

# **CSE 222**

# **Graduate Networking**

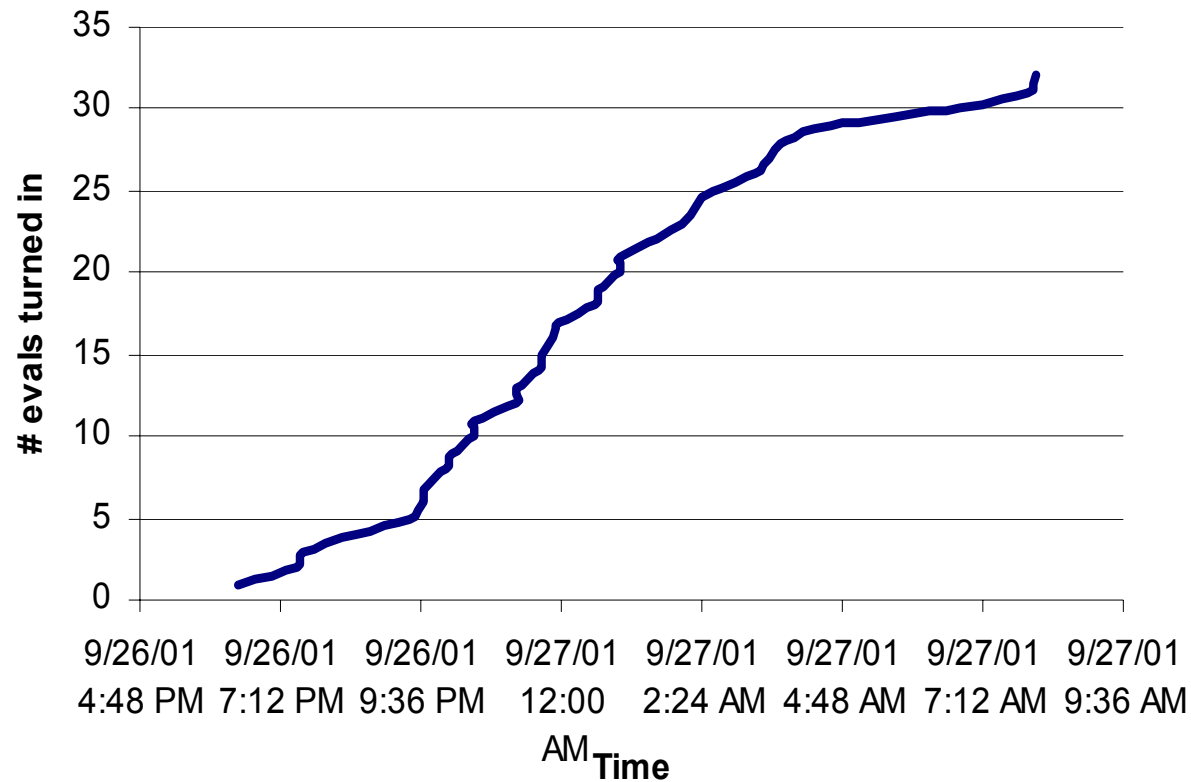
**Fall 2001**

**Lecture 3: Internetworking and  
Reliable Transmission**

Stefan Savage

# Average hours of sleep increases

Eval turnin-in time distribution



# Eval quality

---

- Generally very good (85%)
- One common mistake
  - ◆ Pure descriptions
    - » “The end-to-end paper talks about the end-to-end argument. It talks about it for file transfer, and for encryption, and for...”
  - ◆ Really looking for a bit more thought
    - » Summarize the **key ideas** in your own words, not the organization of the paper
    - » Critique the ideas
      - What makes the ideas important or significant?
      - What assumptions are they based on?
      - How might they apply (or not) to problems not discussed in the paper?

# Overview

---

- **Internetworking**
- **Reliable Transmission**
  - ◆ Retransmission
    - » Stop and wait
    - » Sliding window
  - ◆ Forward Error Correction

# Internetworking

---

- Cerf&Kahn74, “A Protocol for Packet Network Intercommunication”
  - ◆ Foundation for Internetworking and hence, the Internet
  - ◆ We’ll talk about the reliability issues later
- Key internetworking issues
  - ◆ Internetworking as a separate service layer
  - ◆ Addressing
  - ◆ Packet size
  - ◆ Error detection
  - ◆ Gateway services

# How to connect different networks?

---

- Gateways
  - ◆ Connect different networks together
  - ◆ One half of gateway is in each network
- Two options
  - ◆ Translation: Gateway translates directly between different network formats
  - ◆ **Indirection**: Gateway translates between local network format and universal “intermediate” format
- Pros/cons of each approach?
- Note impact of economics on decision. This is engineering.

# Addressing

---

- Hierarchical addressing
  - ◆ Global inter-network address
  - ◆ Local network-specific address



- Why hierarchical?
- Assumptions about networks?

# Packet size

---

- Heterogeneous maximum packet size
  - ◆ E.g. Ethernet 1500B, FDDI 4500B, SLIP ~250B
- Options
  - ◆ Mandate minimum packet size
    - » The Internet did this (576B IPv4, ~1500 IPv6)
  - ◆ Fragmentation
    - » Gateway splits packets into smaller acceptable packets
    - » Need way to associate and order fragments from same packet
      - Use unique sequence numbers + byte count field
    - » Need way to know when you have a whole packet
      - Flag indicating end of segment
- Question to think about: how does the unit of retransmission interact with fragmentation?

# Error detection

---

- **Bit errors**

- ◆ Checksum written at end of packet
- ◆ Checked by receiving hosts
  - » Whats wrong with this approach?
- ◆ CRC vs Checksum: performance/detection tradeoff
- ◆ Error distribution assumptions?
  - » “When the CRC and TCP Checksum Disagree”, Stone& Partridge, SIGCOMM 2000.

- **Packet losses**

- ◆ We'll deal with this shortly

# Gateway services

---

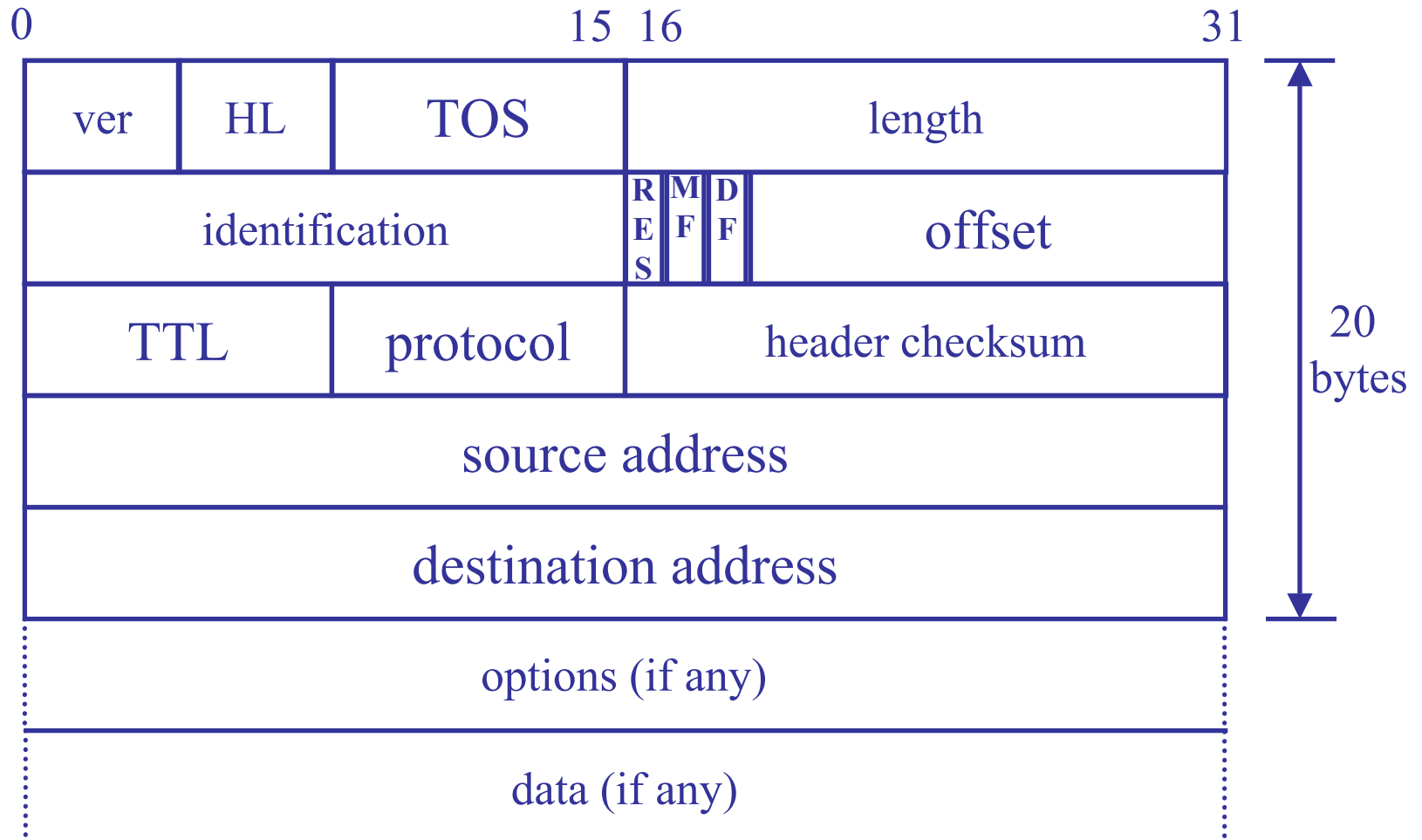
- Things Gateways do
  - ◆ Forward packets
  - ◆ Fragment (must parse sequence numbers/length)
- Things they do not do
  - ◆ Reassembly or duplicate detection
    - » Why?
  - ◆ Error detection
  - ◆ Loss detection/retransmit requests
  - ◆ Failure detection

