Internetworking and the Internet

University College London
Department of Computer Science
Author: Graham Knight

The Internet

The Internet is a collection of **physical networks** connected together by **routers** and **point-to-point links**.

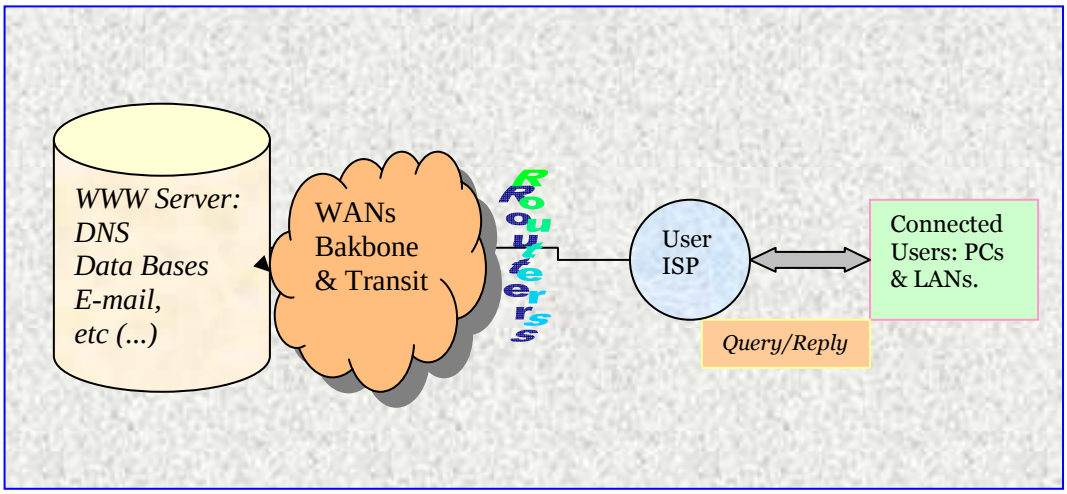
- Physical Networks – LANs, MANs, WANs – anything! (One of the strengths of the Internet.)
- Router – a packet-switch designed to handle Internet Protocol (IP) datagrams.
- Point-to-to point links – cable, optical fibre, microwave, etc.

Most users have a link via the PSTN to connect to an **Internet Service Provider** (ISP). The ISP can support many simultaneous users. The links provide access to a router, which is attached to the ISP's own facility. For a typical ISP, this would probably be a high-speed LAN – fast Ethernet or Gigabit Ethernet perhaps. It will also have **servers**, for example:

- Mail server – receives and stores email on behalf of users.
- WWW server – perhaps including pages of the ISP's service.
- Domain Name Server – see later.

The word "server" is used to mean a program implementing a service, as well as the host on which the service runs. It is possible to run several server programs on one server host.
The ISP will probably have a second router giving access to the rest of the Internet. Typically, this will use a link to a **transit network** which might serve a region or country. There will have to be a commercial agreement under which the transit network agrees to carry the ISP's traffic over the Internet.

The heart of the Internet are the **backbone networks**. These are very high capacity networks which carry traffic between transit networks. Once more, these are owned by separate organisations so commercial agreements will be needed.
Every interface between a host or router and the Internet has an **IP address** – e.g. 129.16.3.1. These addresses appear in the headers of IP datagrams and are used by routers to determine to where a datagram should be sent next.



Normally a client starts with a **domain name** like "www.foo.com". This must be mapped to an address. The Internet has a **Domain Name System** (DNS) to perform this task automatically. Communication usually begins with a message to the ISP's Domain Name System server asking for a mapping. If this is successful, the client now has an address which can be put in the header of an IP datagram and the datagram can then be launched towards its destination. The top-level domains have been fixed for a long time. There are five world-wide generic domains (*com*, *org*, *net*, *edu*, and *int*), two US-only generic domains (*mil* and *gov*), and country code domains (e.g., *us* for the United States, *uk* for the United Kingdom, etc.).

TCP and UDP Ports

· IP addresses identify hosts
· Ports identify processes
– Like telephone extension numbers
– When packet arrives, OS looks at ports and chooses process accordingly.

"Well-known" ports
– HTTP (WWW server) 80
– FTP (FTP server) 21
– SMTP (Email) 25

IP addresses identify hosts. We need also to identify a particular service or process. This is the role of TCP ports. There is a close analogy with telephone extensions; the telephone number identifies the building, the extension number identifies an individual within the building, and so on. Public services use "well-known" ports (see above). When a browser sets up a connection to a WWWserver it must send a TCP SYN PDU with the destination port set to 80.

