Computer Networks and Distributed Systems

Computer Networks – Transport Layer

Course 527 – Spring Term 2014-2015

Anandha Gopalan

a.gopalan@imperial.ac.uk
http://www.doc.ic.ac.uk/~axgopala

Contents

- Transport Layer
  - End-to-end communications
  - End-to-end addressing: ports
- Transport protocols
  - **UDP**
    - Header format
  - **TCP**
    - Header format
    - Connection setup
    - Retransmission
    - Flow/congestion control
    - …

Host-to-Host Communications

- Datagrams transferred between hosts
  - Network/Internet level in stack
- Routed through networks based on IP addresses
  - Source address of sending machine
  - Destination address of recipient machine
- But:
  - No identification of which applications send or receive
  - Only unreliable connection-less datagram service

Protocol Ports

- Use ports to define end points
  - Abstract addressing
  - 16 bit unsigned integer: 0 – 65535

- Processes specify ports to send to or receive from
  - Use operating system calls to bind to port
  - Packets have source and destination ports

Well Known Service Ports

- Other systems know which port to address to
  - Standard services usually run on well known ports

- List of well known services (RFC 1060)

  Ports 0 – 255: Internet Assigned Numbers Authority (IANA)

  Ports 0 – 1023: Privileged UNIX standard services

- Static file with service/port mappings

  Unix: `/etc/services`

  Windows: `\windows\system32\drivers\etc\services`

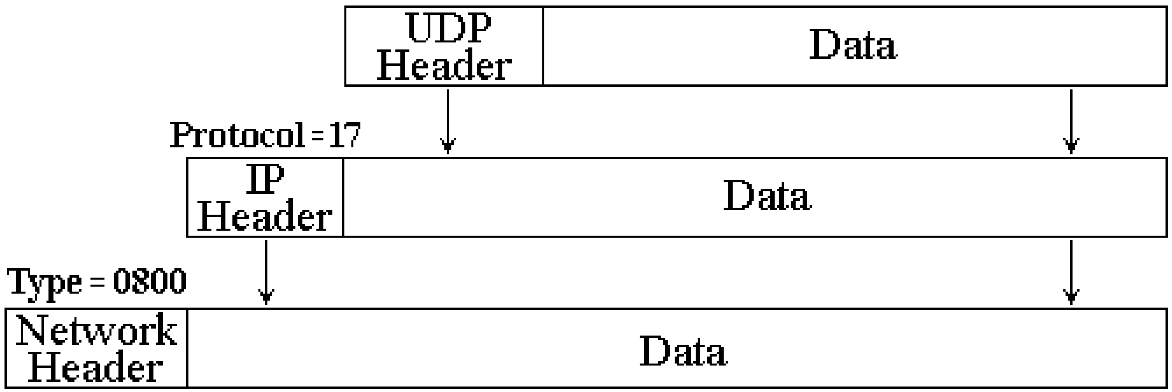
Complete Addresses

- Complete addressing for a datagram (UDP/TCP) describes the sender and receiver in such a way that the services which are communicating are fully defined

- Source IP address and source port
  - Source port = return addr (may be omitted in UDP)
  - e.g. `maidenhair.doc.ic.ac.uk:57992`

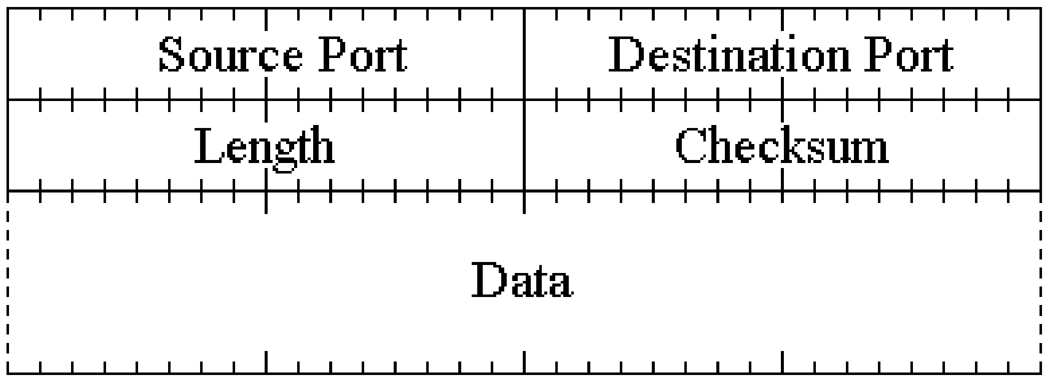
- Destination IP address and destination port
  - e.g. `www.amazon.co.uk:80`