# How IP works today?

---

- **Packet format**
  - ◆ Header checksum
  - ◆ Time-to-live
  - ◆ Protocol demultiplexing
  - ◆ Type-of-Service
  - ◆ Options
- **Addressing**
- **Fragmentation**

# IP Packet Format



# IP addressing

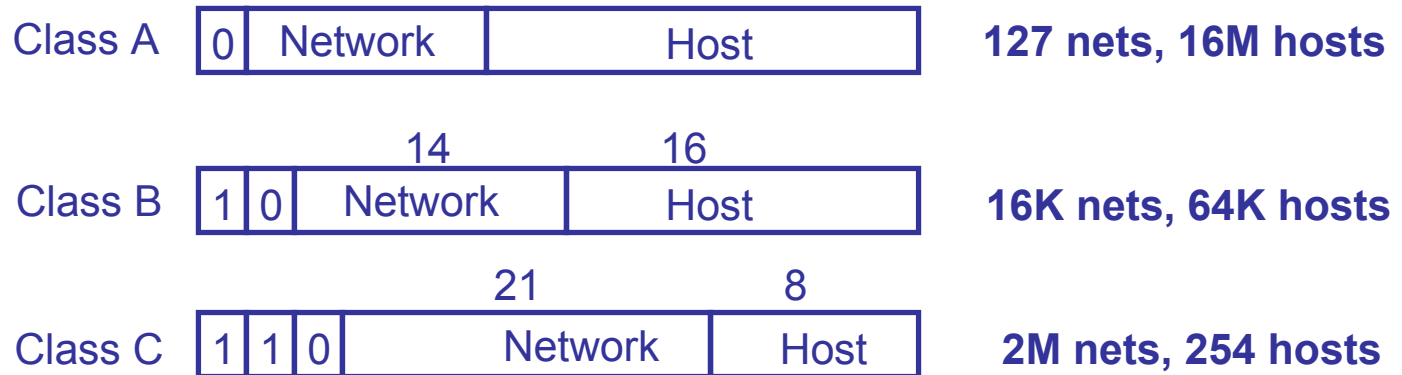
---

- 32-bits in an IPv4 address
  - ◆ Dotted decimal format a.b.c.d
  - ◆ Each represent 8 bits of address
- Network part and host part
  - ◆ E.g. IP address 132.239.15.3
  - ◆ 132.239 refers to the UCSD campus network
  - ◆ 15.3 refers to the host gremlin.ucsd.edu
- Which part is network vs host?
  - ◆ Pre-1993 class-based addressing, static determination
  - ◆ Post-1993 classless addressing, dynamic split
    - » This is what the “network mask” is all about

# Class-based routing (<1993)

---

- Most significant bits determines “class” of address



- Pro
  - ◆ Single lookup to find address
- Con
  - ◆ Fragmentation
  - ◆ Can't aggregate

# Classless addressing (now)

---

- In 1993, Classless Inter-Domain Routing introduced
  - ◆ Routes represented by tuple (network prefix/mask)
  - ◆ Allows arbitrary allocation between network and host address



- ◆ e.g. 10.95.1.2/8: 10 is network and remainder (95.1.2) is host
- Pro:
  - ◆ Finer grained allocation, aggregation of suballocations
- Con:
  - ◆ More expensive lookup: longest prefix match

# Fragmentation

---

- Sender writes unique value in identification field
- If router fragments packet
  - ◆ Copy identification into each field
  - ◆ Differentiate fragments using offset field
  - ◆ MF (More Fragments) flag
- What is impact of fragmentation on retransmission?
  - ◆ No IP-level retransmission
  - ◆ Kent & Mogul, “Fragmentation Considered Harmful”, 1987

# Overview

---

- Internetworking
- **Reliable Transmission**
  - ◆ Retransmission
    - » Stop and wait
    - » Sliding window
    - » TCP
  - ◆ Forward Error Correction

