Protocol Stacks and FTP

Peter Lo

Mission Impossible

- Mission:
  - Deliver a pizza to a specific address
  - BUT... You don't know the route
  - BUT… At each required turnoff a further direction (to the next "hop") will be given
  - like a rally
- Should you choose to accept it?

This week we are going to pull together all that we have done so far and give it some overall structure. We will begin to use a little more of the language that is used in the industry to talk about networking - there will not be many acronyms but there will be a whole new set of terms and concepts provided to refer to what you now are beginning to understand about networks.

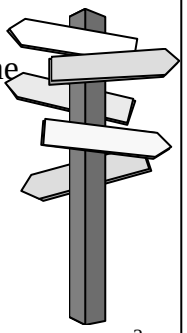
We will do this conceptual re-orientation in the context of an analogy. This analogy, pizza delivery, has a clear relation to the other analogies that we have used earlier like Package Delivery but we are going to add one more step beyond "Reliability" and we are going to think really carefully about the order in which things happen.

You will not be surprised to hear that the delivery you are about to undertake is to an address that you do not know but that you are able to find your way to the delivery address by inquiring at each turn off.

Because you now know quite a bit about networks you will recognise this as routers joining networks.

The "Street Level"

- Addressing information:
  - "*go down Token Ring Boulevard*"
- Protocol:
  - keep following the addressing instructions until you find a direction booth (kindly placed on each corner)
  - ask for directions for the next "hop"...
  - replace the old addressing instructions with the new ("*Turn here and go down Ether Street*")
  - Repeat (until arrival at final destination)



The routers, which we will represent for now as signposts, work like network routers by providing directions to the next "hop" along the way.

The confusing thing is that you, the delivery person, never get the whole picture of where you are going - but the signposts understand the whole geography of the city and they never make a mistake.

We will call this level of navigation the **Street Level.**

The “House Level”

- Addressing information:
  - “*132, Wumpumburger Parade, Bilby*”
- Protocol:
  - The suburb component of the address may be used to get directions to the correct area

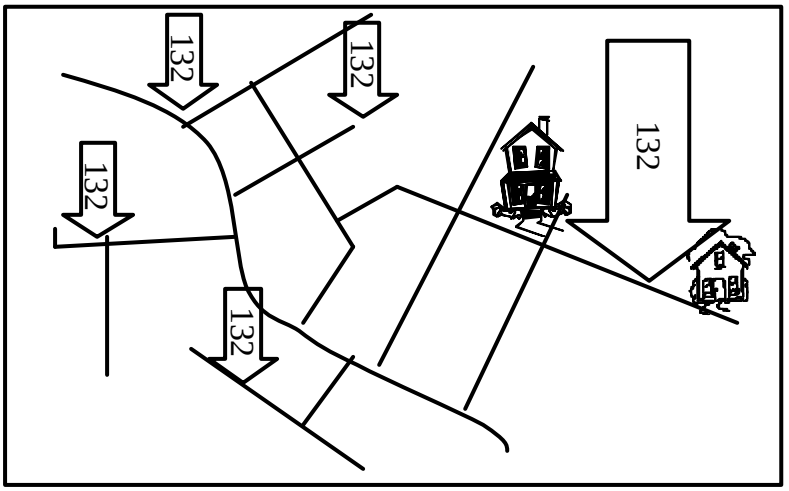
--The street address component may be used within correct suburbs

-- Final delivery is in terms of the “House level”. This only works on the right street though!

Of course Street Level navigation is not going to get you all the way to where you want to go. Sooner or later you are going to have to deliver to a specific house address.

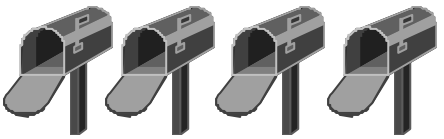
At this point you are going to have to consider **House Level** addressing which you will recognise as the description in this scenario of local delivery. This slide reminds you that the house level addressing only makes sense if you are already in the right street!

House Level - on a street



The "Flat Number Level"

- Addressing information:
  - "*Flat 7*"
- Protocol:
  - use this addressing to determine which door to knock on
  - Let's add some reliability:
    - if some of the pizza is eaten
    - or if the toppings are incorrect
      - take it back and return with correct/complete order



Once you have found the right house you now need to realise that your ordeal is not yet over. The house is divided into flats and you need to knock on the right door! This is **Flat Level** addressing.

In terms of the computer view of things this is saying that we need to get to the right sub-address within the computer. There was a very brief glimpse of this in the practice exam where you had to pick out a **socket address**. The purpose of this extra level of addressing is to take account of the fact that computers are seldom doing only one thing at a time.

