Digital communications

Bryan Ruddy

23 May 2022

We encounter data transfer in many contexts beyond looking at the internals of a computer. The Internet is at the heart of nearly everything we do today: talking to friends, going shopping, and learning about the world. It is also at the heart of many technical tasks, ranging from weather forecasting to health monitoring, and even running factories! But, what is the Internet, really? How and why does it work? Our goal in this module is to give you a high-level overview, so that you understand the basics of how to write code that works with networks and are prepared for future study of the topic.

After completing this module, you should be able to discuss the layers in the five-layer networking model, identify the most common implementations for each level, and discuss how they work. You should also be able to describe the major issues that need to be considered when choosing these implementations, and to justify choices of networking approaches.

The basics of communication

In order to communicate, there are a few basic things we always need to have in place. First, we need to have some sort of *medium* over which we communicate. When we speak, for instance, this medium is sound waves in the air. When computers communicate, this is most often an electrical signal in a wire, though many other media are commonly used. We also need a *target* for our communication, that shares that medium with us. When we speak, our target is everyone close enough to perceive those sound waves. If a computer uses a wire connected only to it and to another computer, only that computer can be the direct target of that communication.

Most importantly, we need a *protocol*, a shared way to interpret the signals sent using that medium. The protocol could be very basic; when someone screams, we interpret that pattern of sound waves as a request for help. This protocol is nearly universal among humans. It can be more sophisticated; if I speak English, other listeners who know English can interpret the pattern of sound waves I produce as abstract concepts. If both parties communicating don't share the same protocol, no communication can happen; if I speak Esperanto, but the person I'm talking to expects to hear English, they will not be able to interpret the sound waves I produce. Protocols can also be layered. For instance, Alice could shout out "Bob, the answer is two!" into a crowded room every minute until Bob shouts back "Roger that, Alice!". This is a protocol layered on top of English that gives Alice some assurance that Bob has received their message.

A good communications protocol can also let us reach targets that don't share a medium with us. In human communication, this could be as organized and controlled as sending a messenger on an airplane to repeat your message to someone on the other side of the world, or as disorganized as a rumour mill. If careful measures aren't taken, and especially if there are many links in the chain between sender and receiver, this can easily turn into a game of broken telephone, where the message delivered is different from the message sent.

Finally, we can characterize a communications system in terms of its *bandwidth*, the rate at which we can communicate. In human communication we might describe this in terms of words per minute; in computers, we usually talk about bits per second.

Data security

As you have most likely experienced many times in your life, communication is not perfect. We are always shouting into a sea of *noise*, random fluctuations in the communications medium that can block or alter the receipt of legitimate messages. Whether we receive or messages successfully depends on the ratio between the strength of our signal (how loudly we shout) and the strength of the noise, the *signal-to-noise* ratio. When the signal-to-noise ratio is high, like whispering in a library or shouting in a restaurant, it is easy to receive the message as intended. However, if it is low, like when whispering at a restaurant, or shouting next to a jet engine, it can become very difficult to receive anything at all.

In engineered communications systems, we have a number of tools that can be used to ensure successful communication at almost any signal-to-noise ratio. In fact, signals like those from GPS satellites might be better described as having noise-to-signal ratios! There are two main forms these tools can take. In the first, we use special encodings for our data that allow for errors in individual bits to be corrected. In the second, we calculate special values from our data that can be transmitted along with it and compared to see if there have been any errors. We will briefly also discuss cryptography, in the context of tools that can be used to protect data against intentional interference.

Error correction

Error-correcting codes are used to trade off the speed of data transmission against the likelihood of errors, by spreading out each bit of data in the message across multiple bits of data transmitted across the medium. If one of the bits being transmitted is changed, then the code still allows the correct bit of the message to be determined. We typically describe an error-correcting code in terms of the number of bits transmitted, n, for a block of m bits in the message, as a (n, m) code.

A simple example of an error-correcting code is the repetition code, where each bit being transmitted is simply sent multiple times. The (3, 1) repetition code is the shortest one capable of correcting errors, sending "000" for a zero bit in the message and "111" for a one bit in the message. Any one bit of the three transmitted can be changed without affecting the message, as shown in the chart to the right. Only if two of the bits sent are changed is the message misinterpreted.