User Datagram Protocol (UDP)



- UDP provides plain, IP-like service
  - Connection-less datagrams, unreliable delivery, no sequence control, possible duplication
  - Good for fast transfer with resilience to packet loss

UDP Header Format

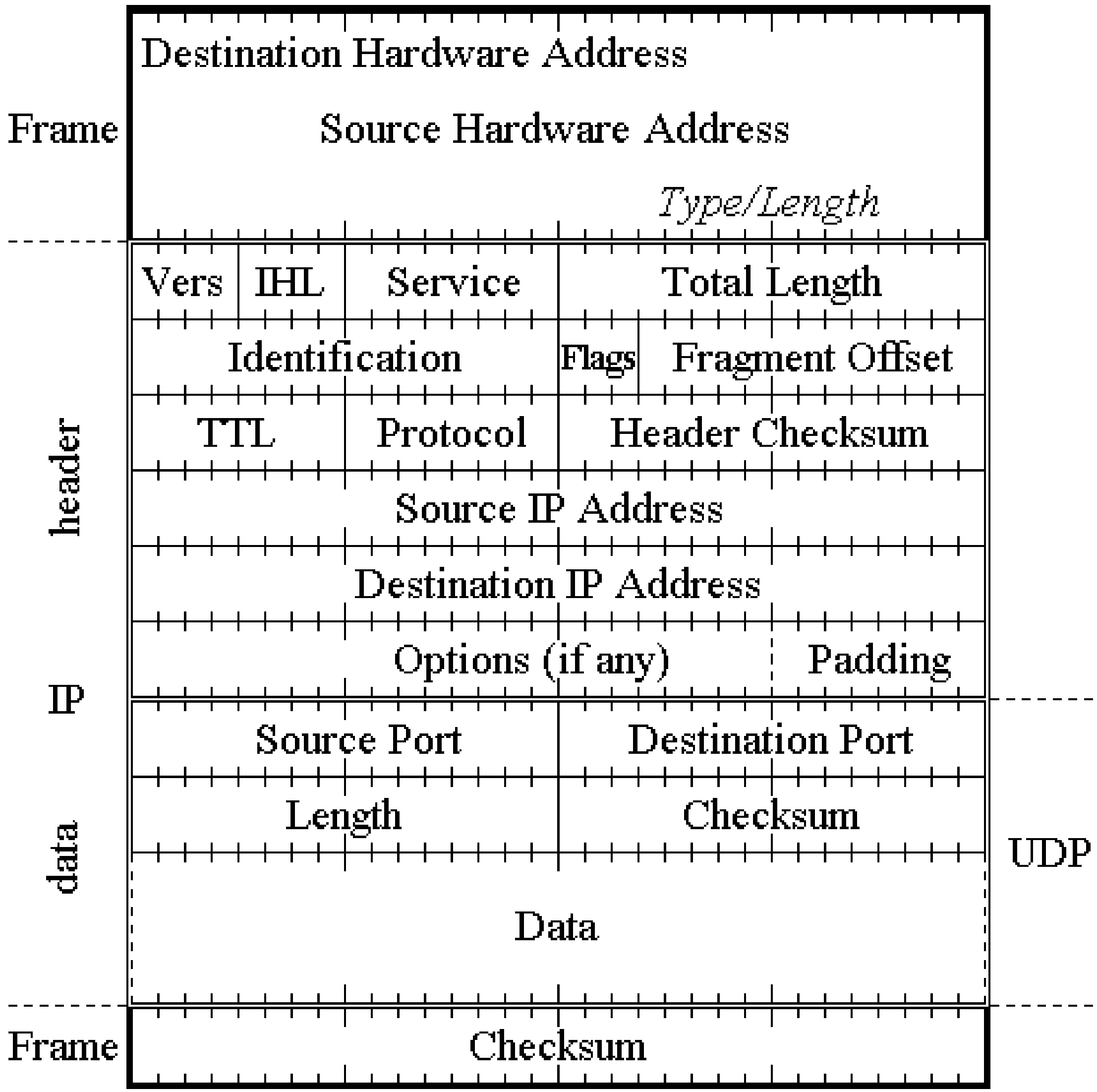


- Source port
  - Optional, 0 if not used, reply-to port if used
- Destination port
  - Host addressing still provided by IP headers
- Length (in bytes)
  - Includes header and data (min. 8 for no data)