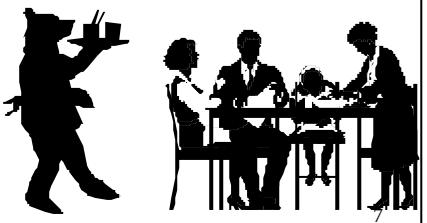
If you are running Windows this is obvious at the level of the different applications you are using at once but it is also the case in a much simpler computer. You have already seen that part way through a PING your computer may need to ARP.

It has also become the convention to introduce some reliability measures at this stage of the game. Perhaps because we are getting close to our final destination it is now time to think about things like whether the pizza has been nibbled a little along the way or whether the topping are right.

We can request a resend - Selective Repeat or maybe Go Back N???

The "Consumption Level"

- Protocol:
  - pay money
  - eat pizza
  - maybe, if it whets the appetite, order some more…
  - wish you'd ordered garlic bread…
    - ask for some to be sent around...



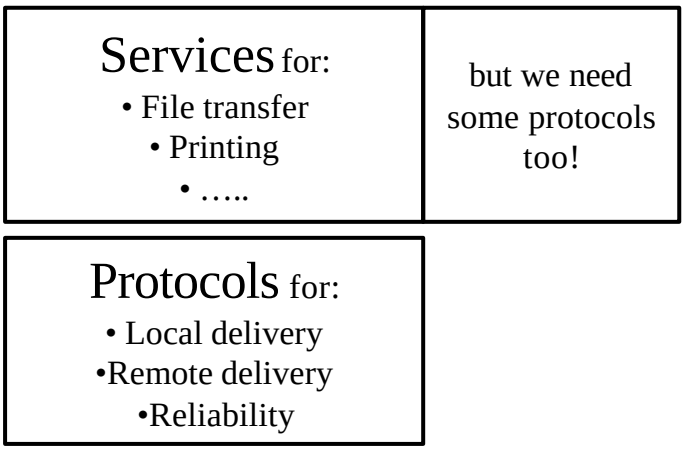
There is one more level to this delivery and this is the actual eating. This **Consumption Level** is where we need to think about things like ordering more or perhaps calling for some garlic bread.

On the networking side we have not considered this level yet. This is where the real work gets done - assuming that all the local **and** remote **and** reliability stuff has worked OK we are now in a position to get some real work done.

In a sense we have reached a watershed in our talking about networks - we have reached the stage where we stop talking only about **Protocols** and can begin to talk about **Services.**

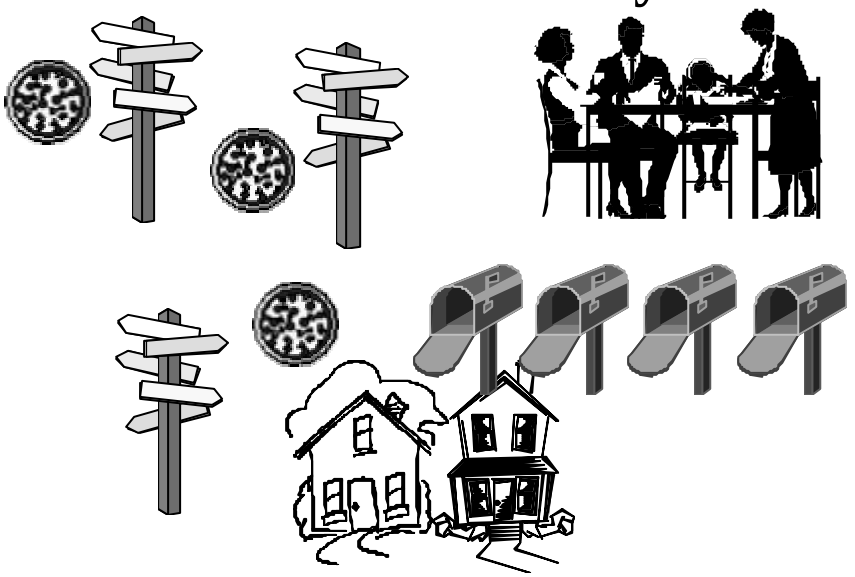
That does not mean that we will no longer think about protocols - each service will require its own protocol - but the level of work that these protocols do is now on a level above simply delivering the packet (or the pizza)

From Protocols to Services



This slide make the point still clearer and really summarises the whole of the Network Protocols and Services unit.

Overview of Pizza Delivery



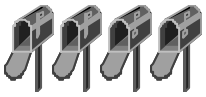
Back to pizza!

In this animation we see the whole delivery spelled out step by step. The end goal, remember, is the eating of the pizza - everything else is just a means to that end.

A kinder view!!!

Consumption Level

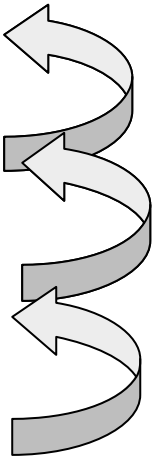
Service



Flat Level - Reliability

House Level - Local Delivery

Street Level - Remote Delivery



This is a less messy view of the delivery and one in which we make an important point about those signposts that showed up in the Street Level addressing slide.