Triplet received	Interpreted as
000	0 (error-free)
001	0
010	0
100	0
111	1 (error-free)
110	1
101	1
011	1

A (3,1) repetition code.[1]

Compared to other, more complex, error correcting codes, the repetition code performs very poorly. It reduces the effective bandwidth for communications to one third of the capacity, but gives relatively little information to overcome errors. Information theory experts have developed more sophisticated codes that can recover data under harsh conditions with less impact on effective data rate. For example, Reed-Solomon coding is used on CDs, DVDs, and barcodes (including QR codes), while convolutional codes are used for Wi-Fi and turbo codes are used for mobile phones. These codes are too complex to describe here, but if chosen carefully can allow for reliable error correction under even extreme noise conditions.

Checksums

In many cases, when errors are rare, it is sufficient to simply determine whether data being transmitted have been corrupted, without correcting the error. We can then request re-transmission of the corrupted data. we do this by calculating a special, additional piece of data, called a *checksum*, and sending it along with the data we wish to protect. The simplest form of such a check is a *parity bit*: one extra bit is stored, calculated such that the number of "one" bits transmitted in a block is always even. This is even parity; one can also have odd parity where the number of ones is always odd. Parity bits can detect single-bit errors, but fail if more than one bit is flipped.

Parity is why the ASCII encoding only uses seven bits; the eigth bit per byte was used as a parity bit.

More common and more robust is a mechanism called a *cyclic redundancy check*, or CRC. This feeds a block of data through a mathematical function that transforms it into a short code, and does so in such a way that the relationship between the data and the check code is nearly random. A single-bit change to the data creates a massive change in the CRC code, and it is statistically extremely unlikely that a multi-bit error could result in the same CRC value. CRC values are easy to compute using electronics hardware, and thus are very common in communications protocols.

Cryptography

Error correction and redundancy checks are effective at protecting data against random noise, but what if someone was specifically trying to alter your data? How could you tell if it was tampered with? Alternatively, how can you make sure that only your intended recipient can read your data? Cryptography provides tools

that can address both these problems.

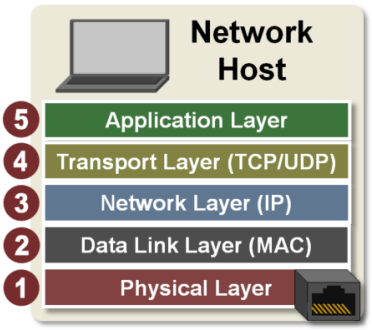
To protect messages against intentional interference, you can use a cryptographic hash function, which works similarly to a CRC. However, unlike a CRC, a cryptographic hash function is designed to work only one way, such that the only way to discover what messages could generate a particular hash is to brute-force check all possible messages until you find a match. On their own, these cryptographic hashes can be used to verify the integrity of data if you have a separate trusted source for the hash. Combined with other techniques, cryptographic hashes can be used to create and verify digital signatures.

This brute-force check is a similar problem to the calculation underlying Bitcoin and other cryptocurrencies.

If you also wish to prevent others from reading your data, you need to use a *cipher* to encrypt it. Both you and your target need to agree on what cipher to use and exchange *keys* for that cipher, so that each can decrypt what the other has sent. A wide variety of cipher algorithms and key exchange algorithms have been developed, and their security is often dependent on subtle details. In general, you should not attempt to implement your own cryptographic functions, but should follow current industry best practices to select one from an external software library.

The networking model

It takes a lot of work to get from a raw means of sending signals to a full-featured and useful communications system. Just as you have experienced when writing other computer software, it is useful to employ the concept of *abstraction* to this problem: build on partial solutions from others, without worrying about the exact details of them. While we might wish to abstract away the entire problem of communications, and let it become someone else's problem, in practice we may wish to do some of the work ourselves, and even if we don't there may be options we need to choose between due to different features and performance. To help with this process, engineers have defined a variety of layering models to describe the different parts of communication that can be abstracted.