UDP Checksum

- Checksum (16 bit using 1s complement sum)
  - Calculated over pseudo header + UDP header + data
  - Pseudo header mimics IP header
    - Source IP address
    - Destination IP address
    - Protocol (17)
    - UDP length
      - Zeros to pad to multiple of two octets
  - Allows detection of changed IP headers by gateways without actually duplicating data

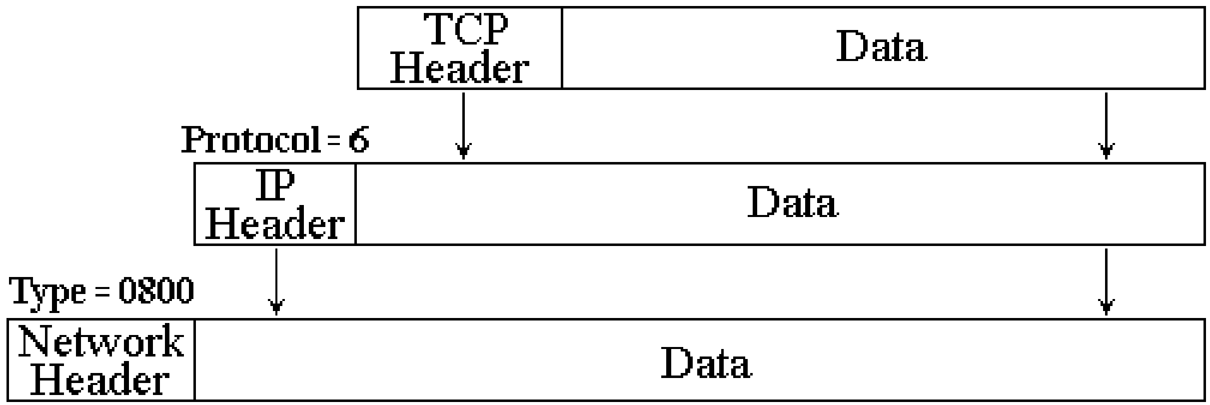
UDP/IP Packet in Ethernet Frame



Reliable Service?

- UDP is unreliable
- Features of reliable service
  - Unstructured stream abstraction
  - Virtual circuit connection
  - Full-duplex connection
- Could build reliable service over UDP
  - Needs to keep track of successfully transmitted packets
  - Add error-correcting & retransmission mechanisms

Transmission Control Protocol (TCP)



- TCP adds a lot to IP
  - Streams with reliable delivery
  - Full-duplex operation
  - Flow control
  - Network adaptation for congestion control
  - Complexity & overheads

Connections

- TCP uses connection as its basic abstraction
  - Rather than just protocol port (receiving datagrams)
- Connection-oriented service on top of connectionless IP
  - Connection identified by pair of endpoints
- Endpoint is (host, port) tuple: (IP addr:port)

  e.g. src: **`146.169.15.121:1069`**
  dst: **`140.247.60.24:25`**

  Web application on **`146.169.15.121`** connects to HTTP port (80) on **`146.169.13.59`**

