COMPSCI 314 S1 C

IPv6, UDP, TCP

ICMP, SSH, FTP

Network Byte Order

Computers have many interesting and different conventions for the ordering of bits in bytes, bytes in words, and words in messages.

To give a uniform interpretation within the network, the Internet Standards specify a *Network Byte Order* for all fields interpreted by the network.

Integers within network headers must be sent with the bytes in decreasing numerical significance, most-significant byte first.

(The most-significant byte is nearest the start of the packet.)

This means that bytes or octets are transmitted in "raster-scan" order, left to right, top to bottom, as we look at the diagrams.

User data may be converted to Network Byte Order at the choice of the user.

Bits within a byte are sent least-significant-bit first; see RFC 2469 for comments on this

IP Version 6 (IPv6)

[Halsall section 6.8]

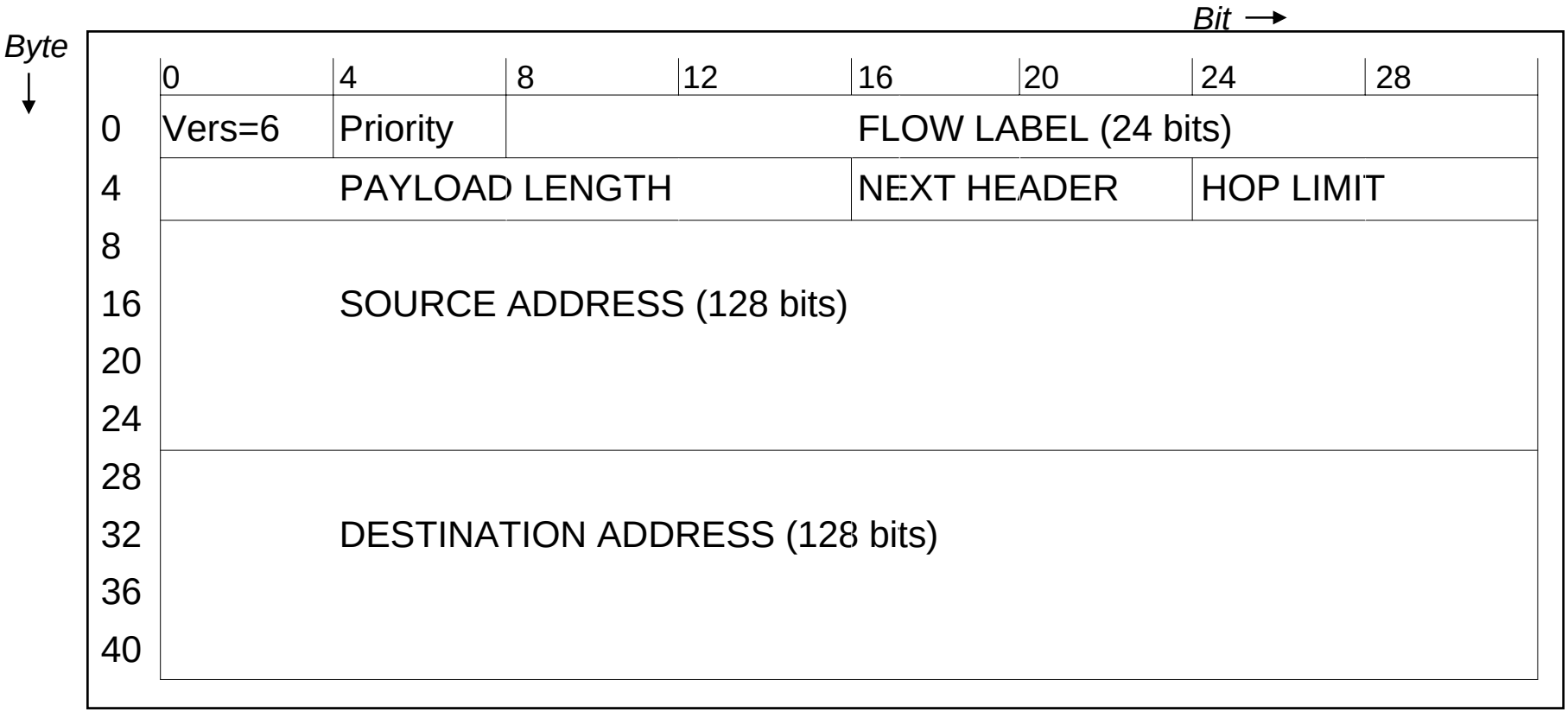
IP Version 6 addresses several problems with the older IP (version 4), especially —

- Exhaustion of the 32-bit address space
- Fragmentation is difficult and expensive for routers
- The IPv4 header is too complex — many aspects can be removed to specialised headers

Other new features include —

- Autoconfigration (site-local addresses)
- Encryption (of header fields and/or user payload)

