Networks: The Internet and Beyond

Professor Richard Harvey FBCS

April 2021

If you have ever experienced the frustration of working in a hotel with poor wifi you will be aware of how critical a connection to the internet is to everyday life. But the internet is a surprisingly recent idea. Transatlantic telephones calls had been a reality since the mid 1920s, but it was not until the early 1960s that one could buy a modem (a modem, or modulator/demodulator is a device that converts digital signals from computers into signals that can travel down telephone wires optimised, usually, for voice communication). Early modems such as the Bell 101[1] or, in the UK, the Datel 1A, had data rates of a few hundred bits per second and they could only connect one computer to another (or point-to-point as it is often called).

Such point-to-point networks were not useless – large corporations often used temporary or leased-line connections to allow bureau computing[2] or to provide access to the mainframe from remote sites. Early home users could use modems to dial-up bulletin boards which would then store the information which others could see when they logged in – a sort of non-realtime Mumsnet. While technically these architectures are networks, they are constrained by the carrier which in this case was the telephone network. Telephony had evolved over many years to provide highly optimised lines for voice communication between two people. Once the call was set-up, the telephony company provided a permanent circuit between the two parties for as long as they were prepared to pay for it. The mention of money at this point is deliberate – almost all the telephone companies in the world had devised a business model which assumed that consumers wanted, needed, and were prepared to pay for a what is called a "circuit switched" network.

There seems to be some agreement that Norman Abramson and a small team of engineers at the University of Hawaii first realised that computer communication was fundamentally unsuited to circuit switching. Their discovery was forced on them by geography. At the time the University of Hawaii had one computer, an IBM 360, but four sites on three of the islands. There were no cables between the sites so it was going to have to be radio. Fortunately, there were a couple of unused spectrum bands so, one them was designated for transmission from the main computer, and the other band was to be used for the "back channel" – communication from the remote terminals back to the main computer [1]. And this is where the interesting bit starts.

Of course, all electronic communication is inherently unreliable – there is interference, electrical noise and so on. When we are speaking on the telephone it is simple for us to say "I'm sorry, could you say that again"[3] but how can this work with computer data? Two approaches are commonplace, and often used together. The first is called Forward Error Correction and is a fascinating idea. Can

[1] There seems to be some doubt about what the Bell 101 modem looked like. Some websites show a lady in fifties garb next to two boxes the size of fridges – I suspect that was not the Bell 101. Either way, by the time modems were needed, there were the size of a couple of shoe-boxes. Fancy ones, like the Datel made electrical connections to the line whereas the early US modems often used "acoustic couplers" which were rubberised cups that coupled speakers and microphones to a standard headset.
[2] Bureau computing is where is powerful machine in one location is used to the work needed in another location. The Lyons Electronic Office (LEO) computers ran the payroll for Ford UK for example [6].
[3] It is a pet peeve of mine how frequently one has to say that on cellular telephone calls. The mobile telephone after years of work has considerably poorer performance than a landline which is one of many indignities of our age.

we add a small amount of data to the original message such that the receiver can, firstly, detect that an error has occurred and, secondly, correct it.

A formal way of doing this was developed by Richard Hamming and published in 1950 [2][4]. Hamming picked up on the idea of a parity digit which is additional bit added to data. If the number of ones in the data is odd then we set that additional digit to one so that the total number of ones in the augmented data is even (this is even parity). Hamming worked out that by adding parity digits computed on subsets of the data it was possible to formally prove that one could detect all n-bit errors and, this was the surprising thing, correct them. An alternative, and less form of error detection is just to add the number of ones in the data (this is called a *checksum*). If on reception the checksum does not match the data then likely an error has occurred. In human communication we tend to check data by applying a semantic check so, when Army HQ receives a message from the front that reads “send three and fourpence, we are going to a dance” we know the message is highly unlikely. Unlike humans the forward error correcting codes developed by Hamming need know semantic knowledge and are based on pure Mathematics[5].

