Ethereal Tutorial

Zebulebu

Righto - here goes. Its pretty long, because its been written for second-line peeps at work who haven't got that much networking knowledge. I'm sure some of you will be able to skip the obvious bits...

What is Ethereal?

Long known as the Rolls-Royce of open-source Network Monitoring tools, Ethereal is a Network Packet Analyzer – allowing you to capture raw packet data and display it in a helpful form. This can be a massive benefit when attempting to troubleshoot misbehaving networks, and the functionality of Ethereal is simply staggering. This presentation will only scratch the surface of what Ethereal is actually capable of, but, once you understand the basics, you will be able to progress to more advanced capture techniques. Pretty soon, you'll be seeing the world in green 1nes and 0eroes, like Neo in the Matrix…

To explain what Ethereal does, its common to use the analogy of an electrician's voltmeter. Although the level of information that can be captured by Ethereal is far more detailed than the basic reading given off by a voltmeter, they perform the same basic function – allowing you to 'look inside the wire' and see what is going on.

Installing Ethereal

The installation process for Ethereal is pretty straightforward. It runs on almost every platform commonly in use today, including most flavours of UNIX, Macs, Linux and Windows. Since the most commonly used O/S is Windows, I'll cover the installation process for that platform here. If you wish to run it on another system, you can find detailed installation instructions here.

Ethereal requires the WinPCap driver installed underneath it to perform the packet capture process. You used to have to download this separately, happily, it now comes as part of the Ethereal program download itself. You can download the latest version of Ethereal here. One thing to bear in mind is that Ethereal is very much a constant 'work in progress'. Improvements to functionality and stability are made on an extremely regular basis, so it is definitely worth checking back to get an updated version regularly.

To start the installation, download the latest stable release from the link above and double-click the .exe file.

Click 'Next' on the Welcome screen

Click 'I Agree' on the License screen

The next screen is where you can fiddle with the installation components of Ethereal. Since the defaults are fine for what you will need, don't change anything, just click 'Next'

On the 'Additional Tasks' screen, decide whether you want shortcuts on your desktop and all that hoo-hah. This is up to you and your personal preferences, but make sure you leave the check box to associate capture files with Ethereal checked. Click 'Next'

Choose the location you want Ethereal installed in, then click 'Next'

The next screen tells you to install WinPCap. As stated above, WinPCap is the 'engine room' of Ethereal – without it, Ethereal cannot capture packets to analyse them. If you want an overview of what WinPCap is, click the button – this will provide you with a brief outline of what it is and does. Make sure you leave the check-box marked 'Install WinPCap' – you decide whether you want to allow non-admin users to capture. You might think this is a no-brainer, security wise, but it can often be helpful to run a network trace whilst logged in as a non-admin user, especially when troubleshooting problems caused by user rights. Just something to bear in mind when you make your choice! Once you've decided, click 'Install'

The installation process begins. It can take some time on slower PCs, but is usually reasonably fast (well under a minute in most cases). When its finished, the dialogue box will change to 'Installation Complete'. Click the 'Next' button one more time.

The Installation Completion dialogue will be displayed, asking you whether you want to receive an update on the current status of Ethereal ('Show News'), and whether you want to run the program there and then. These choices are up to you – it is usually well worth a visit to the Ethereal update page. If you want to dive straight in and start capturing, you may do so at this stage. However, it is still advisable to reboot following the installation – if only to clear out any temp files created by the install process.

Configuring Ethereal to Capture Packets

The configuration process once Ethereal is installed is extremely simple. All you need to do is 'point' Ethereal at the NIC you will be performing the capture from and make a couple of decisions about the manner in which you are going to capture traffic.

Ethereal's interface can seem utterly confusing at first, but once you have performed your first basic trace, you will be off and running and, believe it or not, this can be achieved within less than five minutes of installing it!

First off the bat, you need to configure Ethereal for your NIC. When you open Ethereal, an unfriendly grey window will be displayed. Fear not – the magic is about to begin! First of all, go to 'Capture', then 'Interfaces'. A list of the Network Interfaces on your PC will be displayed. All you need to do is select the interface you will be using, and Ethereal will go off and start peering into the wire. However, at this point it makes sense to configure a few properties on your chosen capture interface, so that Ethereal makes a little more sense to you than it otherwise might. In order to do this, click the 'Prepare' button associated with your capture interface.

This displays a dialogue box where you can configure all sorts of parameters for the capture, including whether you want to resolve names on the network, the timing of the capture, a predefined filter for the capture and a whole range of other options. All you need to concern yourself with at the moment are two of the options available to you.

Promiscuous Sniffing

The first is the check-box labelled 'Capture packets in promiscuous mode'. To understand what this means, you first need a fundamental understanding of how networks operate, and the difference between a 'Hub' and a 'Switch'. If your device is plugged into a hub, since hubs operate by broadcasting packets out of every interface, you will be able to 'sniff' all traffic flowing to and from any other devices connected to this hub. If, however, your device is connected to a switch, this will not ordinarily be the case, as each port on a switch acts as a 'Broadcast Domain' – meaning that only packets destined for the device associated with the MAC address registered to that switch port will be forwarded out of that port. To complicate matters further for you, this isn't strictly true, as all high-end network switches can make use of a feature known as 'Port Mirroring' or 'SPAN ports in Cisco-Speak which enables an administrator to configure an interface on a switch to receive a copy of traffic destined to one, more or all other ports on that switch. However, we won't go into that here – suffice it to say that if you are connected to a 10Mb hub, you may be able to capture in promiscuous mode, if you're connected to any other device (a 10/100Mb hub, a switch or a router for example) you probably won't be able to.

Once you've got your head round that, you may be wondering 'All well and good Zeb, but what on Earth does capturing packets in 'promiscuous' mode entail?'. Well, basically, it enables you to sniff ALL traffic travelling through a hub – not just the traffic destined either to your NIC alone or broadcast traffic. This can be exceptionally powerful – but is also very dangerous to you if, for instance, you're playing around with it at work without the requisite authority! Personally, at this stage of your understanding of Ethereal I would leave this check-box unticked!

Display Options

The second set of options you will want to look at is the 'Display Options' group. In order for you to see exactly what is going on during your trace, in real time, tick the 'Update list of packets in real time' check-box. Once you do this, you can also decide whether you want capture information to scroll up automatically as it is received. This is personal preference – I usually check it, but it is entirely up to you. For this tutorial I strongly advise that you check this box to start with, just so you can see what is going on during the capture. You can always go back and change it later if it gets on your nerves!

At this stage, it isn't worth bothering with applying a filter, since this tutorial will only perform a short capture you are unlikely to see a huge quantity of 'chatty' traffic – but remember later on that, once you are a little more familiar with Ethereal and the way it works, you can preconfigure a filter so that you only collect information relating to a particular application, Protocol, Network Address, MAC address or any number of other options. Did I mention Ethereal was ϑpowerful?

Your First Capture

Now that you've selected the options you require, you're ready to begin your first trace. Go ahead and click 'Start'. You will see that the Ethereal display immediately changes to show three separate windows. You may start to see some traffic instantly – if you have any applications open, or a web browser for instance. Indeed, if you leave this long enough, even without any applications being open you WILL start to see traffic hitting your NIC. This is the 'background noise' of your network, and it can be an interesting experience to watch this traffic, just to give you an idea of how 'chatty' your LAN is. If, for instance, you have any laser printers on your LAN (and I bet you do), you'd be surprised how often they advertise themselves to all and sundry via broadcasts…

Of course, this information isn't that much use to you – what you want to do is capture and analyse a 'real' trace. For that, we'll be using the good old PING, as its short, sharp and simple – whilst still allowing you to get a decent understanding of how to read Ethereal's captures. Incidentally, if you followed my earlier advice and set Ethereal to scroll in real time as packets are captured, you'll be able to follow along with the PING as it takes place. This will definitely aid your understanding of the Ethereal capture process.

Go ahead and open up a command prompt & PING another box on your LAN. It helps to choose a host you know is up – otherwise you may end up confusing yourself when trying to analyse the capture later! If you watch the top window as you perform the PING, you should see a number of lines appear. These lines are the information for each packet that enters or leaves your NIC. You'll recall that ICMP is the protocol used by PING. If you look at the 'Protocol' column, you should see the PING probe detailed as ICMP packets.

What you SHOULD see at this stage, is a series of four PING requests – made from your machine to the machine you PINGed. Each one of these requests SHOULD be followed by a reply from the machine you PINGed. All this, of course, is provided you kept the PING as the default option of four requests!

Once this process has completed, and the PING probe is finished, you can go ahead and click 'Stop'.

You have now completed your first trace using Ethereal! At the moment, this may not be much use to you, since all you have done is watch the PING transaction take place. In the next part of the tutorial, I'll show you how to analyse the trace so that you get a full understanding of how the different elements of Ethereal combine to give you a complete picture of the network session you are tracing unfolded.

Capture Analysis

In the first part of this tutorial we saw how to install Ethereal, configure your interfaces and begin capturing data. This part of the tutorial will cover the basics of analysing your capture files, giving you a basic understanding of how to use the interface, what each window shows and providing pointers to further learning. In the final part of the tutorial, I'll show you how to analyse a capture of something a little more exciting than the PING probe you made earlier.

For now, let's stick with the PING capture you performed in the first part of the tutorial. Providing you followed my advice and stopped capturing straight after your PING probe was complete, you should have no difficulty locating the packets that represent the PING session – they will be at, or near the bottom of the uppermost pane in the Ethereal interface. Helpfully, you'll notice that Ethereal colour-codes packets according to protocol. Your colour for ICMP packets may be different, but as you can see, mine is a lovely shade of violet.

The Ethereal interface

Before we go any further, lets take a closer look at the Ethereal interface so you can familiarise yourself with the different elements it contains. Right at the top of the screen is the traditional menu bar, with the equally traditional toolbar situated just underneath it. These contain commands & options just like any other program. Directly underneath the tool bar there is a series of three windows, displaying the data you have captured. The top window is called the Summary Window. This displays a single-line summary of each packet captured by the program. Underneath this is the Protocol Tree Window – which displays the packet in much more detail. The final pane is the Data View Window. This is where the raw data comprising the packet is displayed. OK – with that explanation out of the way, on with the analysis.

Understanding the Analysis Procedure

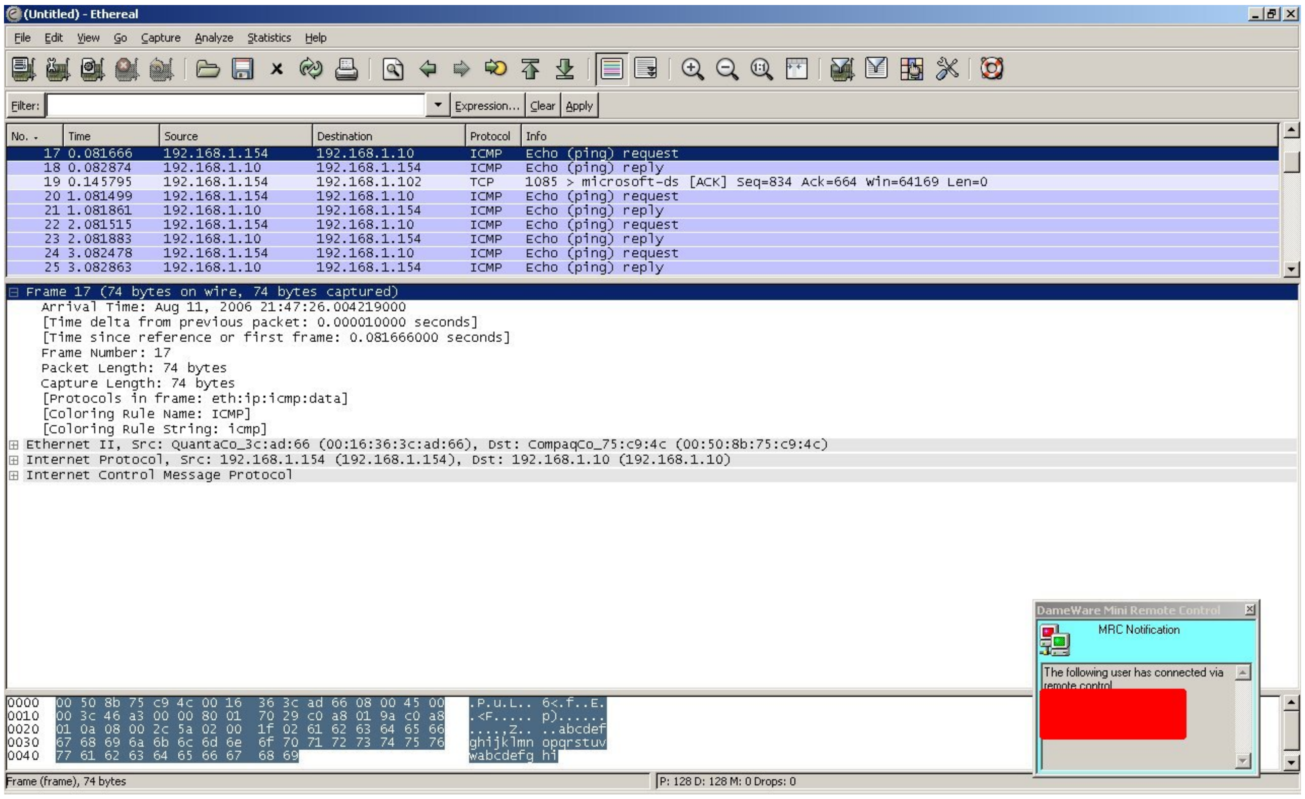
To begin with, let's examine the first packet that represents the start of your PING session. (Now don't get pedantic, I know that there are probably two ARP packets directly above requesting the MAC address of the IP you pinged that represent the REAL start of the PING probe – but we're only interested in the actual ICMP packets themselves at the minute – but give yourself a pat on the back if you were about to raise an objection, it shows you understand basic networking!)

