University of Glasgow

TCP

Networked Systems (H)
Lecture 13

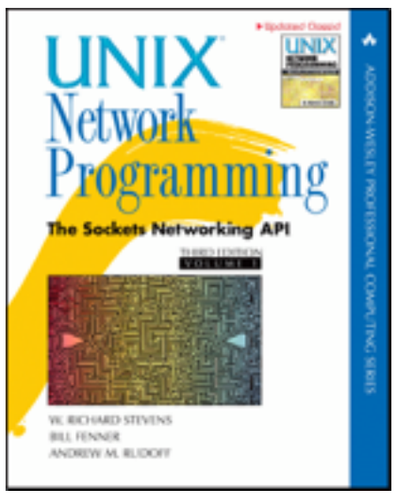
Lecture Outline

- Berkeley Sockets API
- The TCP protocol

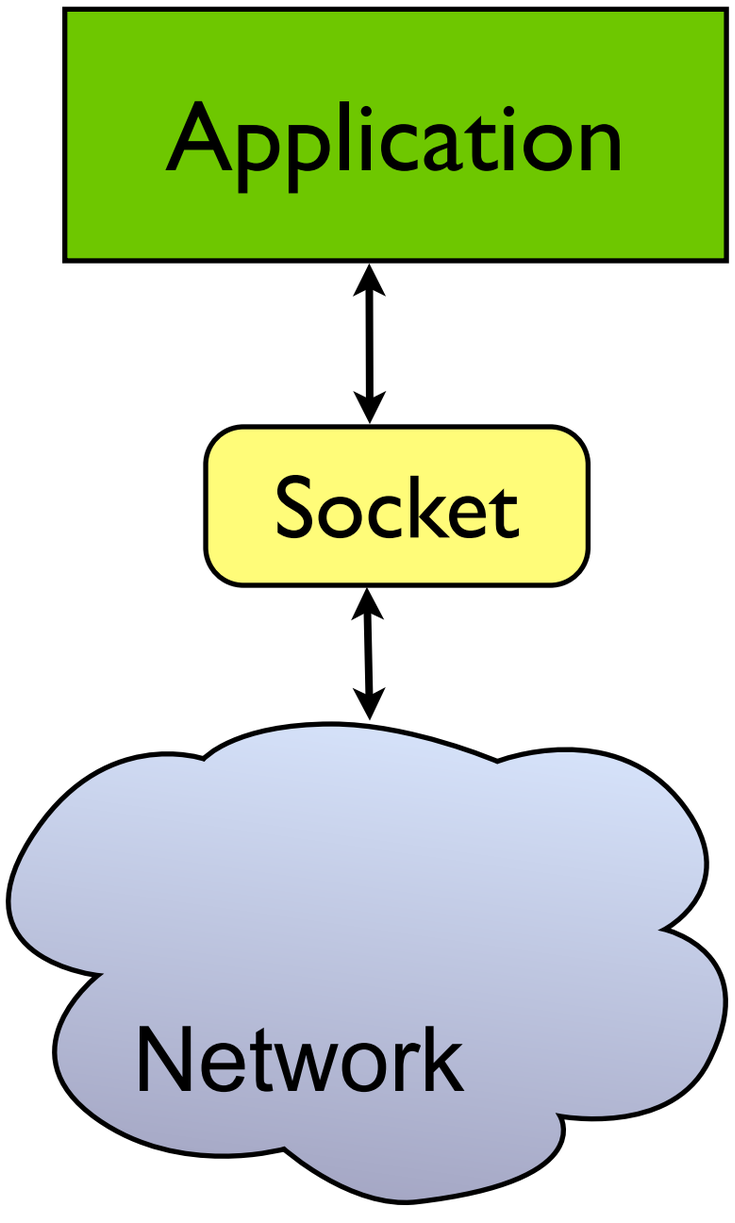
The Berkeley Sockets API

- Widely used low-level C networking API
- First introduced in 4.BSD Unix
  - Now available on most platforms: Linux, MacOS X, Windows, FreeBSD, Solaris, etc.
  - Largely compatible cross-platform

- Recommended reading:
  - Stevens, Fenner, and Rudoff, “Unix Network Programming volume 1: The Sockets Networking API”, 3rd Edition, Addison-Wesley, 2003.