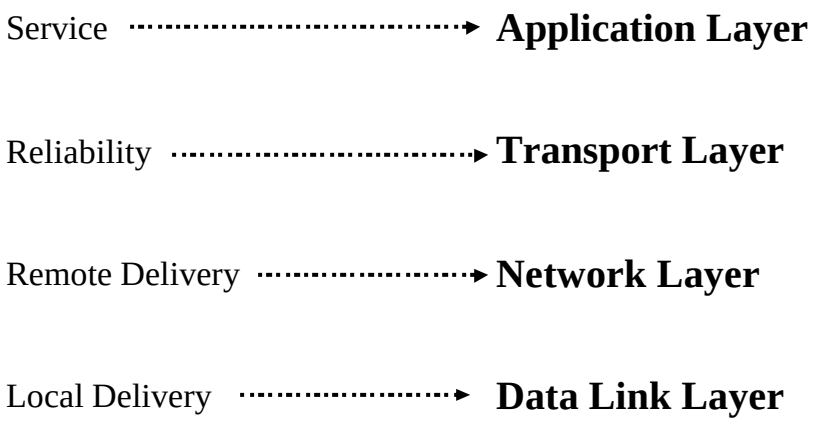
When we spell the delivery out, in terms of the different level of addressing it appears that the Street Level operations take place before the House Level operations. You already have learnt a different way of looking at things - you should be pretty clear by now that the Frame Header (House Level addressing) is the outermost wrapper on the packets in a network. What has gone wrong with our analogy?

The problem lies in those signposts! Really there is no such thing as distinct entities that only act as signposts. When we want to get a direction for our next turn off we go to a special house where there is some geographical knowledge available and they will send us on our way.

This was made clear in the final NetSim exercise where the computers became routers simply by adding a second virtual network card.

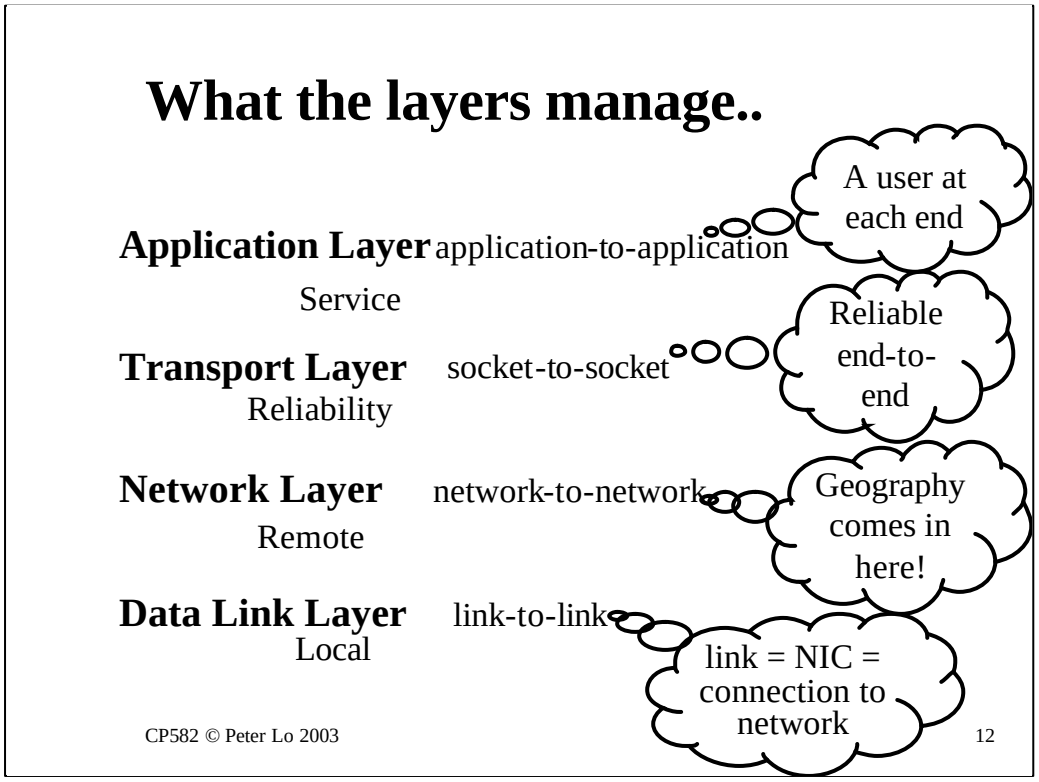
If we make (slightly special) houses the source of our routing information we can re-arrange the layers to look more like what we have become used to.

New Names for Layers



In the real world of networks the terms that are used to describe these different levels of operation are not the ones we have been using so far in NPS.