The IPv6 Header



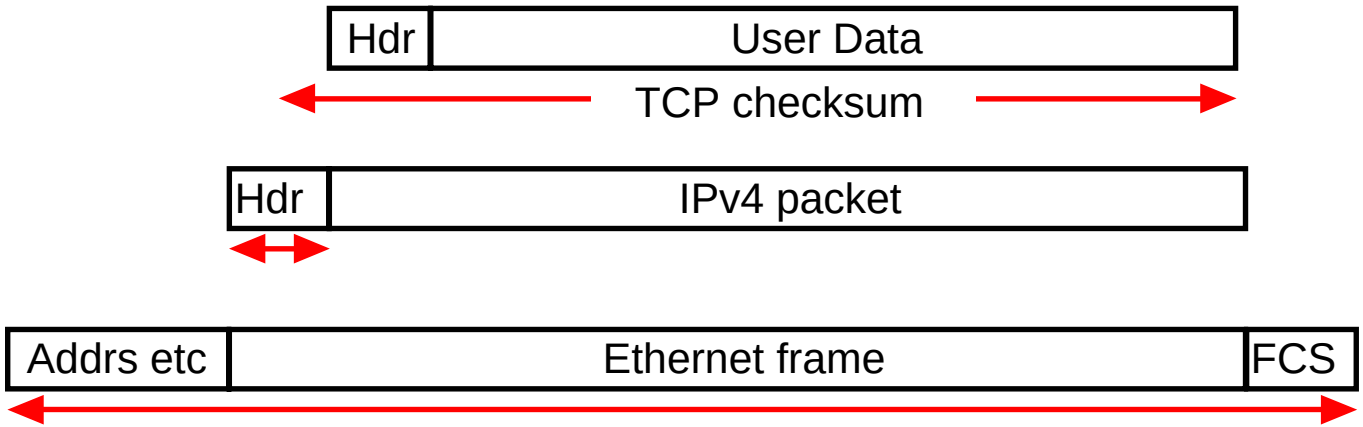
- The introduction is 64 bits long, and each address is two lots of 64 bits for better efficiency in fast processors
- The NEXT HEADER identifies either the header OR (on the last header) the packet protocol (e.g. TCP)

IPv6 Header Fields

- Hop Limit is set to the maximum number of hops and is decremented by each router. The packet is discarded if its hop limit becomes zero.
  It is like the IPv4 'TIME TO LIVE' (as it eventually developed, not as originally defined), but is not decremented by time.
- Flow Label is a unique identifier allocated by the source to a series of related packets. It allows packets to be treated similarly by routers, such as following the same paths. (The flow label and source address may be used to retrieve cached route information.)
  The flow label usually expires if not used for a while.
- Extension Headers are somewhat like IPv4 options. Each header (including the IPv6 header) indicates the nature of the next header, or the payload protocol if there are no more optional headers.
- Checksums are assumed to be handled by the underlying datalink layer and are omitted from the IPv6 header.

Notes on Checksums

TCP and IP were originally designed for unreliable links with poor error checking. They therefore have multiple levels of overlapping checksums. With Ethernet we get that



- TCP check covers whole of user data + TCP header for a complete end- to-end check, including source and destination addresses
- IPv4 check covers just the IP header to minimise switching overheads
- Ethernet check is very strong and covers all of each IP packet
- *IPv6 relies completely on the underlying checksums*

IPv6 Extension Headers

- IPv6 moves some information from the IPv4 header into extension headers which follow the main IPv6 header and have defined formats
- The IPv6 header has a ‘next header’ field which gives the type of the first header, and each header itself has a ‘next header’ field

Header types are —

- Authentication
- Destination options
- Fragmentation
- Hop-by-hop
- Routing
- Security

IPv6 Fragmentation

- IPv6 routers and switches do not fragment packets *en route*; the sender is supposed to ensure that the packet fits within the MTU for the path
- If a user packet is too long, the *sender* must fragment it and create appropriate smaller packets, each with an appropriate fragment header to allow reassembly of the user packet
- There is a way for IPv6s host to *discover* path MTUs
- Summary: fragmentation is entirely the responsibility of the *sender*

Connecting applications

- Data doesn't just get sent from host to host
- Data is exchanged between applications on the hosts concerned
- A typical host runs multiple applications simultaneously: WWW, e-mail, FTP, networked games, SSH or Telnet, database server/client, NetLogin, …
- Problem: IP itself provides the means for routing between hosts, but not for routing between applications
- How do we indicate which application our IP packet's payload originates from, and which application it is intended for?
- Need a higher order protocol to sort this one out! For example, UDP or TCP

The concept of *ports*

These are ‘addresses’ or identifying numbers for users of UDP and TCP, just like protocols or access points at lower layers —

- The LLC layer uses the DSAP to select the service, say IP
- IP uses its protocol field to select a network-layer protocol such as TCP (or ICMP etc.)
- TCP uses the Destination Port to select the end-user service, e.g. FTP or e-mail.
  Similarly, the Source Port identifies the sending user application

More on *ports*

- TCP and UDP, the two most important protocols layered on top of IP, both implement ‘ports’
- An (IP) ‘port’ identifies a sending or receiving application
- Port numbers in TCP and UDP are 16 bit, i.e., we can address up to 65536 applications on each host with each protocol
- Low-numbered ports (< 1024) are reserved for ‘well-known’ services, such as e-mail, web, ssh, etc.
- More on ports later!