TCP Features

- Streams
  - TCP data is stream of bytes
  - Underlying datagrams concealed

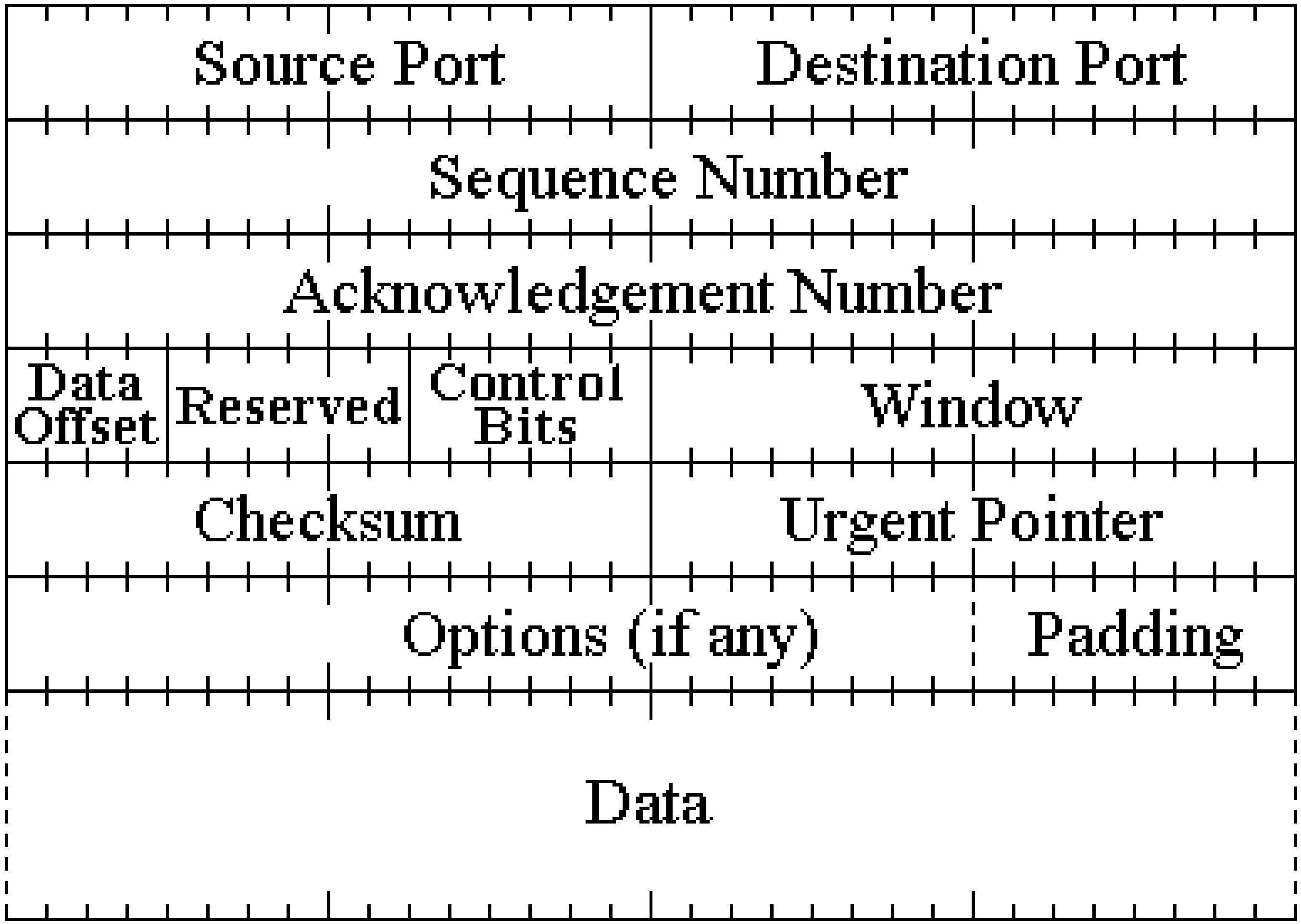
- Sequence numbers for reliable delivery
  - Used to maintain byte order in stream
  - TCP detects lost data and arranges retransmission
  - Stream delayed during retransmission to maintain byte sequence

TCP Features

- Flow control
  - Manages data buffers and coordinates traffic (between sender and receiver) to prevent overflows
    - Fast senders have to pause for slow receivers to keep up

- Congestion control
  - Monitors and learns delay characteristics of network
  - Adjusts operation to maximise throughput without overloading network

TCP Segment Format



TCP Fields: Sequence Numbers

- Sequence number (seq num)
  - Indicates position in stream of 1st data byte in segment

- Acknowledgement number (ack num)
  - Exploit full-duplex connection and use segments to piggyback acknowledgments
  - Ack num = next seq num expected

TCP Fields: Data Sizes

- Data offset
  - Number of 32-bit words in TCP header
  - Needed because of variable length options
- Window size → used for flow control
  - Number of data bytes which may be sent, starting with byte indicated by ack num
  - Recipient should not send more than (window size – bytes sent) without acks (in transit)
  - 0 means “no more data now, please”

TCP Fields: Control + Checksum

- Control bits

  URG: urgent pointer valid

  ACK: ack num valid

  PSH: "push" this segment (transmit promptly)

  RST: reset connection

  SYN: synchronise sequence numbers

  FIN: sender has reached end of byte stream

- Urgent pointer
  - Pointer to high priority data in stream (e.g. error conditions)

- Options
  - Negotiate max segment size, scaling factor for window size, ...