This slide shows the more conventional terms - everything is a **Layer** now.



These layers, given their official names, are also given official sounding job descriptions.

Application Layer

This layer is concerned with communication between two, remote-from-each-other, applications. Ultimately there may be users controlling the applications at either end but often the application may just manage by itself.

Transport Layer

This layer does double duty. It manages communication between sockets (also called **ports**) on remote computers but it also makes sure that the data is intact. It manages reliability.

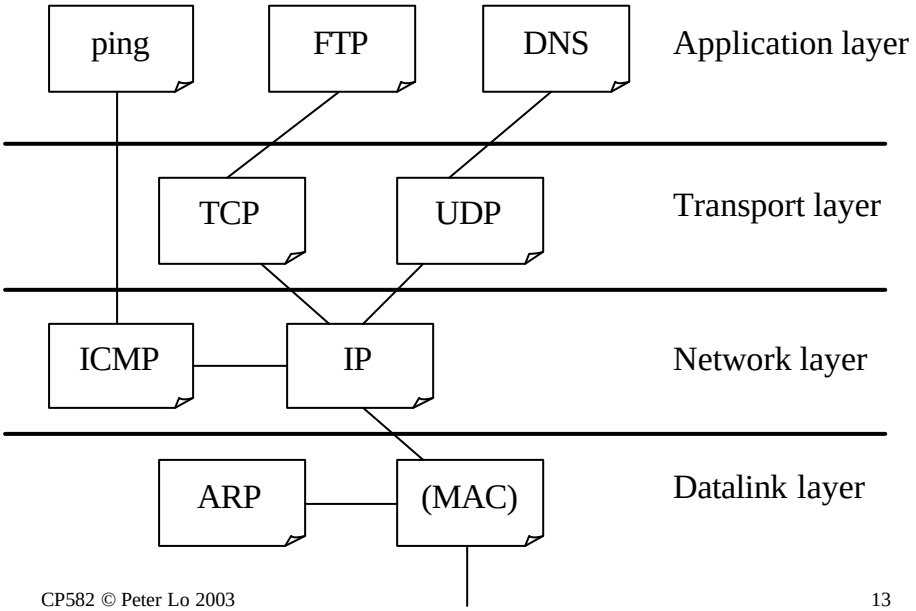
Network Layer

Here the geographical organisation of the network is managed. Data flows from network to network under the control of the Network Layer.

Data Link layer

Another new word! What used to be a NIC is now referred to as a “link”. The Data Link layer manages transfer from Link to Link (easy really)

Layers in TCP/IP



Each of the protocols that we have looked at in TCPIP fit quite neatly into this layer model. This slide, which would have been meaningless a few weeks ago, now says quite a lot about the protocols we have been using.

A set of protocols that work together in this way is what we call a **Protocol Stack**

New term - Protocol Stack

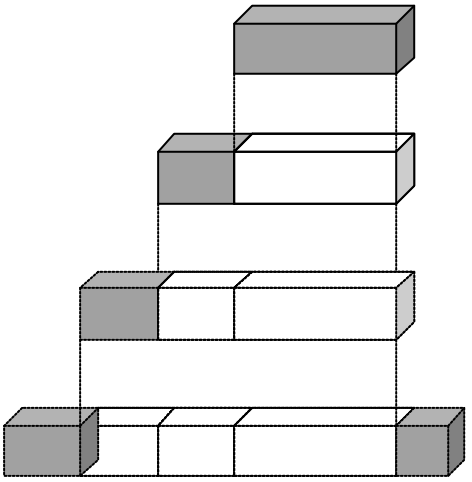
- A set of protocols that work together
  - Implementing a variety of network services
  - Different protocols for various layers
  - Each layer communicates with the corresponding layer at another node.
    - By using headers
    - Which **encapsulate** the previous layer

The title of this week's lecture introduces the term **Protocol Stack**. This is the term we use to describe the way that the dance of the layers takes place for different packets in different protocols.

You have seen examples now in NetXRay of TCPIP and IPX packets and yet the overall view presented when you view the packet is basically the same. TCPIP and IPX are two distinct Protocol Stacks that do the same overall job as each other - using the same view of the world in terms of layers.

It is this universally agreed view of the layers that networking is to be divided into that enables the designers of the different protocol stacks to understand what each other are doing - and maybe even write programs that translate between the different stacks.