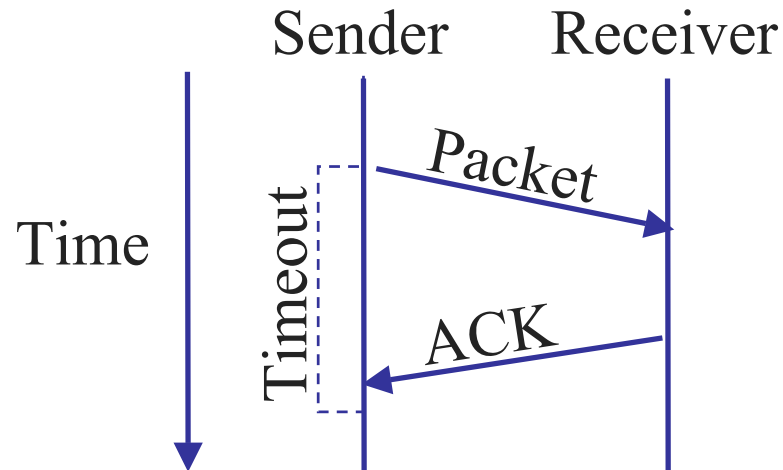
# Reliable Transmission

---

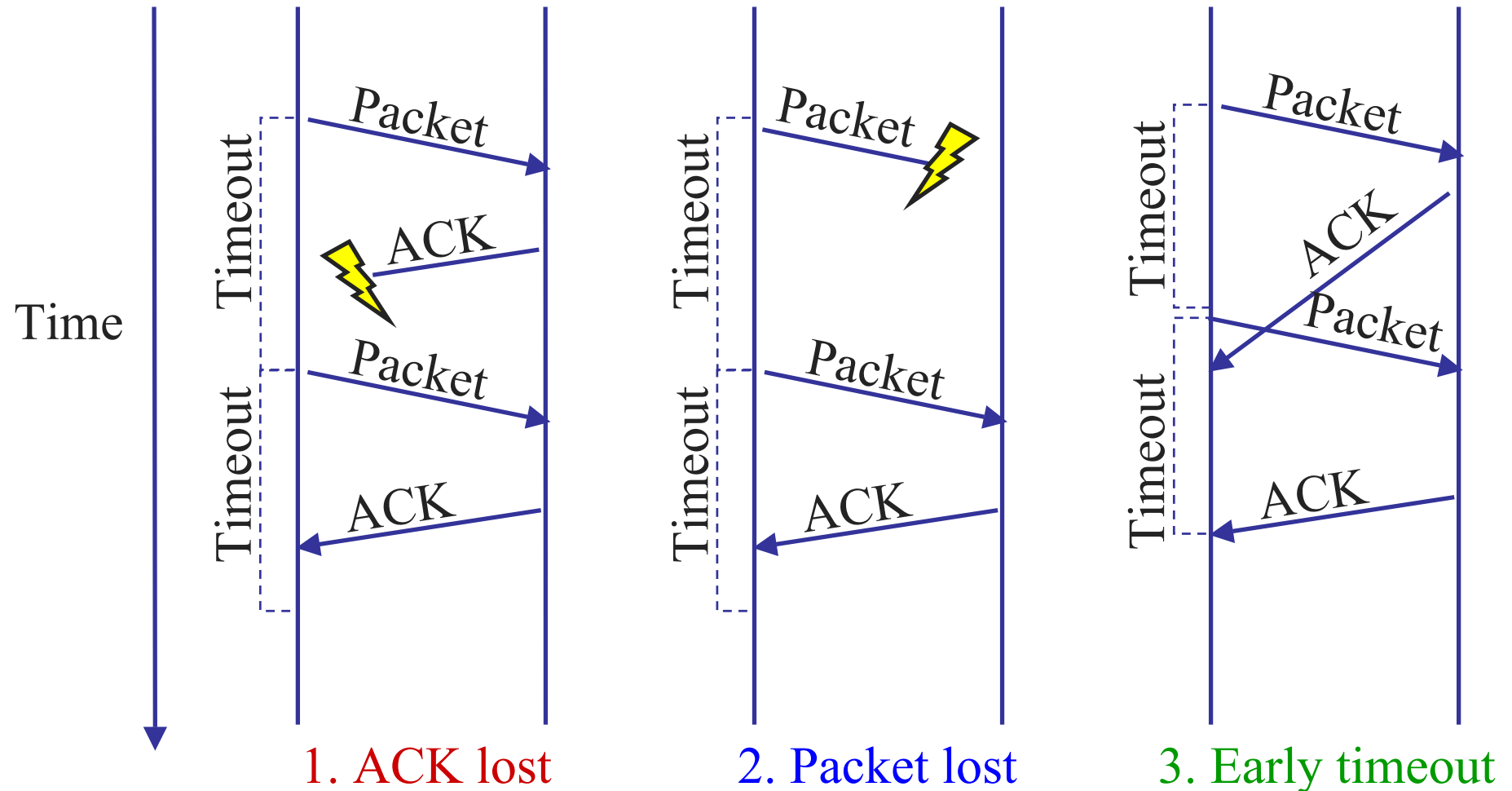
- How do we reliably send a message when packets can be lost in the network?
- Obvious options
  - ◆ Detect a loss and retransmit (**Cerf&Kahn74**)
  - ◆ Send redundantly (Byers et al.98)
- First, simplest reliable protocol: Stop and Wait
  - ◆ Send a packet
  - ◆ Stop and wait until an acknowledgement arrives from receiver
  - ◆ Retransmit if timeout occurs before ACK arrives

# Stop and Wait

---

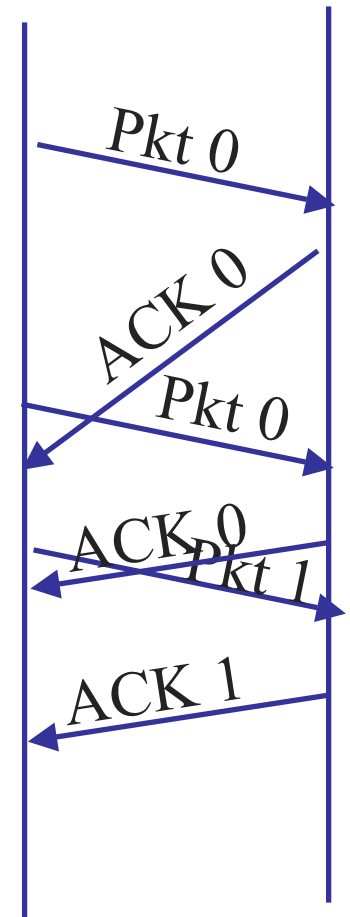


# Recovering From Errors



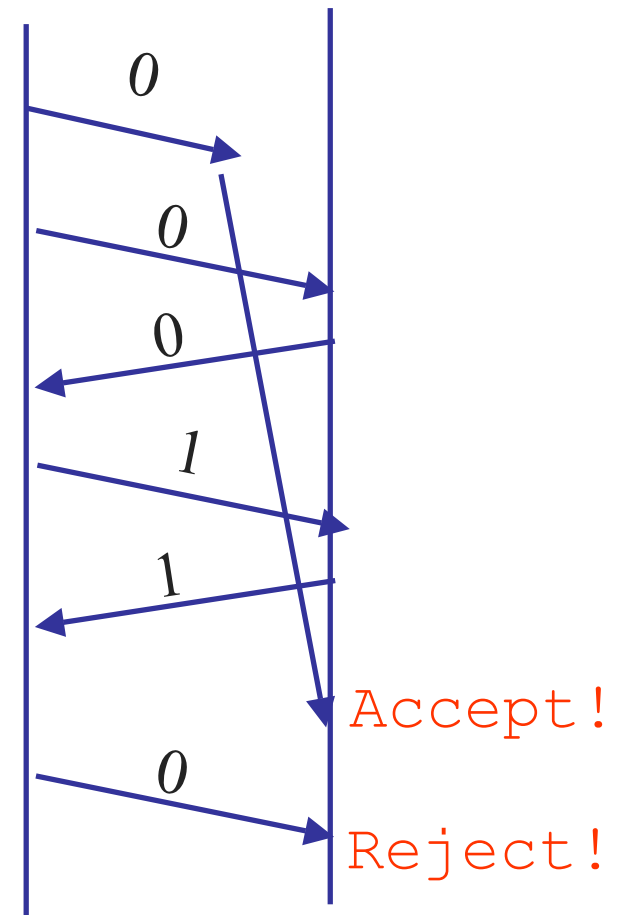
# How does receiver recognize a duplicate?

- Add sequence numbers to packet
  - ◆ Both in data packets and ACKs
- Sequence # in packet is finite, though
- How many bits do we need?
  - ◆ One bit for stop and wait
  - ◆ Won't send seq #1 until receive ACK for seq #0



# What if packets are delayed?

- Never reuse a seq #? Finite...
- Require in order delivery?
- Prevent very late delivery?
  - ◆ TTL: Decrement hop count per packet, discard if exceeded
  - ◆ Seq #s not reused within delay bound
  - ◆ TCP *standard*:
    - » Maximum segment lifetime (MSL)
    - » 120 seconds is recommended
    - » Trust implementations to obey ☺



# What happens if a machine crashes?

---

- How do we distinguish packets sent before and after reboot?
  - ◆ Can't remember last sequence # used
- Solutions
  - ◆ Restart sequence # at 0?
  - ◆ Assume boot takes max packet delay?
  - ◆ Choose seq # at random and hope?
  - ◆ Use stable storage and increment high order bits of seq # on every boot
- Reality: People don't worry about this
  - ◆ Slow reboots, explicit connection management, tolerant users

# Performance issues with Stop and Wait

---

- **Capacity**

- ◆ For a network with bandwidth  $BW$  and delay  $D$ , a sender can transmit  $BW \cdot D$  bytes before the network is “full”
- ◆ Stop-and-**wait** is very inefficient... pipe is empty most of the time

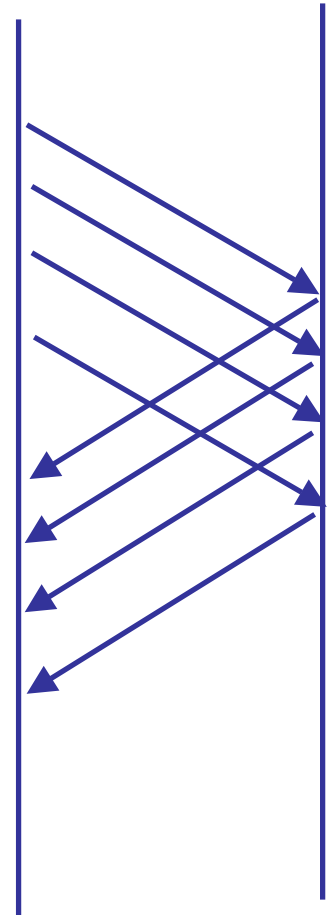
- **Delay**

- ◆ Time to detect loss limited by timeout
  - » How to select timeout?
- ◆ How could you do better than a timeout-based scheme?