If you highlight the first packet in your PING session by clicking on the row representing it in the Summary Window, you'll see this packet turns a dark blue colour – alerting you to the fact that it has been selected. You'll also notice the Protocol Tree Window now shows a different set of data than it did – this is the packet information for the first packet in your PING probe. This is where the meat of the information about the particular packet being examined is located. Whilst the Summary Window provides you with a basic idea of what a particular packet contains, the Protocol Tree Window shows you all the information contained in the packet, including headers, protocol information and payload.

So let's take a closer look at the Protocol Tree Window. All packets contain a set of information that both describes the packet and the data it contains. This information varies from protocol to protocol, but can contain things such as a sequence number, date/time information, source address, destination address, protocol type, MAC addresses and any of a whole host of other things – as well as the data itself. This can make the process of examining each packet a pretty complex and time-consuming task. Luckily, we've chosen an ICMP packet here, which are relatively easy to understand.

The easiest way to look at the Protocol Tree Window is to see each packet as a collection of fields and subfields. You'll see the '+' signs on the left-hand side – if you expand one of these you'll find information related to the particular element of the packet that it represents. In our case, the following elements are present:

The packet's 'descriptive' data, or 'Meta' Data


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
DameWare Mini Remote Control
MRC Notification

```
No. .  Time      Source         Destination    Protocol Info
  17 0.081666 192.168.1.154  192.168.1.10   ICMP   Echo (ping) request
  18 0.082874 192.168.1.10   192.168.1.154  ICMP   Echo (ping) reply
  19 0.145795 192.168.1.154  192.168.1.102  TCP    1085 > microsoft-ds [ACK] Seq=834 Ack=664 Win=64169 Len=0
  20 1.081499 192.168.1.154  192.168.1.10   ICMP   Echo (ping) request
  21 1.081861 192.168.1.10   192.168.1.154  ICMP   Echo (ping) reply
  22 2.081515 192.168.1.154  192.168.1.10   ICMP   Echo (ping) request
  23 2.081883 192.168.1.10   192.168.1.154  ICMP   Echo (ping) reply
  24 3.082478 192.168.1.154  192.168.1.10   ICMP   Echo (ping) request
  25 3.082863 192.168.1.10   192.168.1.154  ICMP   Echo (ping) reply
```

```
⊟ Frame 17 (74 bytes on wire, 74 bytes captured)
    Arrival Time: Aug 11, 2006 21:47:26.004219000
    [Time delta from previous packet: 0.000010000 seconds]
    [Time since reference or first frame: 0.081666000 seconds]
    Frame Number: 17
    Packet Length: 74 bytes
    Capture Length: 74 bytes
    [Protocols in frame: eth:ip:icmp:data]
    [Coloring Rule Name: ICMP]
    [Coloring Rule String: icmp]
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: CompaqCo_75:c9:4c (00:50:8b:75:c9:4c)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.10 (192.168.1.10)
⊞ Internet Control Message Protocol
```

```
0000  00 50 8b 75 c9 4c 00 16  36 3c ad 66 08 00 45 00   .P.u.L.. 6<.f..E.
0010  00 3c 46 a3 00 00 80 01  70 29 c0 a8 01 9a c0 a8   .<F..... p)......
0020  01 0a 08 00 2c 5a 02 00  1f 02 61 62 63 64 65 66   ....,Z.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

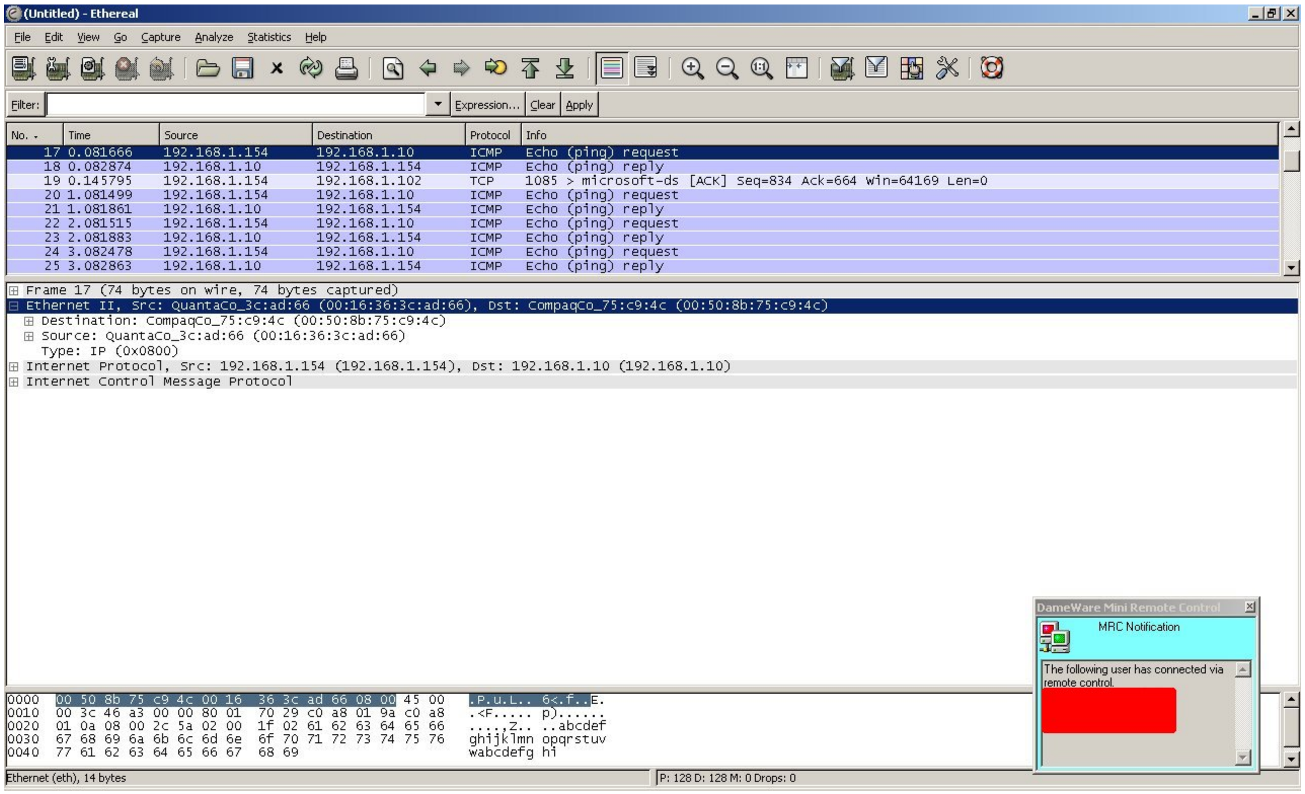
Frame (frame), 74 bytes | P: 128 D: 128 M: 0 Drops: 0

The following user has connected via remote control
[redacted]

This is data that describes the packet and includes Time/Data received, the Frame number, Frame length and a list of the protocols contained within the frame. In the case of this capture, we can see that the Frame Number is 17, that it was transmitted on August 11th at 21:47:26, it is 74 bytes and that the entire frame was captured (74 bytes on wire, 74 bytes captured). Note here that sometimes not all data in a packet is captured, for various reasons. Looking at the MetaData frame is a good way to ensure you have captured the entire packet.

A brief note is necessary here on the layered methodology of network communications. You'll recall from your basic networking training that network communication is a layered affair. Ethereal adopts a top-down approach to displaying the data at various layers of the OSI model – with the top entry in each packet captured being the lowest level of information (the DataLink layer), rising to the highest layer (the Application layer). Since the Packet MetaData is not part of the actual packet– it just describes it – this information should not be considered part of the packet capture itself

The DataLink (Layer 2) Information


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
DameWare Mini Remote Control
MRC Notification

```
No. .  Time      Source          Destination     Protocol Info
    17 0.081666  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    18 0.082874  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    19 0.145795  192.168.1.154   192.168.1.102   TCP      1085 > microsoft-ds [ACK] Seq=834 Ack=664 Win=64169 Len=0
    20 1.081499  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    21 1.081861  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    22 2.081515  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    23 2.081883  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    24 3.082478  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    25 3.082863  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
```

```
⊞ Frame 17 (74 bytes on wire, 74 bytes captured)
⊟ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: CompaqCo_75:c9:4c (00:50:8b:75:c9:4c)
  ⊞ Destination: CompaqCo_75:c9:4c (00:50:8b:75:c9:4c)
  ⊞ Source: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66)
    Type: IP (0x0800)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.10 (192.168.1.10)
⊞ Internet Control Message Protocol
```

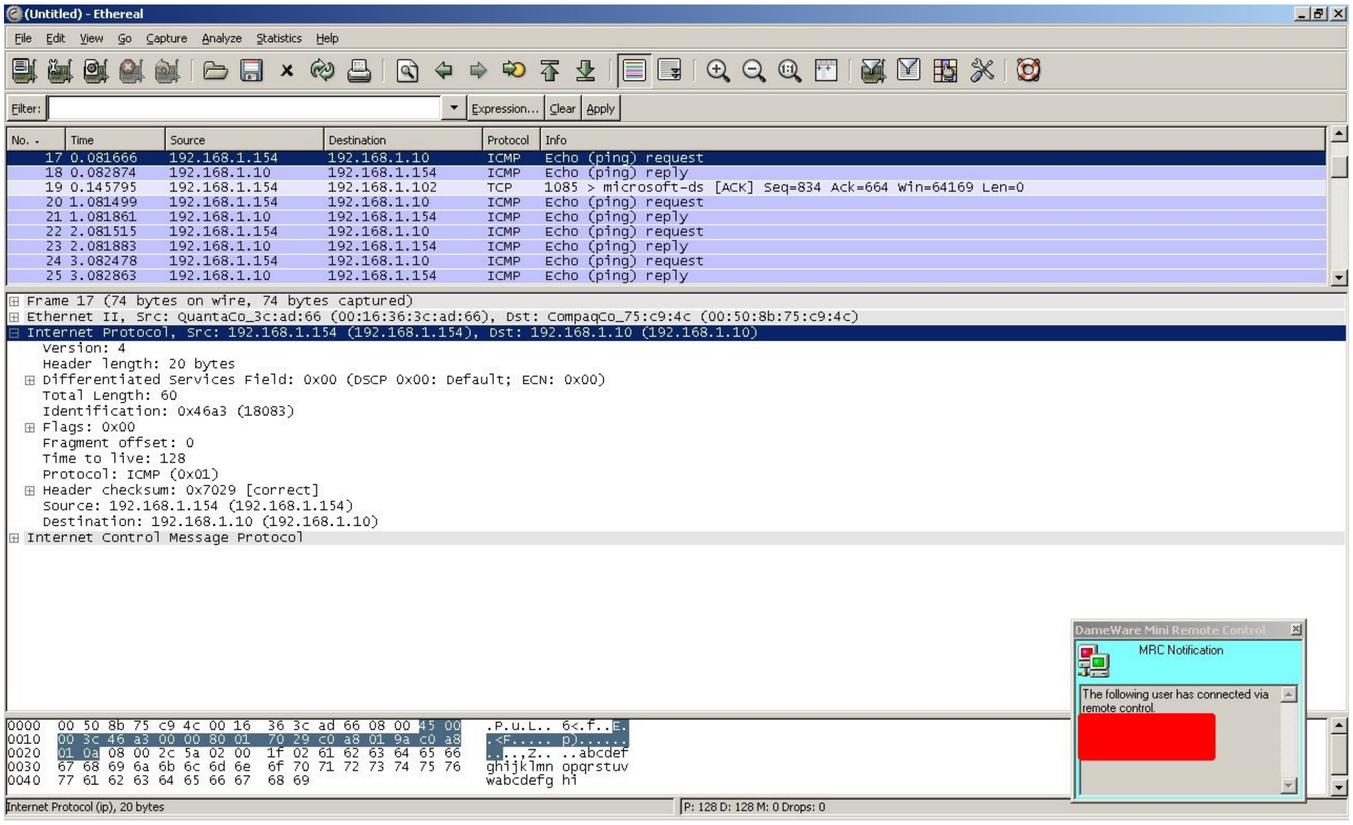
```
0000  00 50 8b 75 c9 4c 00 16  36 3c ad 66 08 00 45 00   .P.u.L.. 6<.f..E.
0010  00 3c 46 a3 00 00 80 01  70 29 c0 a8 01 9a c0 a8   .<F..... p)......
0020  01 0a 08 00 2c 5a 02 00  1f 02 61 62 63 64 65 66   ....,Z.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

Ethernet (eth), 14 bytes   P: 128 D: 128 M: 0 Drops: 0

The following user has connected via remote control.
[redacted]

Underneath the MetaData information is the information pertaining to the DataLink layer which, as any student of the OSI model knows, is the layer representing physical assembly of packets on the wire. In our case, this is the Ethernet II standard – you may also see other types such as Token Ring & FDDI, but the vast majority of the time you will be capturing over Ethernet. As you can see, expanding this layer in our capture shows two more expandable entries – one designating the Destination address, the other showing the Source address. This is an excellent example of the hierarchical nature of Ethereal – to begin with, all you can see for each entry is the description of the interface and it’s MAC address, but if you expand the node you will see more information. In this case it isn’t that much more exciting – just letting you know that the frame is a Unicast frame as opposed to a Broadcast or Multicast. It does, however, serve to highlight that there can be information in a capture that isn’t immediately apparent. Always expand the ‘+’ signs!

