CSCD 330
Network Programming

Lecture 9

Transport Layer
Spring 2018

Reading: Begin Chapter 3

Outline

- Overview of Transport Layer
- Multiplexing / Demultiplexing
- UDP
  - Functions
- Reliable Transport
- State Diagram of Reliable Transport
  - Building it from basics

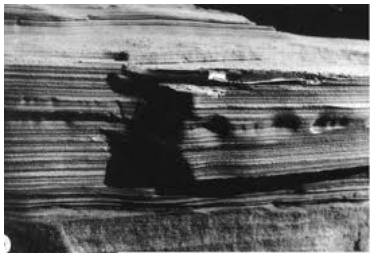
Introduction to Transport

- **Layers 1, 2 and 3**

  Physical, Data Link and Network

  - Concerned with actual packaging, addressing, routing and delivery of data
  - **Physical layer** handles bits
  - **Data link layer** deals with local networks
  - **Network layer** handles routing between networks
- **Transport layer** in contrast, does not concern itself with these “nuts and bolts” matters
  - It relies on lower layers to handle process of moving data between devices

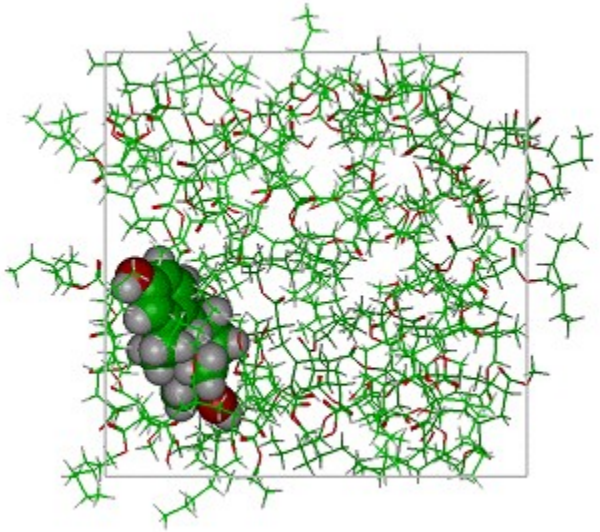
Transport Layer Purpose



- **Transport Layer's Job**
- Provides for communication between application processes on different computers
  - Computers have many applications all trying to send and receive data
    - Example: Web Browser open
      Bit Torrent downloading in the background and
      WoW gaming session open
- Transport layer enables these applications to send and receive data
  - All applications must use same lower-layer protocol implementation

Transport Layer Purpose

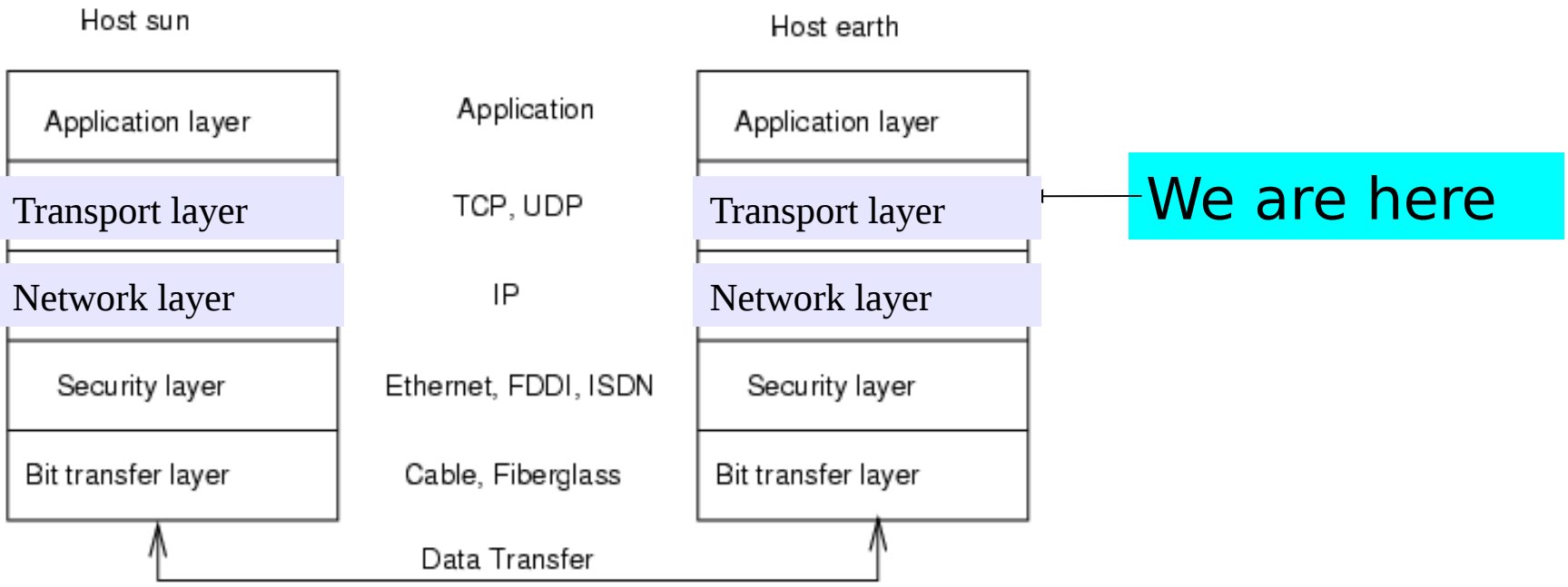
- How Does it Do This?
- Transport layer protocol must keep track of data from each application,
  - Combines this data into a single flow of data to send to lower layers
  - Device receiving information must reverse these operations, splitting data and funneling it to appropriate recipient processes
- Transport layer provides connection services for protocols and applications that run at levels above it
- Categorized as either
  - Connection-oriented services
  - Connectionless services



Transport Layer

Introduction to Transport Layer

- Recall the simplified model we have of today's Internet



Introduction to Transport Layer

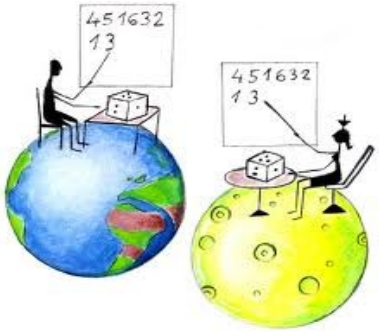
Apps

Transport

- What do we know about the Network Layer?
  - Best effort service
  - No guarantees
  - No orderly delivery of segments
  - Said to be “Unreliable Service”
  - All hosts have at least one IP address

Introduction to Transport Layer

- On top of Network Layer
  - UDP and TCP two main protocols of transport layer



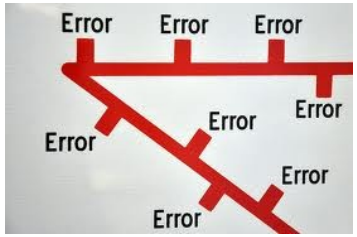
  - Responsibility of this layer
    - Minimum
    - Extend delivery service between two hosts to
    - Delivery service between two processes running on a source and destination host
    - Error checking of packets

BitTorrent

Transport Layer UDP

- Two flavors of Transport Protocols
- UDP – simpler, faster and unreliable
  - What does it do?
    - Delivers packets

    - Checks Errors



That's all it does!

Transport Layer TCP

- TCP – more complex, slower and reliable
  - What does it do?
    - Flow control

    - Sequence numbers for packets
      1,2,3,4,5,6,7 ... 10

    - Acknowledgments and timers
    - Ensures data arrives in order and is correct
    - Congestion control