# Pipelined transmission

---

- Send multiple packets without waiting for the first to be ACKed
- Reliable, unordered delivery:
  - ◆ Send new packet after each ACK
  - ◆ Sender keeps list of unACK'ed packets and resends after timeout
  - ◆ Receiver same as stop & wait
- Prob: What if packet 2 keeps being lost?
  - ◆ Receiver must buffer all packets after 2
  - ◆ Potential buffer overflow



# Sliding Window flow control

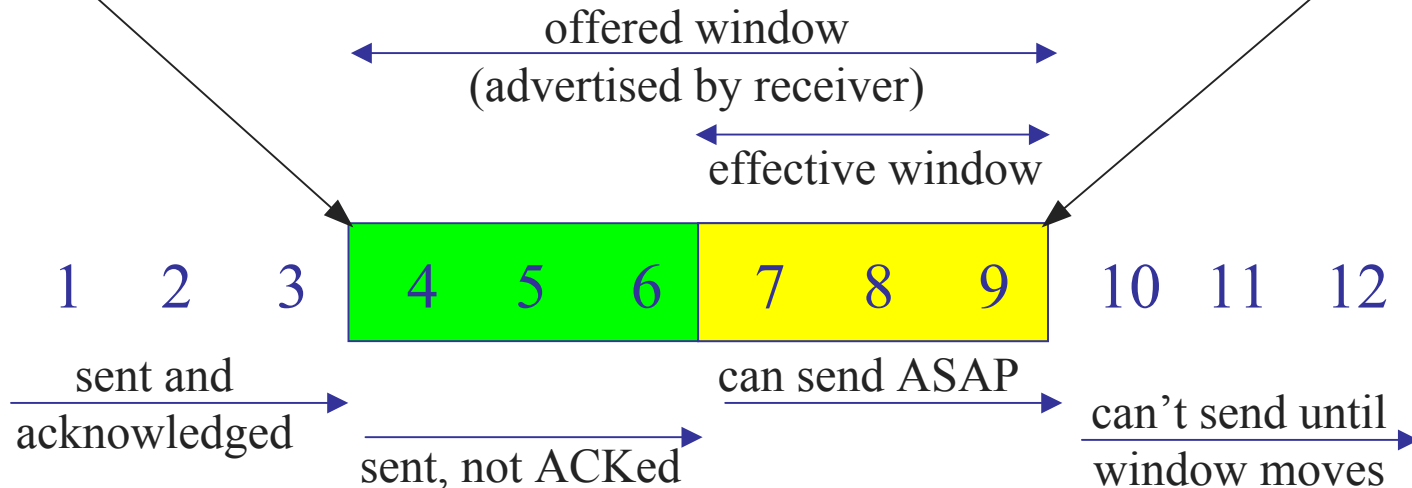
---

- Goal: keep sender from getting too far ahead of lost packets
  - ◆ Receiver advertises amount of buffer space available
  - ◆ Sender can only have this much data outstanding at any time
- Sliding Windows
  - ◆ Offered Window: Available buffer space at receiver
  - ◆ Effective Window: Offered Window – (DataSent – DataAcknowledged)
  - ◆ Sender only sends up to Effective Window limit

# Visualizing a Sliding Window

Left side of window advances when data is ACKed

Right side controlled by size of Advertised Window



# What if we lose a packet?

---

- **Go back N**
  - ◆ Receiver provides cumulative ACKs “got up through packet k”
    - » If multiple packets received, only one ACK needed
  - ◆ OK for receiver to buffer out of order packets
    - » Should you send an ACK for out-of-order packets?
  - ◆ On timeout, sender restarts from k+1
- **Selective acknowledgement (SACK)**
  - ◆ Receiver sends ACK for each packet in window
  - ◆ On timeout, sender resends only the missing packet

# Rough algorithm

---

- Sender
  - ◆ Send full window, set timeout
  - ◆ On ACK:
    - » If it is greater than the last acknowledgement then increase the effective window by the difference
    - » Send additional data up to effective window
  - ◆ On timeout:
    - » Resend next packet not yet ACKed
- Receiver
  - ◆ On packet arrival:
    - » If packet is next contiguous packet, then ACK and deliver contiguous buffer to application
    - » Else, if packet is outside window then drop, else buffer
      - Optionally ack

# Can we shortcut the timeout?

---

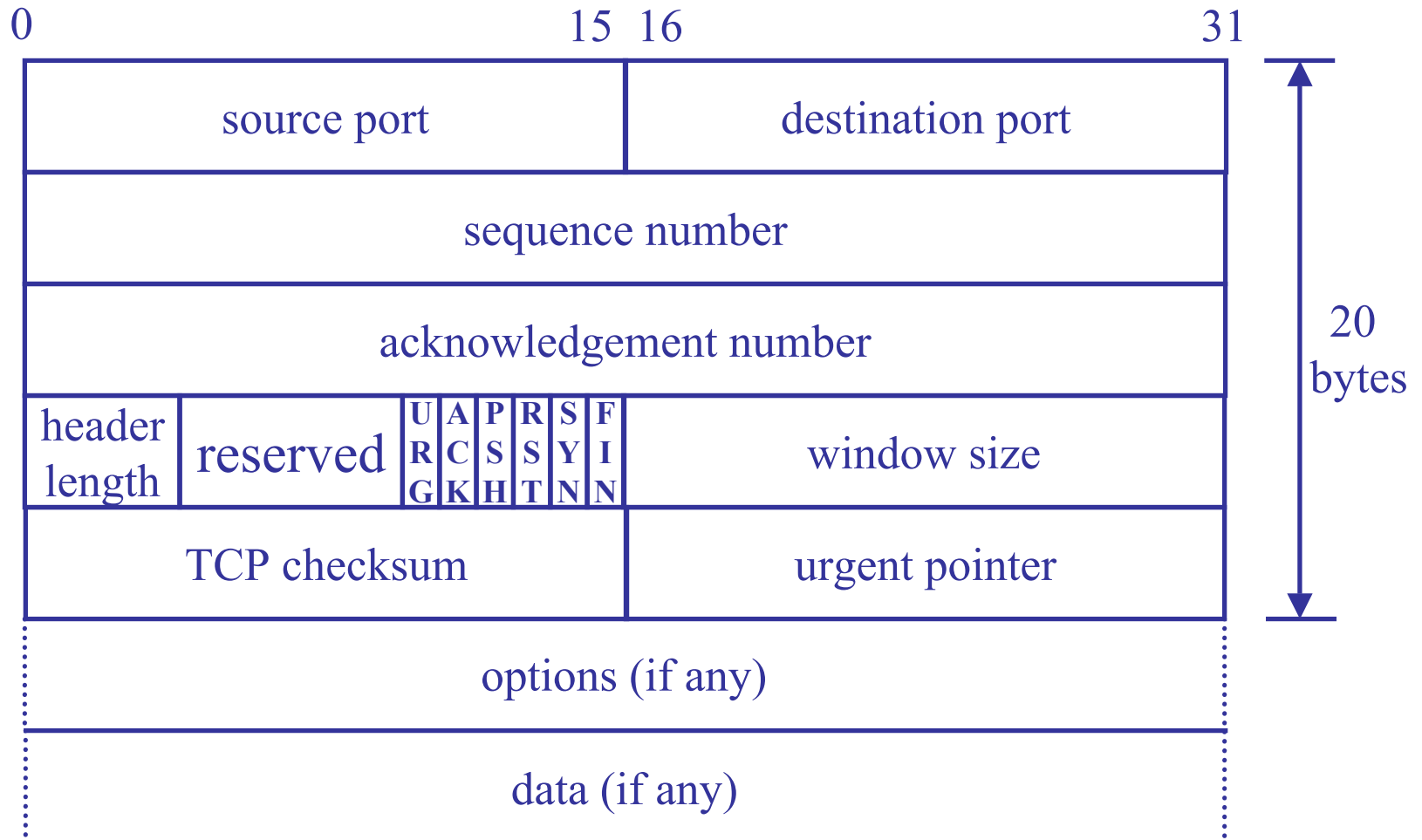
- Problem: If a packet is dropped in the network, the sender has to wait until timeout occurs before reacting
- If packets usually arrive in order, an out of order arrival is *evidence* that a packet might have been lost
  - ◆ Negative ACK
    - » Receiver requests missing packet
  - ◆ Fast retransmit
    - » Receiver ACKs out-of-order packets with seq# of last contiguous packet
    - » When sender receives multiple **duplicate** acknowledgements resends missing packet