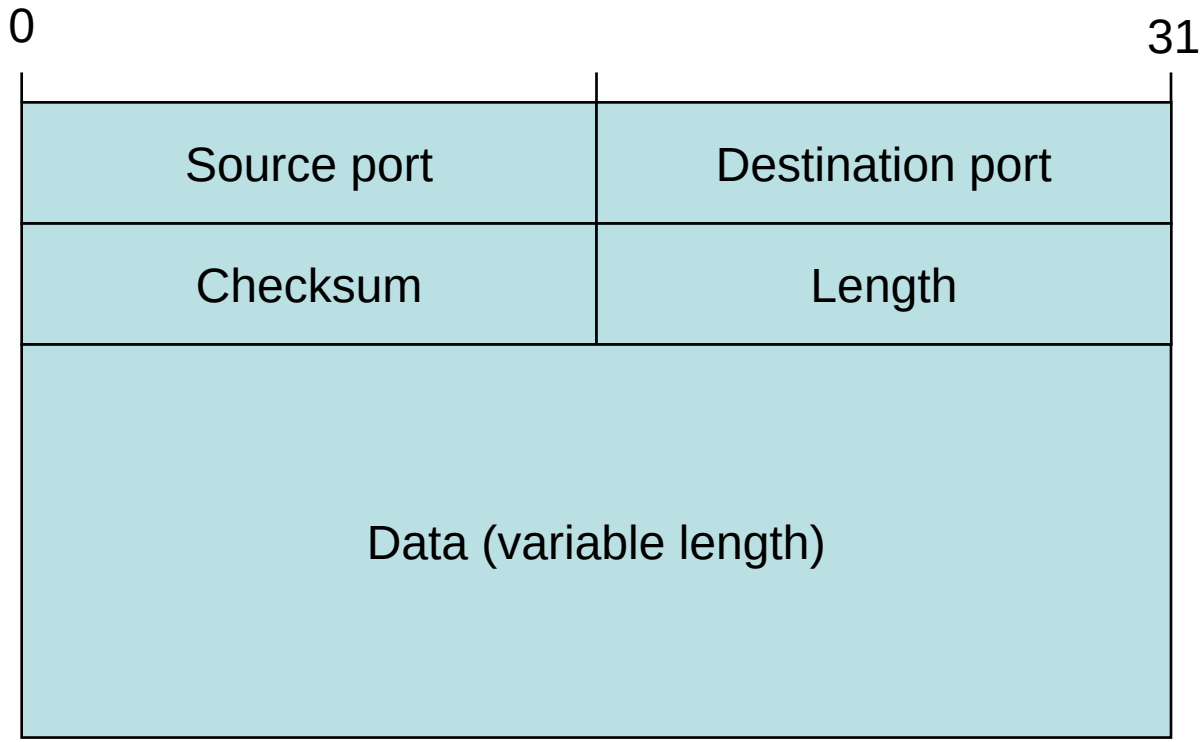
UDP and TCP

- UDP and TCP datagrams are encapsulated inside IP datagrams
- UDP – *User Datagram Protocol*:
  A connectionless best-effort protocol
- TCP – *Transmission Control Protocol*:
  A connection-oriented reliable *byte* stream protocol
- UDP and TCP are 'end-to-end ***transport****-layer* protocols' – the two hosts involved are solely responsible for header, content, error detection, error recovery, etc. Other hosts just forward the encapsulating IP datagram without looking at the content (at least in theory, anyway!)

User Datagram Protocol (UDP)

- UDP is used to send a single piece of information (byte block) from an application on one host to an application on another host via an agreed pair of ports
- Applications on each side are identified by their respective port numbers
- Delivery is ‘best effort,’ but not reliable – lost datagrams are not detected or retransmitted by UDP itself
- UDP datagram is checksum protected for error detection (remember that IP only protects its header by checksum – why?)
- Error recovery must be implemented by the applications themselves (e.g., by means of a protocol on top of UDP)

UDP datagram structure



Advantages and disadvantages of UDP

- Simple building block with almost ‘minimal’ design – easy to use as basis for higher-order protocols with maximal design flexibility ☺
- Widely implemented as part of the TCP/IP suite of protocols – every computer connected to the Internet can speak UDP too ☺
- No reliable delivery or error recovery ☹
- No flow control mechanisms – what happens if one host sends data at a faster rate than the receiving host can cope with? ☹

Transmission Control Protocol (TCP)

- TCP is used to establish a bidirectional ***byte*** stream between two applications on two hosts (a TCP 'connection')
- The byte stream is created by sending variable-length blocks of bytes in a series of TCP *segments*.
  At most one TCP segment per IP datagram
- Applications are identified by their respective port numbers
- Byte stream is reliable – errors and missing data are detected and retransmission is requested
- Flow control mechanisms enable better utilisation of available bandwidth

Sockets

[Halsall section 7.3.1]

- Sockets are software objects that are implemented in some way or another for all languages that need to access a machine's TCP/IP stack
- The applications 'plug into the socket'
- We may distinguish between client sockets and server sockets, although in many cases the same object will handle both socket types