TCP Example – The WWW

Web pages are identified by Universal Resource Locators (URL). URLs have the form <protocol>://<domain name>/<path name>. The <protocol> is usually HTTP (Hyper Text Transfer Protocol) which was specially designed for the retrieval of web pages. However, others are possible – for example FTP (File Transfer Protocol) for server communication. Whatever protocol is specified will be used to retrieve the page from the server. Most of these protocols use TCP to provide reliable sequenced delivery.

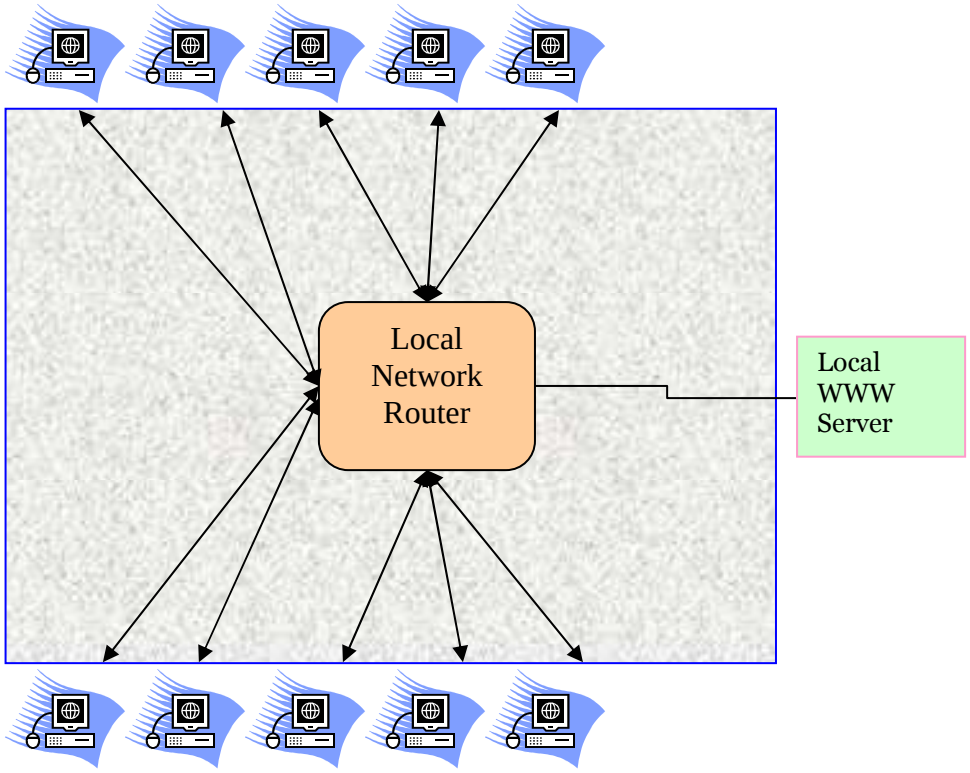
The <domain name> is the name of the host on the WWWserver. This will have to be mapped to an IP address via DNS before the TCP connection can be set up.
The <path name> locates the page within the server system's file system. Most often the pages that are retrieved contain text formatted using the HTML mark-up language. HTML pages usually contain links to other pages. Pages often contain images. The image files are specified by URLs contained in the HTML. Each image must be retrieved using a separate TCP connection. Typically these run in a parallel mode.

TCP and Network Congestion

- Routers can become congested
  – Packets delayed, eventually some are dropped
- TCP knows about dropped packets – retransmits
- TCP implementations slow down (reduce the window)
- ***Congestion is reduced***