# How TCP works today?

---

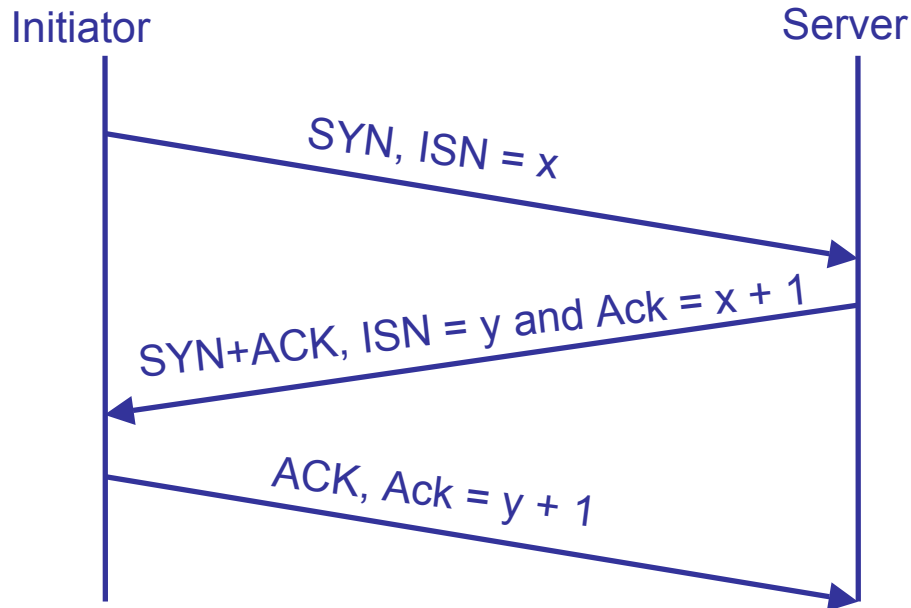
- **Packet format**
  - ◆ Process demultiplexing
  - ◆ Data checksum
  - ◆ Options
- **Connection management**
- **Flow control**

# TCP Packet Format



# TCP Connection Establishment

- How to synchronize initial sequence numbers on each side?
  - ◆ Touched on in Cerf&Kahn, but they didn't know how to do it right
  - ◆ Tomlinson invents three-way handshake in 1975



- Why do we need the last ACK?

# TCP Flow control

---

- Sliding window
  - ◆ Byte granularity for sequence numbers and advertised window
  - ◆ Pro/con of bytes vs packets?
- Go-Back-N where out-of-order packets buffered
- Fast retransmit (invented for TCP in 1988)
- SACK option (just becoming widespread)
  
- Lots of icky details
  - ◆ Window probes, Silly Window Syndrome, Nagel algorithm, MTU discovery, PAWS, etc...
  - ◆ Steven's Book series, "TCP/IP Illustrated" is a great source for these details

# Example Icky Detail: Advertised Window Deadlock

---

- If the receiving process does not empty the buffer (e.g., not scheduled), then the sender fills up the receiver's buffer
  - ◆ Advertised Window is 0
  - ◆ Effective Window goes to 0 when all data is ACKed
- Problem: When can the sender start sending again?
  - ◆ No timeouts because all data is ACKed
  - ◆ No packets from receiver with a new Advertised Window because receiver isn't running
- Solution: Ping with a segment of 1 byte of data
  - ◆ Eventually receiver responds with a new Advert. Window

# And now for something completely different...

---

- When is retransmission not ideal?
  - ◆ Delay sensitivity (e.g. streaming audio)
  - ◆ High-loss, high-delay channels (e.g. satellite)
  - ◆ One-way channels (e.g. television)
  - ◆ Multiple-receivers (e.g. multicast)
  
- Alternative: forward error correction (FEC)
  - ◆ Example: Byers et al98, “A Digital Fountain Approach to Reliable Distribution of Bulk Data”
  - ◆ Some figures and notes courtesy of Byers et al.

# Bulk data distribution

---

- Applications
  - ◆ New release of widely used software (e.g. Internet Explorer)
  - ◆ Push “hot” data to Web caches
  - ◆ Popular video distribution
- Key qualities
  - ◆ Lots of data (10-100MBs)
  - ◆ Multiple receivers – use Multicast for efficiency
  - ◆ Heterogeneous receivers (modems to ethernet)

# Why not use retransmission?

---

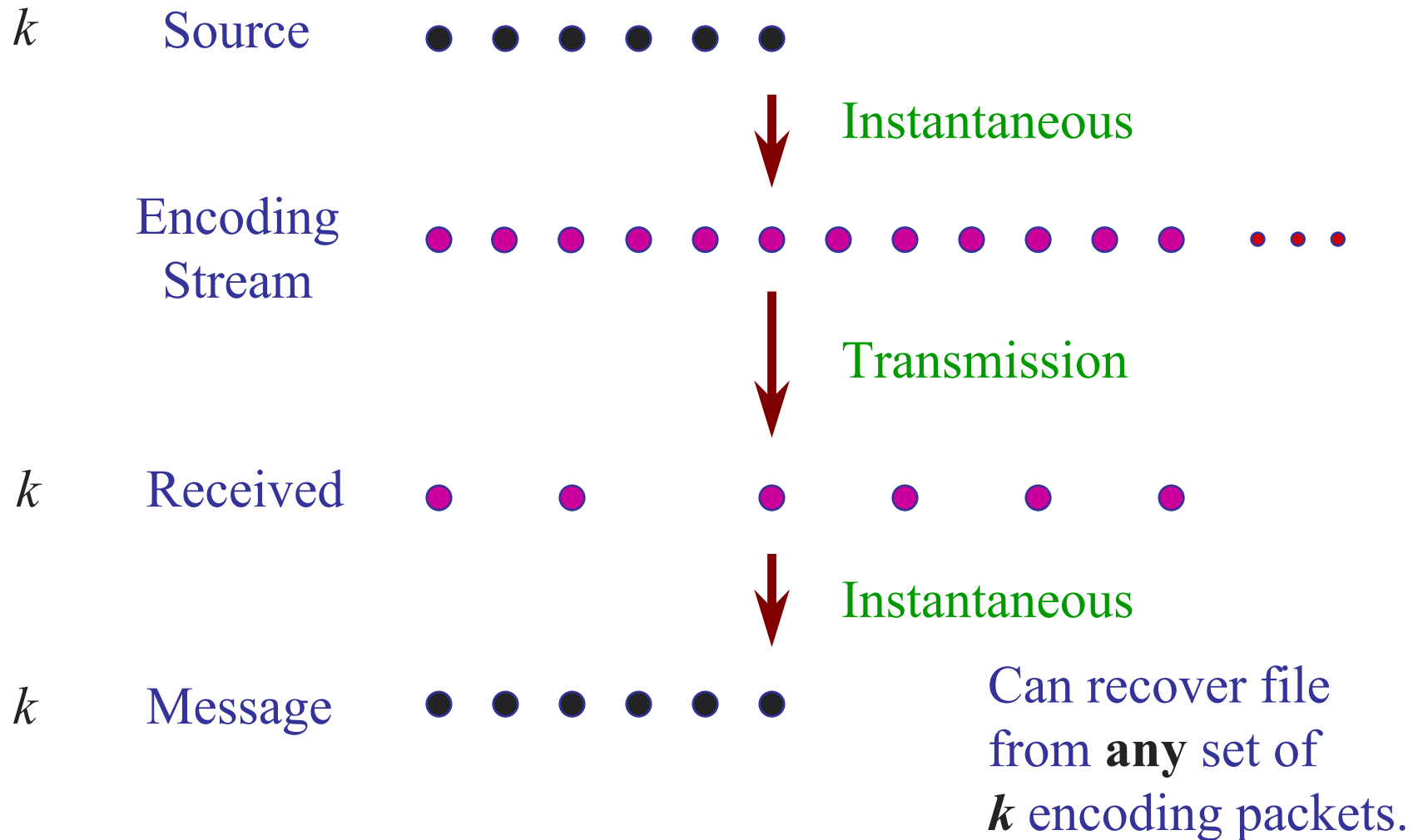
- Lots of receivers
  - ◆ If each one ACK/NAK then hard to scale
    - » Lots of messages
    - » Lots of state
  - ◆ Sender must buffer until all receivers acknowledge
  - ◆ New receivers can't easily join
  - ◆ Loss rates will be high on low BW channels