Server socket particulars

- A server application's socket **binds** to a particular port using a particular protocol (TCP or UDP). This is usually done by setting the port and protocol properties of the socket object
- The port number/protocol pair are unique to the server application on the server host – no other application has the same port number *and* uses the same protocol
- When the server application has been loaded and is ready to receive connections (or UDP packets), the server socket is told to **listen**. This is usually accomplished by means of a method call on the socket object
- For each TCP connection (or UDP client port), the server socket either causes the application to fork a child process, or sets up a connection ID that uniquely identifies the connection or the UDP client
- Note that the server socket is not given an IP address – it can read the server host's own address from the configuration and learns the client's IP address from the incoming datagrams

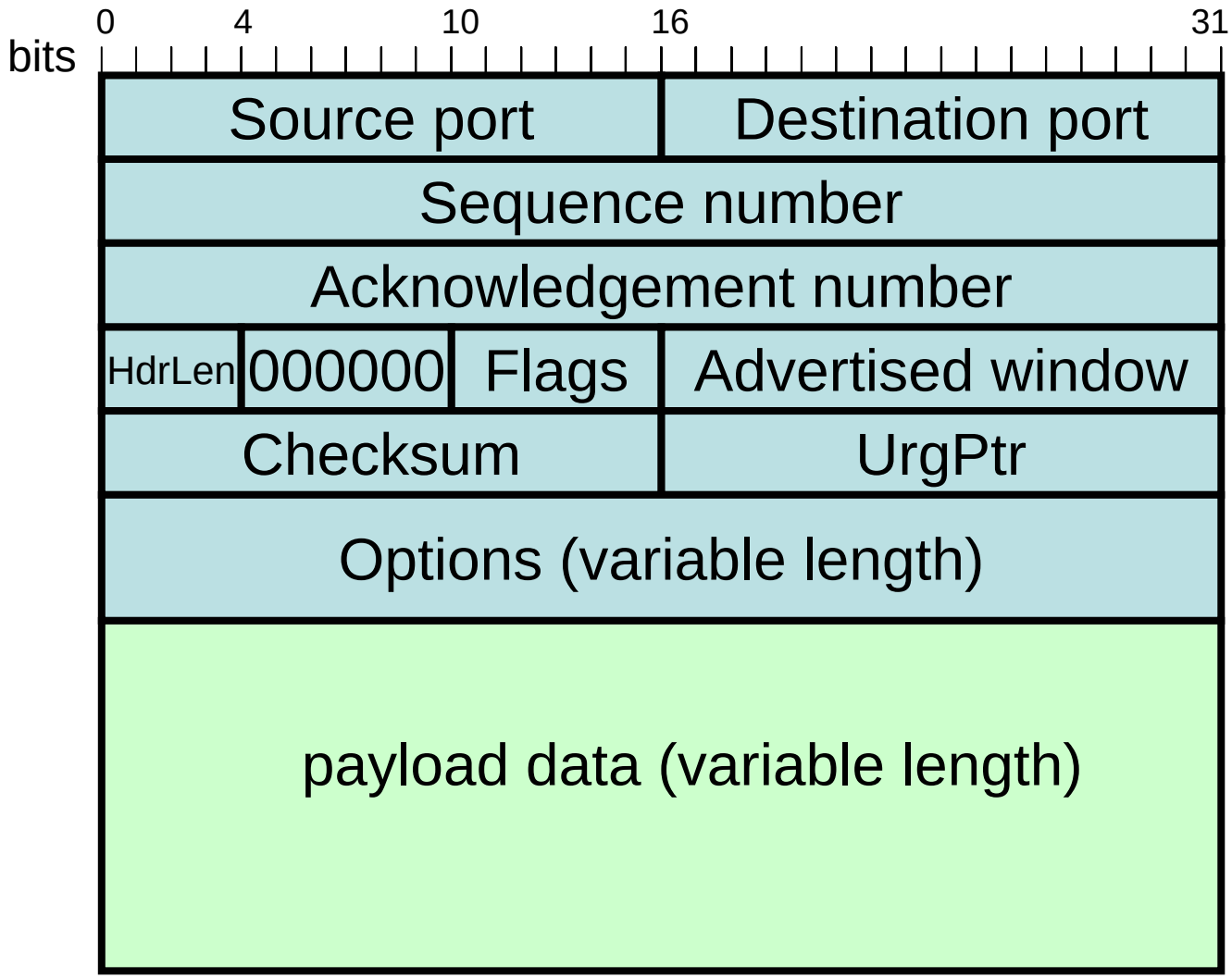
Client socket particulars

- A client application must tell its socket which IP address it wishes to connect to. This is usually done by setting a property of the socket object
- The client application must tell the socket which protocol it wishes to use. This is usually also done by setting a property of the socket object
- A client application's socket usually doesn't specify the port it wishes to use on the client machine
- Rather, it asks the TCP/IP stack for a port from a pool of available ports
- The application then tells the client socket to connect. This is usually done by invoking a method on the client object
- Once the connection is established, the socket generates either an event or calls a callback function in the application in order to notify it of the successful connection

Common mechanisms for both client and server sockets

- Once the connection is established, the socket sets up two buffers: a send buffer, and a receive buffer
- The application writes data into the send buffer by invoking a method on the socket. The socket then tries to dispatch the data to the other end
- If data is received, the socket writes it to the receive buffer. It then notifies the application via an event or a callback function that data has arrived
- The application then invokes a method on the socket object that reads the data out of the receive buffer
- If the application wishes to disconnect, it invokes the associated method on the socket object
- If the other party disconnects, the socket object notifies the application by means of an event or by invoking a callback function

TCP datagram structure



TCP sequence numbers

TCP uses a sliding window protocol, with a variable window size. Sequencing is by a byte sequence count.