The Network Layer (Layer 3) Information


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
DameWare Mini Remote Control
MRC Notification

```
No. .  Time      Source          Destination     Protocol Info
    17 0.081666  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    18 0.082874  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    19 0.145795  192.168.1.154   192.168.1.102   TCP      1085 > microsoft-ds [ACK] Seq=834 Ack=664 Win=64169 Len=0
    20 1.081499  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    21 1.081861  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    22 2.081515  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    23 2.081883  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
    24 3.082478  192.168.1.154   192.168.1.10    ICMP     Echo (ping) request
    25 3.082863  192.168.1.10    192.168.1.154   ICMP     Echo (ping) reply
```

```
⊞ Frame 17 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: CompaqCo_75:c9:4c (00:50:8b:75:c9:4c)
⊟ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.10 (192.168.1.10)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 60
    Identification: 0x46a3 (18083)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: ICMP (0x01)
  ⊞ Header checksum: 0x7029 [correct]
    Source: 192.168.1.154 (192.168.1.154)
    Destination: 192.168.1.10 (192.168.1.10)
⊞ Internet Control Message Protocol
```

```
0000  00 50 8b 75 c9 4c 00 16  36 3c ad 66 08 00 45 00   .P.u.L.. 6<.f..E.
0010  00 3c 46 a3 00 00 80 01  70 29 c0 a8 01 9a c0 a8   .<F..... p)......
0020  01 0a 08 00 2c 5a 02 00  1f 02 61 62 63 64 65 66   ....,Z.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The following user has connected via remote control.
[redacted]

Internet Protocol (ip), 20 bytes | P: 128 D: 128 M: 0 Drops: 0

Now we start to see something a little more worthy of analysis. Since the Network layer protocol is much more interesting than the DataLink layer, there is much more information to be found here. For instance, we see that the Layer 3 protocol in use here is the Internet Protocol (IP). We can deduce from our capture that the two IP addresses assigned to the source & destination addresses are 192.168.1.154 and 192.168.1.10 respectively. We can also see that none of the various flags available in IP have been set, and that the TTL is set to 128. Finally, we note that the protocol that the IP encapsulates is, in this case, ICMP.

As you can see, there is far more to see in this layer. The Network layer information is often extremely revealing and investigation of this particular area of a packet can often lead to diagnosing the cause of network errors and discovering & analysing unauthorised network access (e.g. from Trojans)

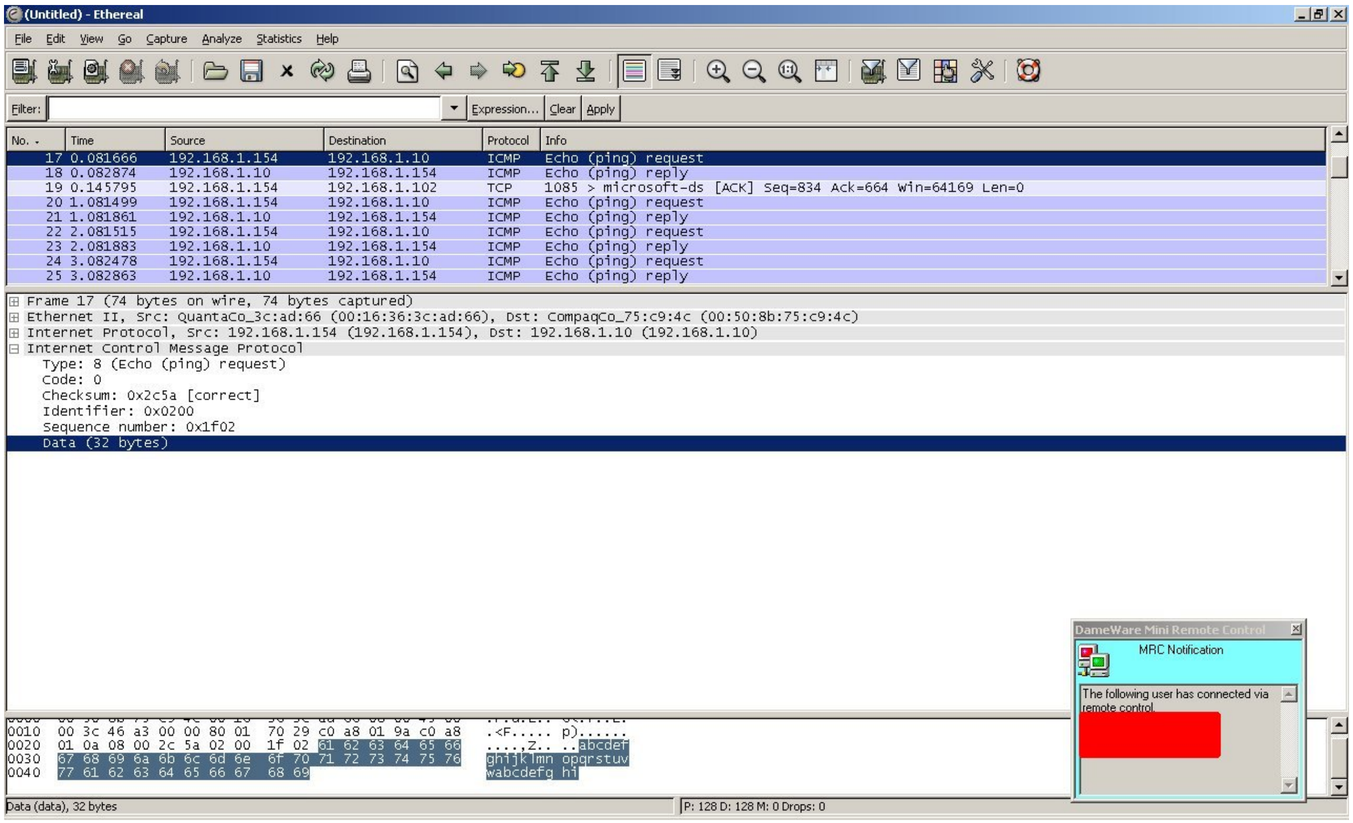
Since ICMP is not actually a part of the Network Layer, but is a feature of the IP protocol, it is displayed below in a separate entry. You can see here the information relating to the ping probe itself – ever wanted to know what type a PING request was in ICMP? Now you know – type 8! You can also see here that the PING was assembled correctly, and view the Sequence Number – this could be very handy for seeing whether transmissions are being received out of order.

Other Layers (Layer 4 and above)

Since our capture here is just to illustrate the features of Ethereal and demonstrate how to use it, we haven't got anything more exciting to look at. However, above Layer 3 sit the higher levels of the stack – Transport, Session, presentation and Application. In the final part of the tutorial, we'll look at capturing an HTTP request, where data pertaining to more layers of the stack will be captured.

For now though, this concludes the information to be found in the Protocol Tree Window. In the final part of this tutorial we'll look at the raw data that makes up the packet itself.

Raw Data


(Untitled) - Ethereal
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply
DameWare Mini Remote Control
MRC Notification

```
No. -  Time      Source         Destination    Protocol Info
  17 0.081666  192.168.1.154  192.168.1.10   ICMP     Echo (ping) request
  18 0.082874  192.168.1.10   192.168.1.154  ICMP     Echo (ping) reply
  19 0.145795  192.168.1.154  192.168.1.102  TCP      1085 > microsoft-ds [ACK] Seq=834 Ack=664 Win=64169 Len=0
  20 1.081499  192.168.1.154  192.168.1.10   ICMP     Echo (ping) request
  21 1.081861  192.168.1.10   192.168.1.154  ICMP     Echo (ping) reply
  22 2.081515  192.168.1.154  192.168.1.10   ICMP     Echo (ping) request
  23 2.081883  192.168.1.10   192.168.1.154  ICMP     Echo (ping) reply
  24 3.082478  192.168.1.154  192.168.1.10   ICMP     Echo (ping) request
  25 3.082863  192.168.1.10   192.168.1.154  ICMP     Echo (ping) reply
```

```
⊞ Frame 17 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: CompaqCo_75:c9:4c (00:50:8b:75:c9:4c)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.10 (192.168.1.10)
⊟ Internet Control Message Protocol
    Type: 8 (Echo (ping) request)
    Code: 0
    Checksum: 0x2c5a [correct]
    Identifier: 0x0200
    Sequence number: 0x1f02
    Data (32 bytes)
```

```
0010  00 3c 46 a3 00 00 80 01  70 29 c0 a8 01 9a c0 a8   .<F..... p)......
0020  01 0a 08 00 2c 5a 02 00  1f 02 61 62 63 64 65 66   ....,Z.. ..abcdef
0030  67 68 69 6a 6b 6c 6d 6e  6f 70 71 72 73 74 75 76   ghijklmn opqrstuv
0040  77 61 62 63 64 65 66 67  68 69                     wabcdefg hi
```

The following user has connected via remote control.
[redacted]

Data (data), 32 bytes | P: 128 D: 128 M: 0 Drops: 0

As you know, computers deal in raw data. The Data Window allows you to take a peek at the actual raw data making up a packet. If you highlight any of the rows in the Protocol Tree Window, you will see, in the bottom pane of Ethereal's interface, the raw data comprising that information. This will not often be easy to understand – to use a good example, let's look at the actual raw data of the PING request. If you highlight the 'Data' field in the ICMP tree within the Protocol Tree Window, you will see the raw data pertaining to this data in the Data View Window. As you can see – a series of Hexadecimal numbers is highlighted. This is the actual data of the 32 byte data field within the ICMP PING request. If you look to the right, you will see the data (again highlighted for you) represented in ASCII format. Did you know that PING requests contained a series of 32 characters in ascending order? Now you do!

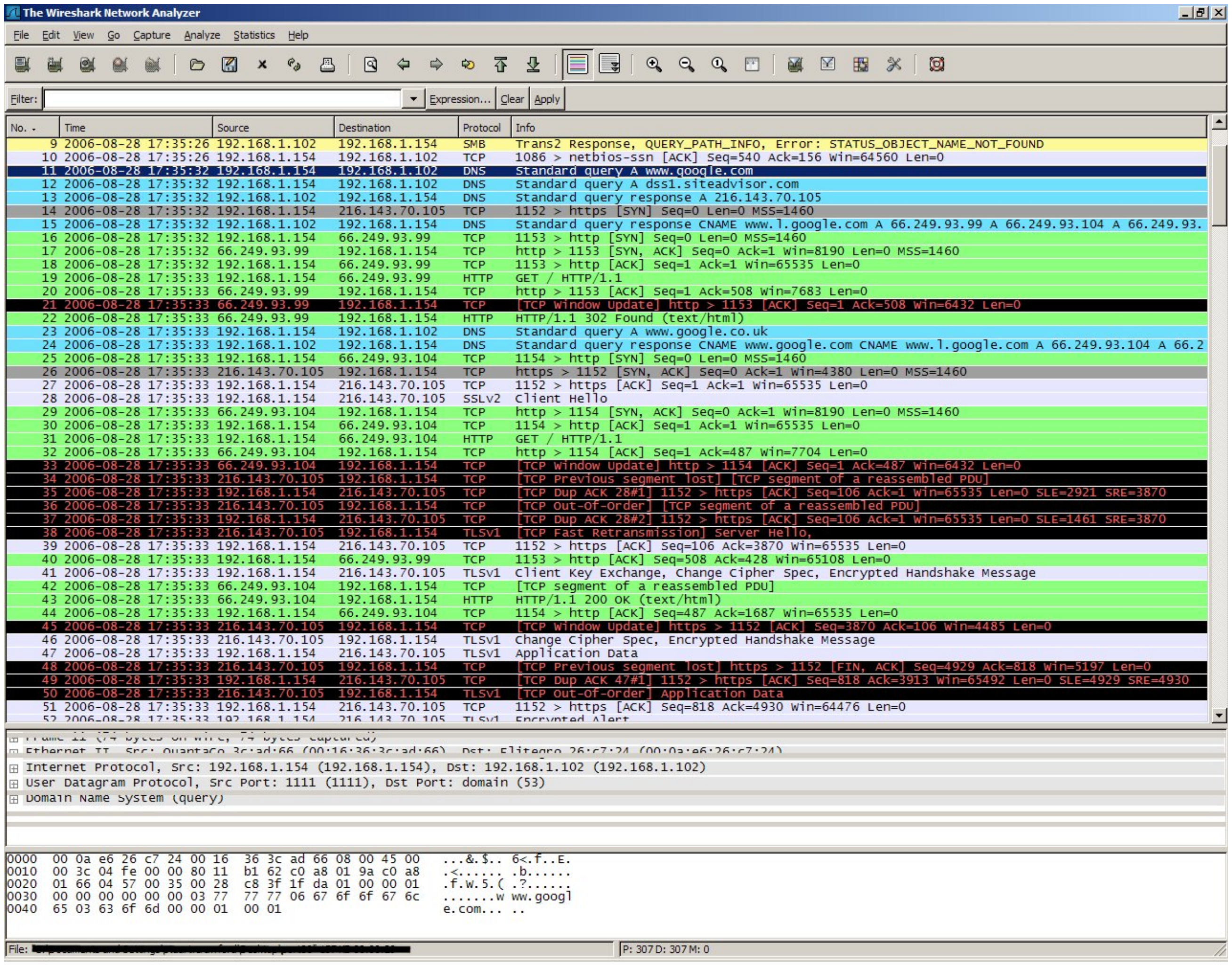
That pretty much concludes the basic overview of capture analysis. If you want to, go ahead and have a look at the other packets ion the PING session, and play around with looking at other types of packets you may have captured.

In the final tutorial we'll look at a capture which is more useful practically than a simple PING probe.

The final two parts of the tutorial will show you how Ethereal can be useful for analysing something a bit more detailed than the basic PING session we examined in the previous lesson. It may be a little more difficult to follow along with this one, but even if you choose not to, this article should help show you how to put Ethereal to practical use.

I've chosen to show you an analysis of some web traffic – specifically, opening your browser, running a search on Google than navigating to the site Google's search facility suggests best matches your search. Whilst not exactly mind-numbing in complexity, this analysis will give you a fair understanding of what it is Ethereal is capable of, and enable you to see how you might be able to put it to good use in other scenarios.

First thing to do is to fire up Ethereal and start a live capture (refer to the earlier lessons for a refresher if you need it). Once Ethereal is running, open the browser of your choice and (if Google isn't already your start page) navigate to Google. Type in 'certforums.co.uk' and hit return – Google should throw up a single link – click on this link to take you to the CertForums home page. Once you're there, you can stop Ethereal capturing live packets. Save the capture so that you can refer to it later if you want, then take a look at the capture in Ethereal's main interface.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression...
Clear
Apply