The five-layer networking model.[2]

For this course, we will adopt a five-layer networking model, following the model used by the maintainers of the main standards underlying the Internet but augmenting it with a layer describing the hardware used. This lowest layer, the *physical layer*, is the only one concerned with hardware, and provides the layer above it with a way to send bits of data from one computer to another. The next layer, the *link layer*, adds a logical structure to the sending of bits, so that we can send a discrete unit of data from one computer to another and have both agree on how to decipher it. Above this is the *network layer*, which describes how data can be sent using the structure defined in the link layer to reach any computer in the world, not just a directly-connected computer. Once we can send data to anyone, the *transport layer* describes how we can send data using the structure defined by the network layer so that it is

robust to any disruptions or irregularities on the network, forming a true connection between two computers. Finally, the *application layer* describes how we do useful things with the data we can now reliably transmit and receive across the connection defined by the transport layer protocol.

The physical layer

The physical layer is concerned with the medium over which data is transferred, and with the protocol relating the state of that physical medium to binary ones and zeros. In computing, there are three main categories of medium: electrical currents in wires, radio waves traveling through free space, and light waves traveling through optical fibres. We will discuss these communications media here, along with the main characteristics of the communications protocols using them.

Wires

Wires form the most basic and most direct communications medium between computers. Wires can be designed to connect pairs of computers directly, or can be used to connect many computers together on a single connection. They can be designed to carry the same kinds of signals used with the data buses inside computers, or can use different protocols specifically designed for long-distance communication.

An ethernet cable.[3]

Wires do have some disadvantages, however. Compared to other media, wires tend to be physically large and heavy. More importantly, the physics of electrical currents limit the rate at which we can change them, and in longer wires the changes must be slower. This means that the bandwidth is limted, especially over longer distances. (Repeaters can be used to maintain bandwidth over distance, but at a cost.) Long wires also tend to pick up noise from the environment, which introduces errors and further limits the achievable data rate.

Optical fibres

Fibre optics are currently the most common and highest-performance medium for communication. Fibre optic cables consist of a transparent core, typically made from glass though plastic can sometimes be used, surrounded by a transparent cladding made from a different material that directs light from the core back to it. The cladding is then covered by a protective sheath, to protect against mechanical damage. Despite this, optical fibres remain very fragile, and sharp bends can damage them. They are also very difficult to join together, requiring nanometer-level precision and even then losing some signal at each joint. Despite their fragility, and the cost of electronics to convert between optical and electrical signals at

each end of the fibre, there are key advantages that drive the use of fibre optics.

The most important advantage of fibre optics is that the laws of physics place few limits on the bandwidth available. Bandwidth is limited by the performance of the electronics at each end, which are easily replaced to upgrade the connection. Optical fibres can also span very long distances with minimal losses to signal-to-noise ratio: tens of kilometers in basic applications, and thousands of kilometers (across oceans) with built-in amplifiers. The undersea optical fibres linking continents typically have bandwidths measured in terabits per second!

Radio waves

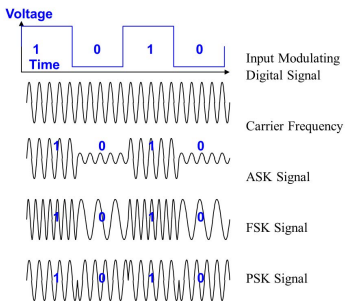
Radio waves are the most versatile communications medium for computers, because they require no fixed infrastructure at all along the link, only at either end. If the right wavelengths of radio waves are chosen, they can easily pass through walls and other obstructions. This flexibility and mobility is a powerful driver for technology development of radio communications.

However, radio waves are an extremely challenging medium to work with. Because the medium is simply free space, the signal strength decays rapidly with distance, following the inverse square law. There is also little separation from others transmitting or receiving radio waves; everyone else's broadcasts look like noise to you, and anyone can receive the messages you send. In addition, the physics of radio waves mean that, to achieve high bandwidths, you have to use wavelengths that are more strongly blocked by walls and other obstacles, and in some cases are even partially blocked by moisture in the air.

Modulation

There are three main ways that we can actually send information using these media. The first is *amplitude modulation*, whereby the strength of the signal we send changes with time. An extreme version of this is called on-off keying, where the signal is turned completely on or off. The second is *frequency modulation*, where we use a signal that varies at high frequency (or is a wave with an inherently high frequency), and change that frequency over time. The last is *phase modulation*, where we shift the timing of the waveform we send to encode information.

