CIS551: Computer and Network Security

Jonathan M. Smith

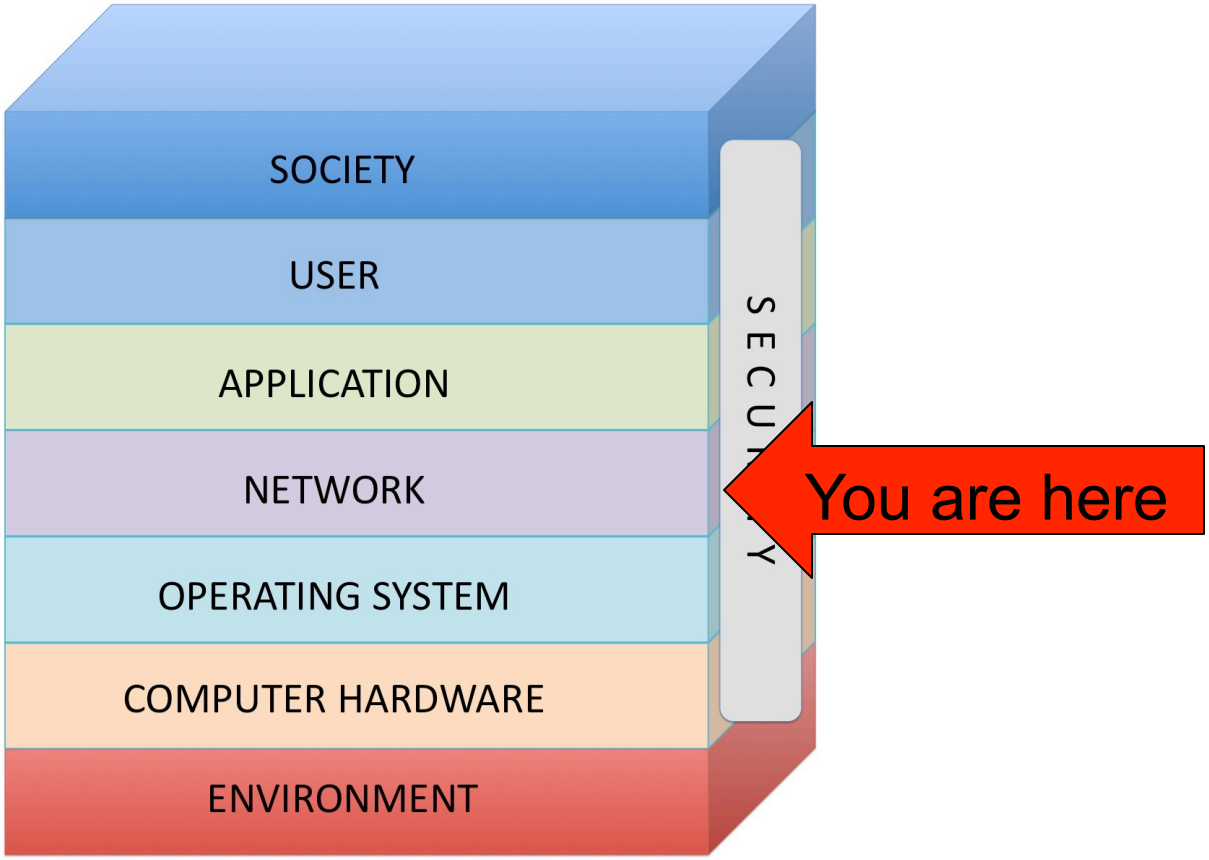
jms@cis.upenn.edu

02/10/2014

CIS551 Topics

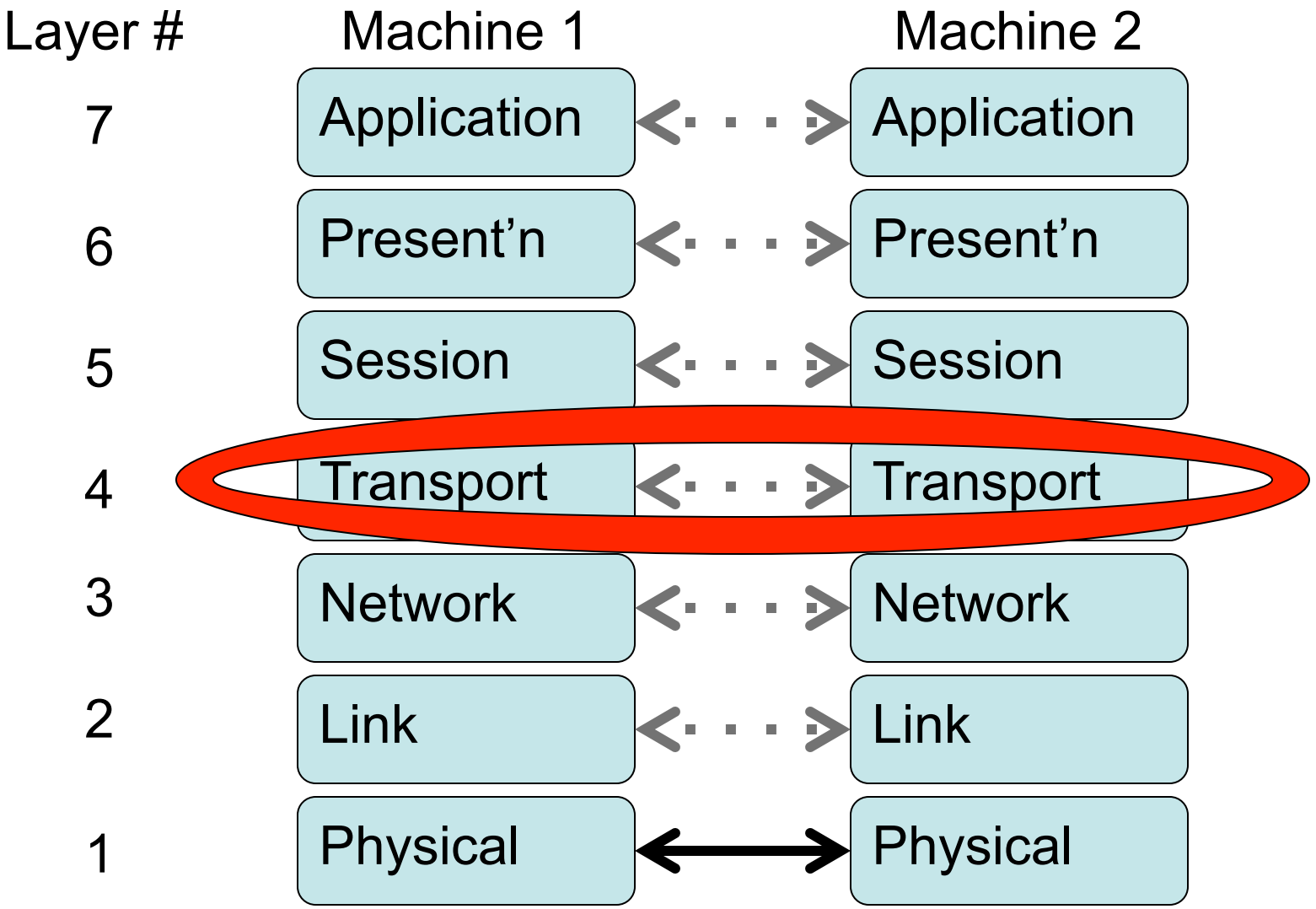
- Computer Security
  - Software/Languages, Computer Arch.
  - Access Control, Operating Systems
  - Threats: Vulnerabilities, Viruses
- Computer Networks
  - Physical layers, Internet, WWW, Applications
  - Cryptography in several forms
  - Threats: Confidentiality, Integrity, Availability
- Systems Viewpoint
  - Users, social engineering, insider threats

Sincoskie NIS model



W.D. Sincoskie, *et al.* “Layer Dissonance and Closure in Networked Information Security” (white paper)