```
No. - Time                Source          Destination     Protocol Info
  9 2006-08-28 17:35:26 192.168.1.102   192.168.1.154   SMB    Trans2 Response, QUERY_PATH_INFO, Error: STATUS_OBJECT_NAME_NOT_FOUND
 10 2006-08-28 17:35:26 192.168.1.154   192.168.1.102   TCP    1086 > netbios-ssn [ACK] Seq=540 Ack=156 Win=64560 Len=0
 11 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS    Standard query A www.google.com
 12 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS    Standard query A dss1.siteadvisor.com
 13 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS    Standard query response A 216.143.70.105
 14 2006-08-28 17:35:32 192.168.1.154   216.143.70.105  TCP    1152 > https [SYN] Seq=0 Len=0 MSS=1460
 15 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS    Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
 16 2006-08-28 17:35:32 192.168.1.154   66.249.93.99    TCP    1153 > http [SYN] Seq=0 Len=0 MSS=1460
 17 2006-08-28 17:35:32 66.249.93.99    192.168.1.154   TCP    http > 1153 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
 18 2006-08-28 17:35:32 192.168.1.154   66.249.93.99    TCP    1153 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
 19 2006-08-28 17:35:33 192.168.1.154   66.249.93.99    HTTP   GET / HTTP/1.1
 20 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   TCP    http > 1153 [ACK] Seq=1 Ack=508 Win=7683 Len=0
 21 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   TCP    [TCP Window Update] http > 1153 [ACK] Seq=1 Ack=508 Win=6432 Len=0
 22 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   HTTP   HTTP/1.1 302 Found (text/html)
 23 2006-08-28 17:35:33 192.168.1.154   192.168.1.102   DNS    Standard query A www.google.co.uk
 24 2006-08-28 17:35:33 192.168.1.102   192.168.1.154   DNS    Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
 25 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   TCP    1154 > http [SYN] Seq=0 Len=0 MSS=1460
 26 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TCP    https > 1152 [SYN, ACK] Seq=0 Ack=1 Win=4380 Len=0 MSS=1460
 27 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    1152 > https [ACK] Seq=1 Ack=1 Win=65535 Len=0
 28 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  SSLv2  Client Hello
 29 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   TCP    http > 1154 [SYN, ACK] Seq=0 Ack=1 Win=8190 Len=0 MSS=1460
 30 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   TCP    1154 > http [ACK] Seq=1 Ack=1 Win=65535 Len=0
 31 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   HTTP   GET / HTTP/1.1
 32 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   TCP    http > 1154 [ACK] Seq=1 Ack=487 Win=7704 Len=0
 33 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   TCP    [TCP Window Update] http > 1154 [ACK] Seq=1 Ack=487 Win=6432 Len=0
 34 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TCP    [TCP Previous segment lost] [TCP segment of a reassembled PDU]
 35 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    [TCP Dup ACK 28#1] 1152 > https [ACK] Seq=106 Ack=1 Win=65535 Len=0 SLE=2921 SRE=3870
 36 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TCP    [TCP Out-Of-Order] [TCP segment of a reassembled PDU]
 37 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    [TCP Dup ACK 28#2] 1152 > https [ACK] Seq=106 Ack=1 Win=65535 Len=0 SLE=1461 SRE=3870
 38 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TLSv1  [TCP Fast Retransmission] Server Hello,
 39 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    1152 > https [ACK] Seq=106 Ack=3870 Win=65535 Len=0
 40 2006-08-28 17:35:33 192.168.1.154   66.249.93.99    TCP    1153 > http [ACK] Seq=508 Ack=428 Win=65108 Len=0
 41 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TLSv1  Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
 42 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   TCP    [TCP segment of a reassembled PDU]
 43 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   HTTP   HTTP/1.1 200 OK (text/html)
 44 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   TCP    1154 > http [ACK] Seq=487 Ack=1687 Win=65535 Len=0
 45 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TCP    [TCP Window Update] https > 1152 [ACK] Seq=3870 Ack=106 Win=4485 Len=0
 46 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TLSv1  Change Cipher Spec, Encrypted Handshake Message
 47 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TLSv1  Application Data
 48 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TCP    [TCP Previous segment lost] https > 1152 [FIN, ACK] Seq=4929 Ack=818 Win=5197 Len=0
 49 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    [TCP Dup ACK 47#1] 1152 > https [ACK] Seq=818 Ack=3913 Win=65492 Len=0 SLE=4929 SRE=4930
 50 2006-08-28 17:35:33 216.143.70.105  192.168.1.154   TLSv1  [TCP Out-Of-Order] Application Data
 51 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TCP    1152 > https [ACK] Seq=818 Ack=4930 Win=64476 Len=0
 52 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  TLSv1  Encrypted Alert
```

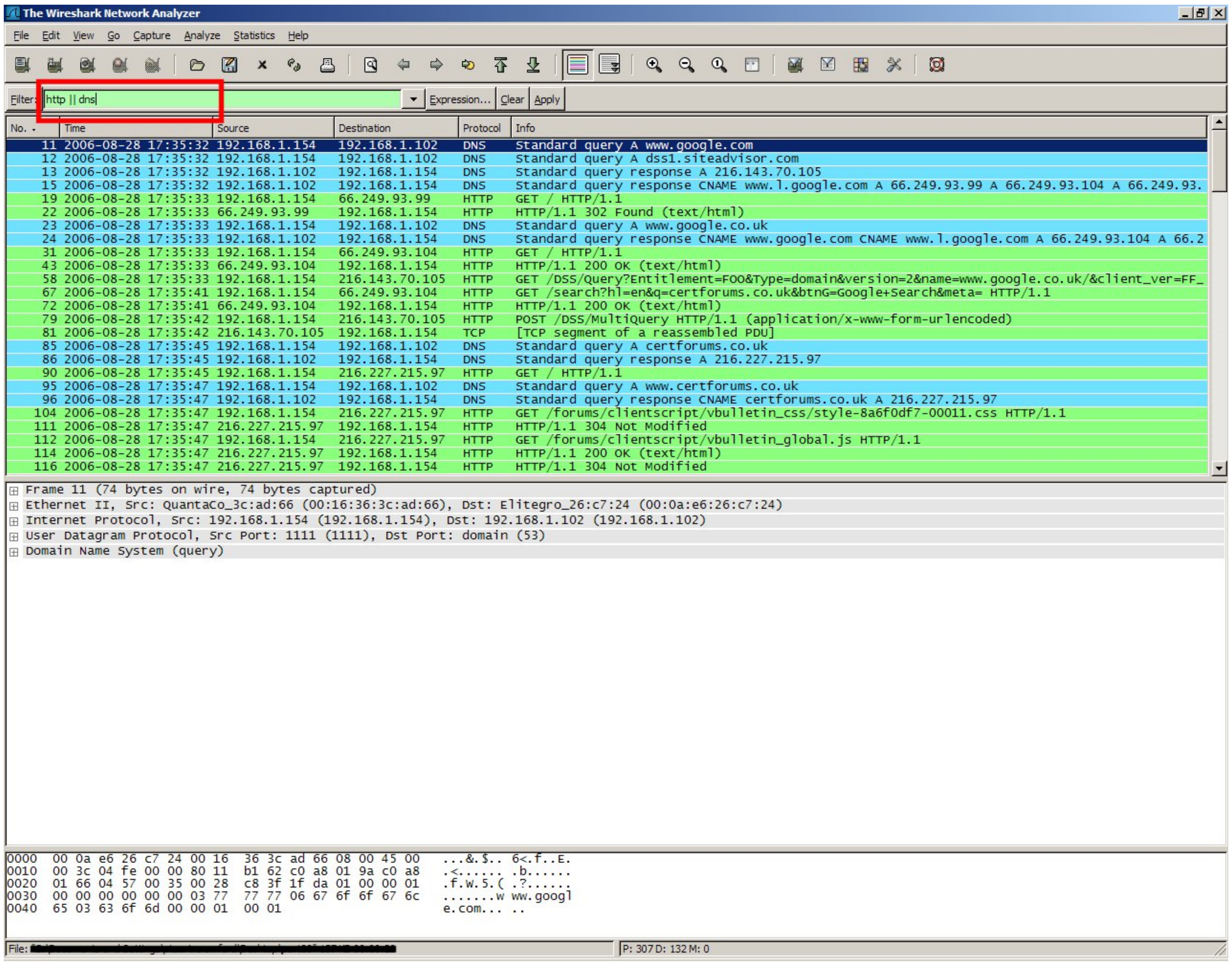
```
⊞ Frame 11 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: Elitegro_26:c7:24 (00:0a:e6:26:c7:24)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.102 (192.168.1.102)
⊞ User Datagram Protocol, Src Port: 1111 (1111), Dst Port: domain (53)
⊞ Domain Name System (query)
```

```
0000  00 0a e6 26 c7 24 00 16  36 3c ad 66 08 00 45 00   ...&.$.. 6<.f..E.
0010  00 3c 04 fe 00 00 80 11  b1 62 c0 a8 01 9a c0 a8   .<...... .b......
0020  01 66 04 57 00 35 00 28  c8 3f 1f da 01 00 00 01   .f.W.5.( .?......
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e.com... ..
```

File: [redacted]   P: 307 D: 307 M: 0

As you can see from the screen capture above, there is a lot of information extraneous to what we are interested in. In order to eliminate this and make the capture easier to follow, I will implement a filter to show only the data we want to see. Filters are an invaluable weapon in Ethereal's armoury – enabling you to perform complex logical operations on an incredible range of traffic (over 42,000 separate protocols & protocol fields at the last count). Here, I'll implement a very simple one. Since I'd like to see both how a website is requested and how that website is then displayed in my browser, I'll implement a filter to display only HTTP & DNS traffic. In order to do this – in the filter dialog box (highlighted in red in the screencap below) I type in:

"http || dns". The two pipes are the logical representation of an 'or' statement – literally, this filter means "show everything that is HTTP or DNS traffic". When I hit return, as you can see below, Ethereal now displays only the traffic I selected in my filter. Neat huh?


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Expression... Clear Apply

Filter: http || dns

```
No. -  Time                Source           Destination      Protocol Info
   11 2006-08-28 17:35:32 192.168.1.154    192.168.1.102    DNS      Standard query A www.google.com
   12 2006-08-28 17:35:32 192.168.1.154    192.168.1.102    DNS      Standard query A dss1.siteadvisor.com
   13 2006-08-28 17:35:32 192.168.1.102    192.168.1.154    DNS      Standard query response A 216.143.70.105
   15 2006-08-28 17:35:32 192.168.1.102    192.168.1.154    DNS      Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
   19 2006-08-28 17:35:33 192.168.1.154    66.249.93.99     HTTP     GET / HTTP/1.1
   22 2006-08-28 17:35:33 66.249.93.99     192.168.1.154    HTTP     HTTP/1.1 302 Found (text/html)
   23 2006-08-28 17:35:33 192.168.1.154    192.168.1.102    DNS      Standard query A www.google.co.uk
   24 2006-08-28 17:35:33 192.168.1.102    192.168.1.154    DNS      Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
   31 2006-08-28 17:35:33 192.168.1.154    66.249.93.104    HTTP     GET / HTTP/1.1
   43 2006-08-28 17:35:33 66.249.93.104    192.168.1.154    HTTP     HTTP/1.1 200 OK (text/html)
   58 2006-08-28 17:35:33 192.168.1.154    216.143.70.105   HTTP     GET /DSS/Query?Entitlement=FOO&Type=domain&version=2&name=www.google.co.uk/&client_ver=FF_
   67 2006-08-28 17:35:41 192.168.1.154    66.249.93.104    HTTP     GET /search?hl=en&q=certforums.co.uk&btnG=Google+Search&meta= HTTP/1.1
   72 2006-08-28 17:35:41 66.249.93.104    192.168.1.154    HTTP     HTTP/1.1 200 OK (text/html)
   79 2006-08-28 17:35:42 192.168.1.154    216.143.70.105   HTTP     POST /DSS/MultiQuery HTTP/1.1 (application/x-www-form-urlencoded)
   81 2006-08-28 17:35:42 216.143.70.105   192.168.1.154    TCP      [TCP segment of a reassembled PDU]
   85 2006-08-28 17:35:45 192.168.1.154    192.168.1.102    DNS      Standard query A certforums.co.uk
   86 2006-08-28 17:35:45 192.168.1.102    192.168.1.154    DNS      Standard query response A 216.227.215.97
   90 2006-08-28 17:35:45 192.168.1.154    216.227.215.97   HTTP     GET / HTTP/1.1
   95 2006-08-28 17:35:47 192.168.1.154    192.168.1.102    DNS      Standard query A www.certforums.co.uk
   96 2006-08-28 17:35:47 192.168.1.102    192.168.1.154    DNS      Standard query response CNAME certforums.co.uk A 216.227.215.97
  104 2006-08-28 17:35:47 192.168.1.154    216.227.215.97   HTTP     GET /forums/clientscript/vbulletin_css/style-8a6f0df7-00011.css HTTP/1.1
  111 2006-08-28 17:35:47 216.227.215.97   192.168.1.154    HTTP     HTTP/1.1 304 Not Modified
  112 2006-08-28 17:35:47 192.168.1.154    216.227.215.97   HTTP     GET /forums/clientscript/vbulletin_global.js HTTP/1.1
  114 2006-08-28 17:35:47 216.227.215.97   192.168.1.154    HTTP     HTTP/1.1 200 OK (text/html)
  116 2006-08-28 17:35:47 216.227.215.97   192.168.1.154    HTTP     HTTP/1.1 304 Not Modified
```