The second idea is ARQ or Automatic Repeat ReQuest protocols. ARQs come is several varieties but the basic idea is that the receiver uses the back channel to ask the transmitter to resend material that it has missed. In its simplest form, every block of data is acknowledged with an ACK signal or, if we are employing some form of FEC, a NACK (meaning we have received some data but can see it is not correct in some way). In the earliest computer connections both of these ideas, FEC and ARQ, were pretty soon established and indeed were known about before the invention of the programmable electronic computer.

Alohanet was unorthodox in that, firstly, it was engineered for computers to send bursts of information, or “packets” as they were called (Abramson argued that most computers would want to transmit large bursts of information quite quickly and then there would be long periods of silence). He turned out to be right about that, and of course circuit-switching is fundamentally unsuited to that sort of communication. The second unorthodoxy was the idea that all the nodes would communicate on the same frequency. This was literally heresy – even in relatively free-and-easy radio channels such as CB radio or Marine VHF there are strict calling and communication protocols to avoid people speaking over each other. How the heck were computers meant to avoid this?

The conventional approach would have been to have either to have assigned each a node a separate frequency (frequency division multiplexing or FDM) or a time slot on the same channel (Time Division Multiplexing or TDM). FDM was ruled out because there was only one frequency available for the back channel and fortunately there were no radio engineers around to design an FDM system[6] to work within that single channel. TDM means all the computers have to synchronise their clocks which was possible but tiresome because the sync signal has to be transmitted and there are different transmission delays to account for in the system. It also meant most of the slots would be empty most of the time because most of the time the computers were not wishing to communicate.

The ALOHANET solution was to allow any of the stations to transmit when they liked. If the data was received uncorrupted, then the receiving stations sent back an ACK. If there was no ACK then the transmitter backed-off for a random amount of time and tried again. Repeat until some limit.

[4] Claude Shannon, who shared an office with Hamming at Bell Labs, introduced a teaser preview of the result in his famous “A Mathematical Theory of Communication” paper in 1948 [7].
[5] Actually they are really are pure Mathematics in the sense that most of the developments in this area rely on an area of Maths known as Galois Field theory (specifically GF(2)).
[6] It’s not that hard to design and FDM system but by “fortunately” I mean that the lack of the obvious solution meant that they had to invent something which has proved to be incredibly useful

This very simple algorithm turned out to be very efficient and as we shall see is the basis for the ARPANET, ethernet and the internet as we know it today.

The problem is sometimes known as the Byzantine generals problem[7] although, in its full complexity, some of the generals are known to be either unreliable or turncoats. In our simple version there are two generals each with a platoon on a hill. Their enemy lies in a valley between them. To attack and win they need to coordinate and attack at once. General 1 sends a messenger to the other general with the order to attack. Has the messenger been intercepted by the opposition forces and killed? Or maybe their message has been delivered corrupted to "retreat"? Although in this case the problem cannot be solved with zero error, I hope you can see that an ACK message improves things enormously – if General 1 and 2 were to use an ALOHA protocol then General 1 sends "Attack;" if he or she receives "Attack acknowledged" back in a reasonable time then there is some confidence that the message got through. If there is no response, then General 1 resends the message. If we receive "Retreat acknowledged" then we know there has been dirty work at the crossroads and we send the message again. Now clearly this does not circumvent all the sneakiness of Byzantine generals but, reverting to ALOHANET, it is good enough.

In network parlance when two transmitters transmit at once we usually refer to a "collision". Depending on the medium, radio waves, wires, optics and so on it may be possible to spot a collision just by listening to one's own transmission or it may be necessary to design an ARQ to tell transmitters that a packet has been correctly received. Either way, the two transmitters need to be persuaded to "back off" and retry at a different time. If they are both running the same deterministic back-off algorithm then it's like one of those awkward moments when both people speak at the same time, and both try again to speak at the same time. ALOHA introduced the idea of a random back-off time.

Meanwhile, in mainland USA, the Advanced Research Projects Agency (ARPA), now DARPA, had been funding a few universities to get connected via a network called ARPANET. The first connections took place in 1969 and used a protocol called NCP or Network Control Program. However, the internet as we know it today did not really come into being until (i) there was created an iron bureaucracy to run it[8] and (ii) there was a more flexible protocol known as TCP, the Transport Control Protocol. However before describing TCP, we ought to briefly look at the way that packets are formatted and addressed: this is IP the internet protocol.