These methods can be used independently or in combination, and can be binary (as shown to the right) or can use more levels to encode more information simultaneously. (The electronics at each end of the connection convert this to and from binary.) Current high-performance communication technologies typically combine amplitude modulation and phase modulation, or use a sequence of two different amplitude values. In this way, up to 15 bits can be simultaneously encoded (i.e. 32,768 distinct signal levels), where



Modulation schemes.[4]

the signal-to-noise ratio is high enough.

Sharing

Some communications media are inherently shared, like with radio waves, but there are also benefits from sharing others across multiple connections. This can be done in a number of ways, some possible across all media and some specific to one. The basic ways follow from the modulation schemes: media can be divided up into slices of time or slices of frequency, with each allocated to a different user. These allocations need not be constant—a method called frequency hopping is commonly used, where each user is allocated a unique sequence of frequencies to use over time. This more fairly shares the burdens of noise and interference, at the cost of complexity. There is a third common method, called *code division*, where each user is given a different modulation pattern to use, so their signal can be distinguished from others based on its content. Code devision is more intuitively difficult to understand, but forms a key part of modern radio communications.

The actress Hedy Lamarr invented one such frequency division scheme during World War II, though similar approaches were not adopted until about 20 years later.

In optical fibres, we can also share the fibre by using different wavelengths of light, as distinct from frequency modulation. (Technology does not yet exist to perform frequency or phase modulation on the scale of the light waves themselves, but only on a signal produced through amplitude modulation.) This method is used for fibre broadband in New Zealand, where each home in a neighborhood can be allocated a different wavelength of light and share a single fibre back to the central office.

With radio waves, we can also share to some extent by using antennas that broadcast only in one specific direction, forming a beam. There are ways to do this using multiple ordinary-looking antennas, which is why many new Wi-Fi access points have a large number of antennas. 5G cellphones use similar methods to allow more users to share a single tower or antenna in a city.

The data link layer

The physical later allows raw data bits to be sent and received, but gives no structure to the process. In order to direct our communication to a specific target, and be assured they can receive and decipher those bits, we need a shared protocol for use with the physical medium. This is the data link layer, often just called the link layer, and there are a wide variety of different such protocols in use. Here, we will focus on a single such common protocol, Ethernet, and then briefly discuss examples of other common link-layer protocols.

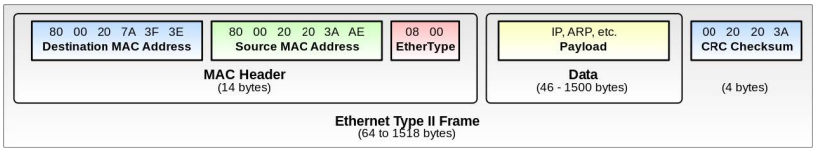
Ethernet

Ethernet is a protocol used to connect computers that share a wire or a fibre optic cable as their physical medium, and the official

standard also includes the many physical-layer protocols it uses. It originally was developed to work with connections where many computers shared a single wire, and thus where there could be *collisions* if two computers attempted to communicate simultaneously, and where computers other than the intended recipient would see each message. It therefore has several features that were intended to alleviate these limitations: each computer has a unique address, and each data transmission is accompanied by a CRC checksum to help detect data corruption caused by collisions.

To ensure each computer only receives the messages attended for it, the Ethernet standard calls for every computer to be assigned a unique 48-bit address, called a Media Access Control (MAC) address. These are assigned by the device manufacturers, and with 280 trillion possible addresses every device should have a unique address. Every message on Ethernet comes with both the MAC address of the sender and the MAC address of the intended recipient, so that computers can ignore messages intended for others and know how to reply to the messages intended for them. If two devices on an Ethernet network have the same MAC address, it is likely that the entire network will malfunction until one of the offending devices is removed.

You can often find the MAC address written on a label or sticker on the computer hardware, as 12 hexadecimal digits.