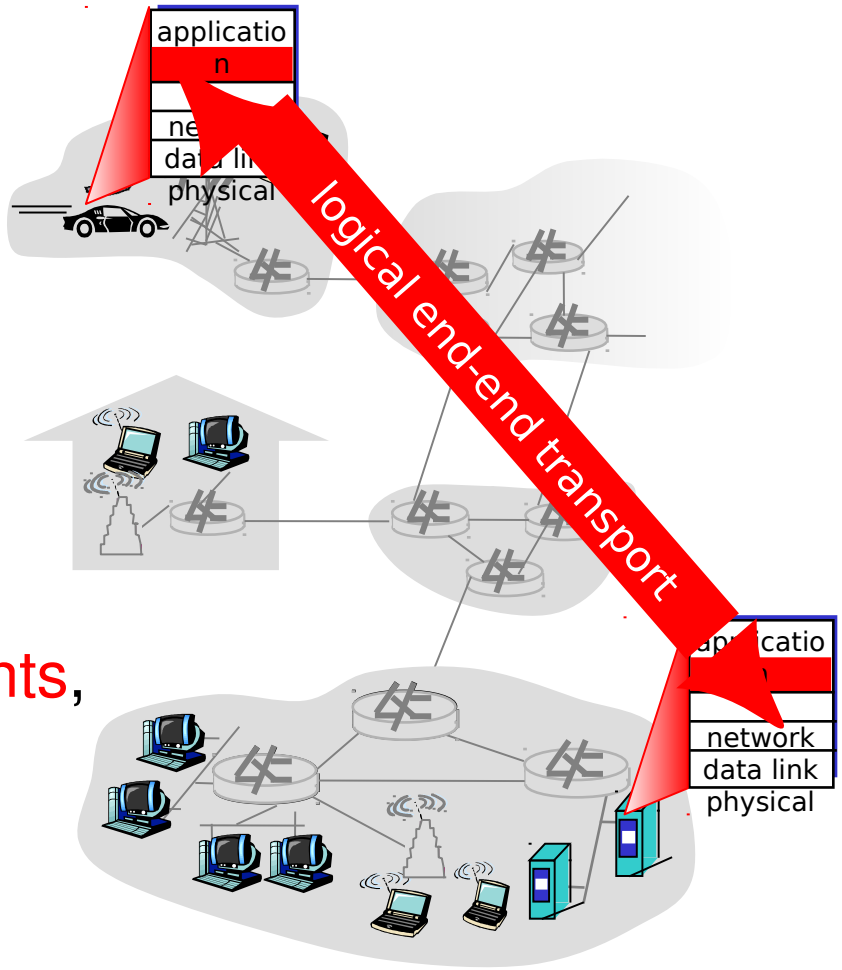
Introduction Transport Layer

Goals

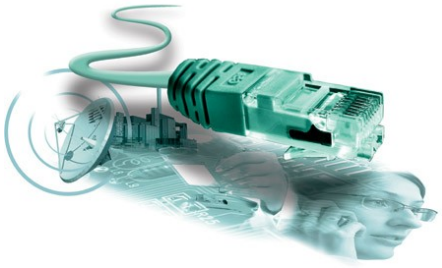
- Understand principles behind transport layer services
  - Multiplexing/demultiplexing
  - Reliable data transfer vs Unreliable data transfer
  - Flow control
  - Congestion control

Transport services and protocols

- Provide Logical Communication between application processes running on different hosts
- Transport protocols run on end systems
  - Sender side
    Breaks messages into segments, pass to network layer
  - Receiver side
    Reassembles segments into messages, pass to application layer

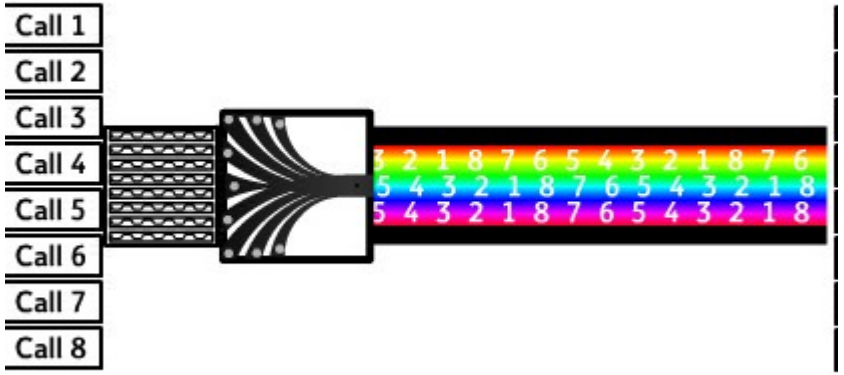


Transport vs. Network Layer



- Network layer
- Logical communication between hosts

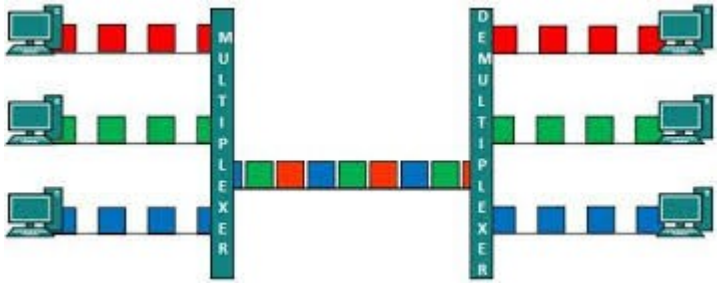
- Transport layer
- Logical communication between processes
  - Relies on and enhances, network layer services



Multiplexing and demultiplexing

In General

Multiplexing and Demultiplexing



- Transport protocol does multiplexing and demultiplexing
  - Sends and Delivers packets to correct application
  - Say single host runs four network applications
    - FTP Session,
    - 2 SSH Sessions,
    - HTTP Session
  - TCP Layer must keep track of four applications and their separate streams of data

Multiplexing and Demultiplexing

- Transport protocol does multiplexing and demultiplexing
  - Gathers data from different applications,
    - Through sockets,
    - Encapsulates each data chunk with headers and passes them to network layer

    -> Multiplexing
  - Delivers data to receiving socket of application

    -> Demultiplexing

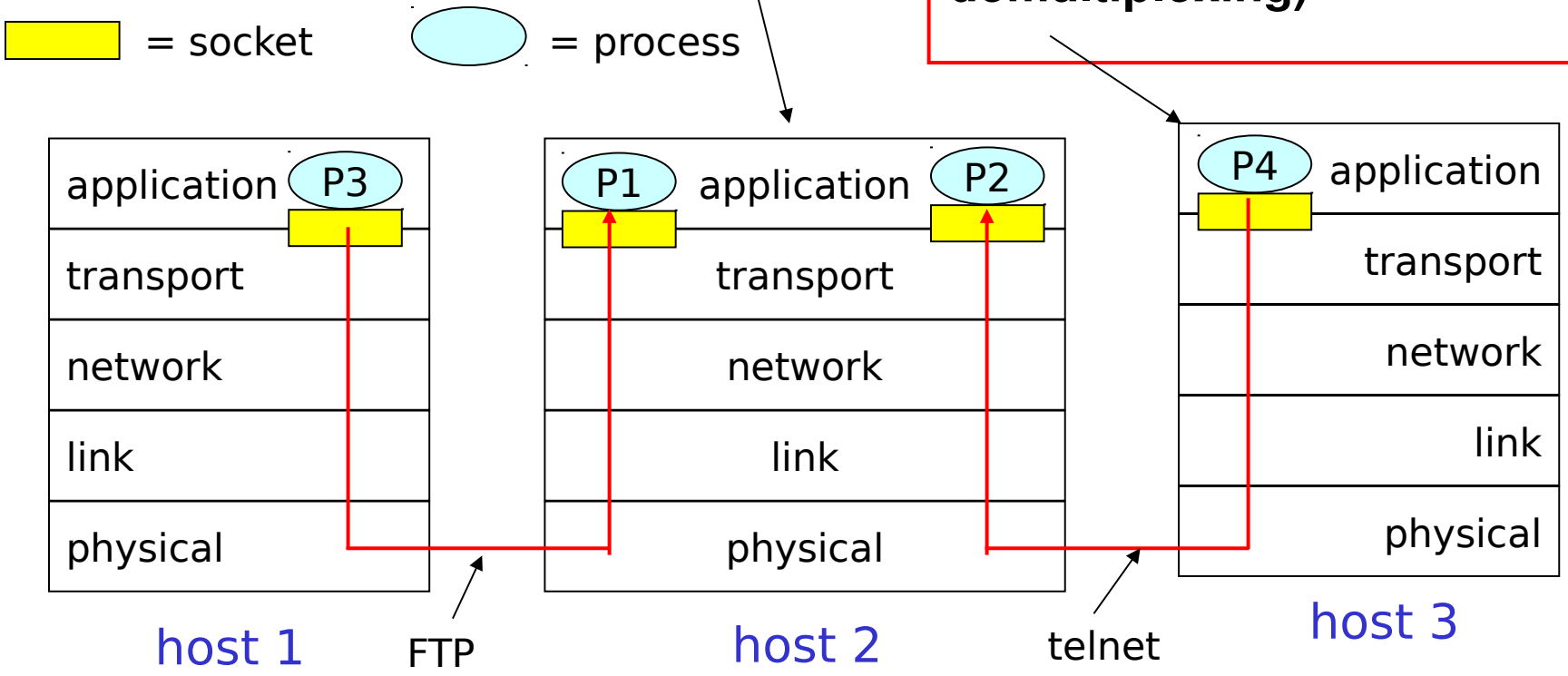
Multiplexing/demultiplexing

Demultiplexing at rcv host:

Delivering received segments to correct socket

Multiplexing at send host:

Gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)