IP specifies that every packet of information contains two parts: a header and the payload. Together they form a *packet* or *datagram*[9]. A datagrams is not just any collection of data, the critical feature is that it contains enough information to be routed from the source to destination computer without any additional chit-chat between the computers. A good analogy is that it is like a postcard – it contains both the address and the information and, like a postcard, it is sent off into the network and the network manages to route to the receiver without have to hold-open a circuit. Unlike a telephone call, it is designed for a *connectionless* network. As soon as one decides to send data via datagrams, users essentially lose control of the routing information – in principle parts, or all, of that angry email you sent to your elected representative may have travelled via Manchester, Milwaukee, Moscow or Mecca.

[7] One of the important contributors to this problem Leslie Lamport wrote that he wanted to get away from the previous name which was the Chinese generals problem. In the full version of the problem some of the generals might be either unreliable or renegades so it's easy to cause offence. Albania was closed at the time so he decided on "Albanian Generals" only to be told that there were a great number of Albanian immigrants in the USA. Hence Byzantine generals (https://www.microsoft.com/en-us/research/publication/byzantine-generals-problem/)
[8] It is fair to say that the instigators of the internet's management practices would be horrified to referred to as a bureaucracy – they saw themselves merely as custodians of good ideas – more on this in a moment.
[9] The word datagram is a hybrid of the words data and telegram.

The specification of these datagrams was published in September 1981 in a document known as an RFC (Request for Comment). RFC 791, edited by one of the founders of the internet John Postel, defines IP, the internet protocol. The essence of RFC 791 is to define the format of a *header* that contains two 32-bit blocks that give the source and destination of the packet. The header also tells intermediate computers on the route (*routers* as we now call them) whether this packet can be split, or *fragmented*, into smaller ones. This was a critical feature in the 1980s: buffer memory was expensive so it was unreasonable to expect all routers to be able to handle very long packets. The standard demanded that all systems be able to handle packets of up to 576 bytes[10]. Since the header has a maximum size of 64 bytes this effectively imposed a minimum standard on routers of being able to handle data of 512 bytes.

One of the successes of the internet has been the standardisation process. It is fair to say that the process of computer standardisation can be nauseatingly slow. The FORTRAN language for example was devised in 1954. It was standardised in 1978 and that was not particularly slow! The internet is not like that. The Internet Engineering Task Force (IETF) which governs the internet has had a few iterations on its structure (it's now a subset of the Internet Society which carries the risk insurance) but the essential ideas are the same – standards, information[11] and experimental documents are published as RFCs. Once published, they stay published although subsequent RFCs may render previous ones obsolete or "deprecated". One feature of the RFC process is that it is fast.

As I am sure you have noticed IP says nothing about what should be the *payload* in a packet nor does it guarantee to deliver a packet. It certainly does not guarantee to deliver packets in the order in which they were sent. For this, higher level protocols are needed. These are usually specified as packets which themselves fit into the payload section of IP packets. As a computer will be running multiple applications, all communicating in different protocols, there is usually a need for some internal addressing – these "ports" are equivalent of the numbers of flats, or apartments, in a large block at the same address. One of the simplest protocols is called UDP, the User Datagram Protocol (RFC 768) and is the most basic packet-within-a-packet protocol. UDP has two additional numbers, the source port and destination port, which are used by the networking software in your computer to route the packets once arrived, a length parameter (which tells the networking software how much data is to come) and a checksum. The checksum breaks the rule of packet-within-a-packet because it is computed not only over the port addresses but also the IP addresses so it provides some additional robustness against corruption[12].