Checksum (16 bit using 1s complement sum)

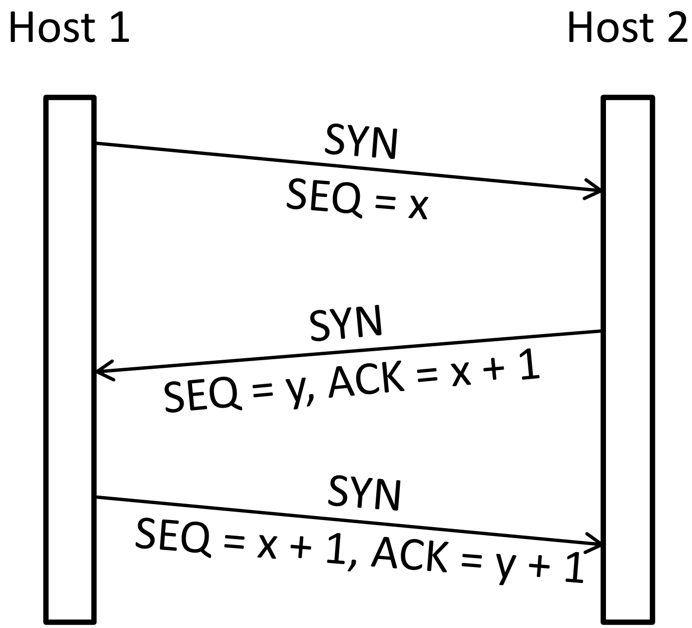
- Same as for UDP with pseudo header

Connection Control

- Two hosts must synchronise seq nums
  - Controls packet order and detects loss + duplicates
- Use SYN segments to establish connection
  - Establish initial sequence num (ISN)
  - Stream positions are offsets from ISN
    - 1^{st} data byte in segment = ISN + 1
- ISN chosen randomly
  - Need to be unique over life-time of connection
  - Starting at 0 bad idea because of old packets…

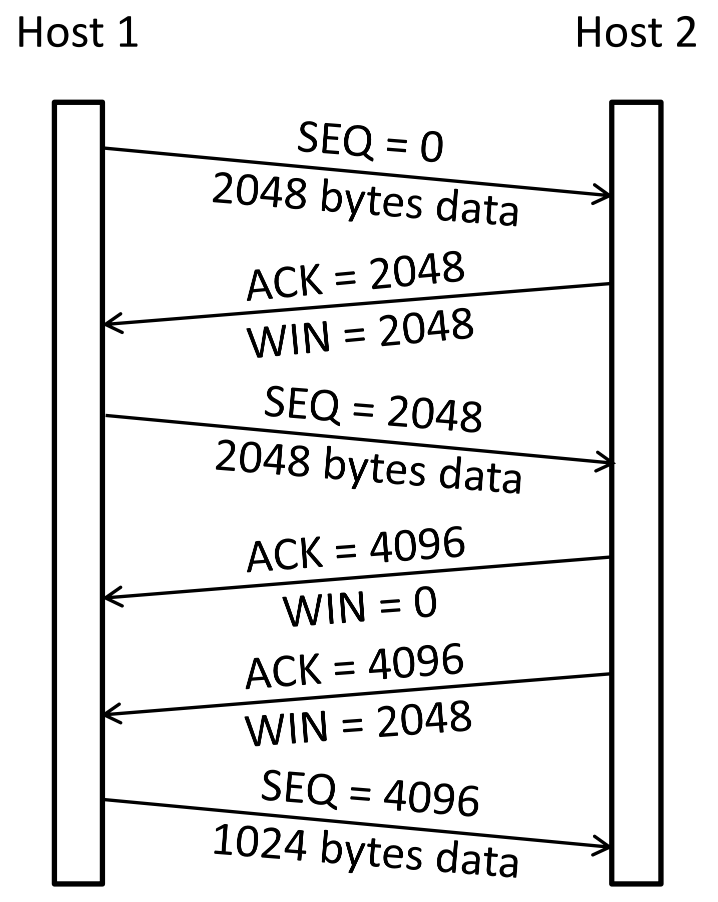
Connection Establishment

- Three way handshake
  - Establishes connection
  - Sender and receiver agree on seq nums
  - Works when two hosts establish connection concurrently