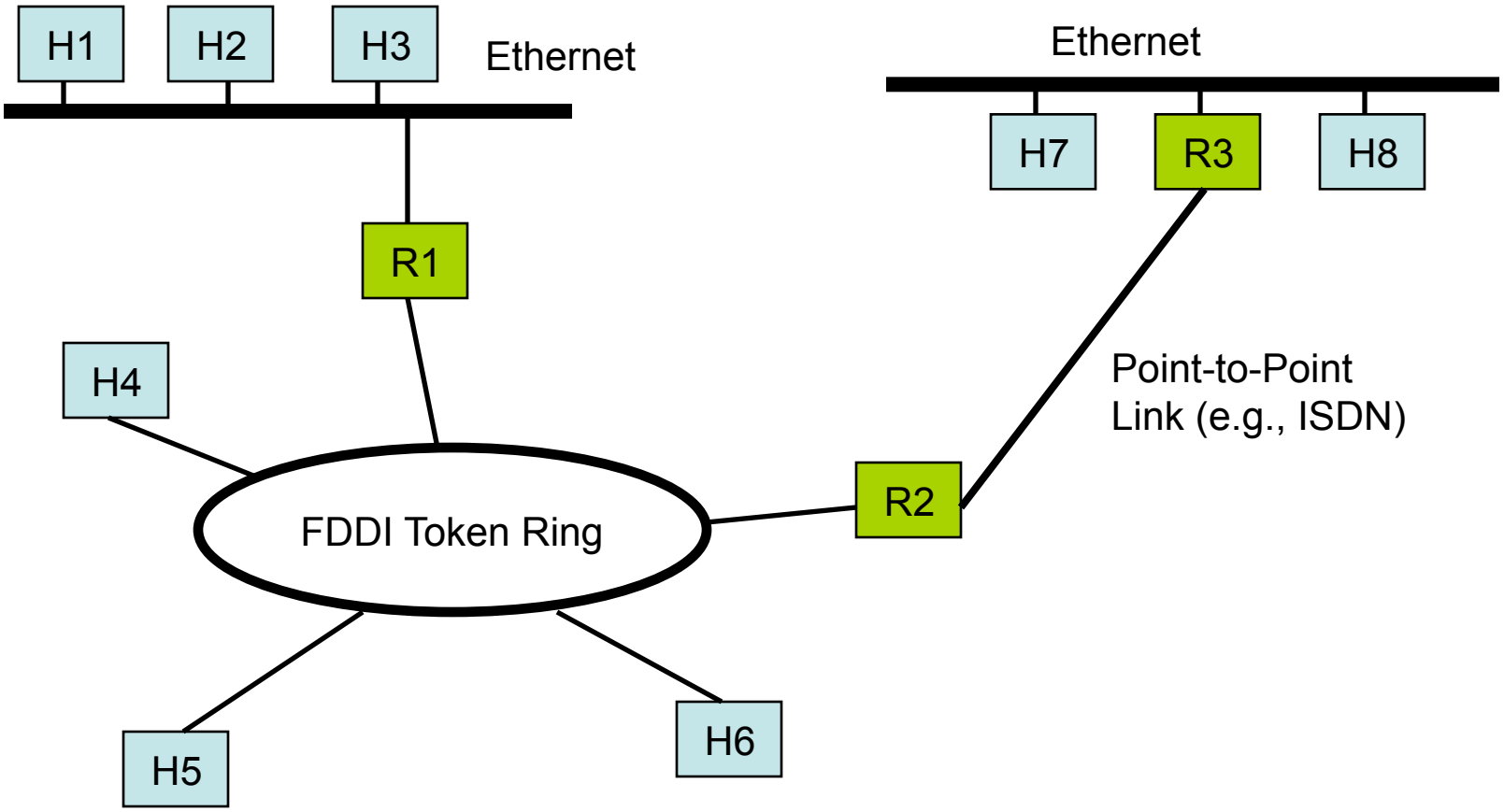
7-layer OSI network model



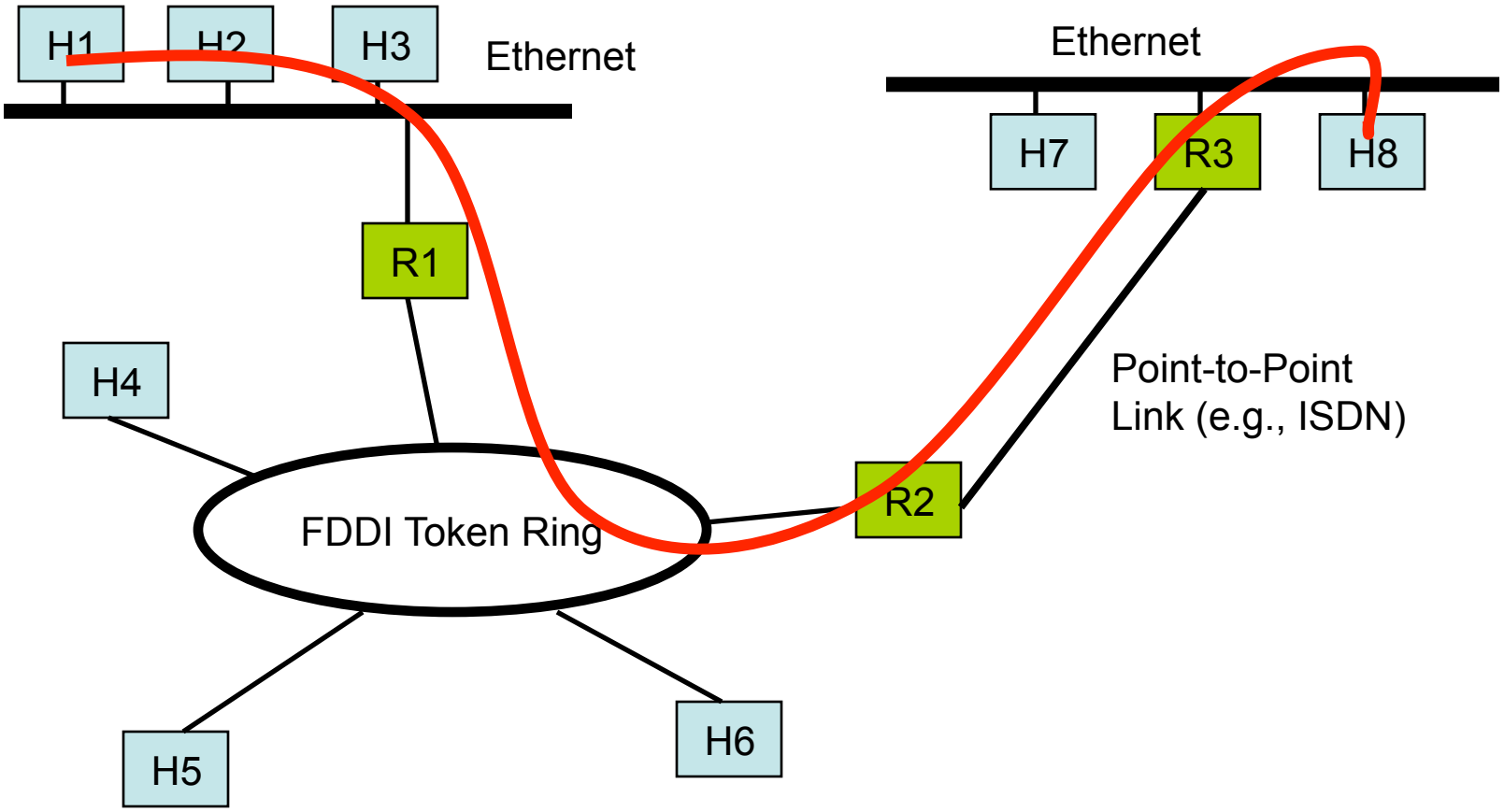
Applications vs. Networks

Application Requirements	Network Characteristics
Reliable, Ordered, Single-Copy Message Delivery	Drops, Duplicates and Reorders Messages
Arbitrarily large messages	Finite message size
Flow Control by Receiver	Arbitrary Delay
Supports multiple applications per-host	…

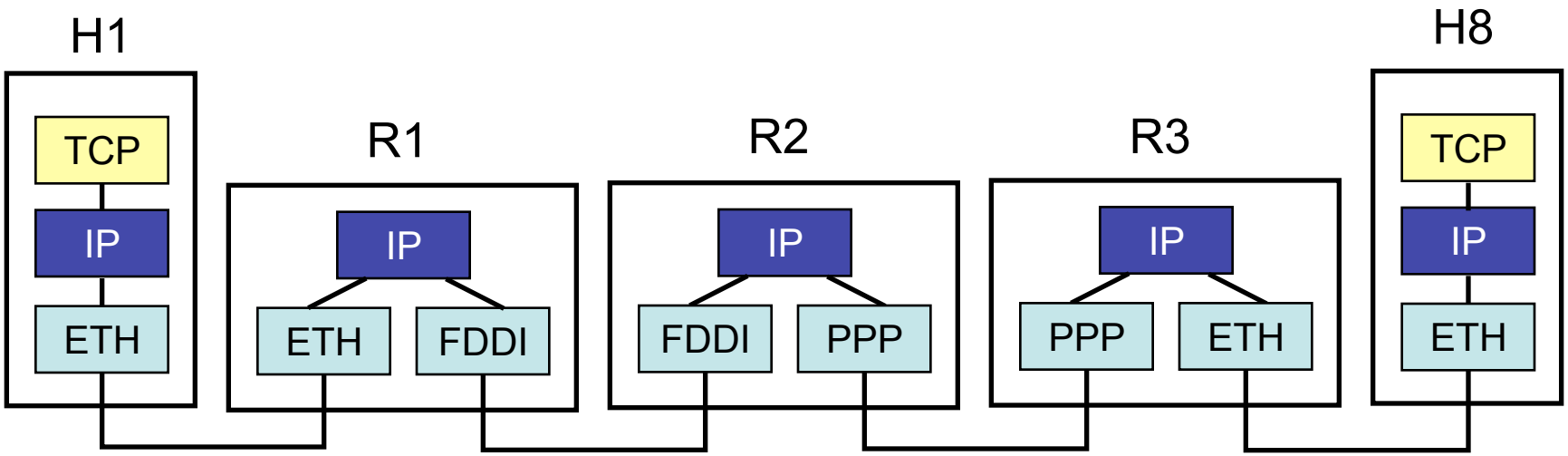
Internetworks (diagram from Peterson & Davie)