UDP is a connectionless protocol that delivers packets, or not, in any order. Programmers may want that model but most of us want to establish a connection with a remote computer and then send information reliably, and in order. For that we need TCP – the Transmission Control Protocol – as described in RFC 793 also dated from September 1981. TCP and IP go together and as they are the indispensable duo (the Morecombe and Wise of the Internet) people often mix them up or refer to them jointly as TCP/IP. The aim of TCP is to give the programmer the impression that there is a continuously connected wire between two computers (even though the reality is lumps of packets flying about all over the place). This is called a *virtual circuit*. It is the protocol used by your web-browser when it communicated with www.gresham.ac.uk to download this transcript. As with UDP, the TCP header contains the source port and destination port numbers, recall these help with routing withing your PC, but it also includes three extra fields: a sequence number (TCP gives the impression of providing data in sequence); an acknowledgement number and a window. As we saw earlier with ALOHANET, acknowledgement is one of the tactics used to cope with any Byzantine generals that might be hanging around.

[10] Postel called bytes Octets to, I assume, avoid confusion with computers that were not using 8-bit words.
[11] And April Fool's Day jokes.
[12] The checksum is said to be computed over a "pseudoheader" and is example of one of several forward error control mechanisms within the basic architecture of the internet.

In TCP a connection can either be *closed*; *listening* or *established*. Once the connection is established then are a variety of states possible since each computer is essentially signalling that they are ready to send data or ready to receive. These are necessarily fiddly as, given we are dealing with the internet, communication can be broken at any time so there is a lot of to'ing and fro'ing. To give a flavour of this I have included the basic *hand-shaking* protocol from RFC 793 for establishing a connection. If I was to convert this into human terms then there are two people who want to talk. Initially B is listening. A says "I want to talk, and I shall number by packets from 100". B says "I acknowledge your need to talk, and I shall number by packets from 300. My acknowledgement is labelled with 101 because 100 was your first packet so I'm now waiting for the next which is 101". A says "OK. I acknowledge your acknowledgment. I am at 101 and you are 301". Then A says "Here is the data"…

```
    TCP A                                                TCP B

  1.  CLOSED                                               LISTEN

  2.  SYN-SENT    --> <SEQ=100><CTL=SYN>               --> SYN-RECEIVED

  3.  ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN,ACK>  <-- SYN-RECEIVED

  4.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>       --> ESTABLISHED

  5.  ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA> --> ESTABLISHED

          Basic 3-Way Handshake for Connection Synchronization
```

TCP start-up is designed as a triple handshake because the connection may break at any time leaving, for example, A expecting to send data to B who has gone offline.

There are several observations that follow immediately from TCP/IP. Firstly, it is bulky (think of all of that header data) which tends to encourage longer packet sizes for efficiency. Secondly it is too complex for hardware[13] so as soon as you connect your computer to the internet it is using up CPU cycles running the TCP/IP stack. Thirdly, there seem to be a number of questions left unanswered which relate to exactly how zillions of these packets are managed. How do computers know the addresses of remote machines, or even their own address? How is communication managed so that the internet doesn't collapse under the weight of everyone screaming for attention at the same time? And finally, given that anyone along the route can see everyone's packets (a technique called *packet sniffing*), how to we prevent everyone seeing everyone else's data?

The first one is relatively easy to deal with: there is another set of protocols associated with naming. In the early years of the internet the University of East Anglia asked for and got, for free, a Class B internet address so that every computer in UEA will have an address that starts with 139.222.xxx.yyy. Computers at UEA are either statically assigned to particular addresses by a human (the network guy) or, more likely, when a new computer comes up on the network it asks UEA's domain name server (DNS) for an address which it is assigned dynamically. Routers also consult the DNS so they can build routing tables which allow address fields in packets to be over-

[13] There are some specialised devices known as TCP Offload Engines (TOEs) which are used in Gigabit and 10Gb ethernet network interface cards (NICs) but they themselves can cause problems due to the difficulties of dealing with updates. These are described in, among other places, a paper with the amusing title "TCP Offload is a dumb idea whose time has come"

written with new addresses if necessary. This Network Address Translation, or NAT, is now a routine fixture of the internet as IP version 4 did not allow enough space in the packets for the addresses, so addresses have to be shared. The NAT system routes all the packets from all computers through smaller number of addresses by modifying the IP address fields and using different port numbers – it's a complex bodge and doesn't work for all protocols hence the pressure to switch to IP version 6 which has larger addresses.