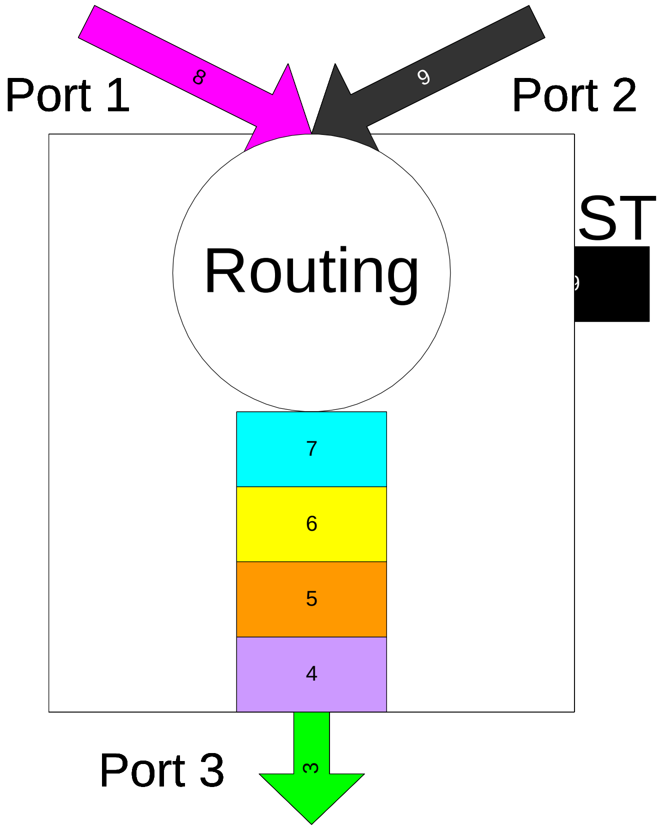
TCP Window Management

- Host 2 has 4k buffer
- Sent data controlled by WINdow field
- Sender does not have to fill receiver's buffer with each segment
- SEQuence and ACKnowledgement indicate what has been sent/received



Congestion

Congestion occurs in routers and in medium access



- Multiple incoming links can saturate single outgoing link
- Slower outgoing link can be saturated by one incoming link
- Routers use store-forward
  – Process each packet before sending
  – If buffer becomes full, drops packets

TCP Congestion Control

- Packet loss mostly due to congestion and not error
  - Detect congestion by considering packet loss
  - Change transmission rate to adapt to congestion

- TCP sender maintains 2 windows
  - Receiver window (flow control) and congestion window
  - Uses whichever currently smaller

TCP Congestion Window

- Congestion window based on network conditions
  - Windows grows and shrinks based on packet loss
  - Different algorithms for finding optimal size
    e.g. additive increase, multiplicative decrease

- Requires efficient timeouts to detect loss
  - TCP measures RTT and adjusts timeouts

- Lots of complexity to ensure throughput, fairness, …

Slow Start

- What is the initial Congestion Window (W)?
  - The initial value of W is MSS, which is quite low for modern networks

- In order to get quickly to a good throughput level, TCP increases its sending rate exponentially for its first growth phase (1 MSS per good acknowledgment)

- After experiencing the first loss, TCP cuts W in half and proceeds with a linear push