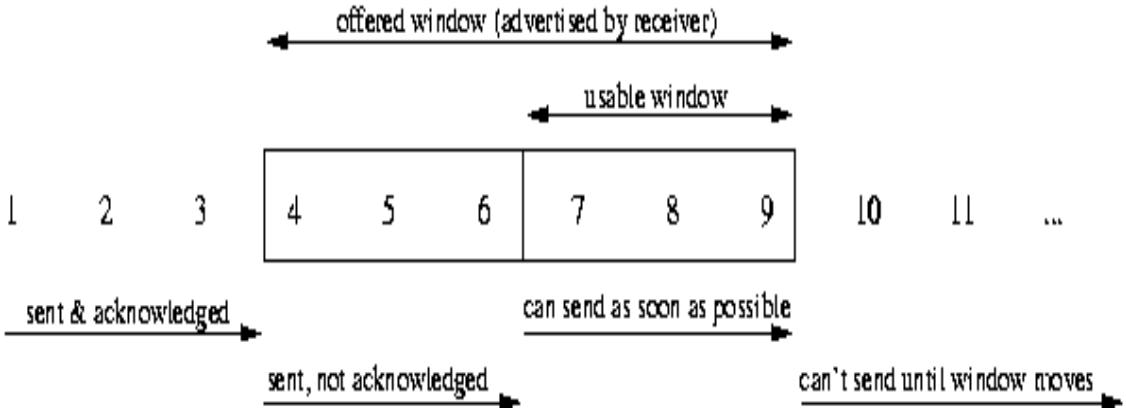
The Internet can become congested when many IP datagrams converge on a single router. Queues will form at the router causing increased end-to-end delays. Eventually the router will begin to discard ("drop") packets. TCP can recover dropped packets by re-transmitting these pakets. However, if not handled carefully, any such packet retransmissions simply add to the congestion.

Most TCP implementations implement a "congestion avoidance" algorithm. When they detect a dropped packet they reduce their transmit window so as to slow down their transmission rate. If all hosts affected by the congestion behave in this way then the congestion will be reduced. Several important and growing applications do not use TCP. These include voice and video applications, as will be seen at a later stage. Efforts are presently being made to give these applications "TCP-like" behaviour so that they can become good "network citizens".



The Sliding Window

Sliding window is a technique for controlling transmitted data packets between two network computers (nodes). Reliable and sequential delivery of data packets is required such as when using the Transmission Control Protocol (TCP). In this technique each data packet includes a unique consecutive sequence number used by the receiving computer to place data in the correct order. See the following **Figure**.



The technique places varying limits on the number of data packets that are sent before waiting for an acknowledgment signal back from the receiving computer. The number of data packets allowed is called the *window size*. The limits on window size vary depending on the rate at which the receiving computer can process the data packets and on the capacity of its buffer, i.e. its internal storage.

If the application in the receiving computer processes the data packets at a slower rate than the sending computer is sending them, the acknowledgment signal from the receiving computer will tell the sending computer to *decrease* the number of packets in the window size in the next transmission, or to temporarily stop transmission to free the buffer. If, on the other hand, the receiving application can process the data packets faster than the sending computer is sending them, the acknowledgment signal will tell the sending computer to *increase* the number of packets in the next transmission.

For efficient transmission, the transmitter must not stop sending pakets for unnecessarily long time intervals. This will happen if the receiving computer sends an acknowledgment signal to stop and does not send another signal to begin transmitting when its buffer has available space or is empty. The flow control in TCP is based on a *sliding window* protocol which allows the sender to transmit multiple packets before it stops and waits for an acknowledgement. This leads to faster data transfer since the sender does not have to stop and wait for acknowledgement after each time a packet is sent. The sliding window protocol can be visualized as in the above figure.

In the figure, the bytes are numbered 1 through 11. The window advertised by the receiver is called the *offered window* which covers bytes 4 through 9, meaning that the receiver has acknowledged all bytes up through and including number 3, and advertised a window size of 6 which is relative to the acknowledged sequence number. The sender computes its *usable window* which is how much data it can send immediately. As data are sent and acknowledged the window moves to the right and the relative motion of the two ends of the window increases or decreases the size of the window.

Voice over IP

– Share one network for voice and data
– Efficiency of statistical multiplexing:
– Silence suppression

• … but there are problems:
– Must have low end-to-end delay (< 100ms)
– Needs low jitter

Using IP for voice, rather than the traditional approach, is attractive for several reasons:

- The IP market is very dynamic which drives own equipment costs.
- IP is very flexible. It is easy to introduce new, high-capacity technologies under IP while keeping the basic switching mechanisms the same.
- Using one network for voice and data rather than two saves money.

However, because of store-and-forward and queuing delays at routers it is hard to meet the low end-to-end delay requirement for interactive voice. There is also a problem of high delay variance (jitter) which makes it hard to deliver a constant stream of bits to the remote codec (coder-decoder). At present, voice quality tends not to be as good as the traditional approach unless the network is small and lightly loaded. However, mobile phones are gradually becoming more reliable and with better quality.