The sequence number is the sequential number of the first byte of this segment. (The very first byte is given a *random* sequence number.)

- The acknowledgement number is the number of the next byte which the receiver expects. It acknowledges all preceding bytes of the data stream. This field is used only if the ACK flag is set
- Note that if the acknowledgement from one segment is lost, a later acknowledgement may well replace it, with no error seen

The Header Length is the length of the TCP header, in 32-bit 'words,' of all the options field(s), OR is the offset, in words, where the payload (data) starts.

TCP flags

There are six TCP flag bits:

- SYN — Synchronise sequence numbers, especially for start-up
- ACK — Acknowledgement number is valid
- FIN — Finish of byte stream, i.e. close connection (in *one* direction)
- RST — Reset the connection
- PSH — Data so far is to be forwarded immediately, TCP shouldn't wait before sending it (not very useful these days)
- URG — Urgent pointer field is valid (almost never used)

TCP *window* and *checksum*

Receive Window

- Used for flow control, it tells how many bytes can be received beyond those acknowledged. The sender knows how many it has actually sent when it receives the acknowledgement, and therefore how many beyond that can still be received. The receiver may vary the window size.

Checksum

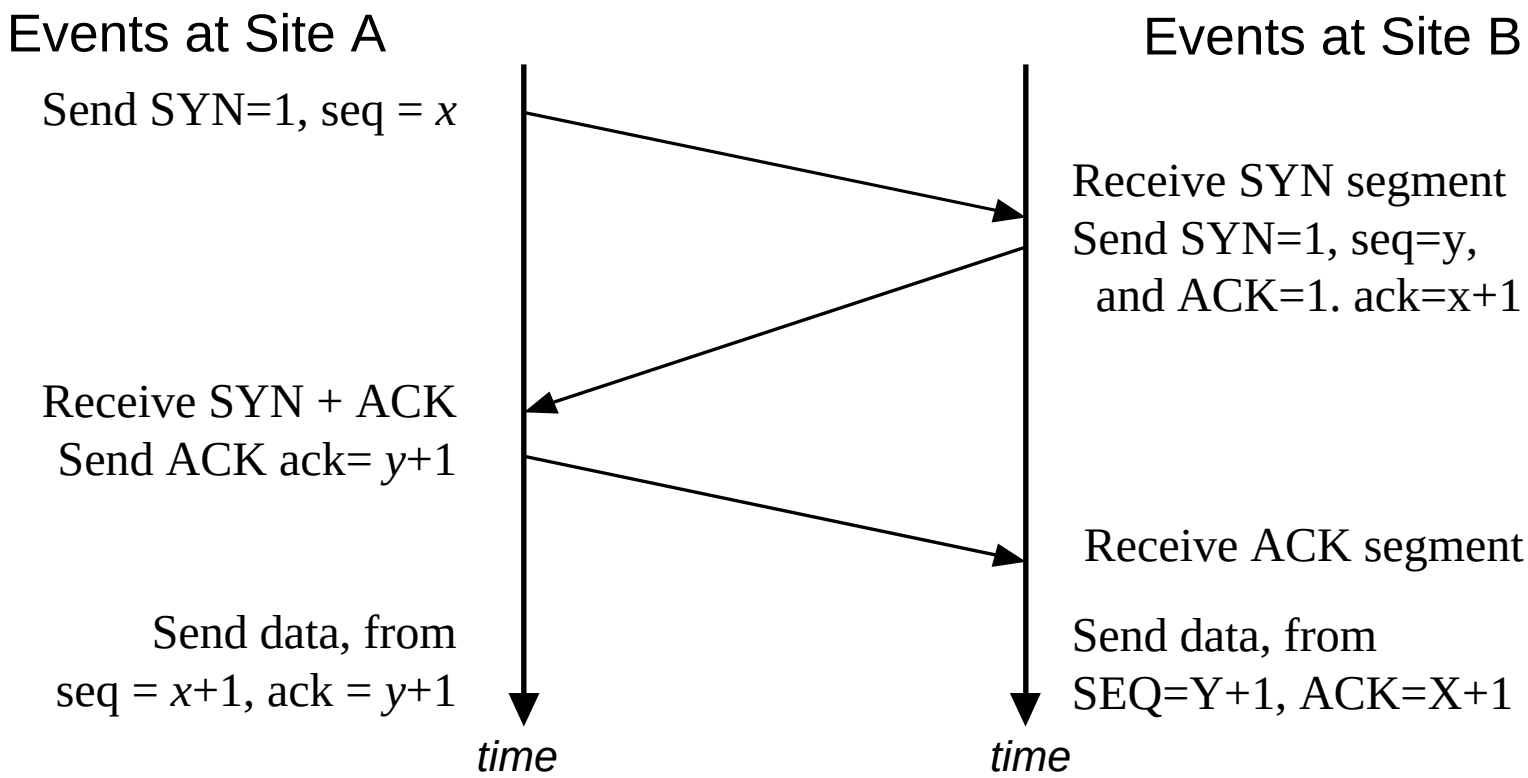
- Take the 1-s complement sum of the segment (as 16- bit words) and store its complement as the checksum. (The checksum includes a 32-bit ‘pseudo- header’ containing the IP addresses, protocol and length)

TCP Connection Protocol

- TCP uses a modified 3-way handshake, to guard against lost or duplicated packets or segments.