Encapsulation

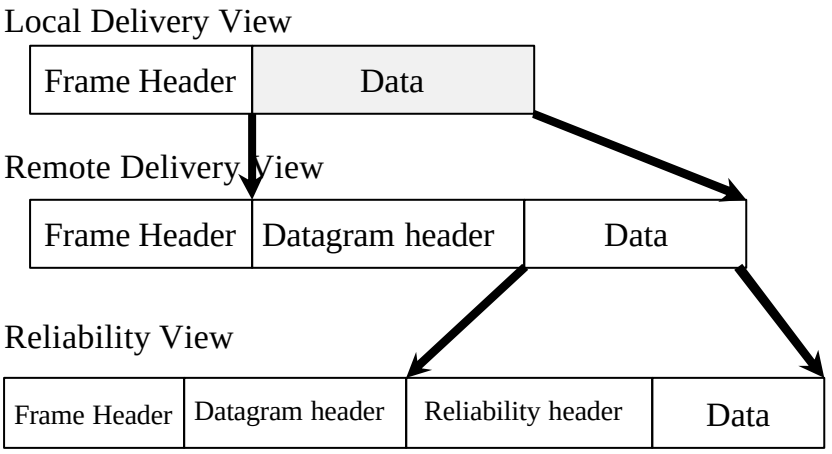


This is the key to it all - **encapsulation** - of one layer's header and data by the next layer's header. In the very first lecture I used the term "black boxing" to refer to this same process.

This slide should remind you of one from Week 5 where the reliability header was added onto the beginning of the packet producing a packet that had four separate "pieces" in the end.

Here we are doing the same thing but specifically with TCP/IP sent over Ethernet. Notice the CRC that is added to the Ethernet frame.

We've seen this!

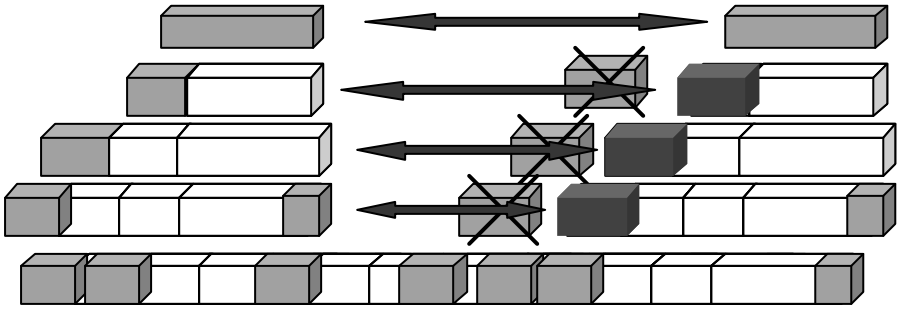


This simply shows encapsulation from another point of view.

What constitutes "data" from the Local Delivery point of view actually contains a header and data when viewed from the Remote Delivery view point.

Similarly the Remote Delivery "data" consists of data and header when we talk in terms of reliability.

Peer to Peer Communication

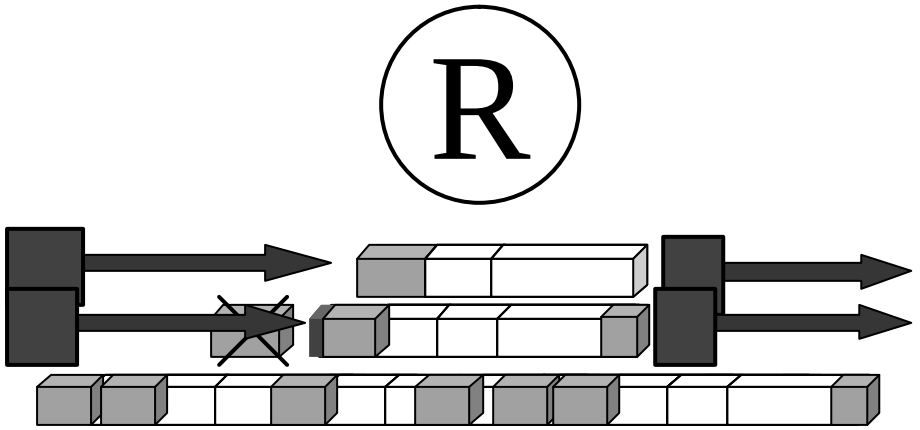


Peer-to-peer communication refers to the way that two nodes communicate with each other. The peer-to-peer stuff is not just happening in terms of the total purpose of the packet but also at each layer below.

Every layer at the sender is using its special header to communicate information to the **corresponding layer** at the receiver. When the receiving layer has finished using the information in the header the header is discarded and the rest of the packet passed up to the next higher layer in the receiver. The identical layer in each of the two computers communicate via the header of that layer. These layers are the **peers.**

As an example consider the sequence number that is put into the reliability header by the sender. This number is "just data" for the lowest (Data Link) and second (Network) layers but when it reaches the Transport layer in the receiver this header is inspected and the sequence number is used to generate an acknowledgment number. Once this has been done the header is finished with and the inside data (the "real message") is passed on up to the Application layer.

What about a Router?



The peer-to-peer communication that takes place when there is a router along the way only extends up to the Network layer. The router treats all the rest of the packet as “just data”.