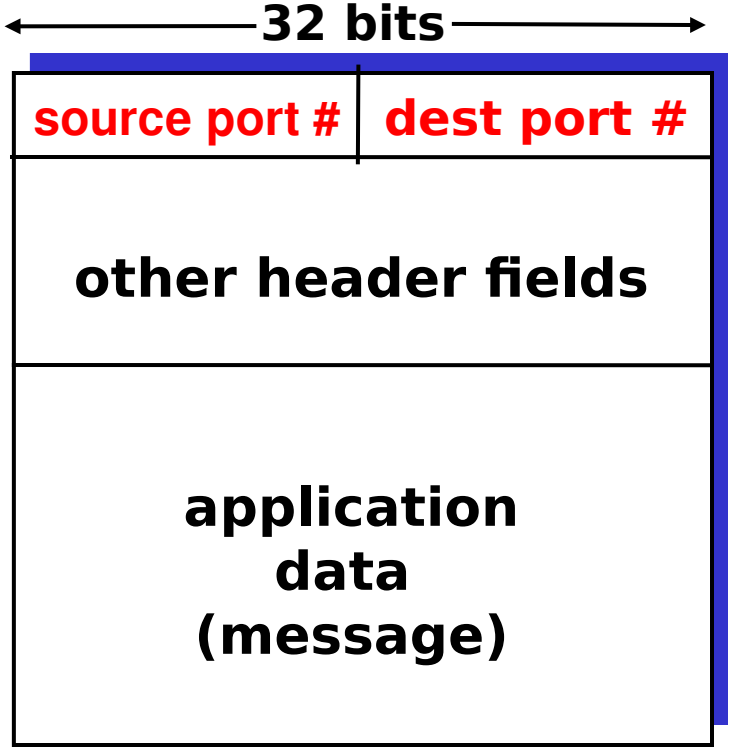


How Demultiplexing Works

Host Receives IP Datagrams

- Each Datagram has source IP Address, Destination IP Address
- Each Datagram carries 1 transport layer segment
- Each Segment has source/destination port number

Host Uses IP Addresses and Port numbers ----> direct segments to appropriate socket



Generic TCP/UDP segment format

UDP Demultiplexing

Connectionless UDP Demultiplexing

- Sockets have port numbers

DatagramSocket mySocket1 = new DatagramSocket(12534);

DatagramSocket mySocket2 = new DatagramSocket(12535);

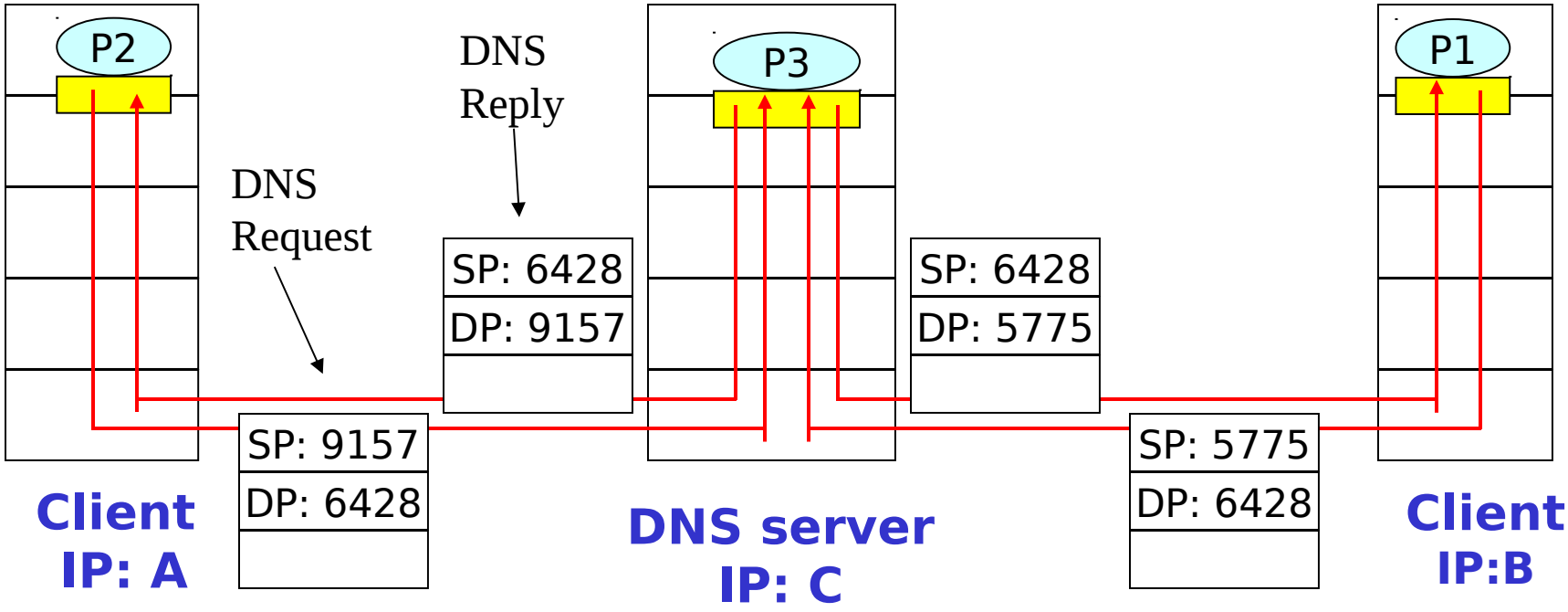
- When host receives UDP segment
  - Checks destination port number within data
  - Directs UDP segment to socket with that port number

- Note: Above are examples of UDP Servers
- UDP Socket identified by two-tuple:

(dest IP address, dest port number)

Connectionless UDP Demultiplexing

DatagramSocket serverSocket = new DatagramSocket(**6428**);



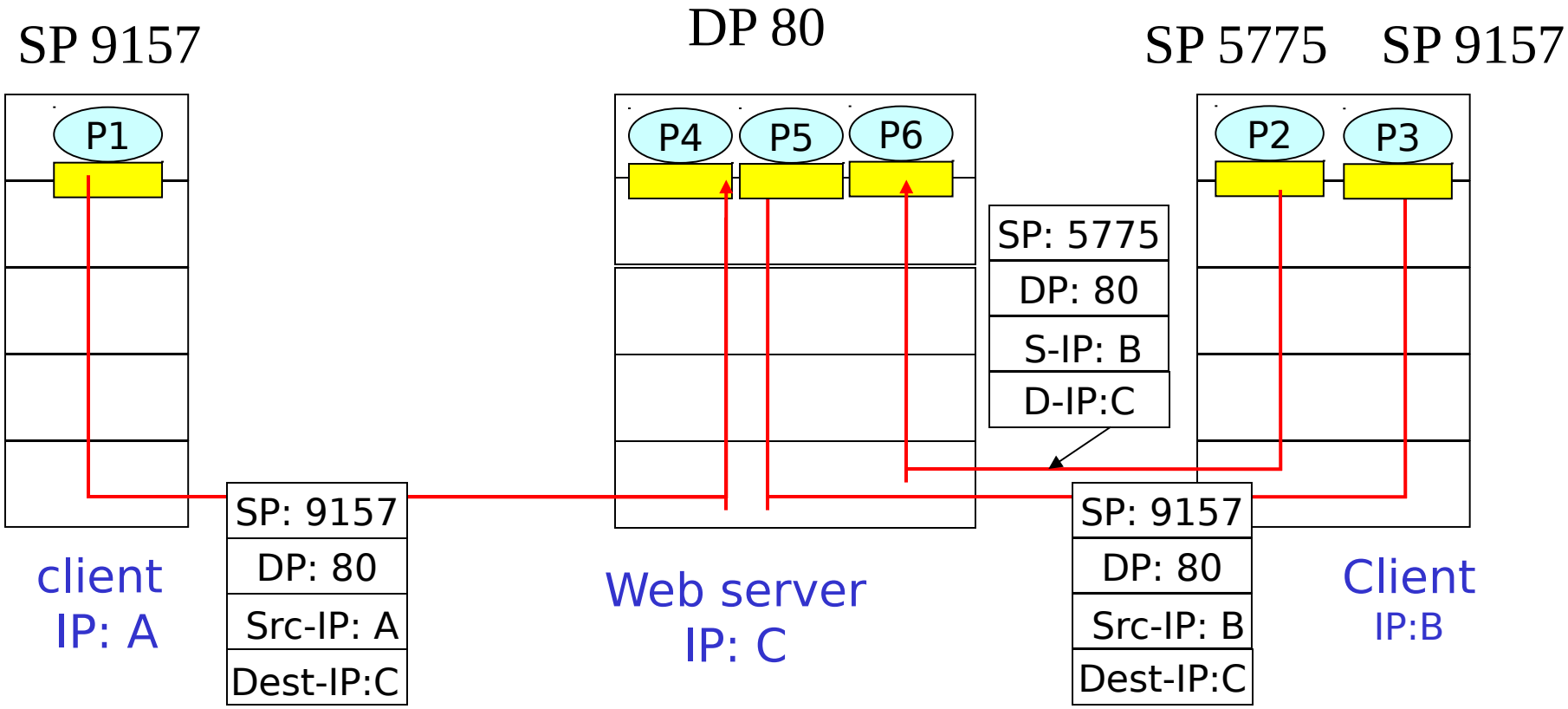
Dest Port (DP) provides “send address”
Src Port (SP) provides “return address”