Internetworks (diagram from Peterson & Davie)



Transport is *host-host* (diagram: Peterson & Davie)

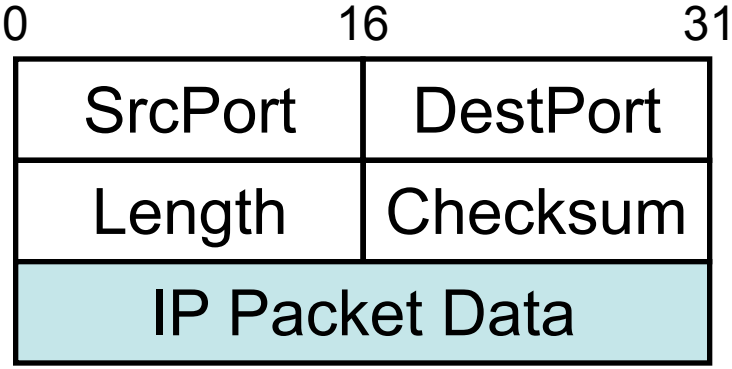


Transport protocols are *end-to-end*

Transport: “end-to-end” services

- User Datagram Protocol
  - Best-effort packet service
  - Thin overlay on basic IP service
  - Used by many “real-time” services
- Transmission Control Protocol
  - “virtual circuit” protocol
  - Reliable bytestream
- Start with UDP, but focus on TCP/IP

