Transport: Basics

G54ACC

Lecture 10

richard.mortier@nottingham.ac.uk

Recap

- UDP connectionless, unreliable
- TCP *connection oriented*, providing *reliability*
  - Provides *flow control* and *congestion control*

- Congestion control will be covered in the next lecture

Contents

- Overview
- User Datagram Protocol
- Transmission Control Protocol

Contents

- Overview
  - Transport layer
  - Port numbers
- User Datagram Protocol
- Transmission Control Protocol

The Transport Layer

- Process-to-Process communication
  - Cf. Host-to-Host (or interface-to-interface) from IP
  - I.e., Multiplexing within host
- UDP, RFC768
  - Best effort, unordered, datagrams
- TCP, RFC793
  - Reliable, ordered, bytestream
- ...with respect to the *process* not the host

Port Numbers

- Provide multiplexing of host's network connection between processes
  - (src IP, dst IP, src port, dst port, proto)
    5-tuple uniquely identifies a *flow* in the Internet
    - At a given moment in time anyway
  - Destination port often identifies service
- 16 bit number space
  - [0, 1023]: *well-known*, usually require root privileges
  - [1024, 49151]: *registered*, often out-of-date
  - [49152, 65535]: dynamic/private/*ephemeral*

Contents

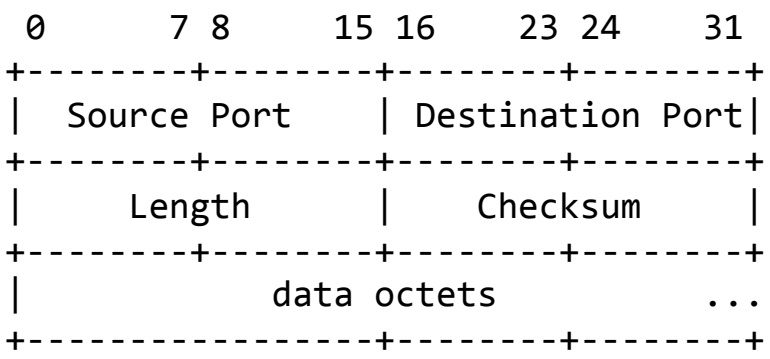
- Overview
- User Datagram Protocol
  - Header format
  - Pseudo-headers
- Transmission Control Protocol

User Datagram Protocol, UDP

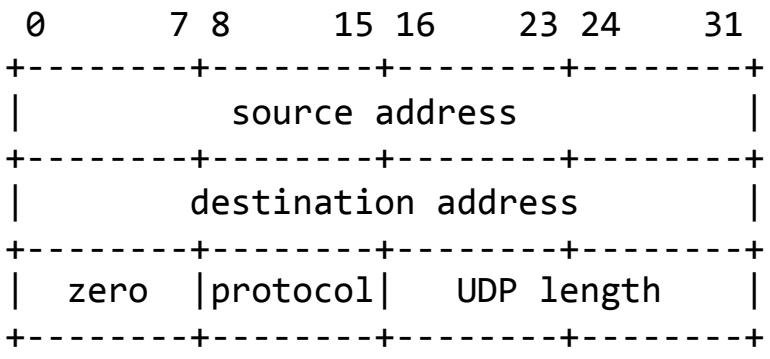
- About the simplest possible transport protocol
  - Provides simple datagrams over IP's packets
  - Adds a *datagram checksum* and *port numbers*
- Checksum covers *pseudo-header* and data
  - If you receive data, it was (probably) intended for you and was (probably) transmitted correctly
  - Integrity, not reliability
- Ports enable multiple processes per host to use IP networking simultaneously
  - Source port can be zero

UDP Headers

- UDP Header
  - Port numbers, length
  - Checksum protects against bit errors



- UDP Pseudo-header
  - Prefixed to UDP header
  - IP addresses, IP protocol, length
  - Why?



