Networking  
CM30078/CM50123

Russell Bradford

2023/24

### 1. Link Layer Protocols

We now turn to some other link layer protocols

*Serial Line Internet Protocol* (SLIP) is an early protocol used on modems to encapsulate IP traffic over serial (telephone) lines

It is a *point-to-point* protocol, meaning it links just two machines to each other: the normal requirement in early dial-up systems

### 2. SLIP

SLIP frame

SLIP frame

A very simple frame encapsulation with a terminating byte (hex) c0; also often a starting c0 byte, too

### 3. SLIP

So how to send data that contains the byte c0?

Use *byte stuffing*

To send c0 actually send two bytes db dc

The pair db dc is reconstructed as c0 at the other end

Stuff db as the pair db dd, which the other end reconstructs as db

A minor expansion of data, but it enables transparent transmission of data

### 4. SLIP

There is no frame size limit, but it was suggested you should support at least 1006 bytes

296 bytes was common (40 bytes of TCP/IP headers plus 256 bytes of data)

Larger frames have relatively less overhead, but at 9600b/s (a typical early modem speed) 1006 bytes takes roughly 1 sec to transmit

If there is a bulk transfer of full-sized frames at the same time as an interactive session, the interactive session’s frames would have to wait 0.5 sec on average to get through, much too slow

An interactive response of over 100-200ms is felt to be slow

### 5. SLIP

296 is a compromise: not too slow for interactive, not too small for bulk transfer, but not particularly good for either

On more modern modems (56Kb/s) it was increased to 1500 bytes

**Exercise** Compute the average latency for 296 byte frames on 9600b/s; and 1500 byte frames on 56Kb/s

**Exercise** And how big a frame could we have on a 10Mb/s Ethernet for the same latency?

### 6. SLIP

SLIP has several problems

Only IP in the next layer is supported (no type field in frame)

The ends must have pre-agreed IP addresses: no mechanism for agreeing addresses

No checksum: even though telephone lines were noisy and created data corruption

No authentication: no way to check *who* is connecting

### 7. PPP

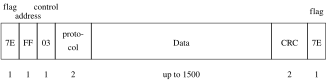
Thus the *Point-to-Point Protocol* (PPP) was developed

Like SLIP it is a point to point protocol

It has three parts

* A framing layout for packets
* A link control protocol (LCP) for managing and configuring links
* A set of network control protocols (NCP) to manage network layer specific options

### 8. PPP



PPP frame

* Frame delimiters 7E, start and end
* Address (always ff), Control (always 03)
* Protocol: tells us what the next layer is, e.g., IP is 0021, LCP is C021 and NCPs are 80xy
* Cyclic redundancy check to spot corruption
* But no address fields (**Exercise** why not?)

### 9. PPP

* Up to 1500 bytes of data (but can be negotiated)
* Values are escaped (like SLIP) by 7d
* 7e 7d 5e
* 7d 7d 5d
* , where 7d 20, so, e.g., 017d 21

### 10. PPP

NCPs can negotiate extras, like compression, frame size, etc.

And authentication, e.g., passwords

While it was devised to be used over telephone modems, PPP is still actively used, e.g., in PPP over Ethernet (PPPoE) as it allows authentication of a connection

Current FTTC products use (IP over) PPPoE over VDSL to pass authentication to the ISP

**Exercise** Read about this

**Exercise** Look at the configuration of your home ADSL or VDSL modem

### 11. Link Layers

Several other link layers exist: too many to talk about in any detail

We have already seen the Ethernet frame for a local area network

There are many link layers for carrying data over long distances, at high data rates, both electrical and optical

### 12. Link Layers

<+(0)-> For example, *Asynchronous Transfer Mode* (ATM) was popular for a while

Designed by telephone engineers, it was really a *connection oriented* digital voice network into which you could squeeze data packets

It has fixed-length frames of 53 bytes (48 data plus 5 header) — good for voice, not so good for data

**Exercise** Read about ATM and PPPoA, that layers (IP over) PPP over ATM, as used in ADSL and DOCSIS

### 13. Link Layers

*Multiprotocol Label Switching* (MPLS) was designed post-ATM when the technology decisions that drove the design of ATM were deemed no longer applicable

Designed by network engineers to be a general long-distance network, it is much better suited to modern data networks

**Exercise** Read about MPLS and how BT uses it in its 21C Network

### 14. Networks

#### Ethernet Link Layer

We want to move up to the network layer: but before doing so there is one more remark on the link layer

Recall Ethernet. The data on the wire:

Ethernet frame

Ethernet frame

### 15. Networks

#### Ethernet Link Layer

The interesting fields here are the *addresses*

The addresses allow a frame to identify the intended destination (and source)

This works well enough when the destination is on the local Ethernet network

Which is shared (or switched), so the frame has no problem being seen by the destination host

### 16. Networks

#### Ethernet Link Layer

What to do when the destination is non-local?