TCP Demultiplexing

Connection-oriented TCP Demultiplexing

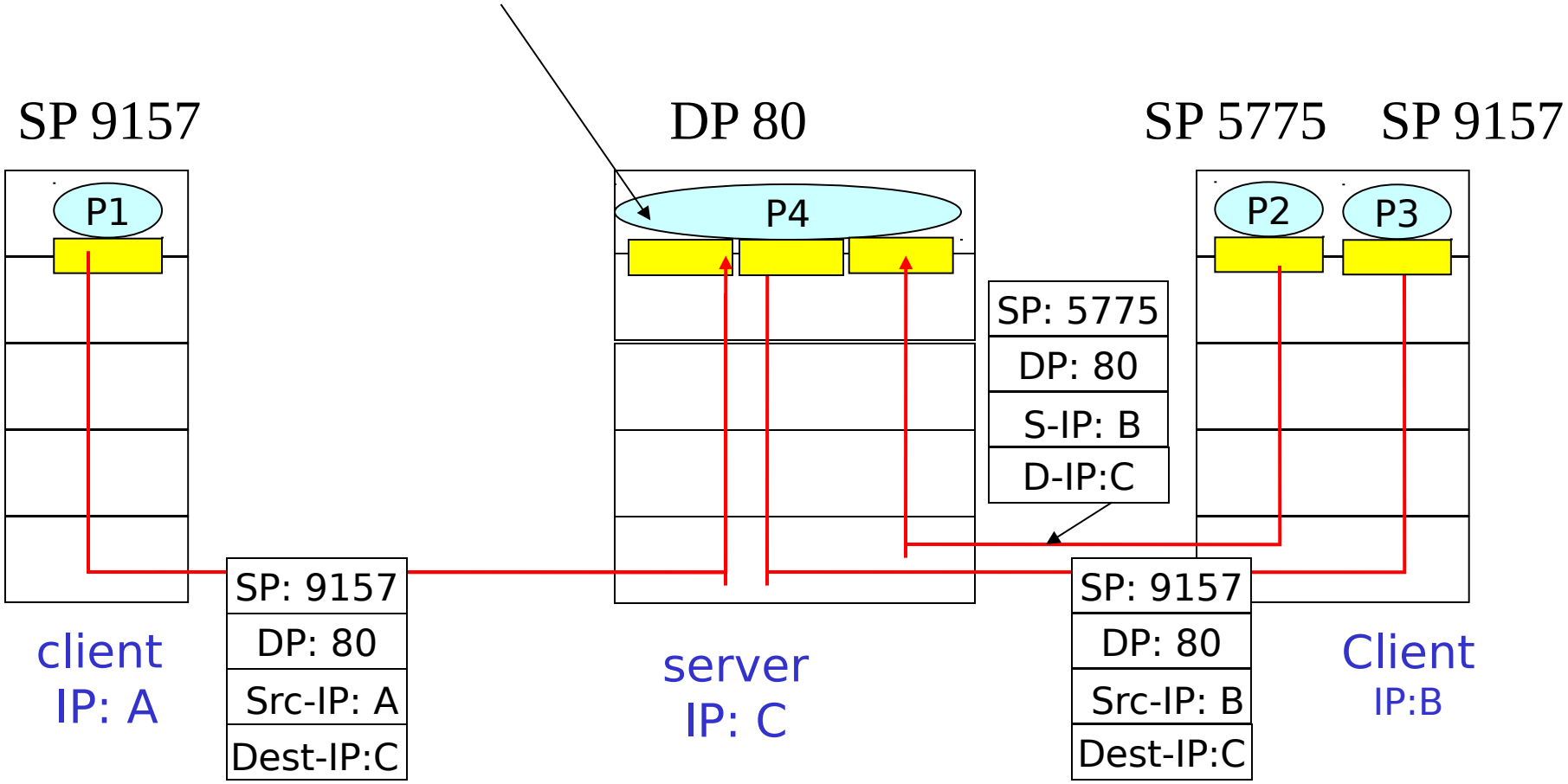
- TCP socket identified by 4-tuple:
  - Source IP address
  - Source port number
  - Dest IP address
  - Dest port number
- Receive host uses all four values to direct segment to appropriate socket
- Server may support many simultaneous TCP sockets:
  - Each socket identified by its own 4-tuple
- Web servers **can** have multiple different connections for each connecting client
  - **Why is that?**
  - Non-persistent HTTP will have different socket for each request!!!

Connection-oriented TCP Demultiplexing



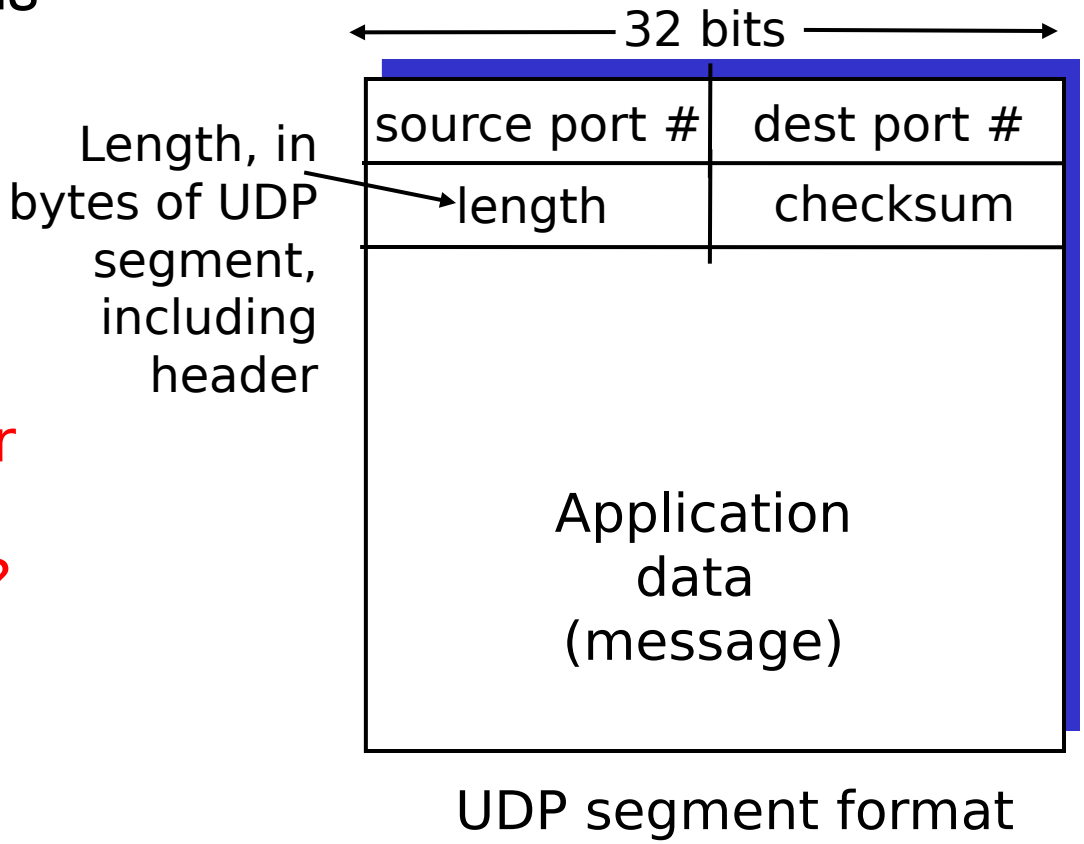
Note: Two machines with same Source Port (SP) = 9157, Is that allowed?