Contents

- Overview
- User Datagram Protocol
- Transmission Control Protocol
  - Header format
  - State machine: setup, teardown
  - Priority, precedence
  - Reliability, timeouts
  - Host considerations
  - Flow control

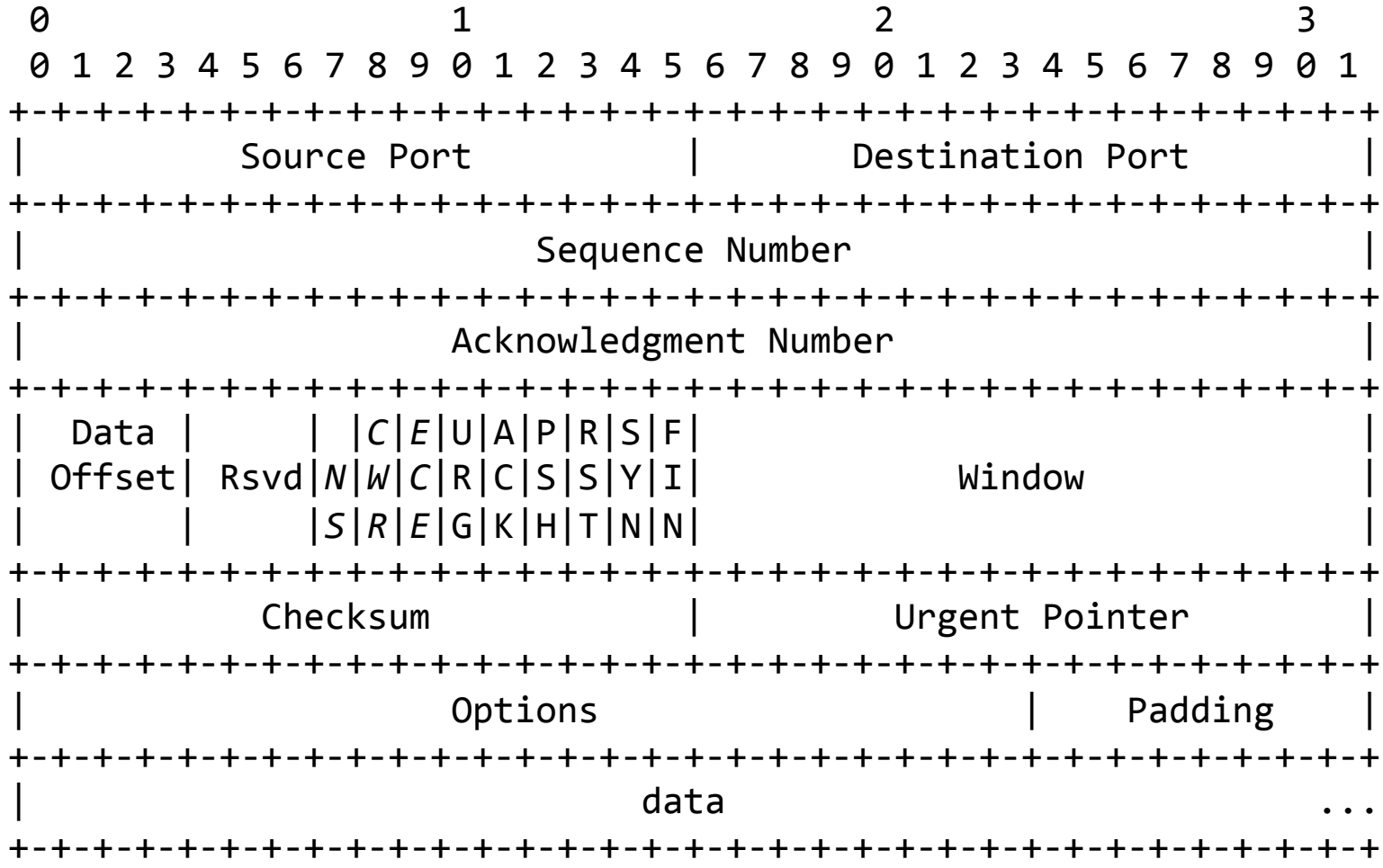
Transmission Control Protocol

- Reliable...
  - It will retransmit data if it gets lost
- ...ordered...
  - Data guaranteed to arrive in the order sent
- ...bytestream...
  - No need for application to segment data
  - However, it is common to insert delimiters
- **Key problem**: Network is *asynchronous*
  - No timing guarantees on, or indication of, delivery
  - Use timers to infer loss – but how to set them?