We can’t simply treat the world as a shared medium and broadcast the packet to everybody

And the network at the destination might not even be an Ethernet and will not have an Ethernet address

So we need *hardware independent addresses* to identify hosts that work independently of the physical network

In the Internet Protocol, these addresses live in the network layer

### 17. Networks Link Layer

#### IP

The network layer used in the Internet Protocol is called the *Internet Protocol* (IP)

It has the major function of dealing with *routing*, determining where a packet should go

Amongst a lot of other stuff, the IP header has network layer addresses

These are hardware independent, and in the same format across the entire Internet

### 18. Networks

#### IP

Each host on the Internet has an IP address that identifies it uniquely over the entire Internet

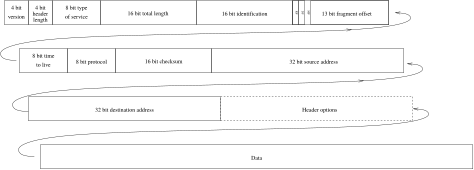
At least, that was the original intention

This is certainly no longer true, for reasons we shall explore later

But, for now, assume this is true

### 19. Networks

#### IP Header

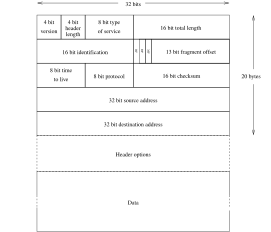


IP header

A bit hard to read, so conventionally we stack the header vertically

### 20. Networks

#### IP Header



IP header (usual layout)

### 21. Networks

#### IP

The source and destination addresses are both four bytes long: we shall come back to the other fields shortly

10001010001001100010000000001110 is an example IP address, a 32-bit value

This is 2317754382 in decimal: not terribly easy to work with

So for convenience we write this as 138.38.32.14, decimal representations of four 8-bit values. The dots are purely to make the number visually easier to read

But, importantly, there is *structure* in an IP address which helps with routing

### 22. Networks

#### IP

In this example, 138.38.32.14. the first half 1000101000100110 (138.38) is a 16-bit *network* address, which identifies the University of Bath

And 32.14 is a 16-bit host address, which identifies a single machine on the University’s network

Note that we write 138.38, but this should really be thought of as 1000101000100110, a bunch of 16 bits

Always remember that the dotted decimal notation is just a convenient way of writing a chunk of bits: there are no decimal numbers in the header!

### 23. Networks

#### IP

This division into network and host parts helps immensely in routing, as all packets destined for the University of Bath can be routed in the same manner

Only when a packet reaches the University is some local knowledge of the network needed

Indeed, the host part of this address splits further into *subnet* addresses that help local routing within the University

But the main point for now is that this IP address is independent of Ethernet and so can be used regardless of the hardware used

### 24. Networks

#### IP

So we have hardware independent addresses: but, now, there is an new problem

Suppose I want to send a packet to <138.38.3.40> on the local network. My data is (ultimately) encapsulated in an IP packet, with my IP address as source and <138.38.3.40> as the destination

(The question of how do we know the destination IP address must be answered later)

Now the IP packet must be further encapsulated in a hardware frame, Ethernet in this example. The OS can’t send the packet on the physical medium until it knows the Ethernet address of the destination

### 25. Networks

#### IP

Ethernet does not know about IP addresses: it can carry any kind of data

IP does not know about Ethernet addresses: it can run on many kinds of hardware

And this separation of layers, as we know, is desirable

### 26. Networks

#### IP

We need some kind of *address discovery*, so given the IP address we can find the corresponding Ethernet address

This is done by the *Address Resolution Protocol* (ARP)

ARP is a very simple link-layer protocol that essentially broadcasts a special frame on the local medium to the effect of “who has IP address <138.38.3.40>?”

All hosts on the local network hear this broadcast and the host with that address replies “Me: and I have Ethernet address <08:00:20:9a:34:dd>”

(There are questions of security here…)

### 27. Networks

#### IP

The OS gets the ARP reply and can now use this information to write the correct address in the Ethernet frame

Only now can the original packet be sent

### 28. Networks

#### IP

We don’t want to use ARP for *every* packet we send, so there is an ARP cache kept by the OS kernel that records the relation <138.38.3.40> <08:00:20:9a:34:dd>

Entries in the cache time out and are removed after, say, 20 minutes

This is in case the host using <138.38.3.40> goes away and is replaced by a different host that uses the same IP address, but has a different Ethernet address: recall IP addresses are *not* associated with the hardware

Once expired, the next packet to <138.38.3.40> will need a fresh ARP