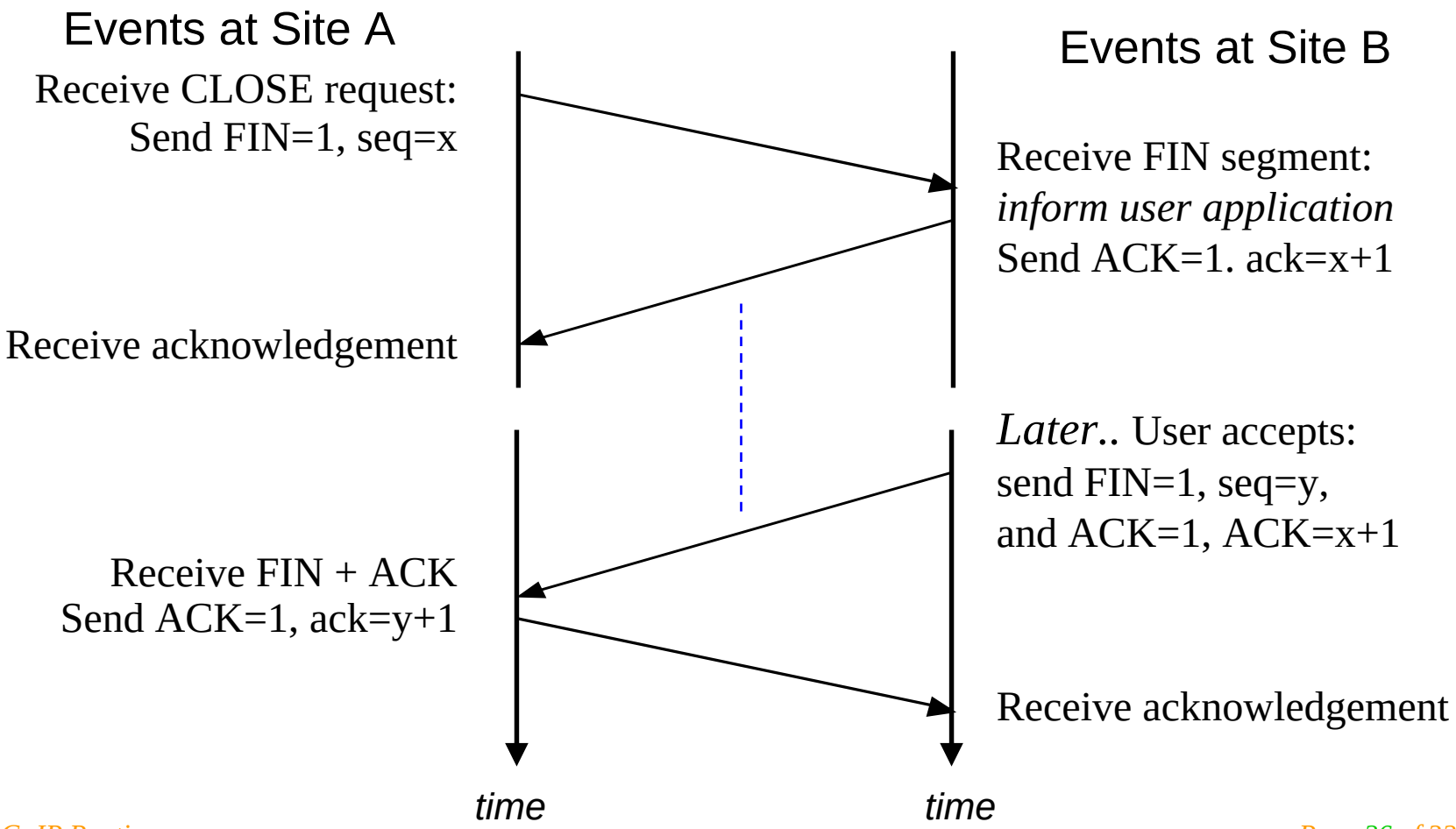
  [Halsall section 7.3, pp 450-452]

- On the diagram we use an Initial sequence number of 1. *Actual starting sequence numbers are chosen at random!*



TCP Disconnection Protocol

- Again, use a modified 3-way handshake, to guard against lost or duplicated packets or segments



How TCP achieves reliability

- Two mechanisms: checksum in each datagram and sequence numbering of bytes with acknowledgement
- Each datagram contains a sequence number for its first payload byte. Missing datagrams are detected by a sequence number that is too high compared to the number of bytes actually received
- Received bytes are acknowledged by returning the expected sequence number of the next expected byte to the sender, with the ACK flag set
- Sender resends unacknowledged bytes after a certain timeout
- Checksum detects errors in TCP datagrams – datagram is dropped at receiver and correction is done by retransmission