A General Principle

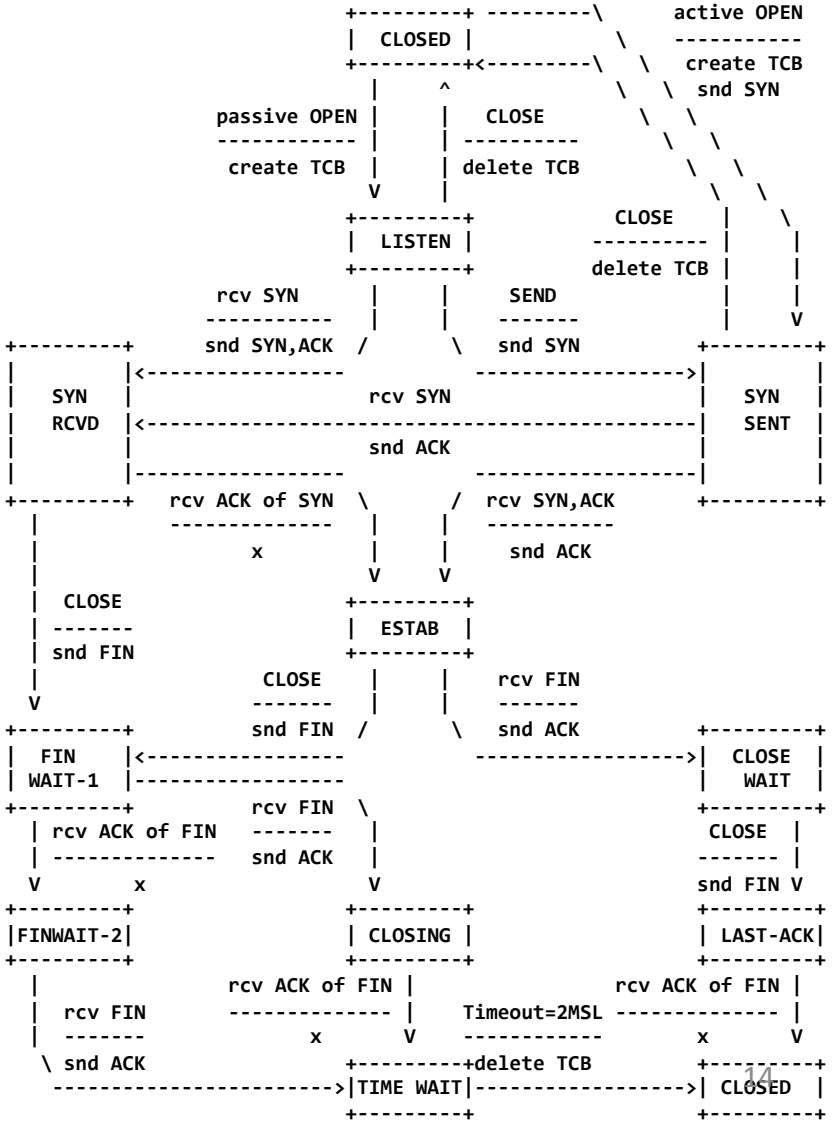
- "*TCP implementations will follow a general principle of robustness: be conservative in what you do, be liberal in what you accept from others*." RFC 793
  - Followed in most of the good Internet protocols
  - Although consider the trade-off between robustness and enforcing upgrade/evolution

TCP Header


0 1 2 3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
Source Port
Destination Port
Sequence Number
Acknowledgment Number
Data Offset
Rsvd
C E U A P R S F
N W C R C S S Y I
S R E G K H T N N
Window
Checksum
Urgent Pointer
Options
Padding
data
...