The second question is about congestion control, and it arose because in October 1986 the internet stopped working. Three nodes in relatively close proximity, the Lawrence Livermore Lab, Berkeley and Livermore Berkeley were connected using 32Kbits per second links. Suddenly the link speed dropped to 40 bits per second. At first, the operators they suspected there was a bug in their relatively new version of TCP which was running under Unix[14]. It turned out that there was a bug but not in the sense that we know it – the software was running as demanded, but TCP was getting its knickers in a twist.

TCP is an example of a "self-clocked" protocol. The transmitter sends a burst and waits for an ACK. Once it receives an ACK, then it can send another burst. But how long should those bursts be? If we restrict them to one packet, and we are on a satellite connection, then it might take a second or more for the ACK to arrive – one packet every second is ridiculously slow. What if we send a great big burst of data? Well, that jams up the network so it cannot be effectively time shared so, if we have low bandwidth networks either we deprive everyone else space or our jumbo packet is dropped which means we have to retransmit it which is inefficient.

The solution was define a new parameter, the congestion window, which controls the size of the packets. When starting a connection, the transmitter sets the window to one (and sends a single packet). On each ACK we increase the window by one. We keep increasing until we reach the receiver's advertised window size (in TCP the receiver sends back their buffer size as part of the acknowledgement protocol). If at any point we do not receive an ACK then it is a fair assumption that there is congestion, and we should reduce our sending quickly. Van Jacobson, the man who devised the first TCP congestion algorithm, recommended that the congestion window be halved. Halving might seem a bit dramatic, but assuming you were using 100% of the link, congestion implies one other user has appeared so the link bandwidth has to be shared 50:50. The general version of congestion control therefore has additive increases in data sent and multiplicative decreases in the event of congestions (AIMD congestion control as it is known). The effect of AIMD is to give a data transmission graph that looks like a sawtooth and people will sometimes refer to data flows as riding the TCP sawtooth. That in a nutshell is TCP Tahoe – the world's first packet congestion control algorithm.

If you have followed me so far then well done! Congestion control is not an easy topic and it is still a subject of debate and research – Wikipedia lists fifteen separate algorithms for TCP/IP congestion control – and different types of channel, WiFi, or satellite for example, demand different congestion control algorithms. Furthermore, the reasons for congestion are not at all well understood even by the cognoscenti. A reasonably recent annoyance is an effect called bufferbloat which particularly affects home broadband modems [3]. Let's imagine I am copying a large file from my work computer to home. My broadband company has provided a modem with a large buffer – it looks to me like everything is going fine – I am slinging packets at the buffer – "give me more," cries the modem "I can handle it". Meanwhile, upstream, there is some mild congestion. The buffer gets fuller but my computer does not know to slow down – the buffer is huge. Meanwhile I'm trying to connect to Gresham College to book my next seat. My ACK packets also go into the buffer but they take an age to be transmitted because they are backed-up behind the buffer, so the Gresham college website assumes I am on a super low bandwidth connection so it throttles back via congestion control and thus nothing happens. Some unscrupulous broadband providers rather like buffer bloat

[14] A version of Unix that we now call the Berkeley System Distribution or BSD.

as it allows them to sell you a more expensive faster connection whereas the problem was very poor latency caused by bufferbloat. These delays are now seriously out of hand – I measured round trip times that were sufficient to send a signal to the moon and back between my home and Gresham College which is under 150km away.

The solution is *tail-drop* – we should use smaller buffers that simply destroy packets they cannot accept. Destroyed packets trigger the congestion control algorithm to work and it throttles back to adapt to the bandwidth available

In their book "Algorithms to Live By" [4] by Brian Christian and Tom Griffiths draw some fascinating analogies with everyday life and the science of networking. They range over the justice system in Hawaii, the Peter principle, queuing for doughnuts and ants[15]. I'll confine myself to one of his observations which is that when one goes on holiday it is customary to write an "Out of office" message that implies that all the messages one receives will be queued up waiting for reading on return. Indeed, many organisations insist their employees set such messages. This is classic bufferbloat since it gives the senders the false impression that the vacationer has available to them infinite processing capacity when they return from holiday, so that they will indeed be able to process the queue. Tail drop would be more efficient and more humane for both the sender and receiver[16].