```
⊞ Frame 11 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: Elitegro_26:c7:24 (00:0a:e6:26:c7:24)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.102 (192.168.1.102)
⊞ User Datagram Protocol, Src Port: 1111 (1111), Dst Port: domain (53)
⊞ Domain Name System (query)
```

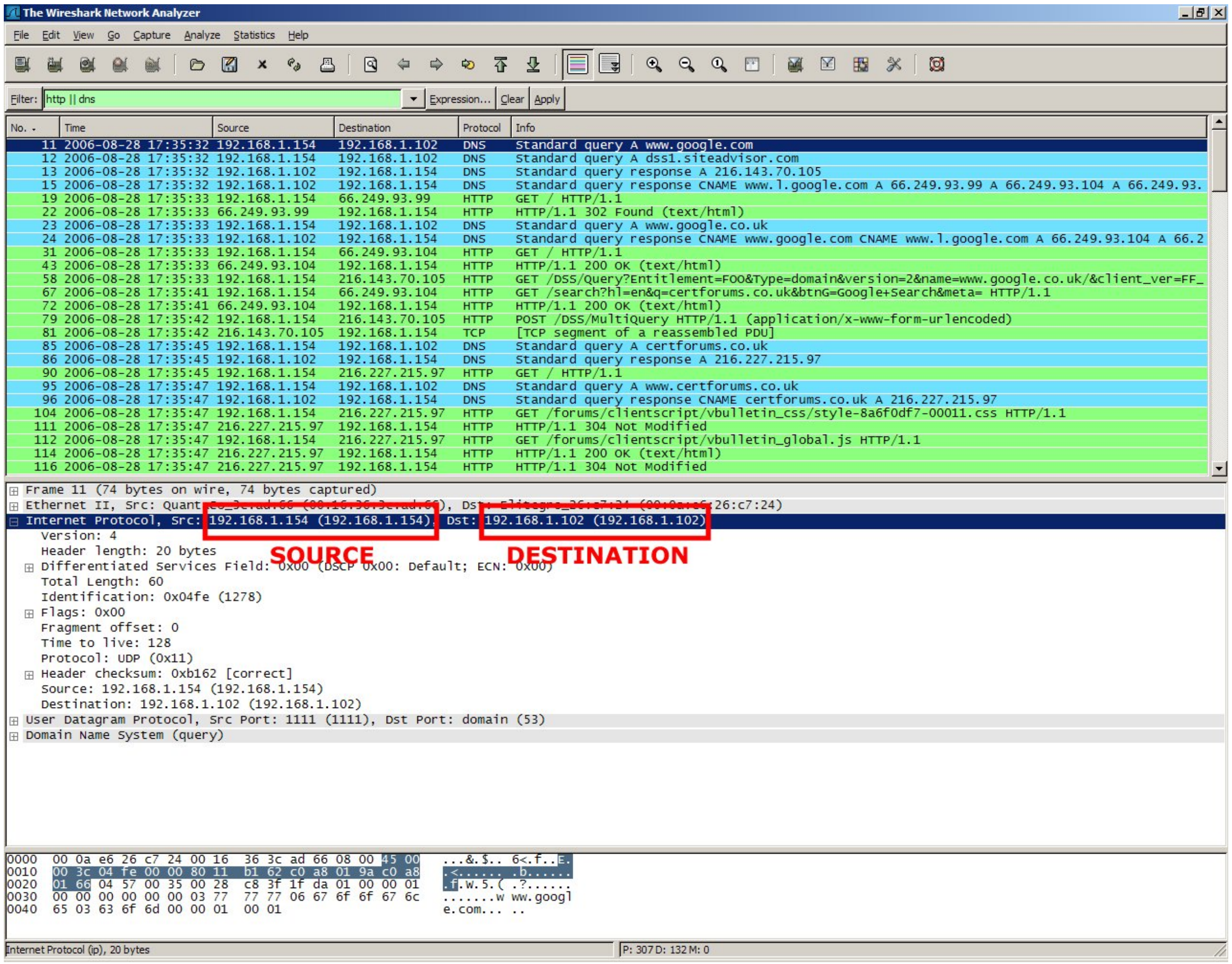
```
0000  00 0a e6 26 c7 24 00 16  36 3c ad 66 08 00 45 00   ...&.$.. 6<.f..E.
0010  00 3c 04 fe 00 00 80 11  b1 62 c0 a8 01 9a c0 a8   .<...... .b......
0020  01 66 04 57 00 35 00 28  c8 3f 1f da 01 00 00 01   .f.W.5.( .?......
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e.com... ..
```

File: [redacted]   P: 307 D: 132 M: 0

One thing I should mention at this point – the 'Time' column in your capture will probably look different to mine. This is because, by default, Ethereal shows the time column as the amount of time that has elapsed since the first packet was captured. At the moment, how you display the time is purely a matter of preference. When I'm looking at something as basic as this I prefer to see the time the capture was made, so I change this option in Ethereal. If you want to do the same, you can do this by going to 'View' – 'Time Display Format' and selecting one of the options there.

OK – now that we've got the traffic that we wish to look at displayed correctly, lets take a look at it. From your basic networking knowledge you should know that, in order to display a website in a browser, you first need to find that website on the net. As you know, in order to do this, your browser makes a DNS request for the URL you specify.

Here, I've cheated a little bit. Since Google is set as my homepage, the DNS query is made as soon as I fire up my browser. As you can see from the first line in the capture, I made a 'standard query' for the 'A' record associated with the domain 'www.google.com'. Drilling down into the protocol Tree window under 'IP' you can see that the SOURCE of this request is the IP address 192.168.1.154 (my PC) and the DESTINATION is 192.168.1.102.


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter: http || dns
Expression... Clear Apply
SOURCE
DESTINATION

```
No. -  Time                Source           Destination      Protocol  Info
   11 2006-08-28 17:35:32 192.168.1.154   192.168.1.102    DNS       Standard query A www.google.com
   12 2006-08-28 17:35:32 192.168.1.154   192.168.1.102    DNS       Standard query A dss1.siteadvisor.com
   13 2006-08-28 17:35:32 192.168.1.102   192.168.1.154    DNS       Standard query response A 216.143.70.105
   15 2006-08-28 17:35:32 192.168.1.102   192.168.1.154    DNS       Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
   19 2006-08-28 17:35:33 192.168.1.154   66.249.93.99     HTTP      GET / HTTP/1.1
   22 2006-08-28 17:35:33 66.249.93.99    192.168.1.154    HTTP      HTTP/1.1 302 Found (text/html)
   23 2006-08-28 17:35:33 192.168.1.154   192.168.1.102    DNS       Standard query A www.google.co.uk
   24 2006-08-28 17:35:33 192.168.1.102   192.168.1.154    DNS       Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
   31 2006-08-28 17:35:33 192.168.1.154   66.249.93.104    HTTP      GET / HTTP/1.1
   43 2006-08-28 17:35:33 66.249.93.104   192.168.1.154    HTTP      HTTP/1.1 200 OK (text/html)
   58 2006-08-28 17:35:33 192.168.1.154   216.143.70.105   HTTP      GET /DSS/Query?Entitlement=FOO&Type=domain&version=2&name=www.google.co.uk/&client_ver=FF_
   67 2006-08-28 17:35:41 192.168.1.154   66.249.93.104    HTTP      GET /search?hl=en&q=certforums.co.uk&btnG=Google+Search&meta= HTTP/1.1
   72 2006-08-28 17:35:41 66.249.93.104   192.168.1.154    HTTP      HTTP/1.1 200 OK (text/html)
   79 2006-08-28 17:35:42 192.168.1.154   216.143.70.105   HTTP      POST /DSS/MultiQuery HTTP/1.1 (application/x-www-form-urlencoded)
   81 2006-08-28 17:35:42 216.143.70.105  192.168.1.154    TCP       [TCP segment of a reassembled PDU]
   85 2006-08-28 17:35:45 192.168.1.154   192.168.1.102    DNS       Standard query A certforums.co.uk
   86 2006-08-28 17:35:45 192.168.1.102   192.168.1.154    DNS       Standard query response A 216.227.215.97
   90 2006-08-28 17:35:45 192.168.1.154   216.227.215.97   HTTP      GET / HTTP/1.1
   95 2006-08-28 17:35:47 192.168.1.154   192.168.1.102    DNS       Standard query A www.certforums.co.uk
   96 2006-08-28 17:35:47 192.168.1.102   192.168.1.154    DNS       Standard query response CNAME certforums.co.uk A 216.227.215.97
  104 2006-08-28 17:35:47 192.168.1.154   216.227.215.97   HTTP      GET /forums/clientscript/vbulletin_css/style-8a6f0df7-00011.css HTTP/1.1
  111 2006-08-28 17:35:47 216.227.215.97  192.168.1.154    HTTP      HTTP/1.1 304 Not Modified
  112 2006-08-28 17:35:47 192.168.1.154   216.227.215.97   HTTP      GET /forums/clientscript/vbulletin_global.js HTTP/1.1
  114 2006-08-28 17:35:47 216.227.215.97  192.168.1.154    HTTP      HTTP/1.1 200 OK (text/html)
  116 2006-08-28 17:35:47 216.227.215.97  192.168.1.154    HTTP      HTTP/1.1 304 Not Modified
```

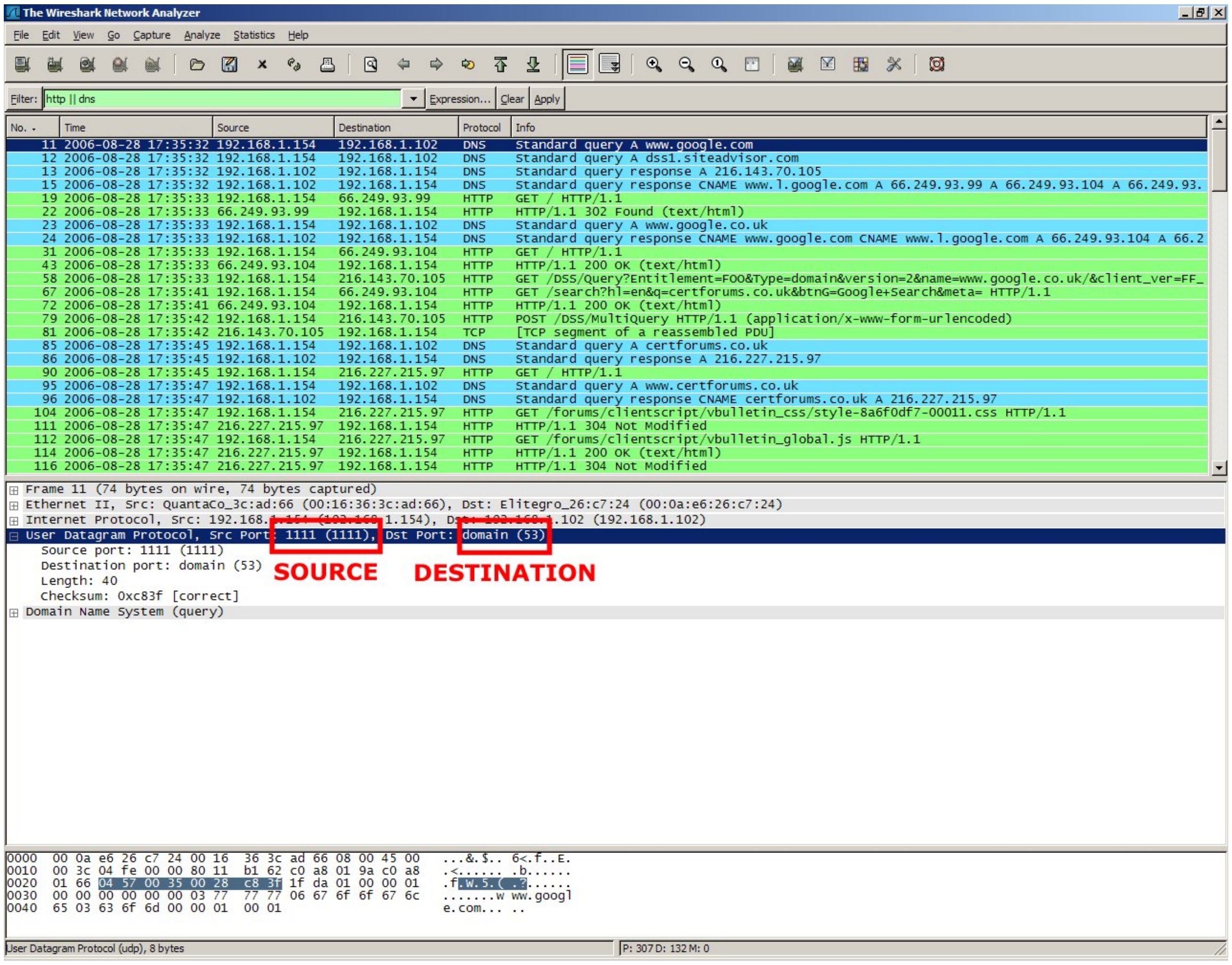
```
⊞ Frame 11 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: Quant[illegible]), Dst: [illegible]26:c7:24)
⊟ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.102 (192.168.1.102)
    Version: 4
    Header length: 20 bytes
  ⊞ Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)
    Total Length: 60
    Identification: 0x04fe (1278)
  ⊞ Flags: 0x00
    Fragment offset: 0
    Time to live: 128
    Protocol: UDP (0x11)
  ⊞ Header checksum: 0xb162 [correct]
    Source: 192.168.1.154 (192.168.1.154)
    Destination: 192.168.1.102 (192.168.1.102)
⊞ User Datagram Protocol, Src Port: 1111 (1111), Dst Port: domain (53)
⊞ Domain Name System (query)
```