In the animation we see the Network header being inspected, not discarded, but a new Data Link header being written before the packet is relayed to the next router or the ultimate destination.

The red arrows, which show the peer-to-peer relations stretch forward, or backward, to the next router or node in the chain.

File Transfer Protocol

- A detailed look at an Application protocol
  - We can now assume **all** that has come before
  - Black boxing!!

So this is the point at which we officially make the shift from **Protocols** to **Services.**

The service that we will look at is the File Transfer Protocol which is an **Application Layer** protocol for transferring files between computers.

FTP's Connections

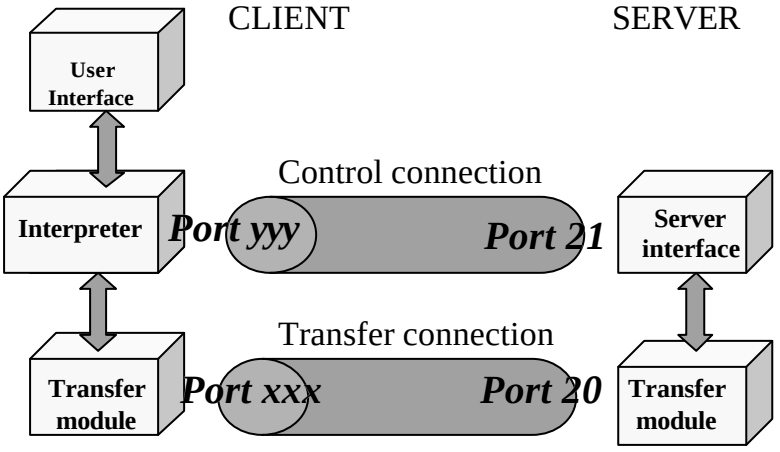
- FTP uses two TCP connections
- Control connection
  - This was what we saw in the week 5 lab
  - It stays open for the duration of the FTP session (FIN at the end)
- Data transfer connection
  - a new one opens for each file transfer
  - such connections are optimised for maximum throughput

FTP enables a node to have a session with a remote host during which a number of different files can be transferred in either direction.

The "session" is in the form of a Control Connection - this is a long-lived TCP connection that begins with the first three-way handshake that you captured two weeks ago and remains in place until you Quit.

During this time each new task is given a new connection - a Data Connection - which is a completely separate TCP connection. Each Data Connection has its own three way handshake to initialise it and a FIN at the end of the transfer.

FTP Modularity



The Client remains connected to the Server throughout the FTP session via the control connection. This is always on **Port** 21 (an internal address within the client or server) Thus "Port 21" almost comes to mean "FTP".

The Transfer connections are opened and closed as required by the commands that the user has entered. They are on different, negotiated, port addresses at the client end but always on Port 20 for the server.

User Interface Commands

For human use!

```
ftp> help
Commands may be abbreviated.  Commands are:
        delete          literal         prompt          send
?       debug           ls              put             status
append dir              mdelete         pwd             trace
ascii disconnect        mdir            quit            type
bell    get             mget            quote           user
binary glob             mkdir           recv            verbose
bye     hash            mls             remotehelp
cd      help            mput            rename
close   lcd             open            rmdir
ftp>
```


These are the commands that you can type in at the User Interface of FTP. If you have a graphical interface for your FTP then the GUI will be designed to implement all these commands.

Control Connection Commands

- Commands sent “on the wire”
- Some common ones are:
  - LIST filelist
  - PASS password
  - STOR filename
  - RETR filename
  - TYPE type
  - USER username

Uses 3-4 byte ASCII for commands

These are the commands that are sent and received on the wire. They correspond quite closely to the User Interface commands.

If a GUI were involved at the client node the commands on the wire would be **identical** with those sent by the command line version of the program.

Control Connection Replies

- 3-digit numbers
  - for computers
- (Optional) character string
  - for people
  - e.g.
    - 125 “Data connection open; starting transfer”
    - 200 “Command OK”
    - 331 “User name OK, password required”
    - 425 “Can’t open data connection”
    - 500 “Unrecognised command”

The traffic from the server to the client on the control connection is in the form of replies that are designed to be read, primarily, by the client computer. Each response has a code number that has a specific meaning.

There is also a text response that can be turned on in order to give humans a chance of understanding what is going on! We will use this in the lab.

Data connection use

- The data connection is used for
  - sending files (in either direction)
  - sending results of directory listings
- Why does it close?
  - In certain modes of transmission, end-of-file is indicated by closure of the data connection

The data (or transfer) connection is used whenever there is something other than a command or response to send. This includes directory listings as well as files moving in either direction.

The sender indicates that the end of the file has been reached by closing the data connection.

How does it work?