Connection-oriented demumtiplexing Threaded Web Server

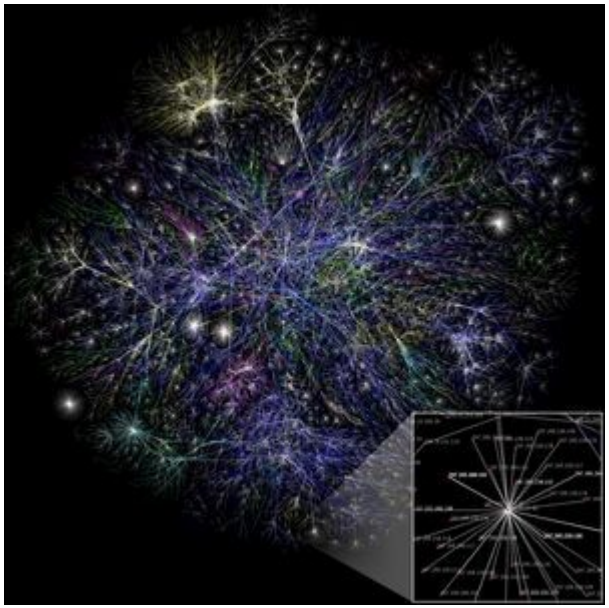


UDP: More

- Often used for streaming multimedia applications
  - Loss tolerant
  - Faster
- Other UDP uses
  - DNS
  - SNMP
- Reliable transfer over UDP
- How would you do it?
- Add reliability at application layer
  - Application-specific error recovery!



UDP segment format



Connection Oriented Transport

Transport Reliability

- What does it mean to be reliable if you are a transport protocol?
  - No data is corrupted
  - All data is delivered in order in which it was sent
  - All data is delivered, not lost
  - Delivered when needed, time aspect important

Reliable Data Transfer

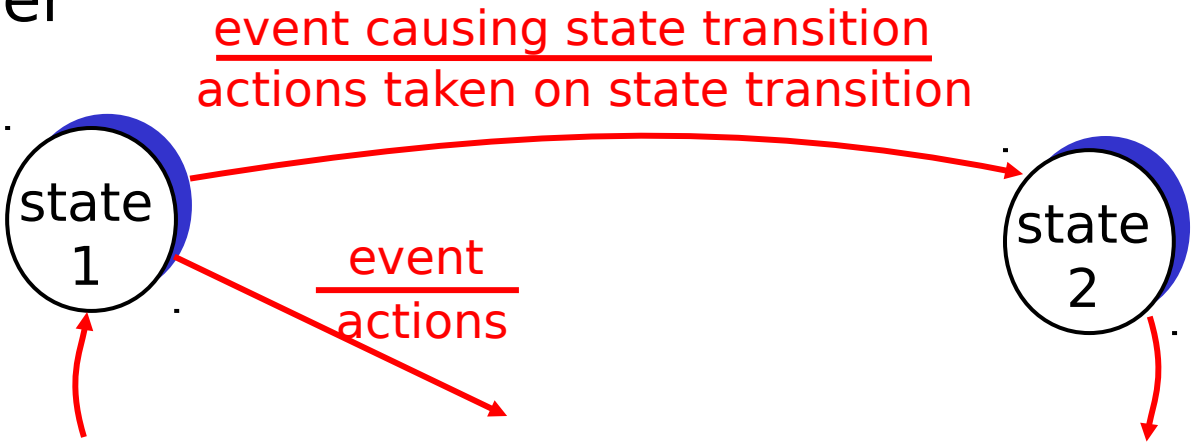
- Next few slides,
  - Build a reliable transfer protocol step by step
  - So, you can see the design decisions that went into the design of TCP to make it reliable
  - Have a protocol that works perfectly to deliver data reliably …

Reliable data Transfer

We'll

- Incrementally develop sender, receiver sides of reliable data transfer protocol (**rdt)**
- Consider only unidirectional data transfer
  - But control info will flow in both directions!
- Use Finite State Machines (FSM) to specify sender, receiver

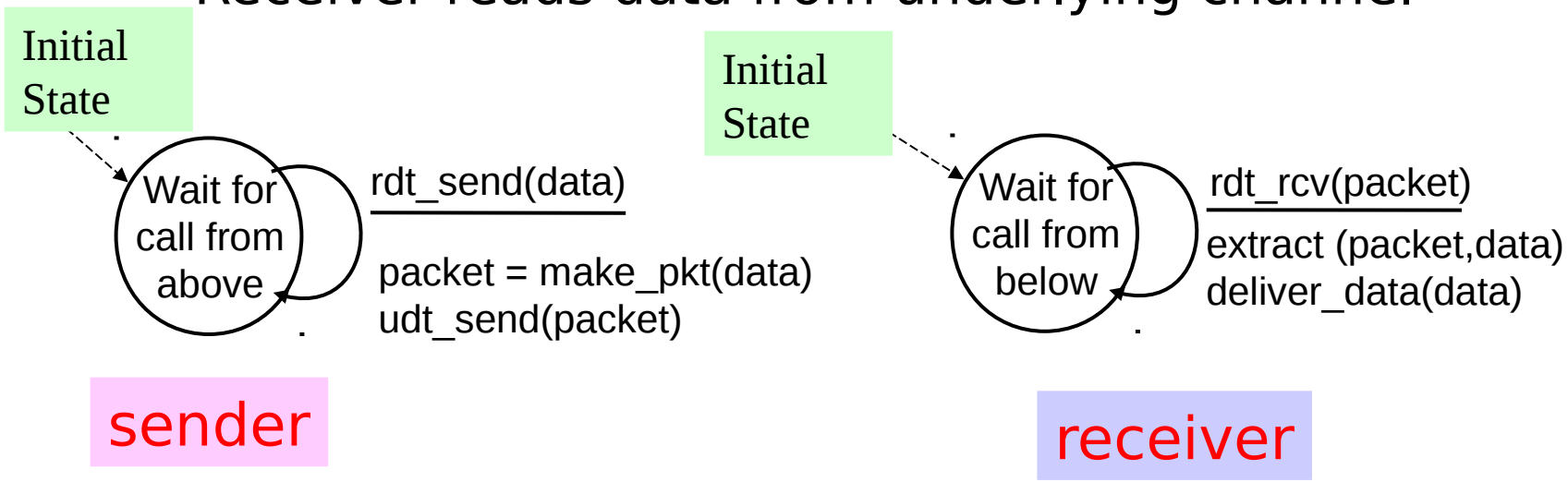
State: When in this "state" next state uniquely determined by next event



Rdt1.0: Reliable transfer over reliable channel

- Underlying channel perfectly reliable
  - No bit errors
  - No loss of packets

- Separate FSMs for Sender, Receiver
  - Sender sends data into underlying channel
  - Receiver reads data from underlying channel



Rdt2.0: Channel With Bit Errors

- What do we need to add to protocol to deliver packets with Bit Errors?

Rdt2.0: Channel With Bit Errors

- What do we need to add to protocol to deliver packets with Bit Errors?
  - Method to detect errors
  - Method to send back a negative acknowledgment
  - Recognition of negative ack -> resend of packet

Rdt2.0: Channel with bit errors

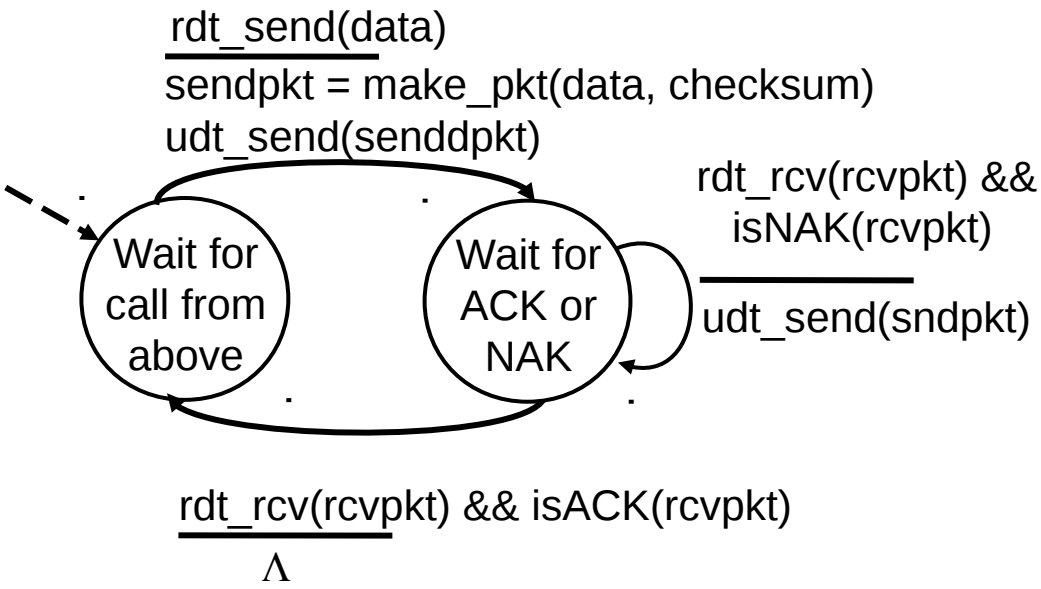
- Underlying channel may flip bits in packet
  - Checksum to detect bit errors