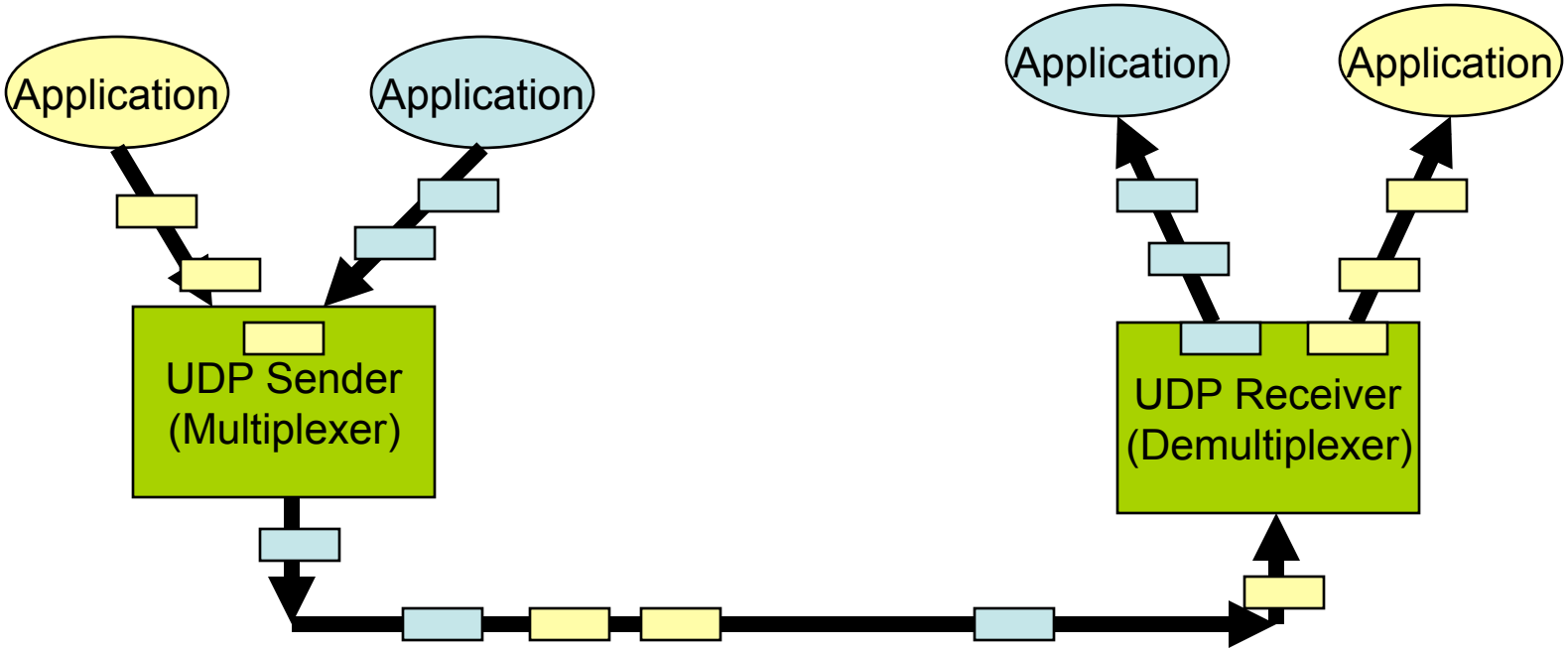
User Datagram Protocol (UDP)



- Minimalist transport-layer protocol
  - Exposes IP packet functionality to application level
  - *Ports* identify sending/receiving process
    - De-multiplexing information
    - (port, host) pair identifies a network process

UDP End-to-End Model

- Multiplexing/Demultiplexing with Port number



Using Ports

- Client contacts Server at a *well-known port*
  - SMTP: port 25
  - DNS: port 53
  - POP3: port 110
  - Unix talk : port 517
  - In Unix, ports are listed in /etc/services
- Sometimes Client and Server agree on a different port for subsequent communication
- Ports are an abstraction
  - Implemented differently on different OS's
  - Typically a message queue