# Digital Fountain approach

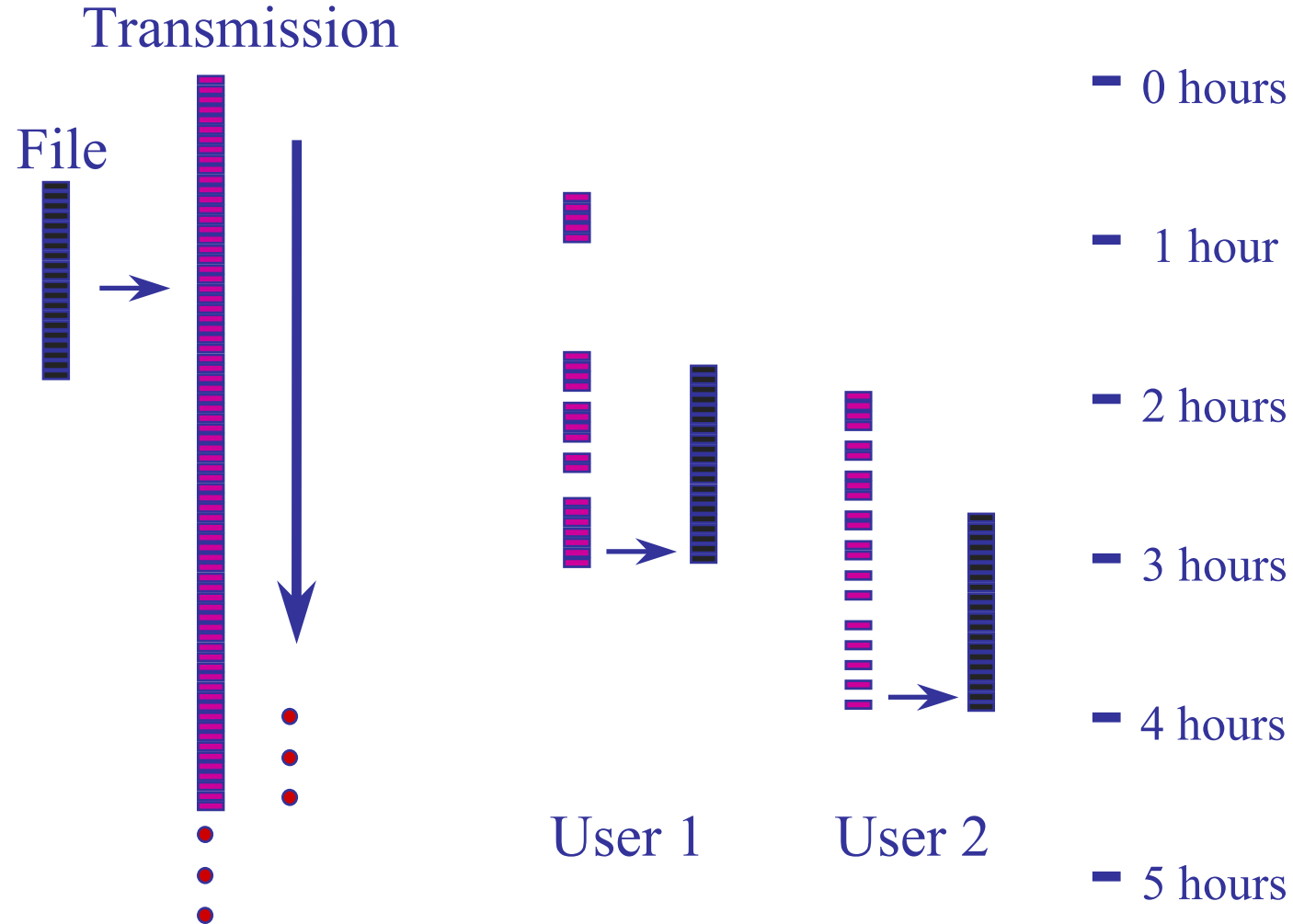
---

- Idea behind Digital Fountain
  - ◆ Use erasure code to redundantly encode  $k$  source packets
  - ◆ Multicast encoded packets continually
  - ◆ Any receiver can reconstruct message from any  $k$  packets
- Low-server load
  - ◆ Client can initiate, leave, continue download without synchronizing with sender
- Does not require two-way communication
  - ◆ Client never communicates with sender
- Tolerates packet loss well
- Low network load for large #'s of clients