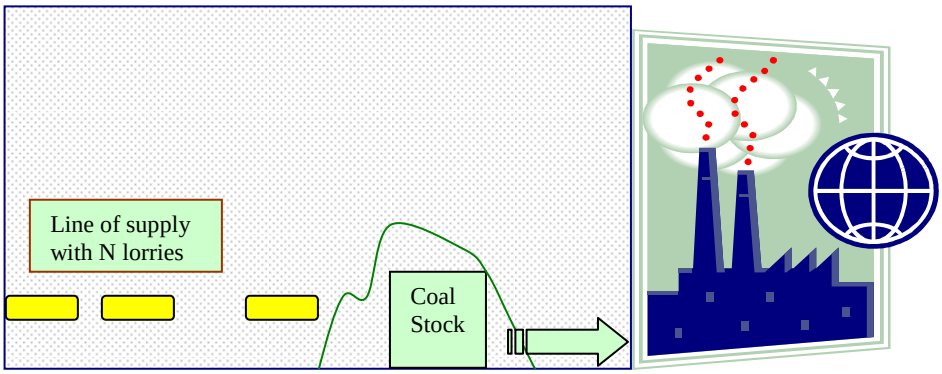
Transmitting Voice

- Need to digitise – sampling by a codec
- Chop bit streams up and allocate to packets
  – Each packet (say) 50 ms
  – Timestamp and Transmit
- Receiver unpacks packets, re-generates bit stream and passes to codec
- … but what about delay and jitter?

Once the voice has been digitised by a codec it is relatively straightforward to packetise it. One decision is how much voice to put in each packet. Large packets minimise the header overhead for transmission. However, they increase store-andforward delay and more speech is lost when a packet is dropped. Hence, the need for an *optimum* paket size. Packets are timestamped. The timestamps are used by the receiver when a packet is lost or delivered out of sequence.

Handling Jitter

- Receiver needs a regular stream of packets
  – If jitter => it will ***not*** be regular
- Power station requires a regular stream of coal lorries
  – Need to keep stockpile in case lorries are delayed
- Voice receiver has “play-out buffer”
- First few packets stored in buffer (not passed immediately to codec)
- When delay occurs, buffered bits can still feed codec
  – … but buffer imposes an extra delay!



A play-out buffer will be needed at the receiver. This acts in a similar way to a stock of coal at a power station. If deliveries of coal are disrupted then electricity production can still continue using coal already stockpiled. Of course, eventually, the stockpile could run out. On the other hand, an overlarge stockpile would be costly and unwieldy. Again, there is a need for an *optimum* buffer size.

When a voice connection starts the receiver will accumulate a few milliseconds of speech in the playout buffer before it begins sending bits to the codec. If packets are delayed or lost, then the stock of bits in the playout buffer can keep the codec continuously fed.

The problem is that the delay imposed at the beginning when the buffer was initialised is never recovered. Thus a playout buffer tackles jitter at the expense of end-to-end delay. The more jitter there is, the more bits must be held in the play-out buffer and the more delay there will be. If there is a great deal of jitter, the delay will eventually become too much for interactive voice. Note that if the transmission is not interactive - a sports commentary for example - then delays of several seconds are acceptable. In such cases, very large playout buffers can be used and high levels of jitter can be tolerated.

User Datagram Protocol - UDP

- Not all applications will want (or tolerate) ARQ delays, e.g. VoIP applications.
  - Each packet contains (say) 50ms of speech
  - Retransmission might take 1 sec

- Better to lose packet than retransmit
  Then => use UDP *not* TCP
- Same service as IP but adds ports
- End-to-end service (routers do not know about UDP)

We have seen that VoIP transmission becomes tricky when jitter is high. Unfortunately, ARQ procedures such as those in TCP, *increase* jitter since they introduce occasional very large delays when re-transmitting. A voice application would prefer these lost packets to be lost for ever rather than have to deal with the increased jitter. Losing one packet (ie. about 50ms of speech) will not be especially disruptive to a listener.

Therefore, we want an alternative to TCP which does not use ARQ procedures. In the Internet world, this is known as the "**User Datagram Protocol**" (**UDP**). UDP adds nothing to the basic IP service; thus UDP datagrams may be lost, delivered out of sequence etc. However UDP does have a small header which includes ports which are used in the manner previously described for TCP.

