server contact each other and choose the encryption keys they are going to use throughout the session.

Packets 775 onwards: These are data from the server and acknowledges.

c. The HTTP response codes used in the interaction and what they mean (look them up yourself on the Web)

   "HTTP /1.1 200 Connection Established" – The HTTP response code in this is 200 indicating "Ok" and that a connection has been established.

   There were no other HTTP response codes found.