```
0000  00 0a e6 26 c7 24 00 16  36 3c ad 66 08 00 45 00   ...&.$.. 6<.f..E.
0010  00 3c 04 fe 00 00 80 11  b1 62 c0 a8 01 9a c0 a8   .<...... .b......
0020  01 66 04 57 00 35 00 28  c8 3f 1f da 01 00 00 01   .f.W.5.( .?......
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e.com... ..
```

```
Internet Protocol (ip), 20 bytes          P: 307 D: 132 M: 0
```

Hands up anyone who knows what IP address my local DNS ϑserver is?

Underneath the IP tree you can see the UDP data associated with the packet. As you should know, DNS requests are made via UDP port 53. Here you can see that I have opened a local UDP port (1111 in this case) and made a connection to remote UDP port 53.

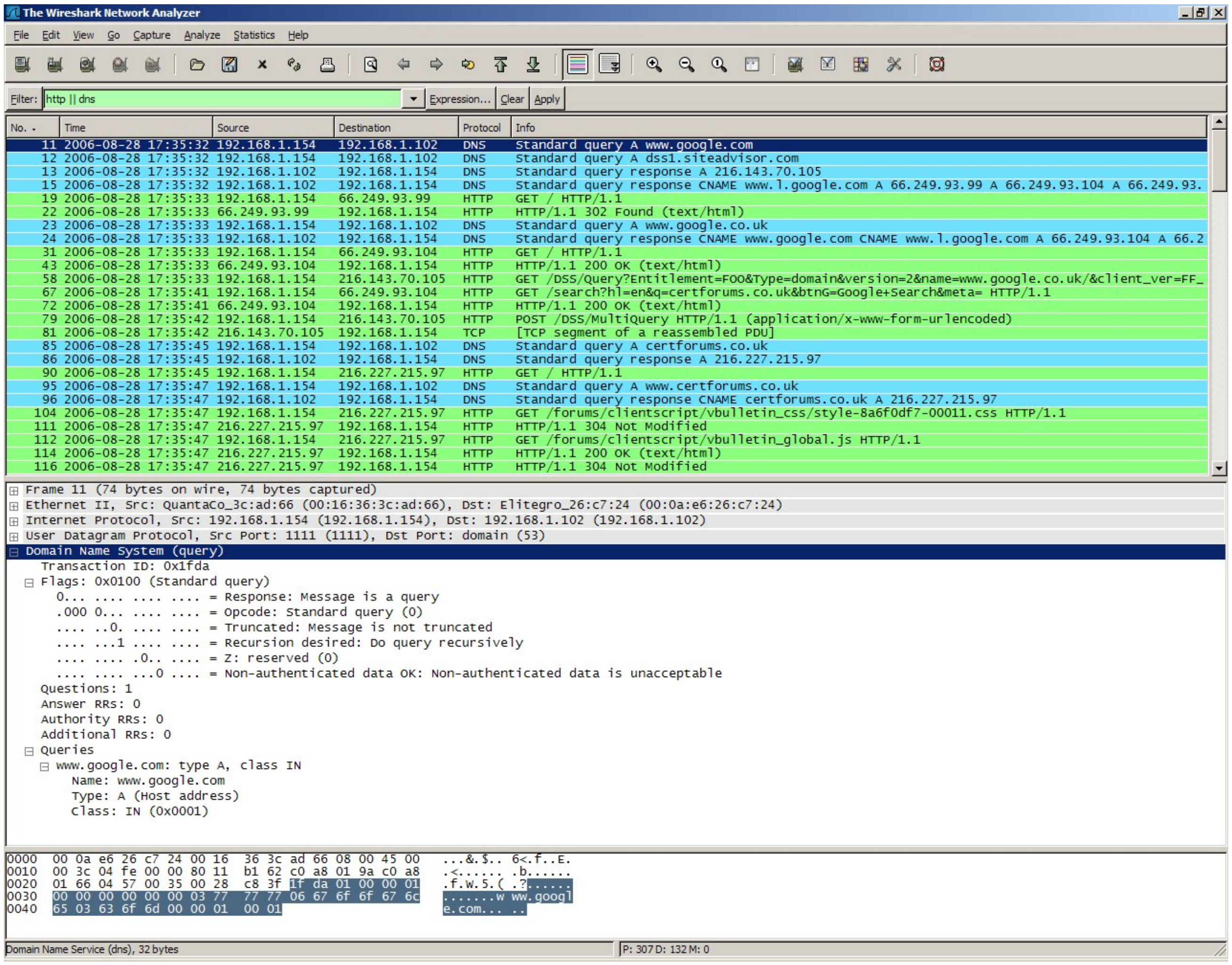

The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter: http || dns
Expression... Clear Apply
SOURCE
DESTINATION
User Datagram Protocol (udp), 8 bytes
P: 307 D: 132 M: 0

```
No. -  Time                Source          Destination     Protocol Info
    11 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.com
    12 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A dss1.siteadvisor.com
    13 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.143.70.105
    15 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
    19 2006-08-28 17:35:33 192.168.1.154   66.249.93.99    HTTP     GET / HTTP/1.1
    22 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   HTTP     HTTP/1.1 302 Found (text/html)
    23 2006-08-28 17:35:33 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.co.uk
    24 2006-08-28 17:35:33 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
    31 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   HTTP     GET / HTTP/1.1
    43 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
    58 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  HTTP     GET /DSS/Query?Entitlement=FOO&Type=domain&version=2&name=www.google.co.uk/&client_ver=FF_
    67 2006-08-28 17:35:41 192.168.1.154   66.249.93.104   HTTP     GET /search?hl=en&q=certforums.co.uk&btnG=Google+Search&meta= HTTP/1.1
    72 2006-08-28 17:35:41 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
    79 2006-08-28 17:35:42 192.168.1.154   216.143.70.105  HTTP     POST /DSS/MultiQuery HTTP/1.1 (application/x-www-form-urlencoded)
    81 2006-08-28 17:35:42 216.143.70.105  192.168.1.154   TCP      [TCP segment of a reassembled PDU]
    85 2006-08-28 17:35:45 192.168.1.154   192.168.1.102   DNS      Standard query A certforums.co.uk
    86 2006-08-28 17:35:45 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.227.215.97
    90 2006-08-28 17:35:45 192.168.1.154   216.227.215.97  HTTP     GET / HTTP/1.1
    95 2006-08-28 17:35:47 192.168.1.154   192.168.1.102   DNS      Standard query A www.certforums.co.uk
    96 2006-08-28 17:35:47 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME certforums.co.uk A 216.227.215.97
   104 2006-08-28 17:35:47 192.168.1.154   216.227.215.97  HTTP     GET /forums/clientscript/vbulletin_css/style-8a6f0df7-00011.css HTTP/1.1
   111 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 304 Not Modified
   112 2006-08-28 17:35:47 192.168.1.154   216.227.215.97  HTTP     GET /forums/clientscript/vbulletin_global.js HTTP/1.1
   114 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
   116 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 304 Not Modified
```

```
⊞ Frame 11 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: Elitegro_26:c7:24 (00:0a:e6:26:c7:24)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.102 (192.168.1.102)
⊟ User Datagram Protocol, Src Port: 1111 (1111), Dst Port: domain (53)
    Source port: 1111 (1111)
    Destination port: domain (53)
    Length: 40
    Checksum: 0xc83f [correct]
⊞ Domain Name System (query)
```

```
0000  00 0a e6 26 c7 24 00 16  36 3c ad 66 08 00 45 00   ...&.$.. 6<.f..E.
0010  00 3c 04 fe 00 00 80 11  b1 62 c0 a8 01 9a c0 a8   .<...... .b......
0020  01 66 04 57 00 35 00 28  c8 3f 1f da 01 00 00 01   .f.W.5.( .?......
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e.com... ..
```

Finally, lets open the DNS tree (remember that Ethereal displays packet data in order from lowest to highest as mapped to the OSI model – therefore the DNS protocol, which operates at the Application layer, is at the bottom of the protocol tree window). Expanding the ‘Flags’ and ‘Queries’ branches, you can see the specific information related to DNS in the packet. Here, we see the ‘ins and outs’ of the DNS query – including the fact that the query is recursive, not truncated, and the query itself (the ‘A’ record for the host ‘www.google.com’)


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter: http || dns
Expression... Clear Apply

```
No. - Time                Source          Destination     Protocol Info
 11 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.com
 12 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A dss1.siteadvisor.com
 13 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.143.70.105
 15 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
 19 2006-08-28 17:35:33 192.168.1.154   66.249.93.99    HTTP     GET / HTTP/1.1
 22 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   HTTP     HTTP/1.1 302 Found (text/html)
 23 2006-08-28 17:35:33 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.co.uk
 24 2006-08-28 17:35:33 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
 31 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   HTTP     GET / HTTP/1.1
 43 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
 58 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  HTTP     GET /DSS/Query?Entitlement=FOO&Type=domain&version=2&name=www.google.co.uk/&client_ver=FF_
 67 2006-08-28 17:35:41 192.168.1.154   66.249.93.104   HTTP     GET /search?hl=en&q=certforums.co.uk&btnG=Google+Search&meta= HTTP/1.1
 72 2006-08-28 17:35:41 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
 79 2006-08-28 17:35:42 192.168.1.154   216.143.70.105  HTTP     POST /DSS/MultiQuery HTTP/1.1 (application/x-www-form-urlencoded)
 81 2006-08-28 17:35:42 216.143.70.105  192.168.1.154   TCP      [TCP segment of a reassembled PDU]
 85 2006-08-28 17:35:45 192.168.1.154   192.168.1.102   DNS      Standard query A certforums.co.uk
 86 2006-08-28 17:35:45 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.227.215.97
 90 2006-08-28 17:35:45 192.168.1.154   216.227.215.97  HTTP     GET / HTTP/1.1
 95 2006-08-28 17:35:47 192.168.1.154   192.168.1.102   DNS      Standard query A www.certforums.co.uk
 96 2006-08-28 17:35:47 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME certforums.co.uk A 216.227.215.97
104 2006-08-28 17:35:47 192.168.1.154   216.227.215.97  HTTP     GET /forums/clientscript/vbulletin_css/style-8a6f0df7-00011.css HTTP/1.1
111 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 304 Not Modified
112 2006-08-28 17:35:47 192.168.1.154   216.227.215.97  HTTP     GET /forums/clientscript/vbulletin_global.js HTTP/1.1
114 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
116 2006-08-28 17:35:47 216.227.215.97  192.168.1.154   HTTP     HTTP/1.1 304 Not Modified
```

```
⊞ Frame 11 (74 bytes on wire, 74 bytes captured)
⊞ Ethernet II, Src: QuantaCo_3c:ad:66 (00:16:36:3c:ad:66), Dst: Elitegro_26:c7:24 (00:0a:e6:26:c7:24)
⊞ Internet Protocol, Src: 192.168.1.154 (192.168.1.154), Dst: 192.168.1.102 (192.168.1.102)
⊞ User Datagram Protocol, Src Port: 1111 (1111), Dst Port: domain (53)
⊟ Domain Name System (query)
    Transaction ID: 0x1fda
  ⊟ Flags: 0x0100 (Standard query)
      0... .... .... .... = Response: Message is a query
      .000 0... .... .... = Opcode: Standard query (0)
      .... ..0. .... .... = Truncated: Message is not truncated
      .... ...1 .... .... = Recursion desired: Do query recursively
      .... .... .0.. .... = Z: reserved (0)
      .... .... ...0 .... = Non-authenticated data OK: Non-authenticated data is unacceptable
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
  ⊟ Queries
    ⊟ www.google.com: type A, class IN
        Name: www.google.com
        Type: A (Host address)
        Class: IN (0x0001)
```