Virtual Circuit model

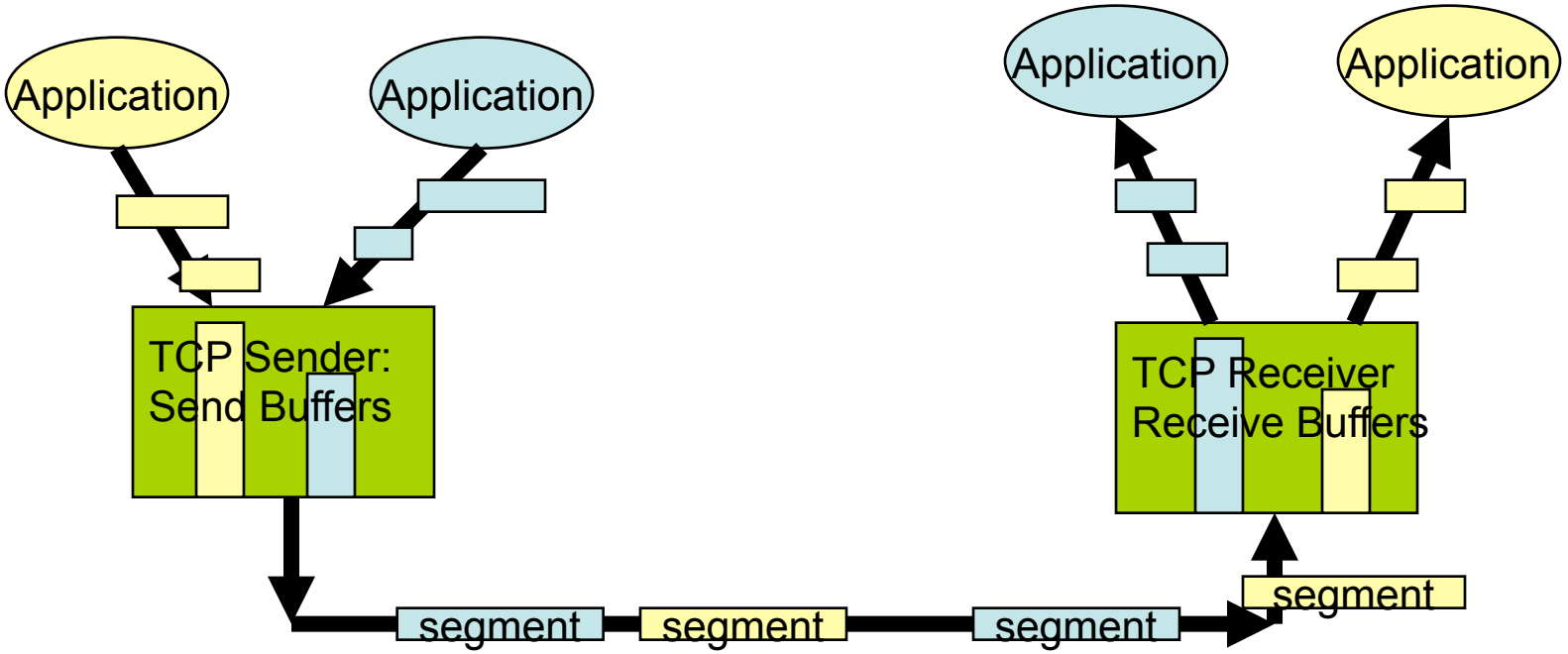
- Reliable, ordered bytestream
  - Problem: network (IP) is best-effort, so no guarantees of delivery, ordering, path, *etc.*
- Transmission Control Protocol:
  - Sequence numbers (for ordering)
  - Acknowledgements/retry (for reliability)
  - Controlled delays given up for reliability

Transmission Control Protocol (TCP)

- Most widely used protocol for reliable byte streams
  - Reliable, in-order delivery of stream of bytes
  - Full duplex: pair of streams, one in each direction
  - Flow and congestion control mechanisms
  - Like UDP, supports ports
- Built *on top of* IP (hence “TCP/IP” usage)

TCP End-to-End Model

- *Buffering* trades delays for losses/errors



Applications use sockets

- Descriptors that provide access to the protocol, e.g.

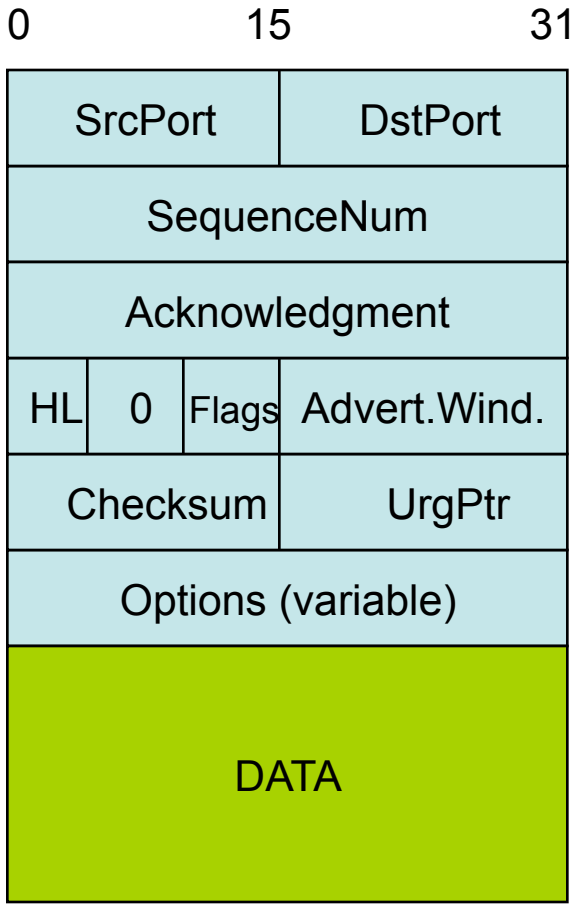
```
sockfd = socket(AF_INET, SOCK_STREAM, 0);
```