VoIP, UDP and Congestion

- TCP does congestion avoidance ("well behaved") but UDP does not
  – Applications which use UDP may be "badly behaved"

- **Solution 1**
  – Make applications well-behaved
  – VoIP could reduce voice quality (hence bit-rate) especially when the network is congested
- **Solution 2**
  – Make the Internet able to defend itself
  – Admission control and traffic policing

VoIP is one of several important Internet applications that use UDP rather then TCP. One problem that arises here is that the congestion avoidance implemented by TCP is lost. This means that, when congestion occurs, all the applications using TCP are slowed down in order to reduce the congestion but UDP applications carry on regardless.

One solution is to make these UDP applications better behaved. For example, a VoIP application which detected congestion could change to a more compressed voice encoding which required less capacity. There might be some loss of voice quality as a result of this.

Another solution would be to change the Internet so that it is able to force applications to slow down. This would be a big change to the Internet which generally has the philosophy of providing a simple, minimal service and pushing as much complication as possible back to the hosts. Nevertheless, more recent work on protocols allow applications to signal to the Internet what capacity they need. The network is then supposed to decide whether it has the capacity to support the requested activity and, if it hasn't, to deny the request. This is "**admission control**". If the request is accepted the network will need to monitor the activity to ensure that the actual capacity used is no more than what was reserved. This is known as "**traffic policing**".

Modelling Data Networks

• View "communications service" as a "black box"
• Service is defined by its interface
– "Service users" make requests
– Service may generate "indications"
– "Service provider" implements the black box

Networks are complicated with many different technologies and protocols. Sometimes these are alternatives, sometimes they are complementary. It is useful to have an abstract way of thinking about networks. This should help us to understand how the various components can be fitted together. The bestknown model is the **ISO 7-layer model** for "Open Systems Interconnection" (OSI). The acronym ISO stands for the International Standardization Organization.

In the 1980s, OSI was seen as as a set of standards that would grow to replace everything else on the Internet. It has completely failed in this for a variety of reasons: technical, political and others. However, the abstract model remains rather like the smile on the Cheshire cat ("Alice in Wonderland").

The starting point is a black-box model of a "**communicatuions service**". The idea here isas follows. A service user can make a "**request**" to a service - for example we can ask it to deliver a piece of data to another user. If the service carries out the request successfully. the receiving user will receive an "**indication**". (Note that communication services do not always succeed in carrying out requests). To take a familiar example, when we pots a letter we are making a request to the postal service. When a letter drops onto our doormat, that is an indication from the postal service.

Confirmed versus Unconfirmed

The ordinary postal service is an example of an "**unconfirmed service**"; once a letter is delivered, the postal service takes no more interest in it. Sometimes we would like the postal service to extend its responsibility and to tell us that the letter was definitely delivered into the hands of the recipient. This is an example of a "**confirmed service**". The recipient has then to tell the service that they have received the indication. This is called a "**response**". Eventually the service tells the sending user that all has been completed successfully. This is a "**confirm**".

Notice that two users can turn an unconfirmed service into a confirmed one by agreeing between them that the recipient will always send an ordinary letter in reply. The difference, however, is that the responsibility for following this procedure rests entirely with the users. With a confirmed service, the service itself takes the responsibility.

The End-to-End Argument

- WWW access needs reliable, sequenced delivery – Use TCP
- Voice over IP needs low delay and jitter – Use UDP
- TCP and UDP are in the "end-systems" (hosts)

We can choose which to use without altering the network

- Suppose IP did ARQ error correction…
- IP – connectionless, unconfirmed

Good design choice

We have seen that different applications require different kinds of communication services; the WWW requires reliable, sequenced delivery and is tolerant of delay and jitter. VoIP is very sensitive to delay and jitter but can tolerate some data loss. Fortunately, two services, those offered by TCP and UDP, are available to meet these requirements. The reason we have the flexibility to choose between the TCP and UDP services is that these services are implemented entirely in the end systems. We don't have to negotiate with the Internet in order to chose between them - it doesn't care what we do.

Suppose we re-invented the Internet with a view to "improving" its service. For example, we might give it the responsibility of delivering data in sequence and of recovering any packets that were lost. This would be fine for the WWW but hopeless for VoIP.

The "**end-to-end argument**" says that services should be simple so as to leave as much flexibility as possible to the end-points of a communication (what we have been calling the Internet "users"). The justification is that only the users really know what they need from the communications between them. Any attempt by a service provider to second-guess the users will, surely lead to trouble. The simple connectionless, unconfirmed service offered by IP is very much in line with the end-to-end argument and is a major contributor to the success of the Internet.