The structure of a modern Ethernet frame.[5]

Data transferred using Ethernet is organized as *frames*, as shown above. Before an Ethernet frame, the physical layer transmits a special eight-byte preamble, which signals to other devices that a frame is about to be transmitted. Next, the frame includes the destination and source MAC addresses, followed by a piece of data called the EtherType that signals what kind of data are being sent. (There are different EtherTypes defined for the various network-layer protocols that work on top of Ethernet.) Following this is the actual data, typically up to 1500 bytes, with a four-byte CRC checksum at the end that verifies the integrity of the rest of the frame. After each frame, the physical layer is required to send at least 12 bytes of zero values, to ensure enough separation between frames.

These protocol features were designed for a system where many computers shared the same wire. However, collisions still caused problems, and modern Ethernet networks include special devices called *switches*, which are directly connected to each computer on the network by a separate wire. The switch reads the destination address in the Ethernet frame, and ensures it is only sent to the computer that should receive it. Because there are never more than two devices sharing a wire, there are no collisions, and data can be transferred quickly and reliably.

Other protocols

Ethernet is a great protocol for organizing data transfer between many computers on a physical network, but we don't always want to communicate within such an arrangement. For instance, when your home computer network connects to your service provider over a phone line or a fibre optic cable, you need to use a protocol that protects the privacy of your connection from other customers of your service provider, and that allows your service provider to turn off your service if you haven't paid! The most common protocol for this is called the *point-to-point* protocol.

Ethernet is not designed to operate on the physical medium of radio waves, where there can be much more interference, many more collisions, and where there is a need to maintain data privacy. The most common family of link level protocols is the set managed by the Wi-Fi Alliance, also known by the overall standard number (IEEE 802.11) or just as "Wi-Fi". Wi-Fi data frames carry additional information, including extra MAC addresses (which are used in mesh networking), data used to help battery-powered devices save energy, and information used to manage data encryption. The Wi-Fi standards also define a set of management and control frames that are needed to connect to access points, set up encryption, and ensure reliable receipt of transmitted data.

The Wi-Fi standards also cover the physical-layer protocols used.

The network layer

Link-layer protocols are suitable for transferring data directly between connected computers, but how do we transfer data when the connection is less direct? To reach long distances, we will often need to pass across many different networks, which may be using different link-level protocols. Your phone may be on Wi-Fi, the wireless access point uses Ethernet to connect to the University network, the University is connected to America using a more exotic protocol to cross a trans-Pacific cable, and your American friend's phone is connected to a backbone network using a 4G cellular phone protocol. Solving this problem is what the network layer is all about.

The most important network layer protocol is also the source of the name "Internet" - the *Internet protocol*, short for "internetworking", and often abbreviated as IP. The Internet protocol defines globally-visible addresses that can be used to identify computers, describes ways that data can be *routed* across networks, and provides a standard data format that is flexible enough to solve problems with different networks having different speeds and different frame sizes.

Addressing

While MAC addresses are meant to be globally unique, they also cannot be changed without significant collision risk, making it

impossible to organize devices using such addresses. Instead, we define *IP addresses*, which still must be globally unique but can now be set by network operators and allocated from a global (or local) pool of addresses. In the first version of IP to achieve widespread use, IP version 4 or IPv4, the addresses only had 32 bits. We typically write these addresses using decimal numbers, like 192.168.0.1; the first number indicates the owner or operator of the network, while the others give steadily greater detail on where the computer is.

Four billion addresses might have been enough in the early days of computing, but with eight billion people on this planet it's not enough for an era of ubiquitious computing. We are thus slowly adopting IP version 6 (IPv6), which has 128-bit addresses. In an IPv6 address, we use the first 64 bits to describe the network the device is on, and the second 64 bits as a unique ID for the device itself. We typically write an IPv6 address using eight hexadecimal numbers, like 2001:0db8:0000:0000:0000:ff00:0042:8329. (We can also shorten this by leaving out the leading zeros and omitting blocks of all zeros, so that our example would be 2001:db8::ff00:42:8329 - still long, but a bit more human-readable!)

This is enough addresses so that every grain of sand on Earth could have one unique address for every other grain of sand, or enough for every molecule in a cubic kilometer of water to have ten addresses to itself!

Each IP version sets aside some addresses for special purposes. For privacy and security reasons, we may want a private network to use addresses that cannot be reached from outside that network. Your home network almost certainly uses one of these, most commonly an IPv4 address starting in 192.168, though addresses ranging from 172.16.0.0 to 172.31.255.255 and those starting with 10 are also private. There are also private IPv6 addresses, starting with the two hex digits fd. IP addresses starting with 169.254 (IPv4) or fe8 (IPv6) are reserved for use when an address is needed that isonly used on a direct connection between two computers.