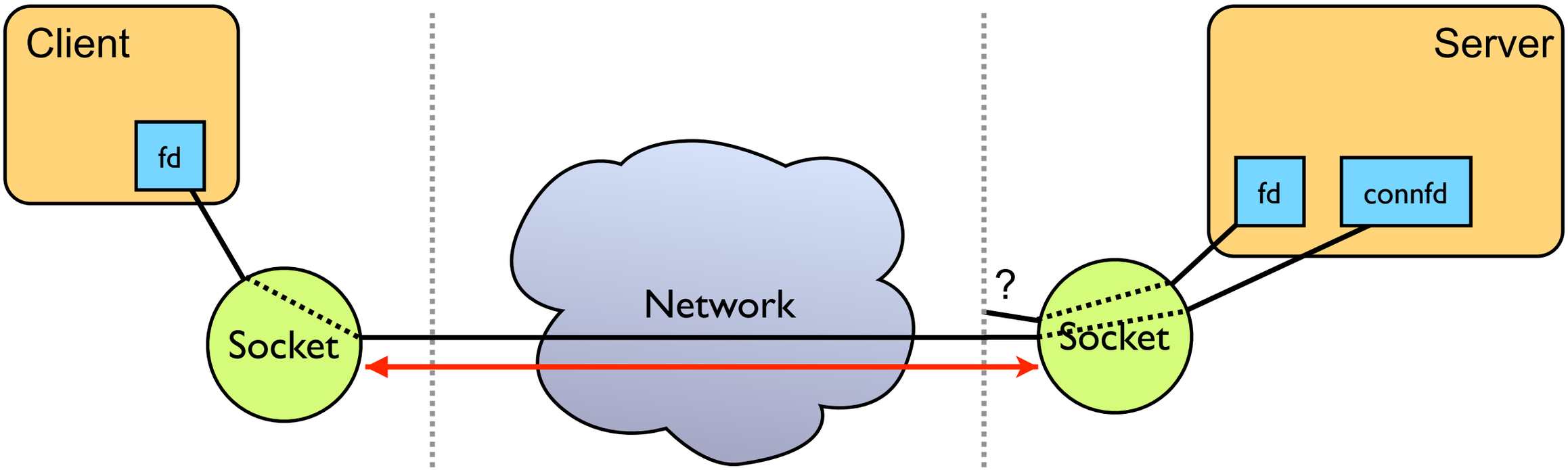


Concepts



- Sockets provide a standard interface between network and application
- Two types of socket:
  - Stream – provides a virtual circuit service
  - Datagram – delivers individual packets
- Independent of network type:
  - Commonly used with TCP/IP and UDP/IP, but not specific to the Internet protocols
  - Will discuss TCP/IP today, UDP next week

TCP Sockets




```
int fd = socket(...)

connect(fd, ..., ...)

send(fd, data, datalen, flags)

recv(fd, buffer, buflen, flags)

close(fd)
```

```
int fd = socket(...)
bind(fd, ..., ...)
listen(fd, ...)
connfd = accept(fd, ...)

recv(connfd, buffer, buflen, flags)

send(connfd, data, datalen, flags)

close(connfd)
```

What services do TCP sockets provide?

- TCP provides five key features:
  - Service differentiation
  - Connection-oriented
  - Point-to-point
  - Reliable, in-order, delivery of a byte stream
  - Congestion control

- These are provided by the operating system, via the sockets API

Client-server or peer-to-peer?