```
0000  00 0a e6 26 c7 24 00 16  36 3c ad 66 08 00 45 00   ...&.$.. 6<.f..E.
0010  00 3c 04 fe 00 00 80 11  b1 62 c0 a8 01 9a c0 a8   .<...... .b......
0020  01 66 04 57 00 35 00 28  c8 3f 1f da 01 00 00 01   .f.W.5.( .?......
0030  00 00 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01                     e.com... ..
```

```
Domain Name Service (dns), 32 bytes                P: 307 D: 132 M: 0
```

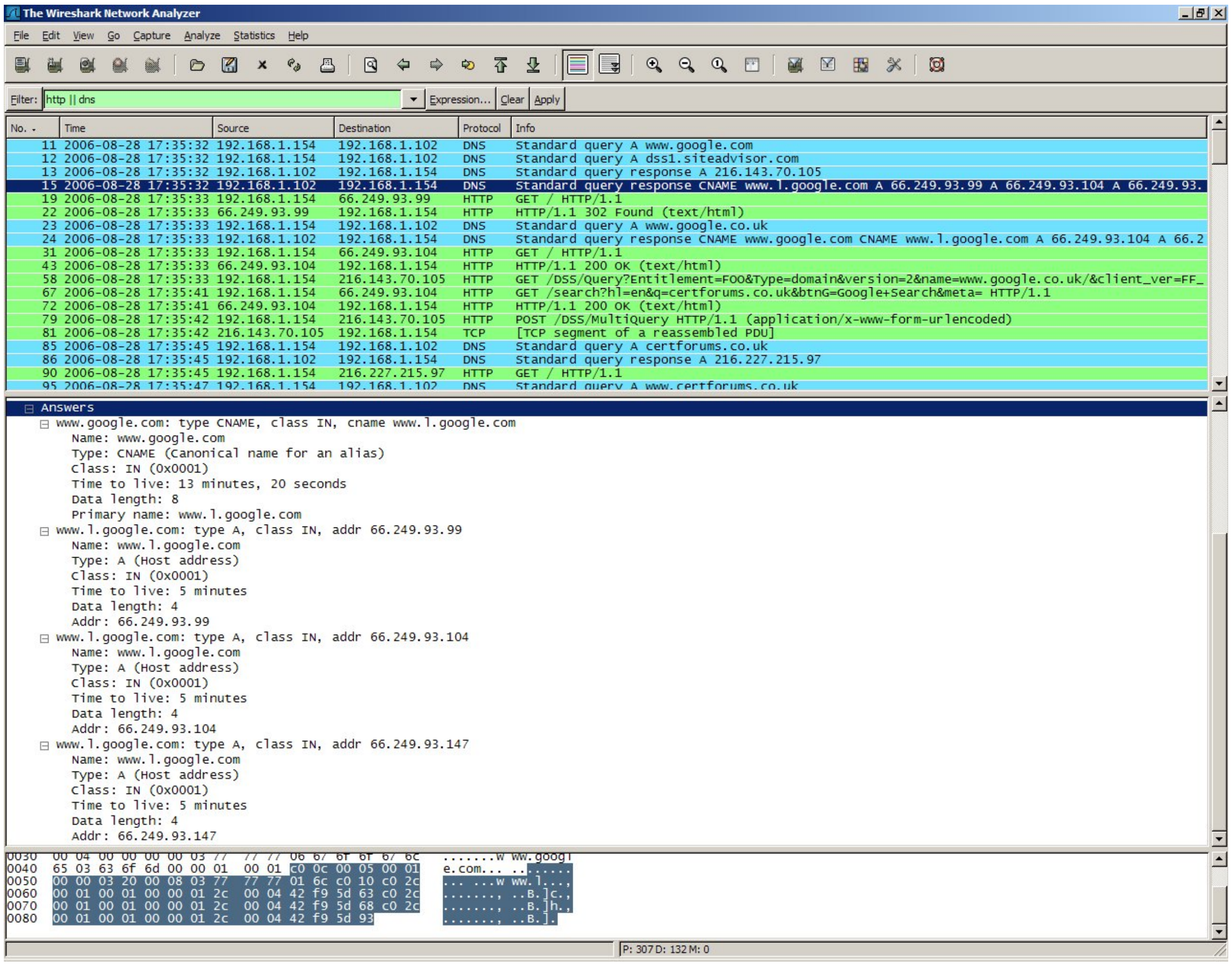
You should already be able to see the benefits of running Ethereal in cases where DNS name resolution is problematic or intermittent – examining the data from a capture filtered out to just DNS traffic would enable you to pinpoint where DNS was falling down. This could prove invaluable in troubleshooting DNS problems – often a cause for consternation for many network admins!

So there you go – your first 'real world' packet analysis. Now lets go a little bit further. Anybody know what the packet listed as '12' in my capture represents? I won't dissect that packet here – I'll leave it as a bit of research for you. Suffice it to say that I find it an invaluable security tool for end users…

Ignoring packet 12 for now then, you can see the result of the DNS query in packets 13 & 15 - the IP address for the host 'www.google.com' is '216.143.70.105' – or is it? Well, actually, no – this is still part of the 'homework' relating to packet # 12 – in fact, it might assist you if you're having trouble finding out what packet 12 represents!

The 'real' response to the DNS query comes in packet 15. As you can see, the DNS query returns a CNAME (Canonical Name) record for something called 'www.l.google.com', providing three separate IP addresses: '66.249.93.99', '66.249.93.104' and '66.249.93.147'. Already we have discovered something interesting about Google's internal infrastructure. It seems that Google's nameservers are set to first return a CNAME in the form 'www.<letter>.google.com' (where '<letter>' appears to be a different letter depending on where you are located or what time of day it is (try this yourself – run two traces on a DNS lookup for Google a couple of hours apart – you'll see that the letter is probably different both times). It appears that a second query is then made, returning three different IP addresses associated with that CNAME. It doesn't take a genius to figure out that this is something to do with Google's fault-tolerance/load-balancing setup – a massive organisation like Google – synonymous with both all-pervasiveness and guaranteed uptime – will obviously have data centres dotted all over the planet. What this DOES highlight, and very well, is that unless you were already well-versed in Google's internal structure, you probably already learned something new from using Ethereal – and you haven't even put the program to any practical use yet!

In order to see the query responses in detail, expand the DNS branch in the protocol tree window and then expand all the entries under 'Answers' – you'll see the CNAME first, followed by the three IP addresses associated with it


The Wireshark Network Analyzer
File Edit View Go Capture Analyze Statistics Help
Filter: http || dns
Expression... Clear Apply

```
No. -  Time                Source          Destination     Protocol Info
    11 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.com
    12 2006-08-28 17:35:32 192.168.1.154   192.168.1.102   DNS      Standard query A dss1.siteadvisor.com
    13 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.143.70.105
    15 2006-08-28 17:35:32 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.l.google.com A 66.249.93.99 A 66.249.93.104 A 66.249.93.
    19 2006-08-28 17:35:33 192.168.1.154   66.249.93.99    HTTP     GET / HTTP/1.1
    22 2006-08-28 17:35:33 66.249.93.99    192.168.1.154   HTTP     HTTP/1.1 302 Found (text/html)
    23 2006-08-28 17:35:33 192.168.1.154   192.168.1.102   DNS      Standard query A www.google.co.uk
    24 2006-08-28 17:35:33 192.168.1.102   192.168.1.154   DNS      Standard query response CNAME www.google.com CNAME www.l.google.com A 66.249.93.104 A 66.2
    31 2006-08-28 17:35:33 192.168.1.154   66.249.93.104   HTTP     GET / HTTP/1.1
    43 2006-08-28 17:35:33 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
    58 2006-08-28 17:35:33 192.168.1.154   216.143.70.105  HTTP     GET /DSS/Query?Entitlement=FOO&Type=domain&version=2&name=www.google.co.uk/&client_ver=FF_
    67 2006-08-28 17:35:41 192.168.1.154   66.249.93.104   HTTP     GET /search?hl=en&q=certforums.co.uk&btnG=Google+Search&meta= HTTP/1.1
    72 2006-08-28 17:35:41 66.249.93.104   192.168.1.154   HTTP     HTTP/1.1 200 OK (text/html)
    79 2006-08-28 17:35:42 192.168.1.154   216.143.70.105  HTTP     POST /DSS/MultiQuery HTTP/1.1 (application/x-www-form-urlencoded)
    81 2006-08-28 17:35:42 216.143.70.105  192.168.1.154   TCP      [TCP segment of a reassembled PDU]
    85 2006-08-28 17:35:45 192.168.1.154   192.168.1.102   DNS      Standard query A certforums.co.uk
    86 2006-08-28 17:35:45 192.168.1.102   192.168.1.154   DNS      Standard query response A 216.227.215.97
    90 2006-08-28 17:35:45 192.168.1.154   216.227.215.97  HTTP     GET / HTTP/1.1
    95 2006-08-28 17:35:47 192.168.1.154   192.168.1.102   DNS      Standard query A www.certforums.co.uk
```

```
⊟ Answers
  ⊟ www.google.com: type CNAME, class IN, cname www.l.google.com
      Name: www.google.com
      Type: CNAME (Canonical name for an alias)
      Class: IN (0x0001)
      Time to live: 13 minutes, 20 seconds
      Data length: 8
      Primary name: www.l.google.com
  ⊟ www.l.google.com: type A, class IN, addr 66.249.93.99
      Name: www.l.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 5 minutes
      Data length: 4
      Addr: 66.249.93.99
  ⊟ www.l.google.com: type A, class IN, addr 66.249.93.104
      Name: www.l.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 5 minutes
      Data length: 4
      Addr: 66.249.93.104
  ⊟ www.l.google.com: type A, class IN, addr 66.249.93.147
      Name: www.l.google.com
      Type: A (Host address)
      Class: IN (0x0001)
      Time to live: 5 minutes
      Data length: 4
      Addr: 66.249.93.147
```