- How to recover from errors?
  - Acknowledgments (ACKs): Receiver tells sender that packet received OK

  - Negative acknowledgments (NAKs): Receiver tells sender that packet had errors
  - Sender retransmits packet on receipt of NAK

- New mechanisms in **rdt2.0** (beyond **rdt1.0**):
  - Error detection
  - Receiver feedback: control msgs (ACK,NAK) rcvr->sender

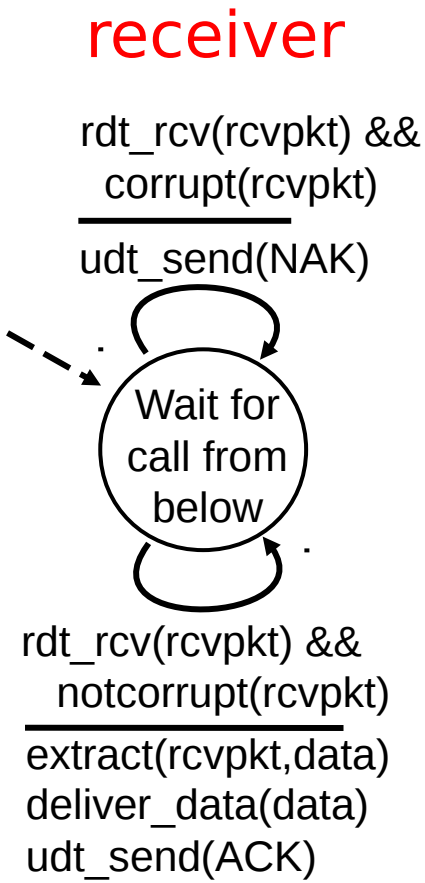
Rdt2.0: FSM specification



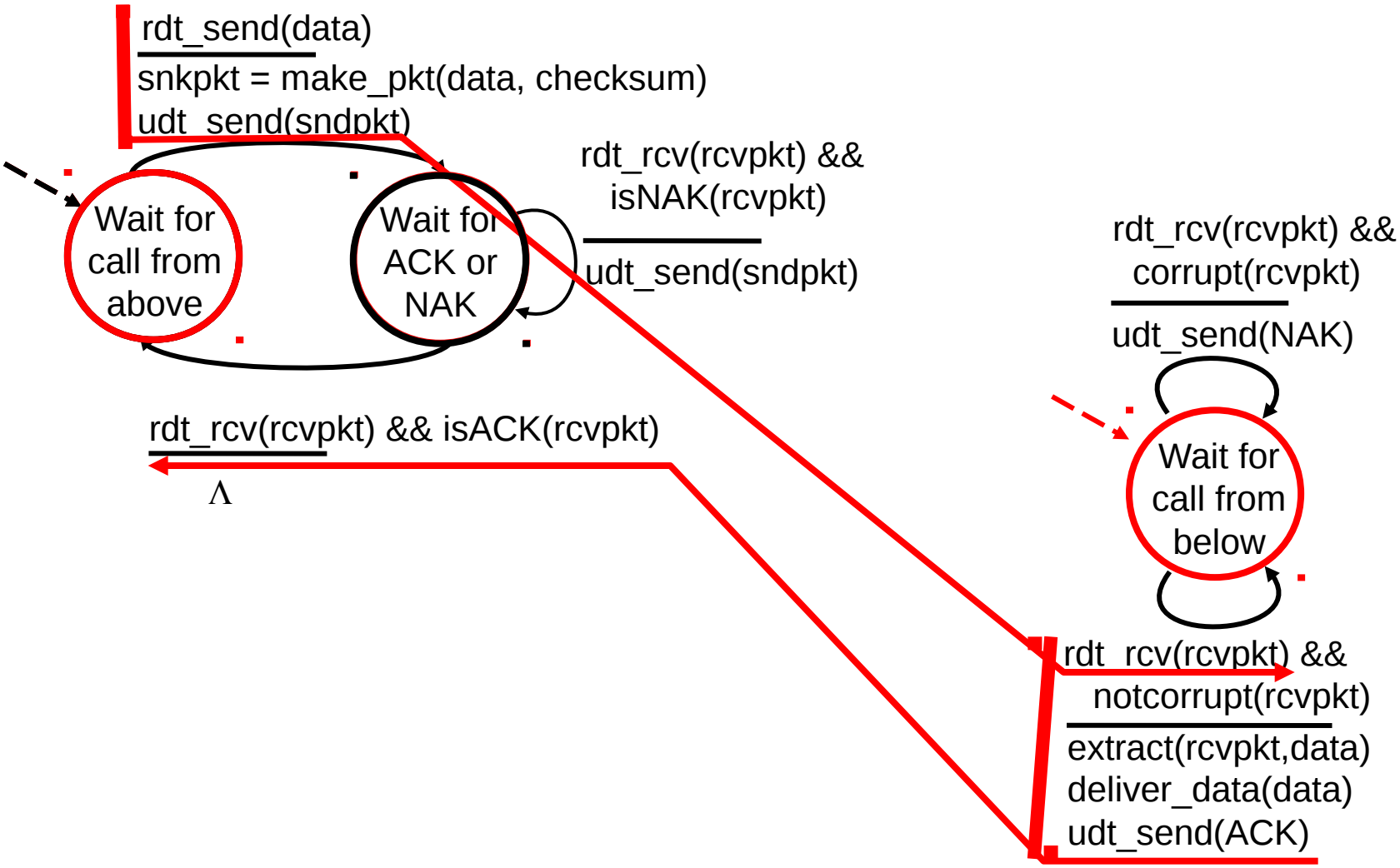
sender

Λ - Means no action on event

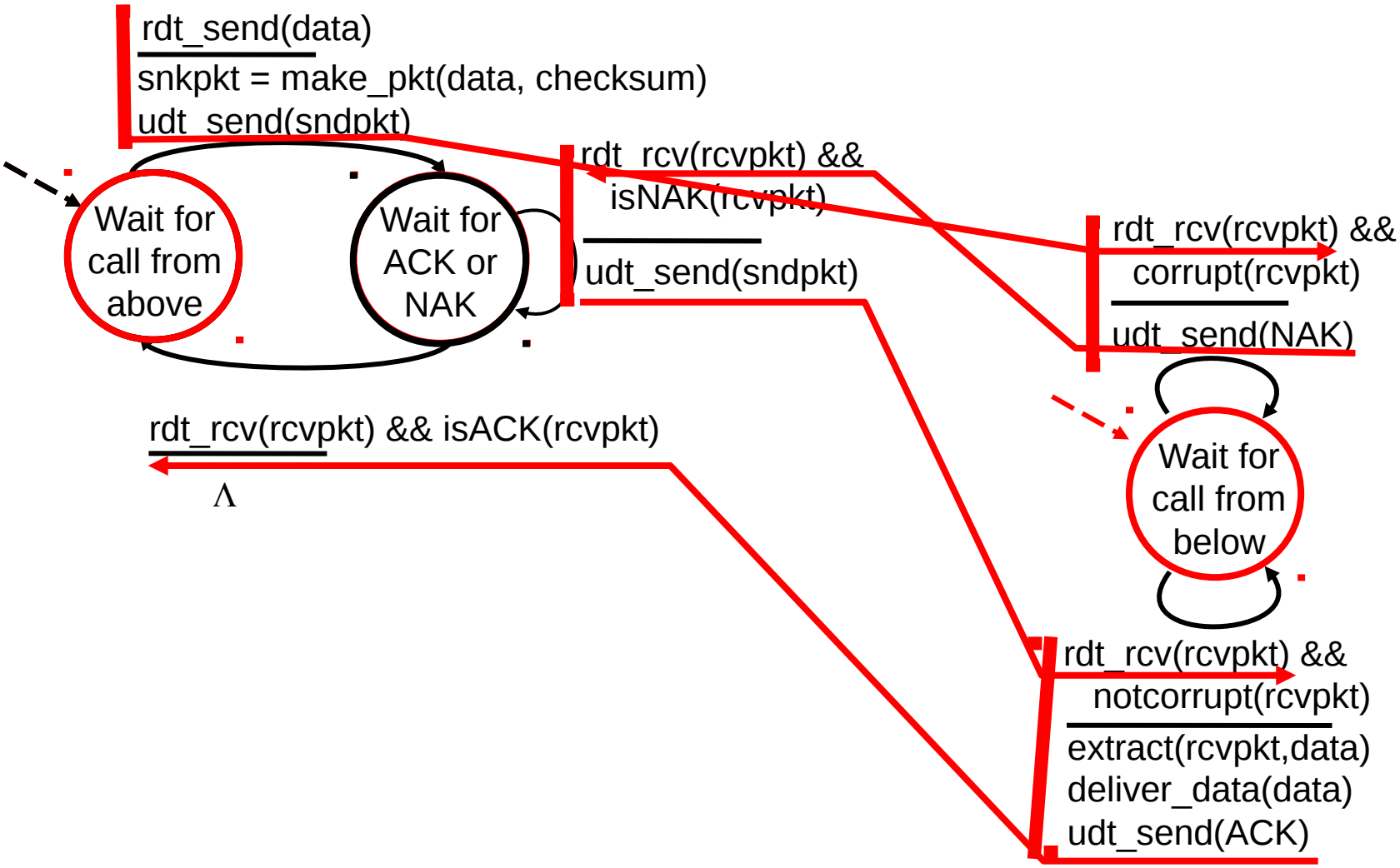
receiver



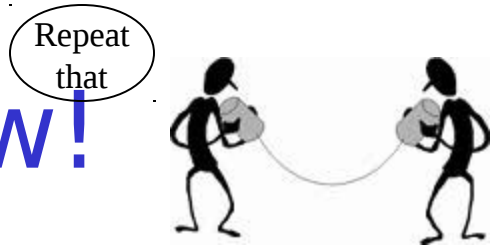
Rdt2.0: operation with no errors



Rdt2.0: error scenario