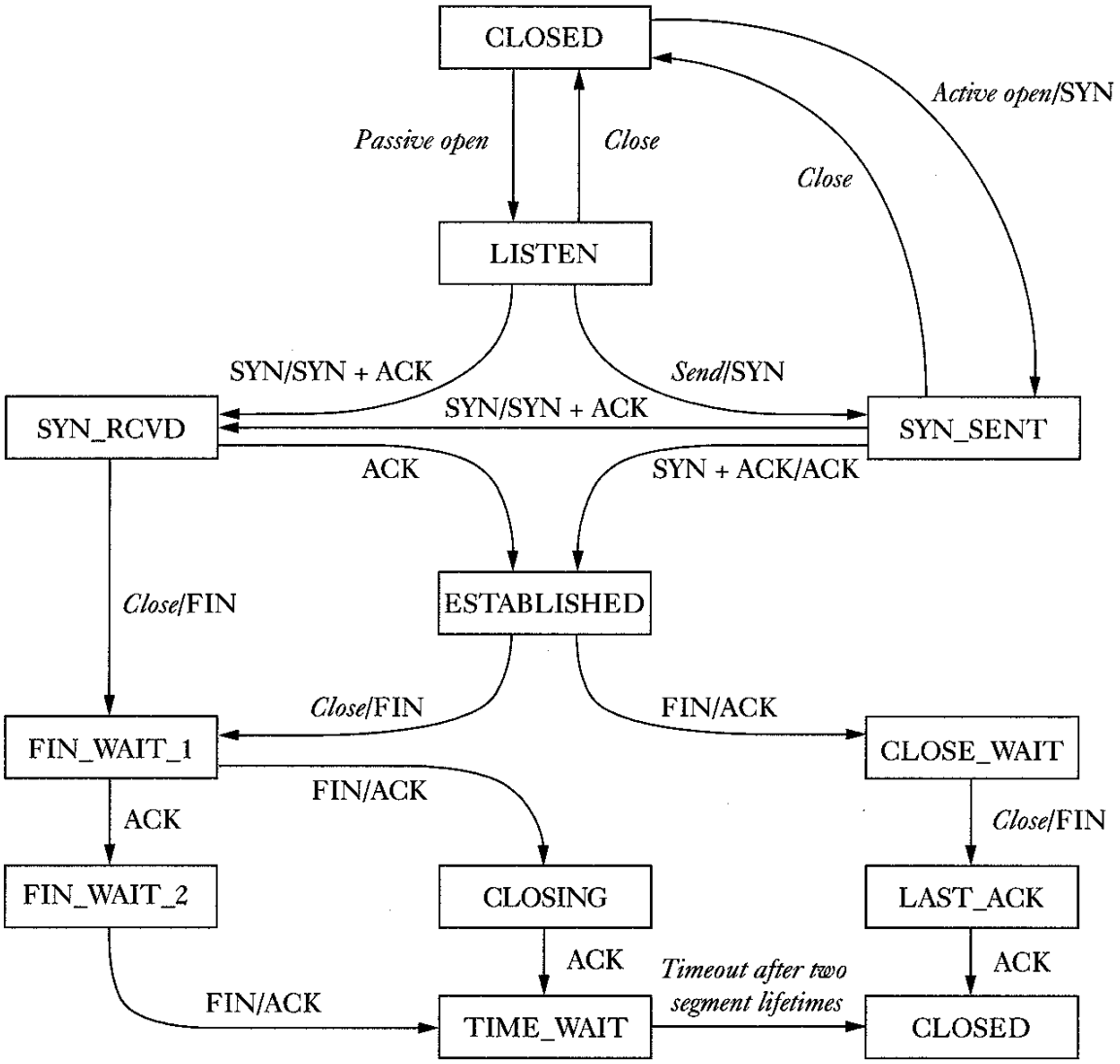
- Read/write access
- Resembles UNIX `pipe` IPC mechanism
- Port number – rendezvous for app – e.g.

```
servaddr.sin_family = AF_INET;
servaddr.sin_port = htons( CIS551_PORT );
connect(sockfd, (SA *) &servaddr, sizeof(servaddr))
```

TCP Packet Format

- Flags
  - SYN
  - FIN
  - RESET
  - PUSH
  - URG
  - ACK
- Fields




CLOSED
Active open/SYN
Passive open
Close
Close
LISTEN
SYN/SYN + ACK
Send/SYN
SYN/SYN + ACK
SYN_RCVD
SYN_SENT
ACK
SYN + ACK/ACK
Close/FIN
ESTABLISHED
Close/FIN
FIN/ACK
FIN_WAIT_1
CLOSE_WAIT
FIN/ACK
ACK
Close/FIN
FIN_WAIT_2
CLOSING
LAST_ACK
ACK
ACK
Timeout after two segment lifetimes
FIN/ACK
TIME_WAIT
CLOSED

TCP Receiver

- Maintains a buffer from which application reads
- Advertises < buffer size as the window for sliding window
- Responds with Acknowledge and AdvertisedWindow on each send; updates byte counts when data O.K.
- Application blocked until read() O.K.