So far, I have pretty much managed to describe how the internet works without getting bogged down in any of the dull stuff. However, I feel I ought to give at least passing time to the idea of *layers*. I'm sure you noticed that TCP was built upon IP and, the world-wide web, is built upon TCP. These "built-upons" describe the layers of a network. In the old days, people ran multiple network standards and there was a good living to be made translating one protocol, say Cambridge token ring, into, say DECnet or Appletalk (to arbitrability pick three defunct standards). To make these translations easier the Open Standards Institute defined seven layers of a network. I fear life is too short to describe them all and in RFC 3439[17] there is an argument against layering which is that the implication that each layer can be separately optimised is a fantasy. Nevertheless, for those of you who feel they must know it, here is the OSI seven-layer model

Table 1: Examples of protocols in the ISO seven-layer model

Layer		Examples
7	Application	web (http; https)
6	Presentation	Mpeg; Mime
5	Session	RPC
4	Transport	TCP, UDP
3	Network	IP
2	Data link	PPP
1	Physical	10-base T Ethernet

I have left one question unanswered from earlier and that relates to security. IP included a field that allowed one to mark packets using the standard US government security markings at the time, restricted, secret, top secret and so on, but appeared unconcerned about any router along the route being able to see all the packets. This has proven to be a major headache and there have had to been various fixes over the years to deal with malicious computer users. I'll talk more about some of the issues in the next lecture but just focussing on internet security the obvious first defence is physical security so it is commonplace to find that key internet sites are highly protected and

[15] Ants also use congestion control and apparently one can aspects of the TCP sawtooth in their behaviour.
[16] There are other good reasons why email is incompatible with a scholarly life and these are explained on Donald Knuth's webpage (https://www-cs-faculty.stanford.edu/~knuth/email.html).
[17] Any particular RFC can always be found at *https://www.ietf.org/rfc/rfcNNNN.txt where NNNN is the RFC number*

replicated. There are, for example, twelve root domain servers operated by twelve independent entities, these are replicated in a total of 1379 entities so, in that case, the chance of failure is low.

When it comes to software security solutions, there are two basic themes. The first is to encrypt the packets so that casual interlopers, otherwise known as the "man in the middle attack," cannot read the data part. This is done in https for example. I hope this works well as it is the method used by our banks[18]! The second, used in conjunction with encryption, is to control the routing. Those of you in employment may have the misfortune to be required to connect to your employer's website by a system known as a Virtual Private Network or VPN. When you start-up your VPN, you connect with a VPN server at your organisation. From then on, all of the packets that would have left from your machine are collected, encrypted and embedded into either UDP or TCP packets (you usually have a choice of which protocol to use). At your employer's site these packets are decrypted and slung out onto the network. The good thing about a VPN is that all packets are encrypted and all of the packet is encrypted so interlopers cannot see with whom you are talking. The bad thing is that all that encryption can be very expensive – you will notice your computer runs hot when the VPN is working and of course the VPN at your employer's site is a now a single point of failure for an attack or, more likely, equipment failure[19]. However, for our discussion the interesting point about a VPN is that it protects your home or mobile internet address since now your packets appear to come from your work address.

This leads us to an important vulnerability of the internet which is that the headers are always unencrypted so, even if we cannot know what is being said, we know who is talking to whom. It is surprising how much information is given away by *traffic analysis*. If there is burst of information from me to my employer and, almost immediately afterwards, we see a burst of traffic from my employer to IKEA, then it's a fair bet that I am ordering some furniture.

In the mid-1990s, the US Office for Naval Research (ONR) funded work to reduce the effectiveness of traffic analysis attacks and the outcome was The Onion Router (Tor). Tor uses yet another internet protocol called SOCKS which allows for another machine to act as a *proxy server*. You tell your machine the name of the proxy, and then your machine wraps your packets with headers destined for the proxy. The proxy rewrites the headers of your packets so they appear to come from the proxy, when they are returned to the proxy they are wrapped and sent back to you, where they are unwrapped and processed as normal. Proxies are quite common in secure networks as it means all external, for example, web traffic can be routed through one machine which is easier to secure than securing everyone. Tor creates labyrinths of circuits between networks of Tor proxy routers. There are quite complicated rules for choosing the route to avoid the possibility that multiple nodes on the route are compromised. However, Tor can still operate securely even if some of the nodes are owned by malicious agents.