# Digital Fountain



# Digital Fountain solution



# Erasure codes

---

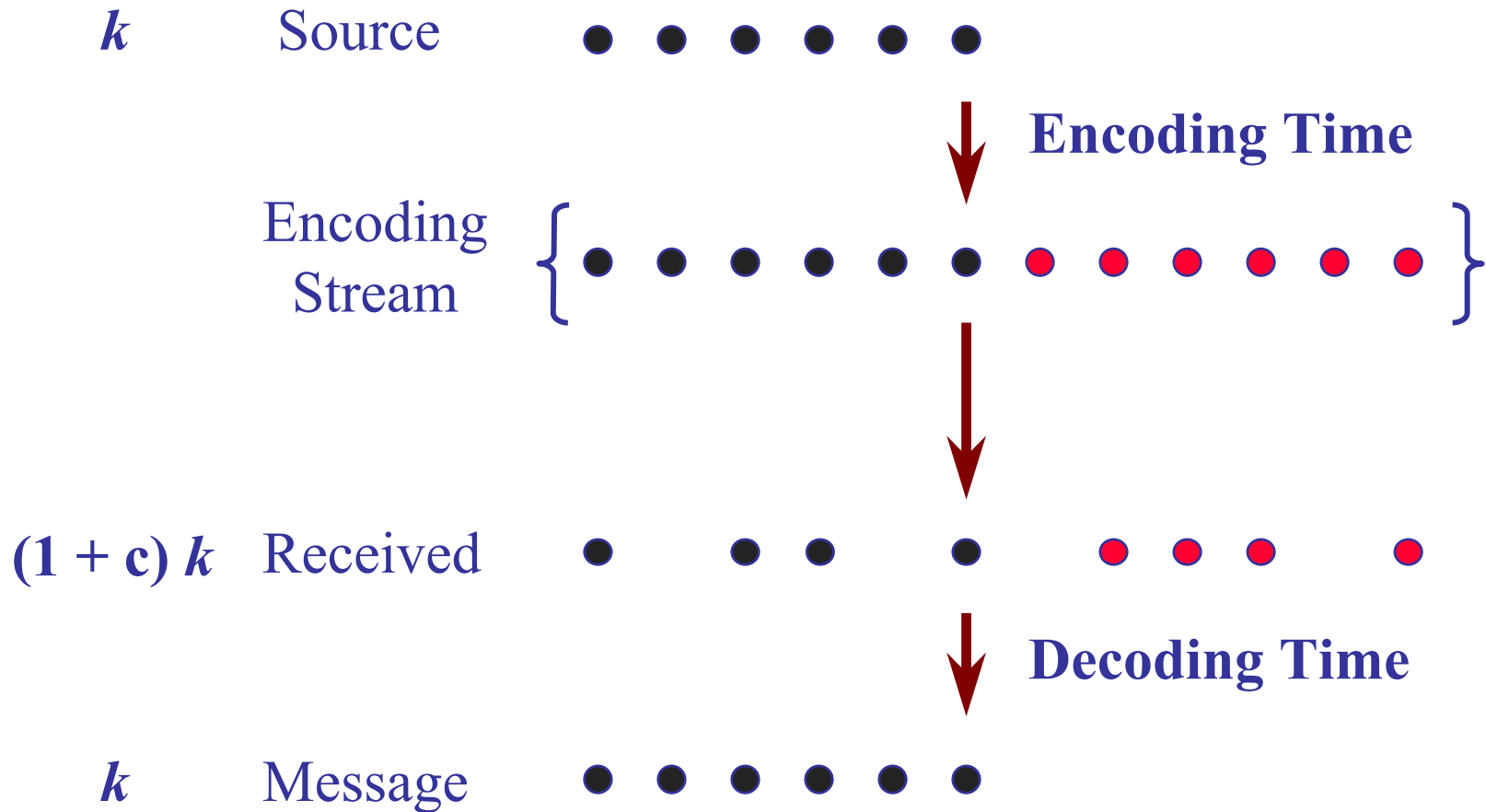
- **Reed-Solomon Codes**

- ◆ Based on dense system of linear equations
- ◆ High decoding complexity (quadratic in  $k$ )
- ◆ Optimal decoding efficiency (1)

- **Tornado Codes**

- ◆ Based on sparse system of equations
- ◆ Low decoding complexity (linear in  $k$ )
- ◆ Suboptimal decoding efficiency ( $1+c$ )
- ◆ But not that bad... ( $c$  usually  $< 0.05$ )

# Approximating a digital fountain



# Timing Comparison

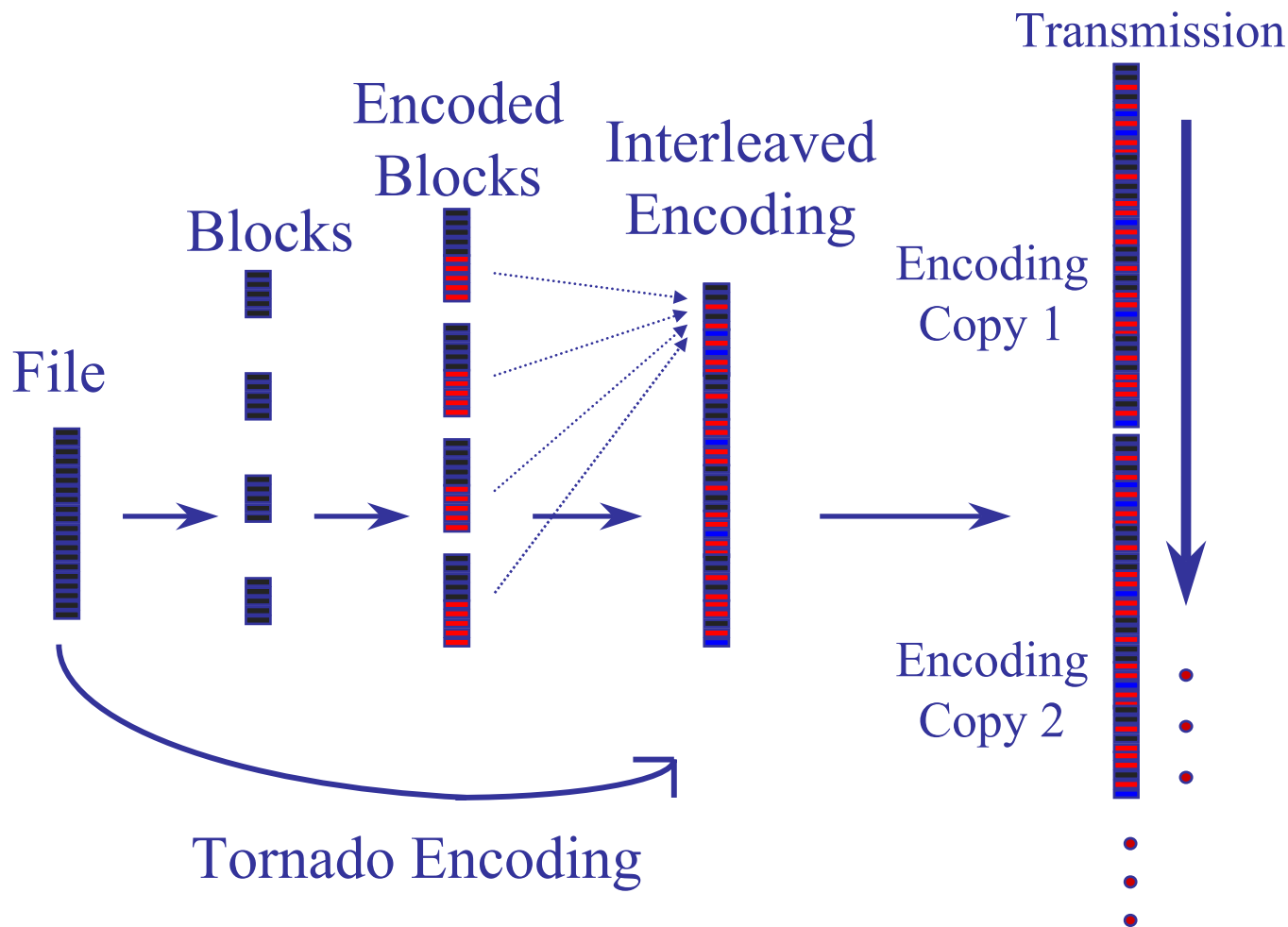
Encoding time, 1K packets		
Size	Reed-Solomon	Tornado Z
250 K	4.6 sec.	0.11 sec.
500 K	19 sec.	0.18 sec.
1 MB	93 sec.	0.29 sec.
2 MB	442 sec.	0.57 sec.
4 MB	30 min.	1.01 sec.
8 MB	2 hrs.	1.99 sec.
16 MB	8 hrs.	3.93 sec.

Decoding time, 1K packets		
Size	Reed-Solomon	Tornado Z
250 K	2.06 sec.	0.18 sec.
500 K	8.4 sec.	0.24 sec.
1 MB	40.5 sec.	0.31 sec.
2 MB	199 sec.	0.44 sec.
4 MB	13 min.	0.74 sec.
8 MB	1 hr.	1.28 sec.
16 MB	4 hrs.	2.27 sec.