- Sockets initially unbound, and can either accept or make a connection
- Most commonly used in a client-server fashion:
  - One host makes the socket `listen()` for, and `accept()`, connections on a well-known port, making it into a server
    - The port is a 16-bit number used to distinguish servers
    - E.g. web server listens on port 80, email server on port 25
  - The other host makes the socket `connect()` to that port on the server
  - Once connection is established, either side can `send()` data into the connection, where it becomes available for the other side to `recv()`
- Simultaneous connections are possible, using TCP in a peer-to-peer manner

Role of the TCP Port Number

Port Range		Name	Intended use
0	1023	Well-known (system) ports	Trusted operating system services
1024	49151	Registered (user) ports	User applications and services
49152	65535	Dynamic (ephemeral) ports	Private use, peer-to-peer applications, source ports for TCP client connections

RFC 6335

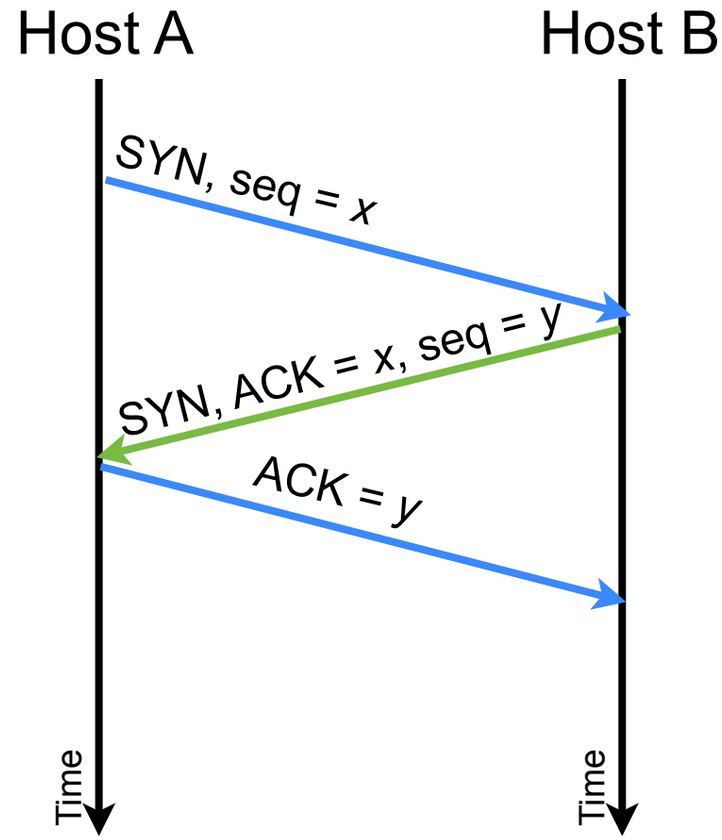
- Servers must listen on a known port; IANA maintains a registry
- Distinction between system and user ports ill-advised – security problems resulted
- Insufficient port space available (>75% of ports are registered)

- TCP clients traditionally connect from a randomly chosen port in the ephemeral range
  - The port must be chosen randomly, to prevent spoofing attacks
  - Many systems use the entire port range for source ports, to increase the amount of randomness available

http://www.iana.org/assignments/port-numbers

TCP Connection Setup

- Connections use 3-way handshake
  - The SYN and ACK flags in the TCP header signal connection progress
  - Initial packet has SYN bit set, includes randomly chosen initial sequence number
  - Reply also has SYN bit set and randomly chosen sequence number, acknowledges initial packet
  - Handshake completed by acknowledgement of second packet
  - Happens during the `connect()`/`accept()` calls
- Combination ensures robustness
  - Randomly chosen initial sequence numbers give robustness to delayed packets or restarted hosts
  - Acknowledgements ensure reliability



Similar handshake ends connection, with FIN bits signalling the teardown

Reading and Writing Data

```
#define BUFLEN 1500
...
ssize_t i;
ssize_t rcount;
char    buf[BUFLEN];
...
rcount = recv(fd, buf, BUFLEN, 0);
if (rcount == -1) {
    // Error has occurred
    ...
}
...
for (i = 0; i < rcount; i++) {
    printf("%c", buf[i]);
}
```

- The `recv()` call reads *up to* BUFLEN bytes of data from connection – blocks until data available
- Returns actual number of bytes read, or –1 on error
- Data is *not* null terminated

```
char data[] = "Hello, world!";
int  datalen = strlen(data);
...
if (send(fd, data, datalen, 0) == -1) {
    // Error has occurred
    ...
}
...
```