Some addresses are used when you want anyone to be able to receive a transmission. These addresses, called *multicast*, are in the range from 224.0.0.0 to 239.255.255.255 (IPv4) or start with ff (IPv6). Typically these are used for special purposes, not for general broadcasts like television.

Finally, there are special addresses reserved for use when you need a computer to talk to itself, with no network connection at all. These are called *loopback* addresses, and are 127.0.0.1 (IPv4) and ::1 (IPv6). Loopback addresses are guaranteed never to be used on any network, and are generally used for testing and software development.

You have used the loopback IP address as part of running the micro:bit simulator in the lab.

Routing

How do you reach someone at a particular IP address, if they aren't on your own network? This is the problem of routing, and it is solved using special computers (called routers) that connect different networks together. For example, if you have home internet, you have a router that is connected to your internet service provider

and to the computers in your home. Your internet service provider has more routers, and ultimately they connect to routers run by companies operating network backbones and trans-oceanic cables.

Your home router often also connects an Ethernet network in your home and a Wi-Fi network.

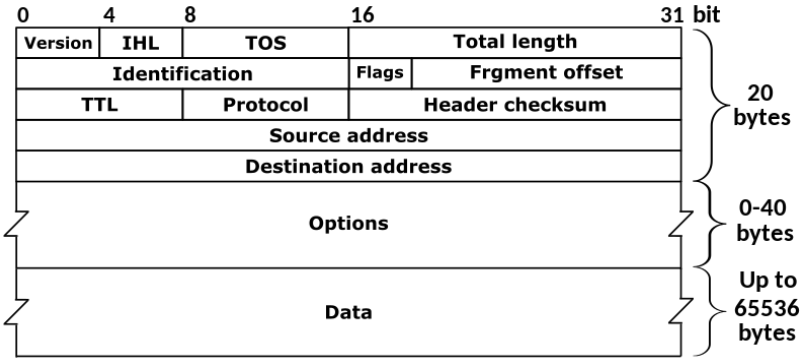
Each router maintains a *routing table*, containing the different IP addresses it can directly reach as well as information about which other addresses can be reached using them. This typically includes some measure of the quality of the connection that can be achieved. Based on the information in the routing table, a router can generate a forwarding table that defines where it should send data for every different destination it might try to reach. For home routers, the routing table is very simple, sending traffic for most addresses to the internet service provider and sending traffic within the local network directly to the correct address. However, routing is a much more complex problem on internet backbones, where it must be done very quickly and without storing detailed information for every possible individual IP address. Routers build their routing tables using information about the network provided by application-level protocols; this does not happen directly within the Internet protocol. The *options* field allows for various debugging tools and experiments, but is rarely used for most traffic.

The problem of routing might look very similiar to your network lab earlier in the semester; Dijkstra's algorithm is one possible approach to solving it.

If the link layer works in therms of MAC addresses, how does the router know which MAC address should be used to reach a particular IP address? This is handled by a different network-layer protocol, called the address resolution protocol for IPv4 and the neighbor discovery protocol for IPv6. These can only work effectively at the local network, because they simply ask all connected computers to identify themselves if they belong to a specific IP address.

Data format

In order to facilitate the movement of data across networks, we need to include some additional information alongside that data. The total set of information transferred at once with IP is called a *packet*, and its structure (for IPv4) is shown below. (IPv6 has a somewhat more complex structure.)



The structure of an IPv4 packet.[6]

Within an IPv4 packet, the *identification*, *flags*, and *fragment offset*

fields are used to allow packets to be split up as they travel if one of the network links requires a smaller packet size; when received at the destination, packets with matching IDs can be reassembled in an order given by the fragment offsets. (The flags indicate whether or not this fragmentation is allowed.) The *TTL* field gives the maximum number of network hops the packet is allowed to travel before the protocol gives up; sometimes a packet can get stuck in a loop if there is a routing problem, and this keeps the network from filling with stuck packets. The *TOS* field is used to prioritize some kinds of data over others, like streaming audio for phone calls. The *protocol* field is used to indicate what kind of transport-layer protocol is in use.