Rdt2.0 Has a fatal flaw!



What happens if ACK/NAK corrupted?

- Sender doesn't know what happened at receiver!

- If Human conversation, just say, please repeat that ... maybe multiple times

- If Computer conversation
  - Add enough checksum bits to correct the error
  - Or, retransmit, but add identifier to packet to say its a retransmission ... this is what TCP does!!!

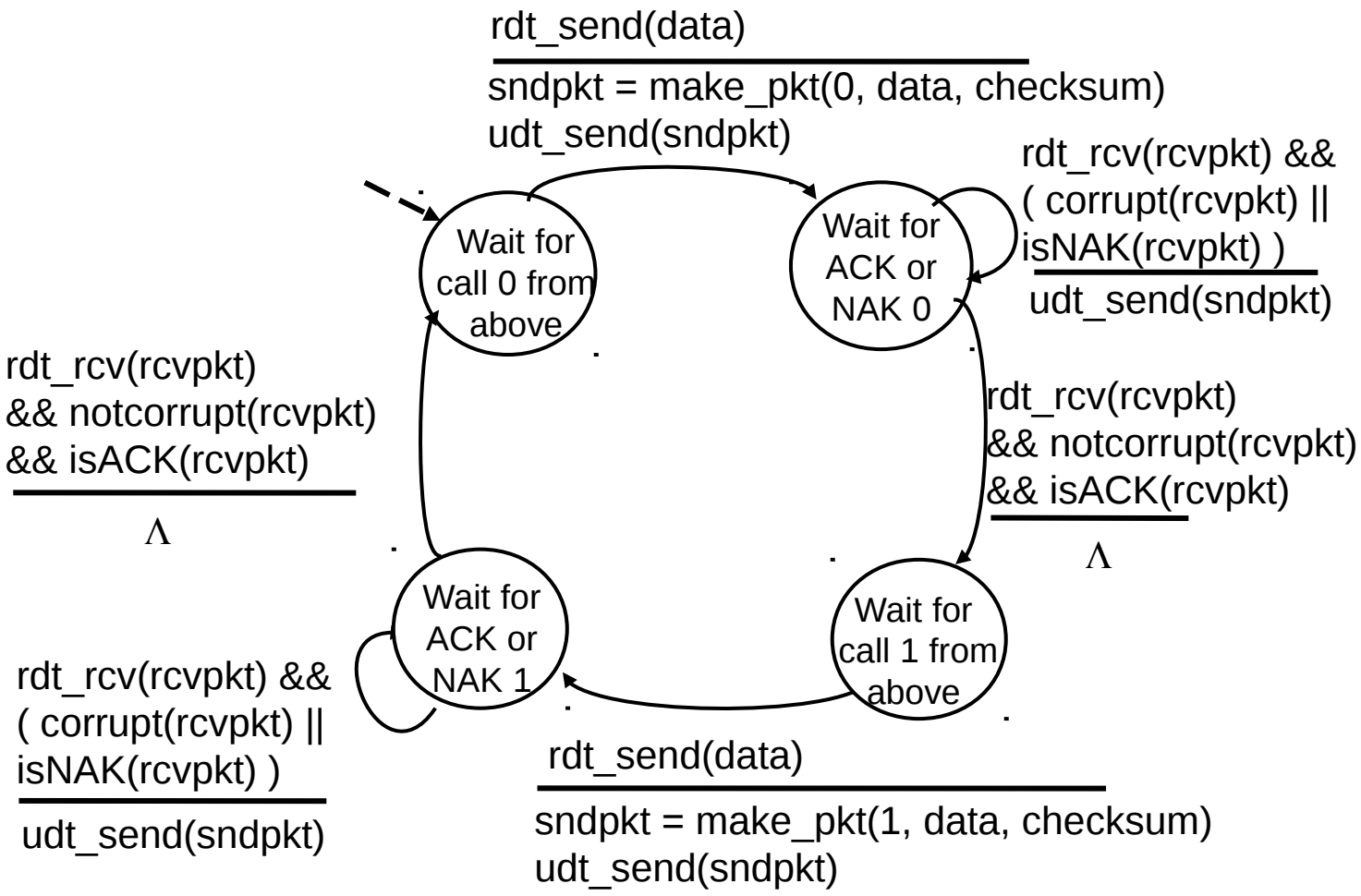
Rdt2.0 Has a fatal flaw!

Handling Duplicates:

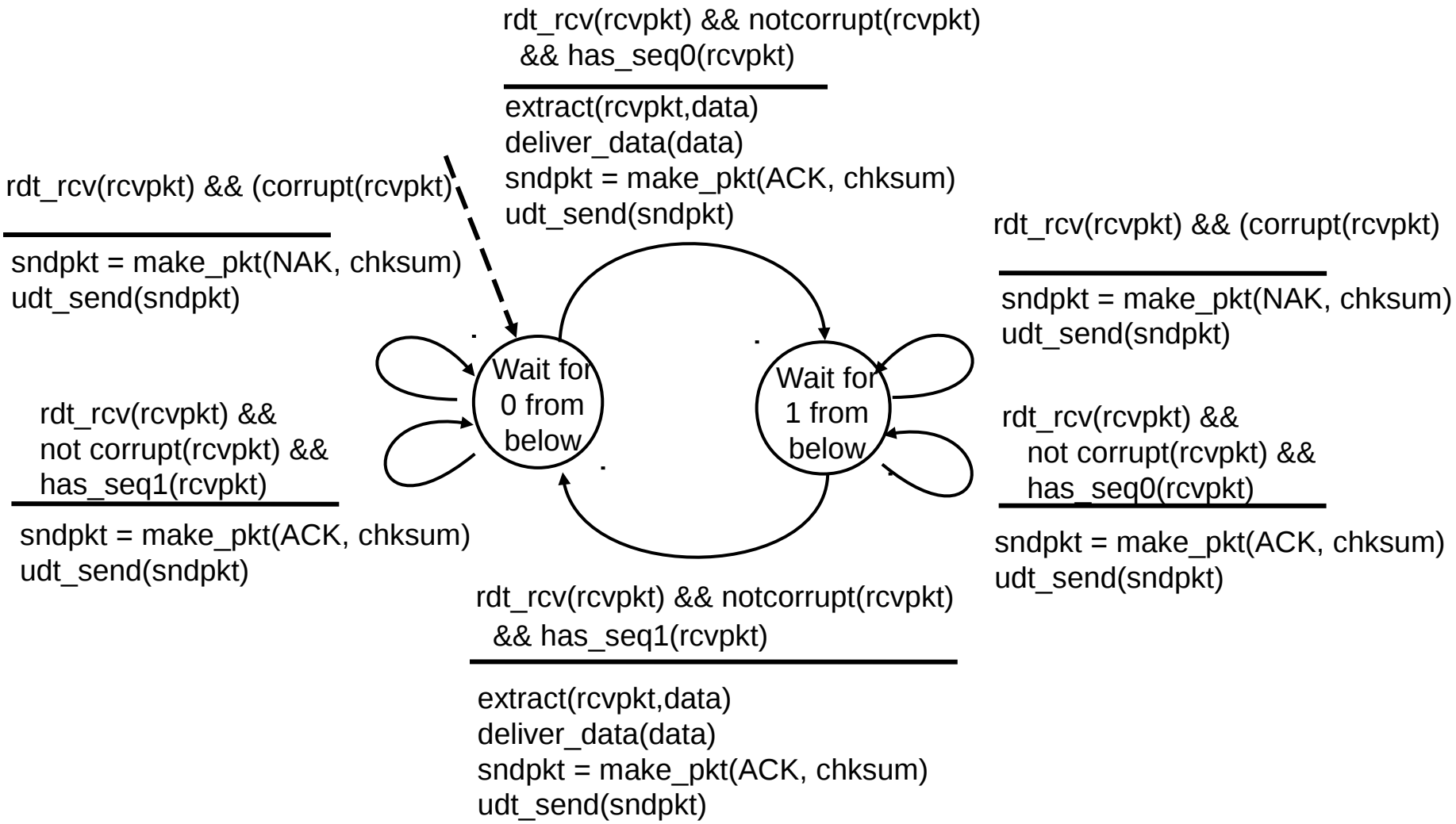
- Sender retransmits current packet if ACK/NAK garbled
- Sender adds sequence number to each packet
- Receiver discards (doesn't deliver up) duplicate packet