Example: Acking TCP datagrams – no error*

- …
- Host A -> Host B: 100 bytes, SEQ 13456
- Host B -> Host A: ACK, expect SEQ 13556 (=13456 + 99 other bytes + 1) to come next
- Host A -> Host B: 200 bytes, SEQ 13556
- Host B -> Host A: ACK, expect SEQ 13756 (=13556 + 199 other bytes + 1) to come next
- Host A -> Host B: 120 bytes, SEQ 13756
- Host B -> Host A: ACK, expect SEQ 13876 (=13756 + 119 other bytes + 1) to come next
- …

* Host A sending data to Host B, simplified

Example: Acking TCP datagrams – with error*

- …
- Host A -> Host B: 100 bytes, SEQ 13456
- Host B -> Host A: ACK, expect SEQ 13556 (=13456 + 99 other bytes + 1) to come next
- Host A -> Host B: 200 bytes, SEQ 13556 (this is lost)
- Host A -> Host B: 120 bytes, SEQ 13756
- Host B -> Host A: ACK, expect SEQ 13556 to come next
- Host A expects acknowledgement with expected Seq 13876. This is not coming, host A times out and resends from the last acknowledged byte ..
- Resend: Host A -> Host B: 200 bytes, SEQ 13556
- Host B -> Host A: ACK, expect SEQ 13876 (=13756 + 119 other bytes + 1) to come next
- …

* Host A sending data to Host B, simplified

Why datagrams may be lost

- Dropped at an overloaded router (too many datagrams in the router's queue)
- Corrupted by bit errors due to noise or interference – dropped at the receiver because checksum doesn't compute to correct value
- Dropped at the receiver because the receiver cannot process the data at the required rate and its buffers are full

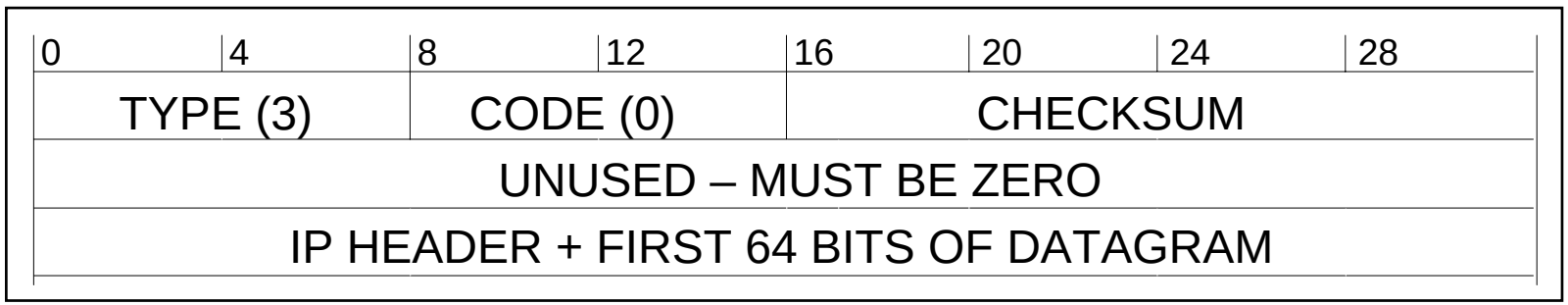
TCP Congestion Control

- Size of TCP's sliding window is varied to control sending rate
- TCP's sending window is usually called its congestion window
- TCP senses congestion by observing *lost packets*, i.e. by watching sequence numbers
- When congestion occurs, the congestion window size is halved – *exponential backoff*
- TCP then increases the congestion window again, one segment at a time

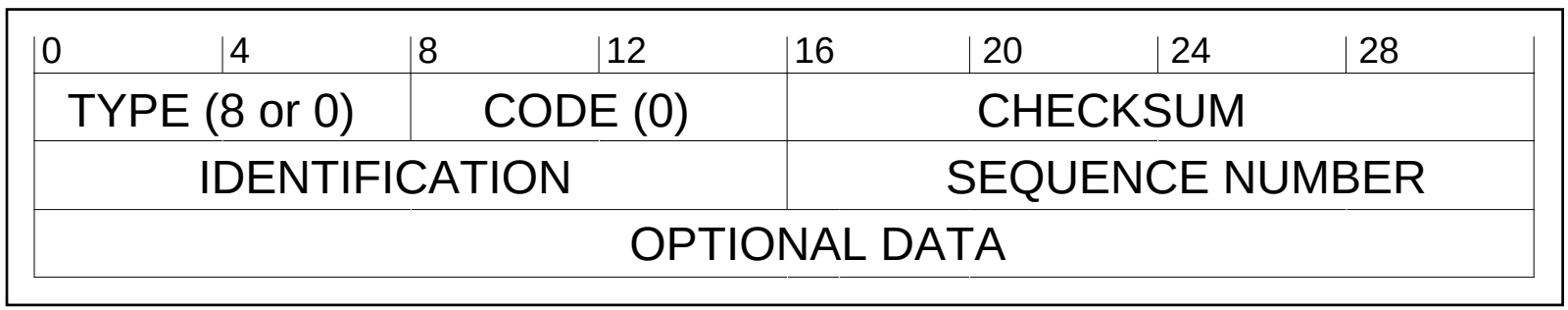
ICMP – Internet Control Message Protocol