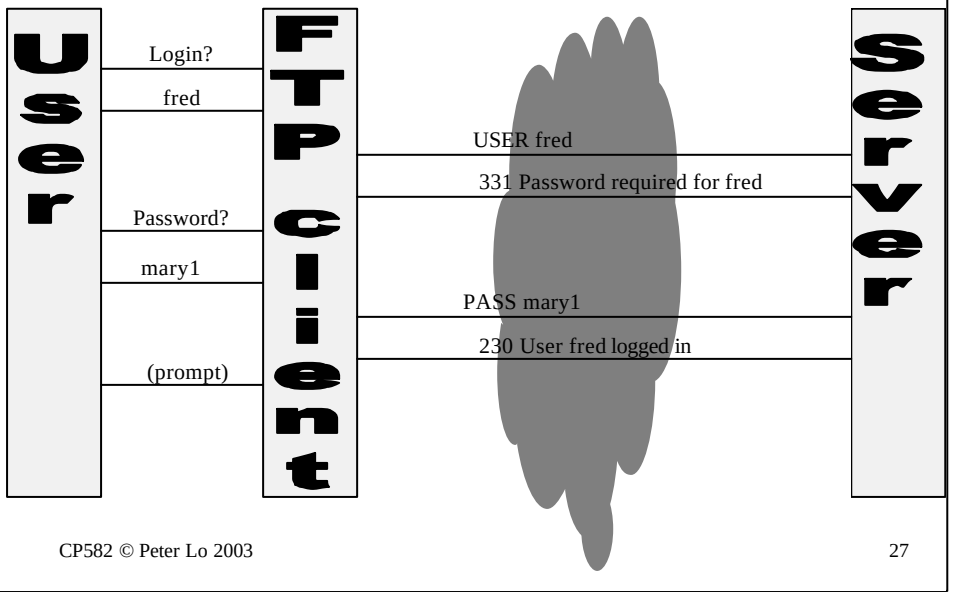
- The data connection is set up like so:
  - 1. The user issues an FTP command that requires a data connection
  - 2. The client program chooses a local port number
  - 3. The client program sends this across the *command* connection with the PORT command
    - PORT n1,n2,n3,n4,n5,n6
      - IP address: n1.n2.n3.n4
      - port address: n5*256 + n6
  - 4. The server receives the client port number, and connects to it. It uses local port 20.
  - Have a look in */etc/services* on a UNIX system

This describes the sequence of steps that is involved in the setting up of a data connection. Note the use of the PORT command on the wire in order for the client to tell the server what Port is going to be used this time.

/etc/services (on a Unix host) is where the standard uses of the port addresses is defined.

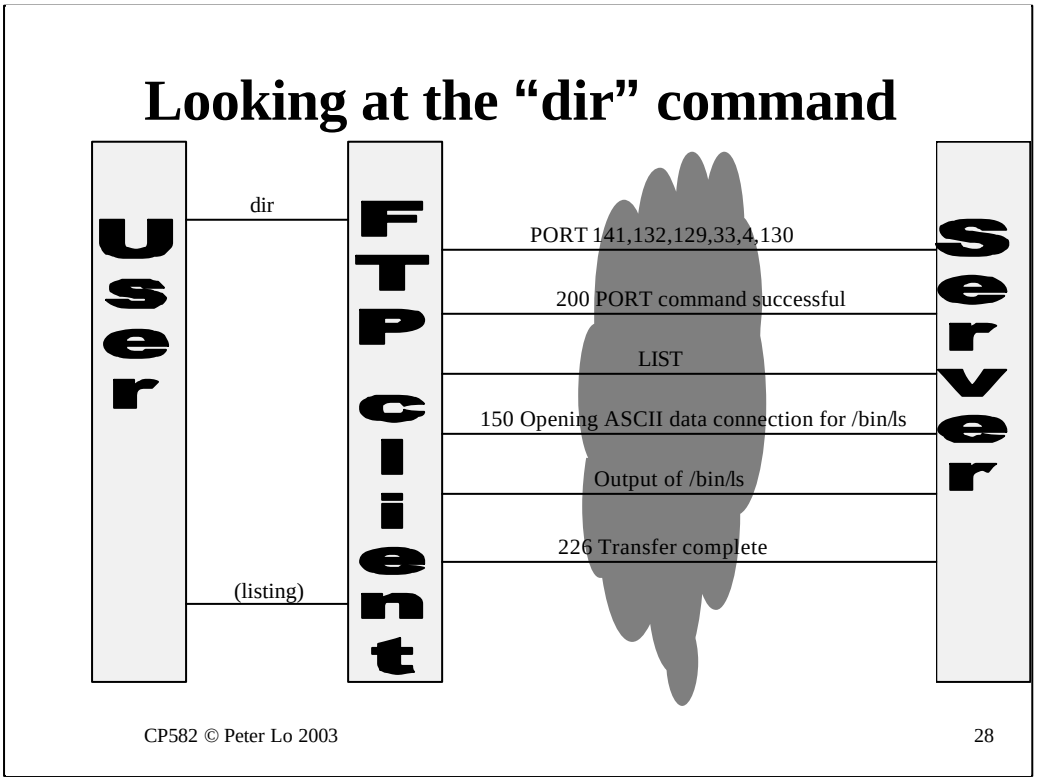
Mapping user-FTP “layers”

Observable by running FTP in debug mode



Here we see an FTP login taking place.