An IPv4 packet uses a checksum to protect the header data, but does nothing to protect the integrity of the data. This is because any corruption of the header could cause routing errors that congest the entire network, while data corruption at worst requires the sender to re-transmit. (Or actually corrupts the data, but that's not the network layer's problem!)

The transport layer

The network layer can get data from one side of the planet to the other, but how do we actually affirm that the data will be useful. We need to establish a connection, so that our recipient knows what to expect, and how to combine a stream of packets into useful data. There are two main protocols used at the transport layer: the *user datagram protocol (UDP)*, used when speed is of the essence, and the *transmission control protocol (TCP)*, used when every byte of data is important.

User datagram protocol (UDP)

UDP adds only a small amount of information to the data being transmitted: source and destination *ports*, to indicate what kind of application-layer protocol is being used, and a length and checksum to maintain data integrity. However, if an entire packet of data transmitted via UDP is lost or missing, or if the data fails verification against the checksum, the data re simply lost, with no means of replacement.

You have used port 8000 with the micro:bit simulator.

UDP datagram header

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Length																Checksum															

The structure of a UDP header.[7]

Because UDP allows data loss, it can only be used where data loss is acceptable, such as in streaming audio or video. Its simplicity also makes it suitable as a substrate for new development of transport-layer protocols and application-layer protocols, which can implement their own techniques for mitigating data loss.

Transmission control protocol (TCP)

TCP is a more complex protocol, which takes specific steps to establish a connection between two computers, the *client* and the *server*. First, the client sends a connection request packet to the server; the server responds with a connection request to the client, and the client must reply with an acknowledgement. This ensures both computers are communicating successfully, and allows each to set a *sequence number* that can be used to detect future missing packets. Once the connection is established, each packet one side sends to the other is responded to with an acknowledgement, and the data sent includes information that can be used to slow down transmission if the recipient cannot handle the rate at which data are ariving. Finally, to close a connection, each side must send the other a termination packet, and must ackowledge the termination sent by the other.

TCP segment header

Offsets	Octet	0								1								2								3							
Octet	Bit	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																								Urgent pointer (if URG set)							
20	160	Options (if *data offset* > 5. Padded at the end with "0" bits if necessary.)																															
⋮	⋮																																
60	480																																

The structure of a TCP header.[8]

TCP packets include additional information to assist with all this back-and-forth. Each packet carries a sequence number, which is incremented every time a packet is sent and can be used to ensure that packets received out-of-order can still be reassembled correctly. When acknowledging a packet's receipt, the sequence number is sent with the acknowledgement, to help detect missing packets. A number of single-bit flags are used to determine the packet type, whether it is part of establishing or ending a connection, or just contains ordinary data. The *window size* is used to indicate the amount of data that the sender can handle receiving, and is used to slow down transmission when needed. A checksum is also used to ensure integrity of the data and the header.

Throughout a TCP session, various timers are used to guard against the possibility of packet loss. Any unacknowledged packets are re-sent after a delay, but every time this happens the delay increases to avoid flooding the network if one party becomes disconnected. Similarly, if one party uses the window size to ask the other to stop sending data, the other will try sending again after a wait even if it never told to re-start. Measures are also taken to prevent overall congestion of the network.

Combined, these features of TCP guarantee that data will be received accurately under nearly all circumstances. However, there are some limitations. TCP can become very slow when the network

is congested, and all the acknowledgements flying back and forth only act to further increase congestion. In addition, malicious actors can break connections and shut down the network by flooding it with unackowledged transmissions. Network engineers have had to go to great lengths to maintain network performance in the face of these limitations.

One of the main ways that the Great Firewall of China prevents unauthorized internet traffic is by sending fake end-of-connection packets.

The application layer

With all of the lower-level abstractions in place, we are now ready to actually send useful data. Different applications, of course, use different data formats, and both parties in the communication need to agree on the data format for the data to be useful. To ease this process, many standard application-layer protocols have been defined. Perhaps the most widely used today is the *hyper-text transport protocol* (HTTP) and its secure variant (HTTPS), used to send web pages and extended to send other kinds of formatted data. However, many other protocols are also in common use, and we will give a brief overview.