```
0030  00 04 00 00 00 00 03 77  77 77 06 67 6f 6f 67 6c   .......w ww.googl
0040  65 03 63 6f 6d 00 00 01  00 01 c0 0c 00 05 00 01   e.com... ........
0050  00 00 03 20 00 08 03 77  77 77 01 6c c0 10 c0 2c   ... ...w ww.l...,
0060  00 01 00 01 00 00 01 2c  00 04 42 f9 5d 63 c0 2c   ......., ..B.]c.,
0070  00 01 00 01 00 00 01 2c  00 04 42 f9 5d 68 c0 2c   ......., ..B.]h.,
0080  00 01 00 01 00 00 01 2c  00 04 42 f9 5d 93         ......., ..B.].
```

On a more mundane level, you can see that the DNS query was returned via remote UDP port 53 to local UDP port 1111, from IP 192.168.1.102 to IP 192.168.1.154 – indicating that the DNS query followed the same path as it did outbound, but in reverse. Of course, we can't see any other information relating to the DNS query in this capture – since my DNS is configured to refer to a local DNS server – which then does the upstream query for me. Your DNS queries may well be different you may see your ISP's DNS server – if you're not sad enough to run your own local DNS like me!

In the final part of the tutorial, I'll look at the HTTP traffic associated with browsing to a site, examine the DNS traffic again, and suggest ways you can experiment on your own to further your learning.

First up, I should probably start out by saying that Ethereal is (and has been for some time) now called Wireshark. Out of habit I STILL call it Ethereal from time to time - but as even I've got used to using the new name now I think the thread titles should be changed.

Last time we looked at Wireshark we were concentrating on how it actually works - this time out we'll be looking at an example of a way you could put it to practical use 'on the job'.

As anyone who has ever worked in tech support will tell you, users are the bane of your life. The same maxim you would apply to small children and savage wild animals should be applicable to 'most users' and 'computers'. They should never be kept in the same room and, in a perfect world, would never come into contact with each other. Unfortunately, since your salary is usually dependent on cleaning up the mess these tards have made of your precious network - and management seem to think they are actually doing something useful with their PCs instead of playing games, sending colleagues who work twelve feet away stupid emails or looking for 'interesting' ways to breach your firewall and download malware, this is seldom an option in most workplaces.

Wireshark can truly be your friend when attempting to troubleshoot a potentially tricky network issue - it's ability to sneak a peek at the raw data travelling across your network can prove invaluable. it has, on many occasions, given me the answer to problems it might have taken me hours to diagnose in less than ten minutes.

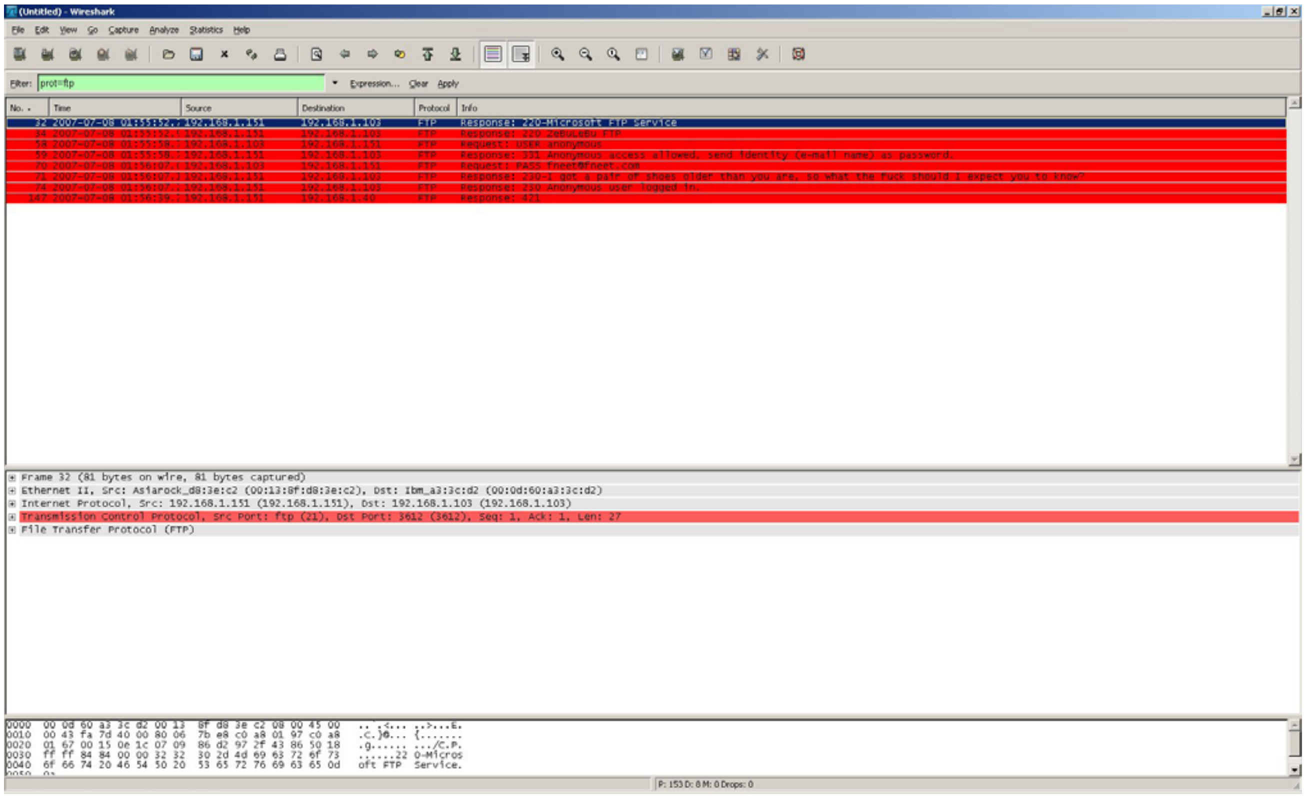
Let's take a simple example. You have a user (let's call him 'Donut') on your network who regularly has to FTP to a machine elsewhere on your network to retrieve a file. The user is using a workstation with the IP address: 192.168.1.40, and the server is located at 192.168.1.151. This user is something of a luddite - he has been told how to logon and retrieve the file but nothing else about how FTP works. He also cannot interpret screen commands, read simple dialog boxes, copy down an error message reliably or keep windows displaying error codes open when requested to do so by first line support. Sound familiar?

This user complains that, 'every day' (lUser speak for 'about once a month') when he attempts to FTP to the address he has been given, he is unable to do so. He says that the connection 'just closes' and the command box 'just disappears'. Whilst you know that there must be a reason for this, and you could probably, (if you didn't end up smashing your head against your monitor in frustration first) tease the correct information necessary to allow you to make a diagnosis of the problem, you decide it will be:

a) more accurate
b) probably faster, and
c) definitely cooler to fire up Wireshark and trace the connection in real time.

Being a good network admin, you've already got a box plugged into a span/mirror port on your switch that allows you to monitor all the traffic belting through that switch (you HAVE got a box like this, haven't you? so you fire up Wireshark and go through the capture process whilst the user is attempting his FTP connection. Just for good measure, you capture for five minutes before and after to see if there may be some symptoms of a problem with the FTP server knocking around.

I won't go through the tedious process of explaining how to use filters & stuff here - if you've got this far, you;re already familiar with filtering out extraneous information from capture files. Lets assume you have already filtered out all traffic that isn't related to FTP in your capture and have been left with something similar to the window below:


(Untitled) - Wireshark

Filter: prot=ftp

```
No. .  Time                    Source          Destination     Protocol  Info
   32 2007-07-08 01:55:52.7 192.168.1.151  192.168.1.103  FTP       Response: 220-Microsoft FTP Service
   34 2007-07-08 01:55:52.9 192.168.1.151  192.168.1.103  FTP       Response: 220 ZeBuLeBu FTP
   58 2007-07-08 01:55:58.3 192.168.1.103  192.168.1.151  FTP       Request: USER anonymous
   59 2007-07-08 01:55:58.7 192.168.1.151  192.168.1.103  FTP       Response: 331 Anonymous access allowed, send identity (e-mail name) as password.
   70 2007-07-08 01:56:07.0 192.168.1.103  192.168.1.151  FTP       Request: PASS fneet@fneet.com
   71 2007-07-08 01:56:07.1 192.168.1.151  192.168.1.103  FTP       Response: 230-I got a pair of shoes older than you are, so what the fuck should I expect you to know?
   74 2007-07-08 01:56:07.2 192.168.1.151  192.168.1.103  FTP       Response: 230 Anonymous user logged in.
  147 2007-07-08 01:56:39.2 192.168.1.151  192.168.1.40   FTP       Response: 421
```

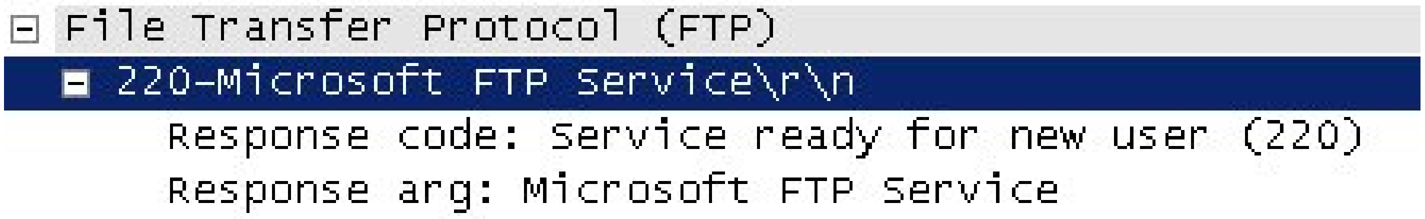
```
⊞ Frame 32 (81 bytes on wire, 81 bytes captured)
⊞ Ethernet II, Src: Asiarock_d8:3e:c2 (00:13:8f:d8:3e:c2), Dst: Ibm_a3:3c:d2 (00:0d:60:a3:3c:d2)
⊞ Internet Protocol, Src: 192.168.1.151 (192.168.1.151), Dst: 192.168.1.103 (192.168.1.103)
⊞ Transmission Control Protocol, Src Port: ftp (21), Dst Port: 3612 (3612), Seq: 1, Ack: 1, Len: 27
⊞ File Transfer Protocol (FTP)
```

```
0000  00 0d 60 a3 3c d2 00 13  8f d8 3e c2 08 00 45 00   ..`.<... ..>...E.
0010  00 43 fa 7d 40 00 80 06  7b e8 c0 a8 01 97 c0 a8   .C.}@... {.......
0020  01 67 00 15 0e 1c 07 09  86 d2 97 2f 43 86 50 18   .g...... .../C.P.
0030  ff ff 84 84 00 00 32 32  30 2d 4d 69 63 72 6f 73   ......22 0-Micros
0040  6f 66 74 20 46 54 50 20  53 65 72 76 69 63 65 0d   oft FTP  Service.
0050  0a
```

P: 153 D: 8 M: 0 Drops: 0

As you can see, this shows that, as well as the traffic associated with Donut's connection (from 192.168.1.40) there is a prior connection from another machine on the network (192.168.1.103).

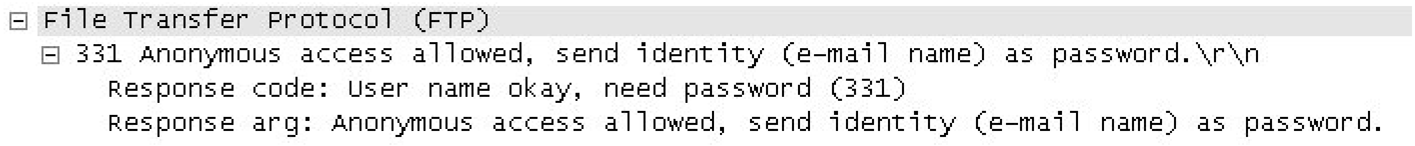
As opening the FTP protocol traffic in the first frame (#32 in the capture) shows:



```
⊟ File Transfer Protocol (FTP)
  ⊟ 220-Microsoft FTP Service\r\n
      Response code: Service ready for new user (220)
      Response arg: Microsoft FTP Service
```

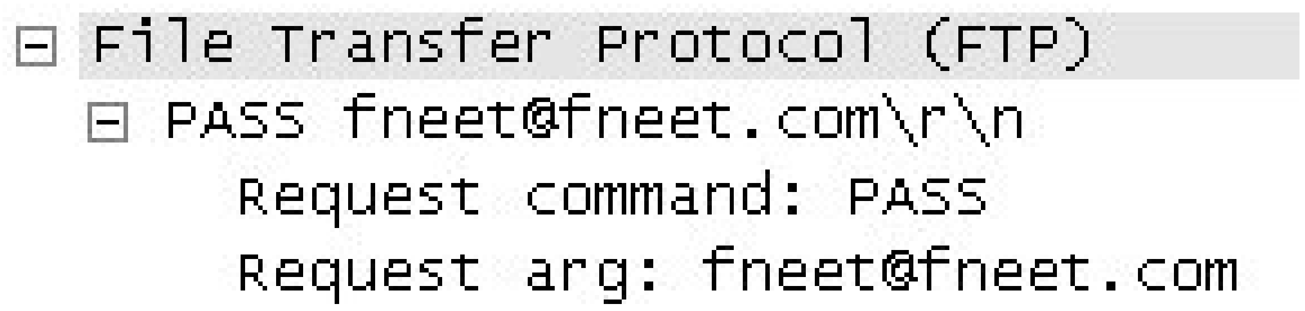
the FTP server was in a 'ready' state, awaiting connections when the earlier login was made. Packet 58 shows us that the user requested an anonymous logon:

and packet 59 shows that anonymous access is allowed to this server:



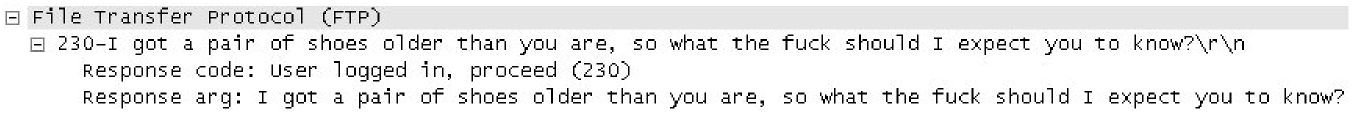
```
⊟ File Transfer Protocol (FTP)
  ⊟ 331 Anonymous access allowed, send identity (e-mail name) as password.\r\n
      Response code: User name okay, need password (331)
      Response arg: Anonymous access allowed, send identity (e-mail name) as password.
```

The previous user was requested to send their identity (suggested as an email address) for logging purposes, which the user subsequently did, in packet #70:



```
⊟ File Transfer Protocol (FTP)
  ⊟ PASS fneet@fneet.com\r\n
      Request command: PASS
      Request arg: fneet@fneet.com
```

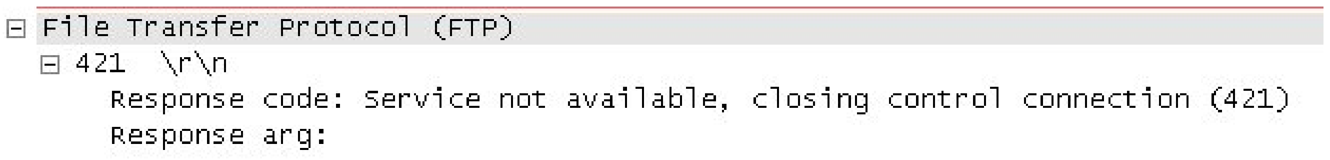
Finally, packet 71 shows that the user was successfully sent the FTP site banner that has been configured on the server:



```
⊟ File Transfer Protocol (FTP)
  ⊟ 230-I got a pair of shoes older than you are, so what the fuck should I expect you to know?\r\n
      Response code: User logged in, proceed (230)
      Response arg: I got a pair of shoes older than you are, so what the fuck should I expect you to know?
```

(reps for anyone cool enough to know where that comes from!)

This proves that there is nothing fundamentally wrong with the FTP server that the user is attempting to connect to. However, the solitary packet associated with traffic from the user's IP address to the server concerned provides everything we need to know in one simple error code:



```
⊟ File Transfer Protocol (FTP)
  ⊟ 421  \r\n
      Response code: Service not available, closing control connection (421)
      Response arg:
```

To be honest, we didn't need to go as far as opening the FTP traffic - we could have seen in the main Wireshark window that the error code was '421'. Armed with this information, plus the fact that an earlier user successfully logged into the server, we can deduce that it is likely the maximum number of sessions allowed on the FTP server has been reached.

This is an extremely simplistic view of what is possible with Wireshark. If you can think of any network-related problem that requires analysing, Wireshark can help. i have used it to sort out routing issues, incorrectly configured subnet masks, malware and worm activity analysis, messenger traffic logging, troubleshooting HTTPS problems, logging portscan activity and hundreds more useful things. It truly is the swiss army knife of any network admin, and learning it should be de rigeur for anyone who wants to further their career in the field.