The User (human!) to FTP client program interactions are shown on the left hand side and the conversation that takes place across the wire on the right.



When the DIR command is used there is a Data Connection opened up as well.

The transport layer - TCP

- Port addresses
  - recall that transport-layer services provide port-to-port connectivity
  - the FTP server waits on port 21 by convention
- TCP provide a reliable transport service
  - TFTP uses UDP, an unreliable transport service
  - As you have seen, TCP is connection-oriented, and uses a 3-way handshake to establish connectivity

The reliable nature of TCP is what underpins FTP.

If we want to be "quick and dirty" there is another protocol **Trivial** FTP (TFTP) that uses UDP for its transport. TFTP has no passwords and user ids and is used, for example, to transfer boot images to diskless stations.

TCP Sequence Numbers

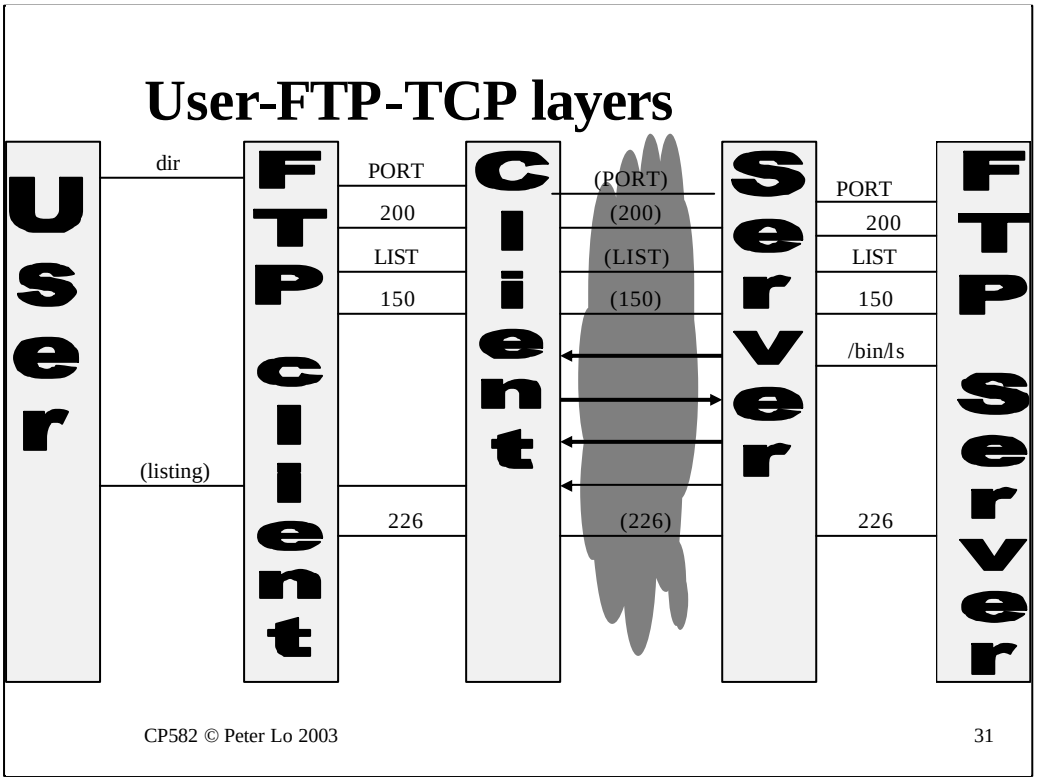
- The Acknowledgment Number is designed to confirm the number of bytes received
  - ACK# = SEQ# + number of bytes received
  - eg
    - Rcvd: Packet with SN=200 containing 50 bytes
    - Send:Packet with AN=250
- In the Lab you will be confirming this

The TCP Sequence Number (a separate number is used by each "end") was negotiated during the 3-way handshake at the beginning of the connection.

From that point on each end uses the Acknowledgment number field to **explicitly** confirm reception of each data byte that was sent.

To make this clear consider an example:

- Packet arrives:
  - Sequence Number of packet: 200
  - Contains 50 data bytes
- Reply sent:
  - Acknowledgment Number: 250
  - Sequence Number: Whatever our current SN is



The final slide shows the lower levels of the FTP server and client managing the additional TCP connection that carries the data that is being transferred.