- This process is called slow start, because of the small initial value of W

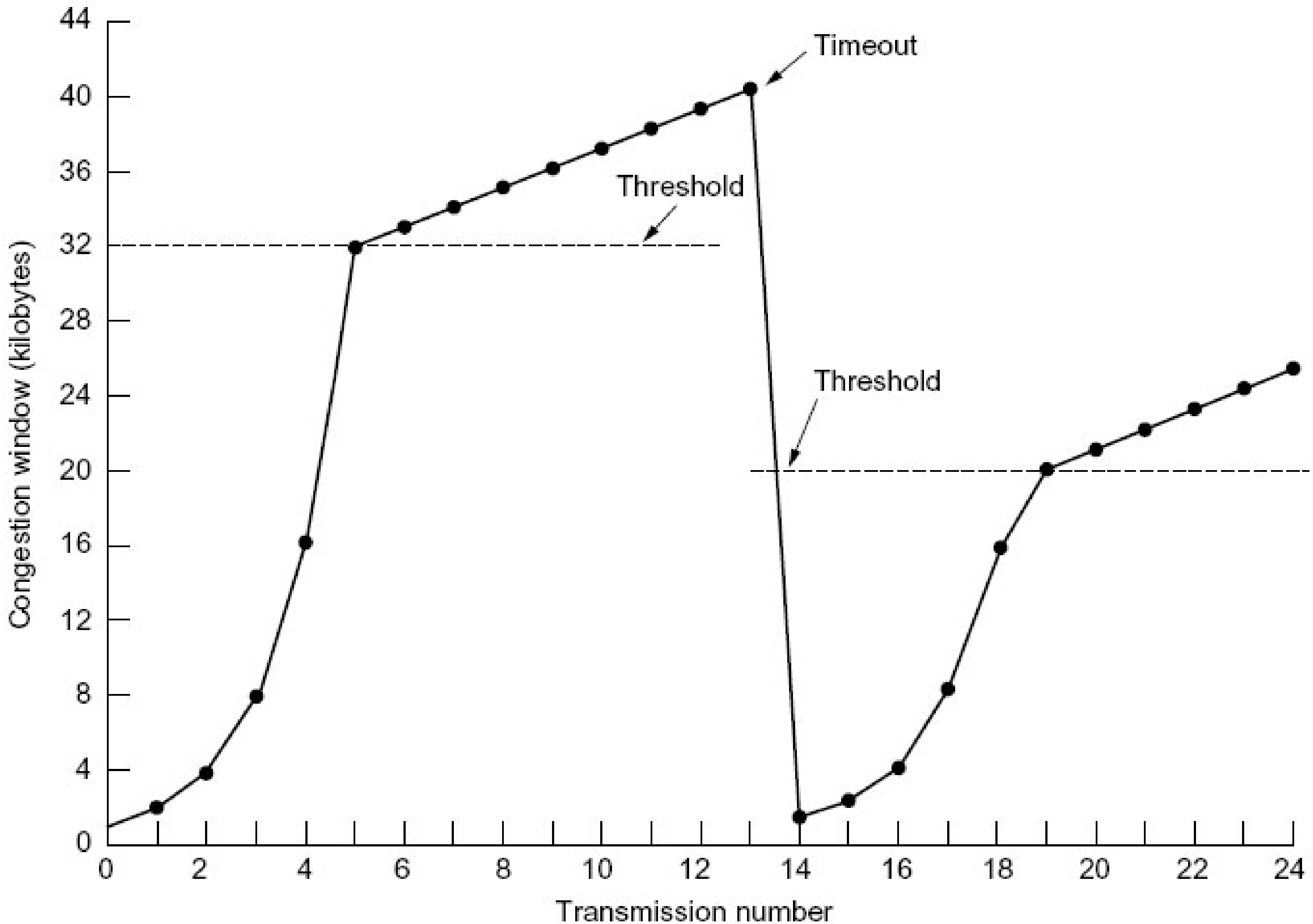
Additive-Increase/Multiplicative-Decrease

- Congestion Window (W) is incremented by 1MSS/W on every good acknowledgement
  - Increase W by MSS every round-trip time RTT
  - Called congestion avoidance
  - e.g., suppose W = 14600 and MSS = 1460, then the sender increases W to 16060 after 10 acknowledgments

- At every loss event, TCP halves the congestion window
  - e.g., suppose the window size W is currently 20Kb, and a loss is detected – TCP reduces W to 10Kb

- Drops congestion window to MSS on a timeout and then do slow-start until W/2 and then do congestion avoidance

TCP Congestion Window



Summary: UDP or TCP?

UDP

- No need for reliability and error detection
  - Message exchanges without transactional behaviour, e.g. DNS, DHCP
  - Real-time apps, e.g. sensor monitoring, video streaming
- Good for short communications
- Efficient for fast networks

TCP

- Need for reliability, error correction, flow and congestion control, or security
  - Terminal sessions, e.g. SSH, Telnet
  - Large data transfer, e.g. web, FTP, email
- Efficient for long-lived connections
- Requires more CPU time and bandwidth than UDP