Tor encrypts each packet using an "onion" of encryption for each hop in the labyrinth. So, if there are four hops in the network then your computer obtains encryption keys from the four routers. It firstly encrypts with the fourth routers key, then the third etc etc. So, when the packet reaches the first router it strips off its own encryption to reveal the address of the second router to which it sends the packet. The second router knows where packet came from and, once it decrypts, it knows where to send it, but it does not know the address of the originator, nor the destination, nor does it know the contents. Thus, routers within the tor network know very little so, even if they are compromised, there is not much they can other than some traffic analysis. The entry and exit nodes or a Tor network are a potential insecurity so great care is used when choosing those.

[18] Actually the https protocol does three things: it encrypts the packets; it allows website owners to avoid people impersonating their website (provided they pay a certification authority) and it also provides a method to prevent packets being inserted or tampered with during transmission.
[19] You can cook sausages on some VPNs.

This has brought us to end of our circuitous tour around networks. In the abstract I promised to talk about the future of networks so I ought to spend a brief interval talking about three themes which are touted as significant for the future: wireless; IoT and security/privacy. Each one of those deserves a lecture in its own right. I regret having to leave out wireless networking in this talk, other than ALOHANET. Wireless brings its own problems – in TCP for example the assumption that if packets are lost it is due to congestion. That may not be the case in wireless, which can lead to poor performance and when packet switching is combined with cellular telephony even more tweaking can be needed[20]. Nevertheless, people hate wires and there is constant consumer pressure to move communication to radio even though radio is inherently less secure and polluting[21]. As we have seen, the protocols associated with the internet are complicated and generally are unsuited to simple computers. This has led to a plethora of lightweight networking protocols that are used principally for devices in the Internet of Things or IoT. You can find our more about the Internet of Things in another Gresham lecture by Martin Thomas [5]. Without being too glib I think I can summarise his lecture by saying that IoT will be a nice idea once it works. However, the final area for innovation and indeed philosophical thought is better (or worse) security and/or privacy. Security and privacy are not the same thing nor is it immediately obvious that more of both is a good thing. These are both topics for the next lecture.

Bibliography

[1] N. Abramson, "The Development of ALOHANET," *IEEE Transactions on Information Theory,* Vols. IT-31, no. 2, pp. 119--123, 1985.

[2] R. W. Hamming, "Error Detecting and Error Correcting Codes," *The Bell System Technical Journal,* vol. XXIX, no. 2, pp. 147--160, 1950.

[3] J. Gettys, "Bufferbloat.net," [Online]. Available: https://www.bufferbloat.net/projects/bloat/wiki/.

[4] B. Christian and T. Griffiths, Algorithms to Live By: The Computer Science of Human Decisions, William Collins, 2017.

[5] M. Thomas, "The Internet of Things," Gresham College, 20 March 2018. [Online]. Available: https://www.gresham.ac.uk/lectures-and-events/the-internet-of-things. [Accessed 12 April 2021].

[6] G. Ferry, A Computer Called LEO: Lyons tea Shops and the world's first office computer, London: Fourth Estate, 2003.

[7] C. E. Shannon, "A Mathematical Theory of Communication," *The Bell System Technical Journal,* vol. XXVII, no. 3, pp. 379--423, 1948.

[20] There was a brief period during the pandemic when people were attacking 5G phone masts. I did wonder briefly if the attacks could from networking engineers who had very carefully optimised their TCP transmission parameters for ethernet and were irritated to find they were not optimal for a new kind of wireless link. Sadly, it turned out that the attacks were not led by network engineers but by luddites who disliked new technology for irrational reasons.

[21] By polluting I mean polluting of the electromagnetic spectrum. People worry about light pollution which disturbs the sleep patterns of humans and animals, but general electromagnetic pollution is also problematic as anyone who has tried to install a wifi system in a large block of flats will be aware.