Connections: State Machine

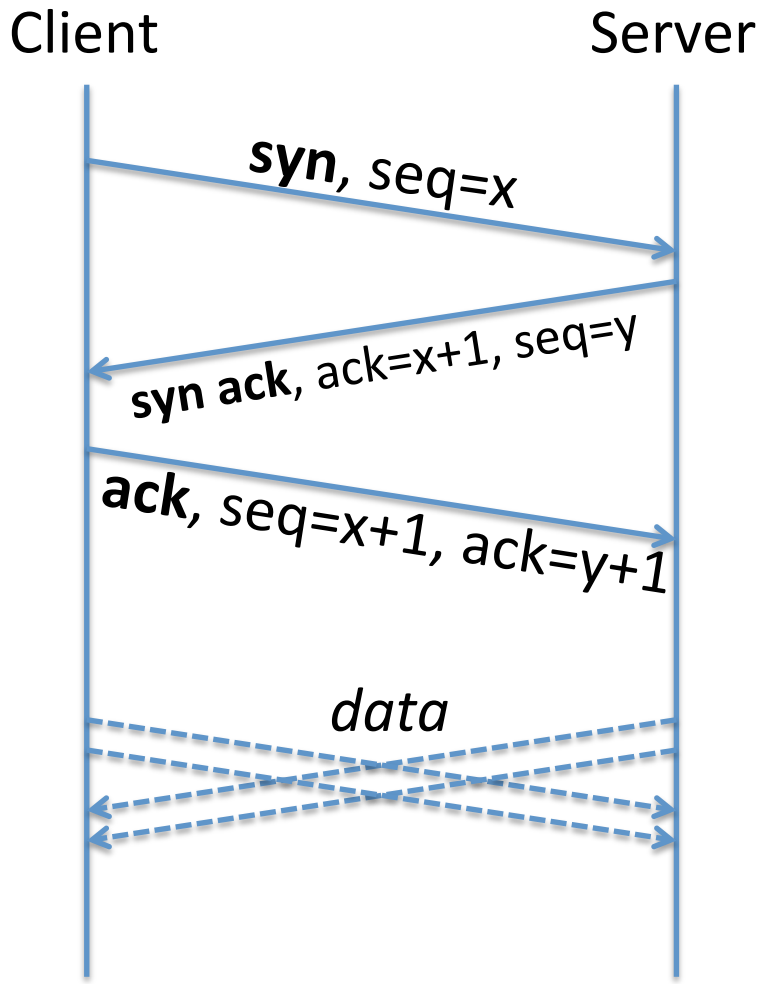
- Quite complex!
  - Why?
- Transitions via:
  - (Sockets) API call
  - Packet Rx/Tx
  - Timer expiry (timeout)
- Typically three phases:
  - Connecting
  - Established
  - Closing
- ...but corner-cases exist!
  - E.g., simultaneous open/close



Connections: Setup

Network is asynchronous

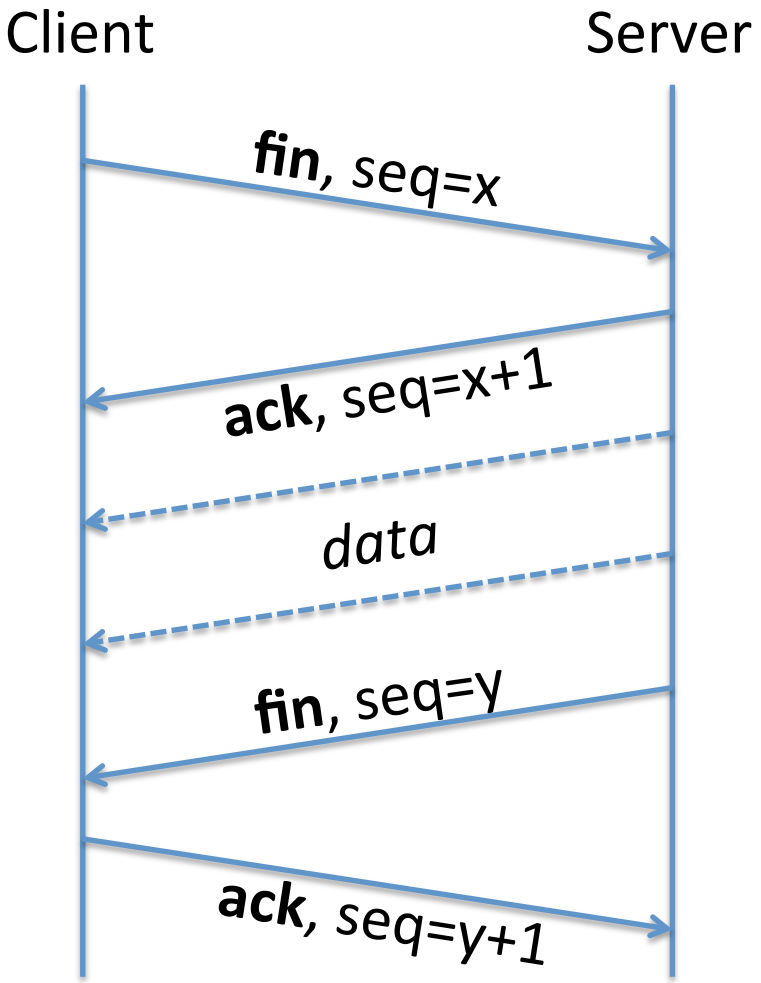
- Both parties must agree
  - Minimum required
  - Can send data with final ACK, but many don't
- Bidirectional connection
  - Data can flow both ways