Tornado Z: Average inefficiency = 1.055

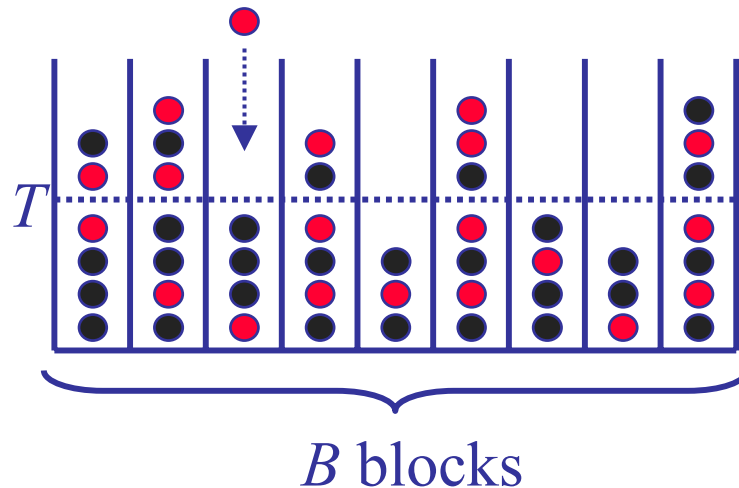
Both codes: Stretch factor = 2

# Interleaving to reduce cost of RS coding



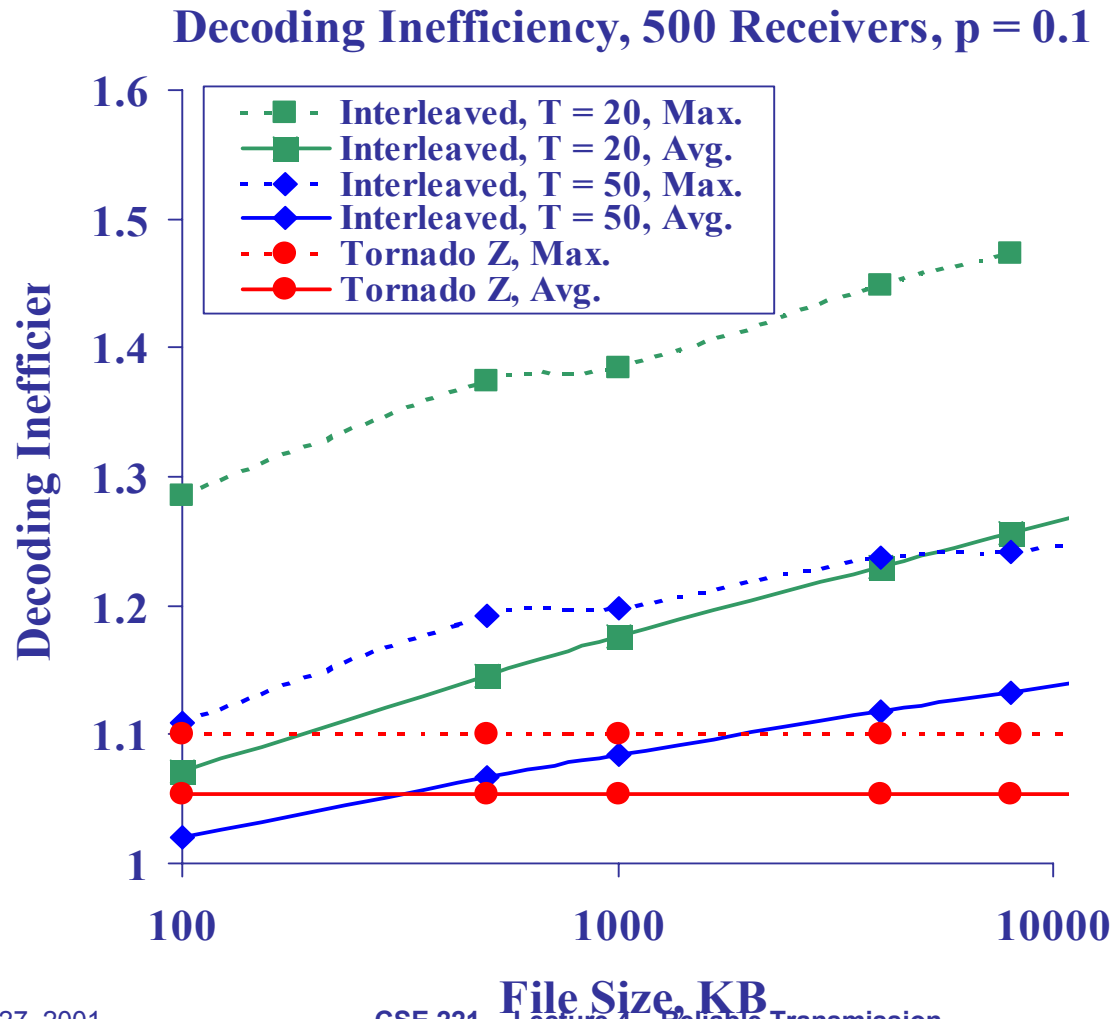
# Drawbacks of Interleaving

- The Coupon Collector's Problem
  - ◆ Waiting for packets from the last blocks:



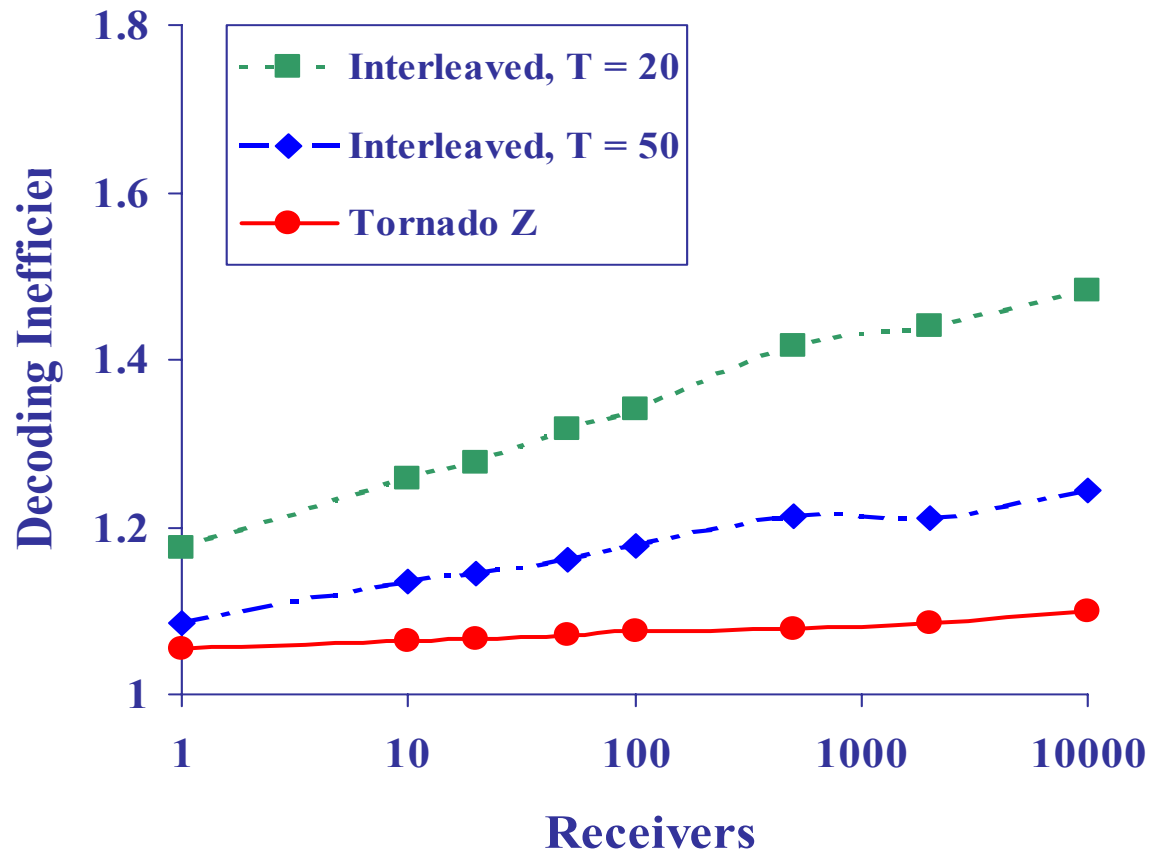
- ◆ More blocks: **faster** decoding, **larger** inefficiency

# Scalability over File Size



# Scalability over Receivers

Decoding Inefficiency on a 1MB File,  $p = 0.1$



# Pros and Cons of Forward Error Correction

---

- Pro
  - ◆ Every packet can be useful for all clients
  - ◆ Well suited to multicast situation
- Con
  - ◆ Sends more data than ideally necessary
  - ◆ Need large block sizes for efficiency

# Summary

---

- Internetworking
  - ◆ Common layer for managing network-level heterogeneity
- Reliable transmission
  - ◆ Detect loss and retransmit
    - » Duplicate detection
    - » Sliding windows
    - » Flow control
  - ◆ Send redundant data and reconstruct
    - » No channel from receiver to sender
    - » Appropriate for large data items in particular settings (multicast)

# For Next Time...

---

- Today we talked about **what** to send... next time we'll talk about **how fast** to send it
- Read 6.3-6.4.1
- Ramakrishnan&Jain and Jacobson&Karels papers
- Next time I'll talk a bit about projects