- The `send()` call sends data via a socket; blocks until all data can be written
- Returns actual number of bytes written, or –1 on error

Record Boundaries in TCP Connections

- If the data in a `send()` is bigger than the data link layer MTU, TCP will send the data as fragments
- Similarly, multiple small `send()` requests may be aggregated into a single TCP packet

- Implication: the data returned by a `recv()` doesn't necessarily match that sent in a single `send()`
  - There often appears to be a correspondence, but this *is not* guaranteed (it may work in the lab, but not when you use it over a different link)

Application Level Framing

Data may arrive in arbitrary sized chunks; must parse and understand the data, no matter where it is split by the network – it's a byte stream (colours indicate one possible split of the data into chunks)

```
HTTP/1.1 200 OK
Date: Mon, 19 Jan 2009 22:25:40 GMT
Server: Apache/2.0.46 (Scientific Linux)
Last-Modified: Mon, 17 Nov 2003 08:06:50 GMT
ETag: "57c0cd-e3e-17901a80"
Accept-Ranges: bytes
Content-Length: 3646
Connection: close
Content-Type: text/html; charset=UTF-8

<HTML>
<HEAD>
<TITLE>Computing Science, University of Glasgow </TITLE>
...
</BODY>
</HTML>
```

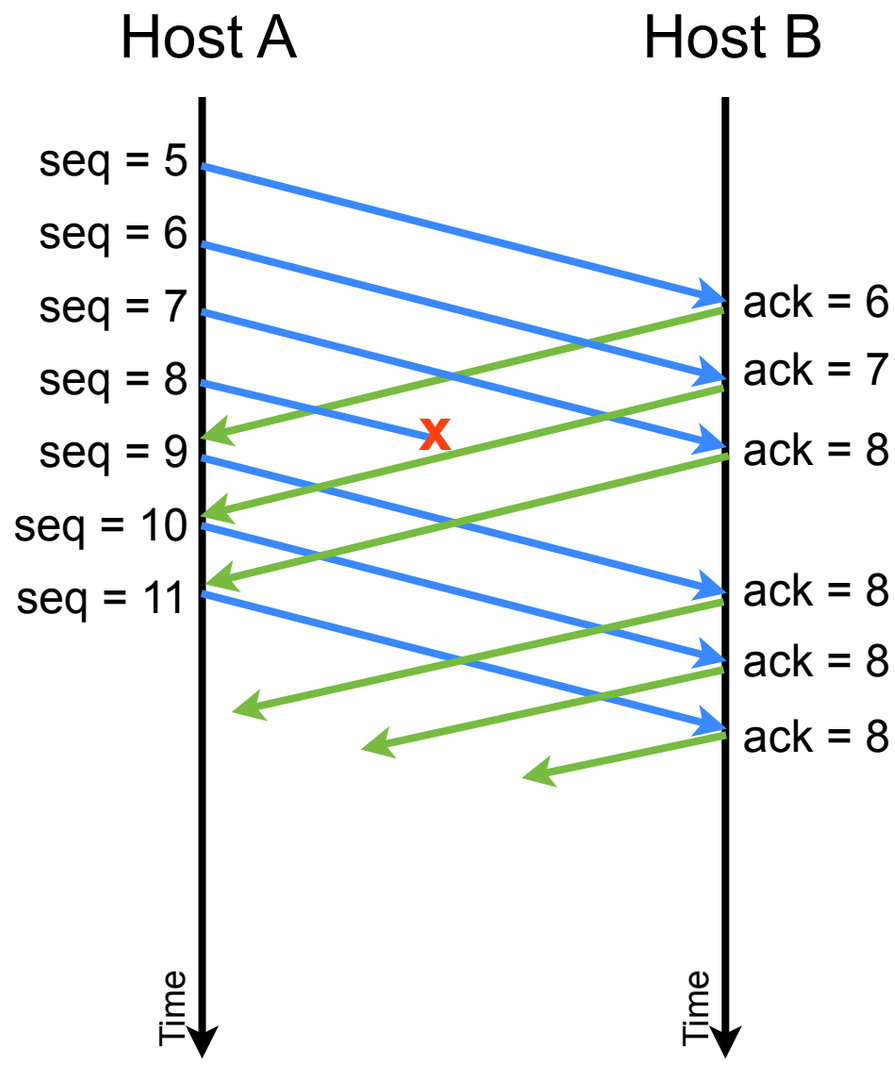
Example: HTTP response

Known marker (blank line) signals end of headers

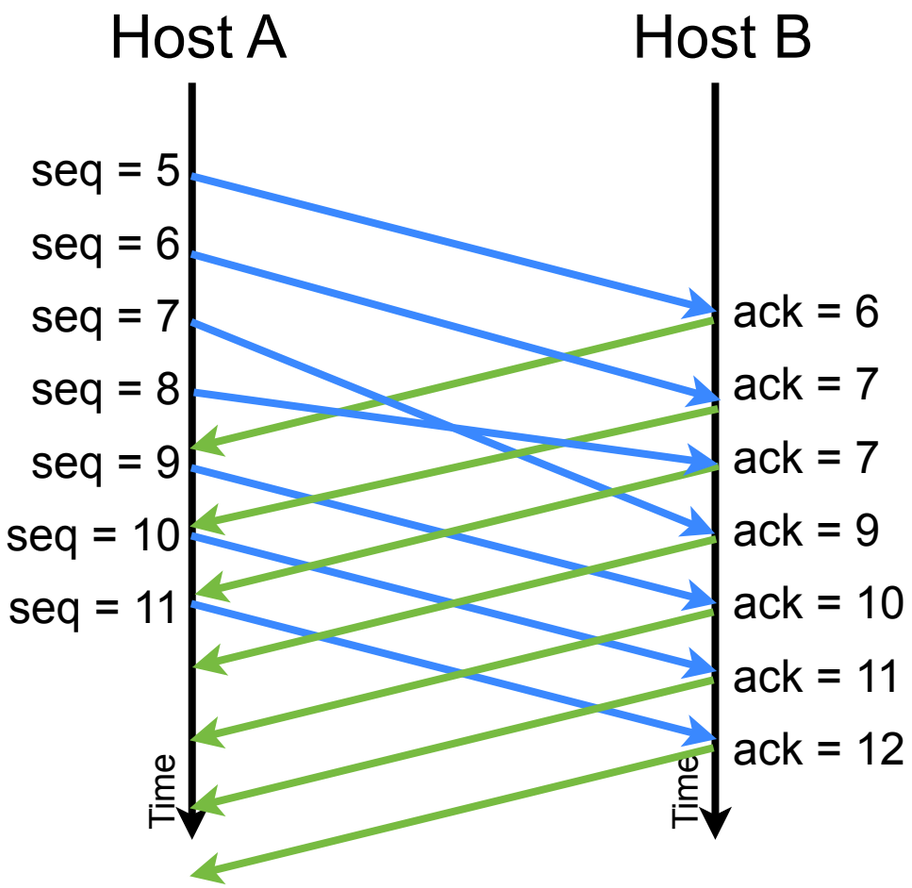
Size of payload indicated in the headers

TCP Reliability

- TCP connections are reliable
  - Application data gathered into packets
  - Each packet has a sequence number and an acknowledgement number
    - Sequence number counts how many bytes are sent (this example is unrealistic, since it shows one byte being sent per packet)
  - Acknowledgement number specifies next byte expected to be received
    - Cumulative positive acknowledgement
    - Only acknowledge contiguous data packets (sliding window protocol, so several data packets in flight)
    - Duplicated acknowledgements imply loss
  - TCP layer retransmits lost packets – this is invisible to the application