Provides support for IP.
ICMP packets have an 8-bit message type, e.g.:

- *Destination Unreachable* (routing failure)



- *Echo Request or Reply* (PING to remote station)



ping and *traceroute*

Well-known applications for simple network testing

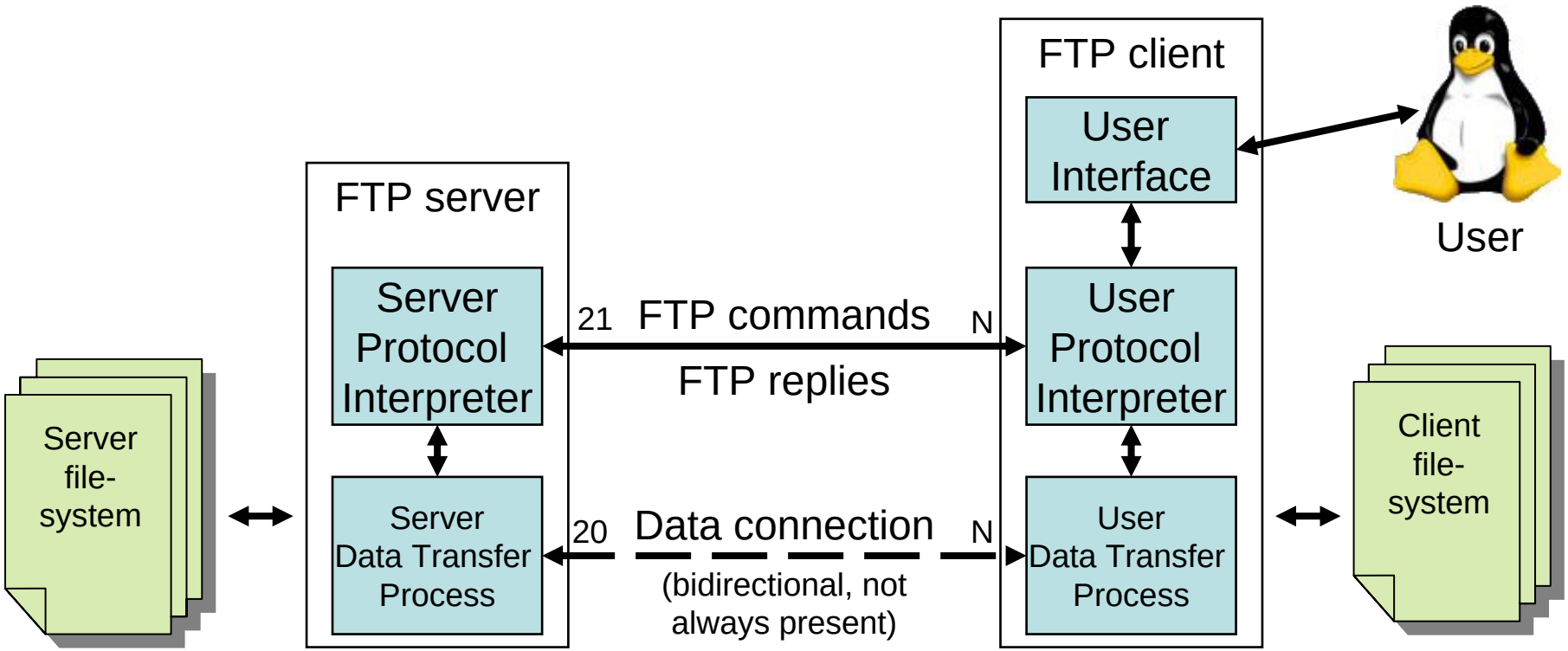
- *ping* lets you check whether a remote machine is reachable and how long it takes to get a reply
- *ping* was often misused for denial-of-service attack. Many firewalls these days will drop ping packets
- *traceroute* is like ping, except that it will also show you all hosts along the route
- *tracert* under Windows
- If a TCP or UDP port on a host is unused, incoming connection requests / UDP packets are answered with an ICMP ‘Port unreachable’ message

SSH - Secure SHell protocol

- *ssh* is an application that uses SSH to provide remote login to Internet hosts over TCP via port 22
- Provides some terminal emulation (like *telnet*)
- *All* data between *ssh* client and remote host is *encrypted*
- Several different mechanisms to authenticate users (without having to send passwords in clear)
- Can run other protocols (e,g, file transfer) through an *ssh* tunnel

FTP – File Transfer Protocol

- Mother of all download protocols!
- Carried over TCP, specification in RFC 959



FTP commands

• open <host>		*open FTP connection to <host>*
• quit		
• lls	[<dir>]	*local list*
• lcd	<dir>	*local change of directory*
• ls	[<dir>]	*remote list*
• cd	<dir>	*remote change of directory*
• get	<file>	*download file*
• mget	<filemask>	*download several files*
• put	<file>	*upload file*
• mput	<filemask>	*upload several files*
• i		*enter binary mode*
• a		*enter ASCII mode* (converts line breaks automatically between different platforms)