Connections: Teardown

- State must be deleted
  - At both ends!
- Instantly
  - Reset, RST
- Politely
  - FIN/FIN-ACK
  - Both directions

➔ Messages can overlap



Priority and Precedence

- TCP presents an *ordered bytestream*
  - The stack can hold on to data before transmitting
  - But not all data in a stream is equal
- Push, PSH
  - Process says to stack "Tx queued data *now*!"
  - In practice "promptly"
  - Why expose in the packet header?
- Urgent, URG
  - Tx indicates "this future data is urgent!"
  - Rx can hurry processing to reach the indicated point

Contents

- Overview
- User Datagram Protocol
- Transmission Control Protocol
  - Header format
  - State machine: setup, teardown
  - Priority, precedence
  - Reliability, timeouts
  - Host considerations
  - Flow control

Reliability

- Basically: retransmit when data lost
- Three problems:
  - How to detect loss?
  - How to avoid loss due to host congestion?
  - How to avoid loss due to network congestion?
- Detect loss via (bytestream) sequence numbers
  - Sender sends seqno
  - Receiver sends ackno
  - Sender can detect how far receiver is through stream

Timeouts

- How long should we wait to retransmit? RTO(RTT)
- RTT estimation: exponential average
  - $RTT_{sample} = time_{ackRx} - time_{packetTx}$
  - $RTT_{estimated} = \alpha.RTT_{estimated} + (1-\alpha).RTT_{sample}$
  - $\alpha = 7/8$
- RTO computation: standard deviation rather expensive
  - Jacobsen/Karels: mean deviation
    - $difference = RTT_{sample} - RTT_{estimated}$
    - $devation \mathrel{+}= \delta.(|difference| - deviation)$
    - $RTO = \mu.RTT_{estimated} + \phi.deviation$
    - $\delta = ¼,\ \mu = 1,\ \phi = 4$
  - Karn/Partridge: retransmits give ambiguous measurements
    - Avoid using retransmitted segments for RTT_{sample}
    - Exponential backoff of RTO: each expiry, $RTO \mathrel{*}= 2$ (max. 60s)

Flow Control

- Rx host needs to put data somewhere
  - It's likely to be busy in an OS scheduling sense
  - Buffer management required for connection
- Rx advertised window
  - Advertise unused buffer space in ACK
- Enables Tx to avoid sending so much that Rx will be forced to drop it
  - ...wasting network (and Tx host) resource
  - Avoiding *network* induced loss is *congestion control*

Connections: Hosts

- Transmission Control Block, TCB
- Contains state used to manage the connection
  - In reality, `struct` is 134 lines in OpenBSD!
- Adds connection state, timers, buffer queue, congestion management details, retransmit details, cached templates, window scaling, path-MTU discovery, &c.

```
Send Sequence Variables
 SND.UNA - send unacknowledged
 SND.NXT - send next
 SND.WND - send window
 SND.UP  - send urgent pointer
 SND.WL1 - segment sequence number used for
           last window update
 SND.WL2 - segment acknowledgment number used
           for last window update
 ISS     - initial send sequence number

Receive Sequence Variables
 RCV.NXT - receive next
 RCV.WND - receive window
 RCV.UP  - receive urgent pointer
 IRS     - initial receive sequence number
```

Summary

- Transport gives per-process demultiplexing
  - UDP: connectionless, unreliable; *vs*.
  - TCP: connection-oriented, sequenced, reliable
- Providing reliability in an asynchronous network is *hard*!
  - If you want it to be a generally usable feature
  - In a wide variety of situations
  - *Congestion control next...*

Quiz (1)

1. What do the source and destination ports identify? Why have both in the packet?
2. Why does the pseudo-header include the IP source and destination addresses?
3. Why do both UDP and TCP introduce padding? Where do they use it?
4. How does a simultaneous open occur, and why does it have to be handled in the state machine?
5. If data is sent with the final ACK in the 3-way handshake, what changes in the picture on sl. 15?

Quiz (2)

6. Why would you ever close a TCP connection rather than resetting it?
7. What is the difference between URG and PSH?
8. TCP is reliable through retransmission – suggest an alternative mechanism?
9. Why do we need to compute the RTO? Why base it on RTT?
10. What is the difference between *flow control* and *congestion control*?