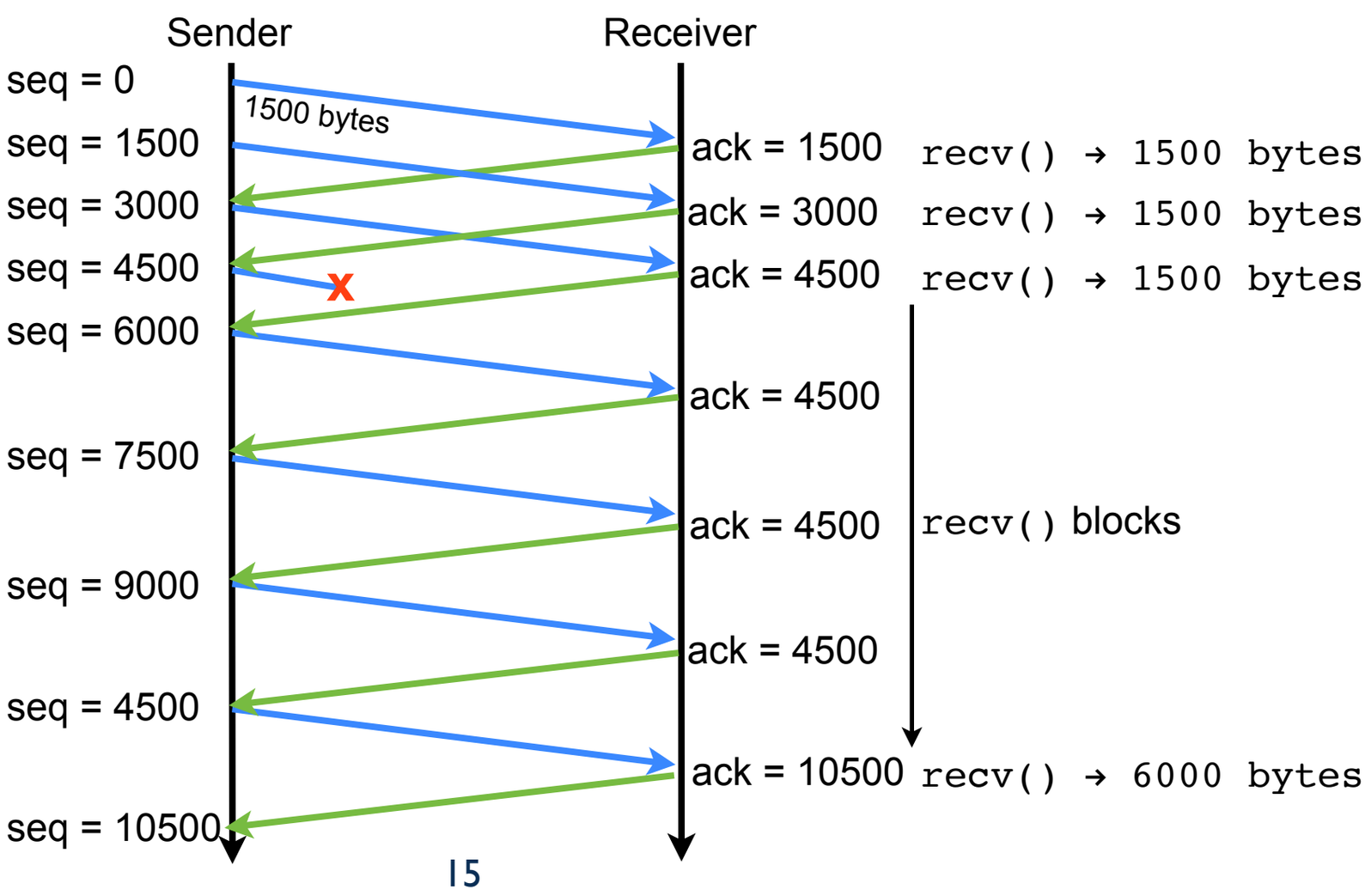
TCP Reliability: How is Loss Detected



- Packet reordering also causes duplicate ACKs
  - Gives appearance of loss, when the data was merely delayed
- TCP uses *triple duplicate ACK* to indicate loss
  - Four identical ACKs in a row
  - Slightly delays response to loss, but makes TCP more robust to reordering

Head of Line Blocking in TCP

- Data delivered in order, even after loss occurs
  - TCP will retransmit the missing data, transparently to the application
  - A `recv()` for the missing data will block until it arrives; TCP delivers all data in-order



Summary

- The Berkeley Sockets API
- Services provided by TCP
  - Reliability
  - Unframed byte stream
  - Head of line blocking
- Next lecture: congestion control