Hyper-text transport protocol (HTTP)

HTTP provides a simple set of text-based commands that can be used by a client to interact with data on a server. The client sends a *request*, and the server sends a *response* that might include resources from the server or information about the status of the request. To work properly, this protocol must use a transport-layer protocol like TCP that can guarantee reliable communication. The basic HTTP requests include the following:

- GET—requests that the server transfer some resource to the client. (i.e. retrieve a web page)
- HEAD—requests that the server send some information about a resource to the client. (i.e. find out how large an image is)
- POST—requests that a resource on the server process some information sent by the client. (i.e. post a question on Piazza)
- PUT—requests that the server add a resource sent by the client (i.e. upload a new web page)

Each request is accompanied by a *uniform resource locator (URL)*, which indicates what resource on the server is of interest to the client, the version of the HTTP protocol being used, and a hostname, indicating which computer should supply that resource. Requests can also include other information as headers, and may include a message body following the headers, a block of data. Some of these requests make no change to the server, like GET and HEAD, while others may cause the server to change some of its data in response.

The server responds to a request with a numerical status code. One common status code you may be familiar with is 404, which means the requested resource cannot be found. It also includes response headers and may include a message body (e.g. the web page requested) following those headers. All of the header data must be represented using printable ASCII charaters, though the message body can be any binary data. The message body is very flexible, and it is commonly used to send audio and video data in addition to just web pages.

One common encoding for header and URL data is Base64, which stores 6 bits per character by using uppercase and lowercase letters, numerals, and a small number of punctuation characters.

HTTPS adds encryption to HTTP, so that no one can eavesdrop on the request or response content, and so that the data recipient can be assured that it has not been tampered with between the client and the server. With the rise of online shopping, electronic bank transactions, and other high-risk forms of data transfer over HTTP, the use of encryption has become essential. Most application software now uses HTTPS by default, rather than HTTP.

Other common protocols

The internet isn't all about web browsing! Common application-layer protocols have been defined for email (SMTP, IMAP, POP), file transfer (FTP, BitTorrent), telephones (SIP), control of machinery (ModBus, MQTT), and even transfer of health data (HL7). Any standard protocol will have readily-available documentation to help you implement it if needed, and usually will also already have software libraries written in your preferred programming language.

Some application-layer protocols work behind the scenes to keep the other parts of the Internet working. For instance, the mapping between human-readable and memorable hostnames, as required by HTTP (e.g. auckland.ac.nz), and IP addresses is handled by the *domain name system (DNS)*. When you connect your phone to a new Wi-Fi network, it is assigned an IP address using the *dynamic host configuration protocol (DHCP)*. Routers populate their routing tables using the *border gateway protocol (BGP)*.

Across all of these protocols, the main thing to keep in mind is that the best practice is to try to use an existing standard if at all possible, because this helps minimise work (through code reuse) and minimise the risk of errors and mistakes. Amateur implementation of encryption is almost guaranteed to fail, for instance. Where there exists no standard application-layer protocol that will do your job, try building one on top of an existing protocol, just as YouTube transfers all its streaming video using special message bodies in HTTP.

Overall, by using these abstraction layers well, you can achieve amazing results and perform astonishing feats. Better still, you can do so while mostly using other people's code!

Figure sources:

[1]From https://en.wikipedia.org/wiki/Error_correction_code (CC BY-SA 3.0)
[2]By Microchip Technology, from https://microchipdeveloper.com/tcpip:tcp-ip-five-layer-model
[3]From https://commons.wikimedia.org/wiki/File:EthernetCableGreen.jpg (Public domain/CC0)
[4]From https://popularelectronics.technicacuriosa.com/2017/03/08/radio-frequency-modulation-made-easy/
[5]From https://commons.wikimedia.org/wiki/File:Ethernet_Type_II_Frame_format.svg (Public domain)
[6]From https://commons.wikimedia.org/wiki/File:IPv4_Packet-en.svg (CC BY-SA 4.0)
[7]From https://en.wikipedia.org/wiki/User_Datagram_Protocol (CC BY-SA 3.0)
[8]From https://en.wikipedia.org/wiki/Transmission_Control_Protocol (CC BY-SA 3.0)