stop and wait
Sender sends one packet, then waits for receiver response

Rdt2.1: Sender, handles garbled ACK/NAKs



Rdt2.1: receiver, handles garbled ACK/NAKs



Rdt2.1: Discussion

Sender

- Seq number added
- Two sequence numbers (0,1). Is that enough?
- Must check if received ACK/NAK corrupted
- Twice as many states
  - State must “remember” whether “current” packet has 0 or 1 sequence number

Receiver

- Must check if received packet is duplicate
  - State indicates whether 0 or 1 is expected packet sequence number
- Note: receiver can *not* know if its last ACK/NAK received OK at sender

Rdt2.2: Improves upon Rdt 2.1 NAK-free protocol

- Same functionality as Rdt2.1, using ACKs only instead of NAK
- Receiver sends ACK for last packet received OK
  - Receiver must explicitly include seq number of packet being ACKed
- Duplicate ACK at sender results in same action as NAK
- Retransmit current packet!!

Rdt3.0: Now Have Channels with Errors and Loss

New Assumption

- Underlying channel can also lose packets

  (data or ACKs)
- Sender doesn't know if data packet or ACK was lost, or just delayed
  - So far ....
    - Checksums, Sequence numbers,
    - ACKs, Retransmissions

- Are these enough to account for lost packets?

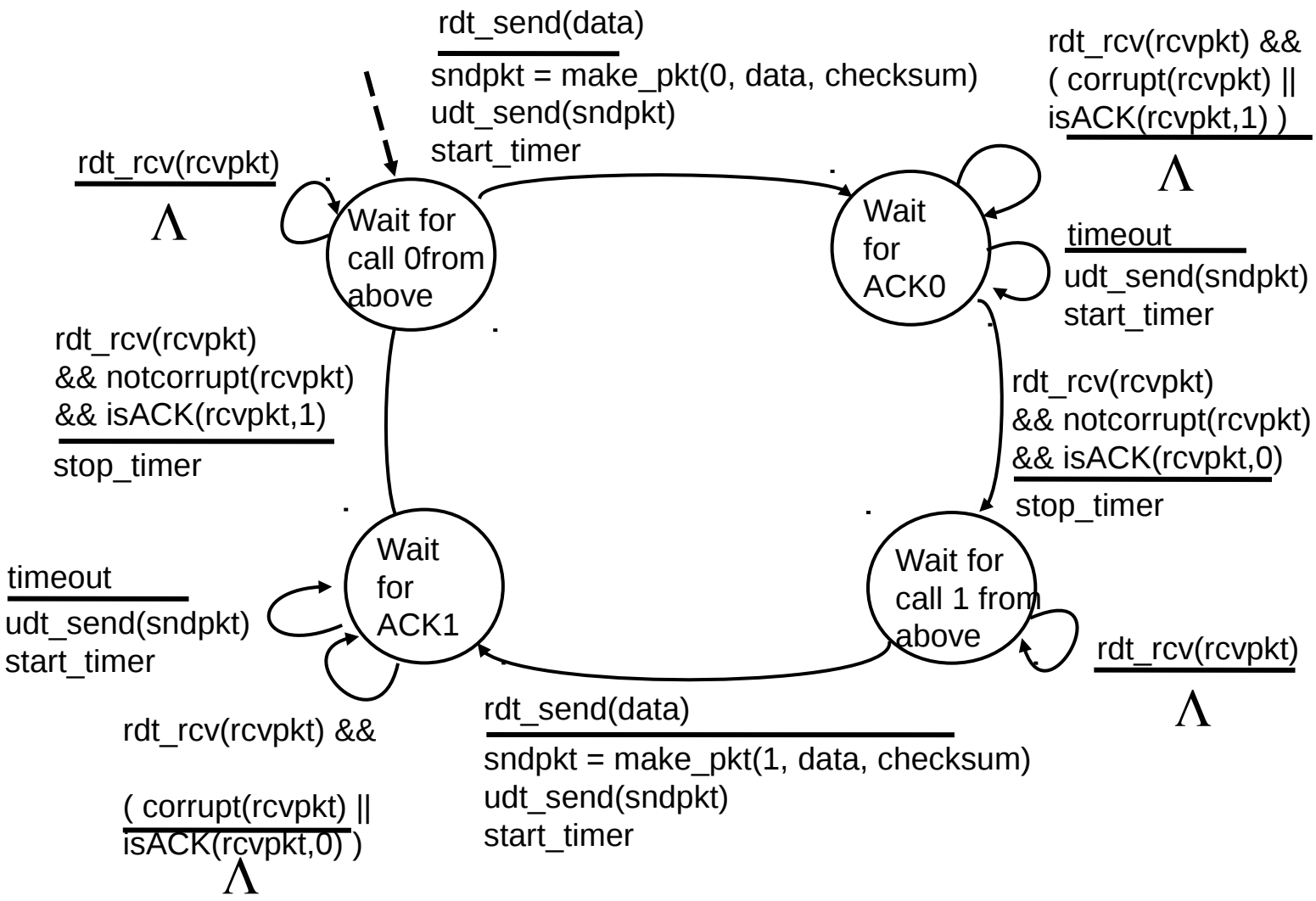
Rdt3.0: Now Have Channels with Errors *and* Loss

Approach: Sender waits "reasonable" amount of time for ACK

- Retransmits if no ACK received in this time
- If packet (or ACK) just delayed (not lost):
  - Retransmission will be duplicate, but use of sequence #'s already handles this
  - Receiver must specify seq # of packet being ACKed
- Requires countdown timer

rdt3.0 sender

Gets more complicated with timer



Summary

- Described what the transport layer does
- Introduced general concepts of a connection-oriented, transport protocol
- Making something reliable is not easy!!
  - Complex, not easy to understand
  - Discussed how you would build in reliability
- Moving on to examine TCP as an example of a reliable protocol

References

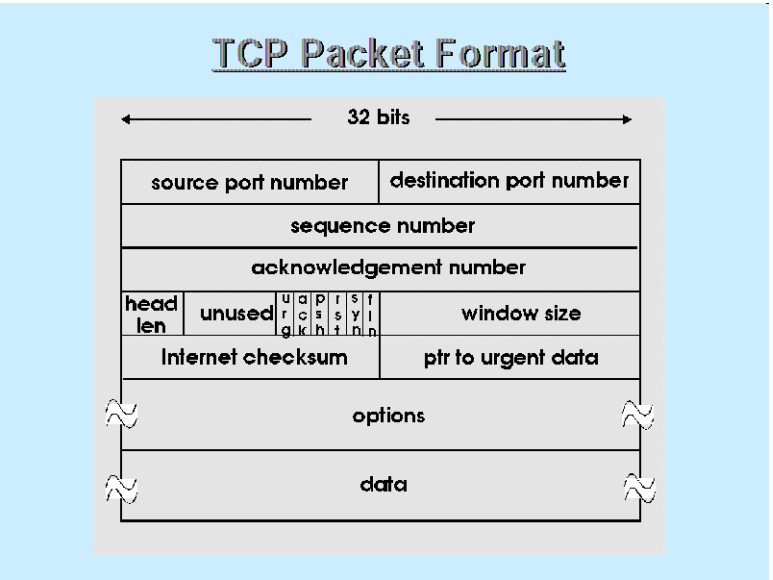
- RFC of TCP v4, 1981

  http://www.ietf.org/rfc/rfc0793.txt

- Original Paper by Authors of TCP/IP Suite

  Vinton G. Cerf, Robert E. Kahn, A Protocol for Packet Network Intercommunication, IEEE Transactions on Communications, Vol. 22, No. 5, May 1974 pp. 637-648

  http://penguin.ewu.edu/cscd330/cerf74-2.pdf



Transport Protocol

Read: Keep reading Chapter 3 - Transport