TCP Sender

- Maintains a buffer; sending application is blocked until room in buffer for `write`
- Hold data until *acknowledged* by receiver (ACK) *as successfully received*
- Implement window expansion and contraction; note difference between *flow* and *congestion* control

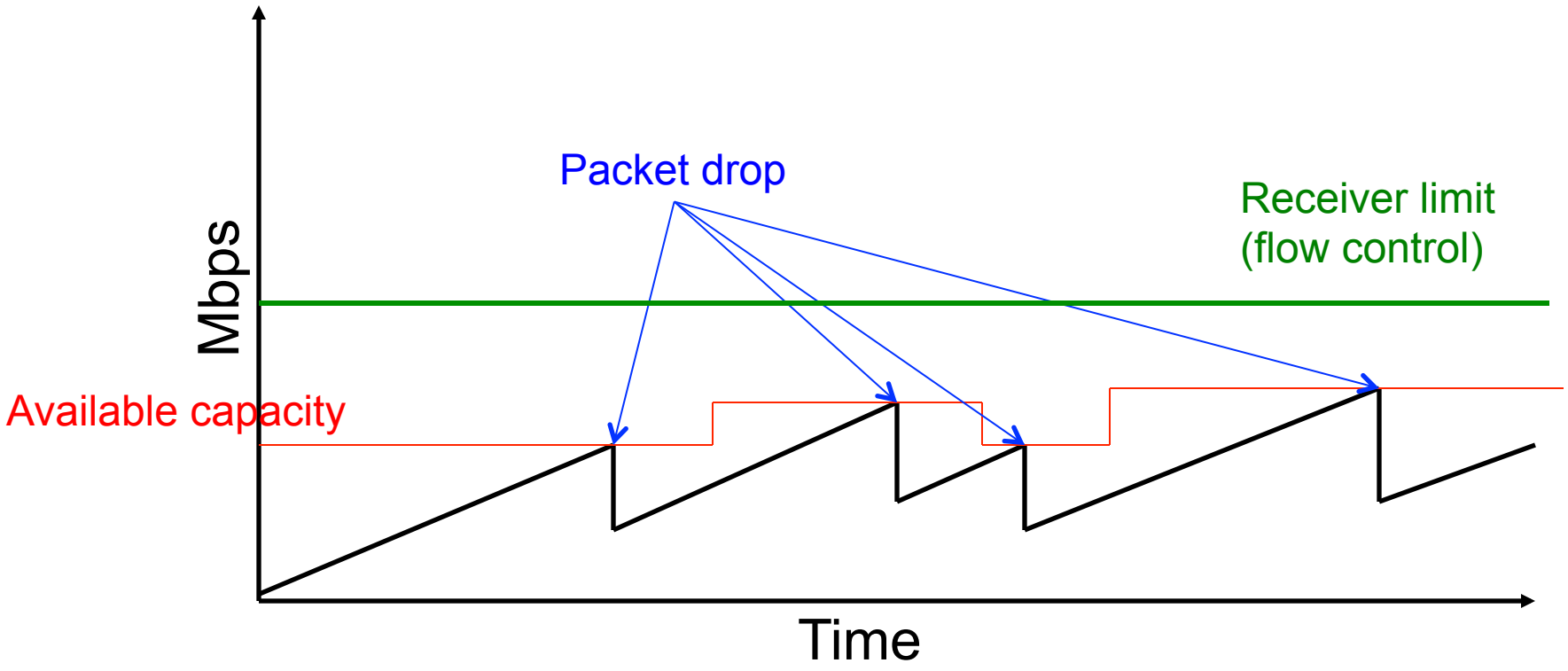
TCP Flow & Congestion Control

- Flow vs. Congestion Control
  - *Flow control* protects the recipient from being overwhelmed
  - *Congestion control* protects the network from being overwhelmed
- TCP Congestion Control
  - Additive Increase / Multiplicative Decrease
  - Slow Start
  - Fast Retransmit and Fast Recovery

Increase and Decrease

- A value (`CongestionWindow`) is used to control the number of unacknowledged transmissions.
- This value is increased linearly until timeouts for ACKs are missed.
- When timeouts occur, `CongestionWindow` is decreased by half to reduce the pressure on the network quickly.
- Hence “additive increase / multiplicative decrease”.

TCP “sawtooth” pattern



Slow Start

- Sending the entire window immediately could cause a traffic jam in the network
- Begin “slowly” by setting the congestion window to one packet.
- When acknowledgements arrive, double the congestion window.
- Continue until ACKs do not arrive (*i.e.*, congestion), or flow control dominates

Network Vulnerabilities

- Anonymity
  - Attacker is remote, origin can be disguised
  - Authentication
- Many points of attack
  - Attacker only needs to find weakest link
  - Attacker can mount attacks from many machines
- Sharing
  - Many, many users sharing resources
- Complexity
  - Distributed systems are large and heterogeneous
- Unknown perimeter
- Unknown attack paths

Syn Flood Attack

- TCP's 3-way handshake:
  - SYN --- SYN+ACK --- ACK
- Receiver must maintain a queue of partially open TCP connections
  - Called SYN_RECV connections
  - Finite resource (often small: e.g. 20 entries)
  - Timeouts for queue entries are about 1 minute.
- Attacker
  - Floods a machine with SYN requests
  - Never ACKs them
  - Spoofs the sending address