CSE 422 Notes, Set 3

- Additional figures are repeated, with permission, from Computer Networks, 2nd through 4th Editions, by A. S. Tanenbaum, Prentice Hall.
- The remainder of the materials were developed by Philip McKinley and Dennis Phillips at Michigan State University

Assignment:

Kurose-Ross: 3.1-3.5, 3.7, 3.8
Outline

- Development of reliable protocol
- Sliding window protocols
  - Go-Back-N, Selective Repeat
- Protocol performance
- Sockets, UDP, TCP, and IP
- UDP operation
- TCP operation
  - connection management
  - flow control
  - congestion control

Transport services and protocols

- Transport layer: essence of computer networking
- provides *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into *segments*, passes to network layer
  - receive side: reassembles segments into messages, passes to app layer
  - Internet choices: UDP and TCP
Internet transport-layer protocols

- **Connectionless, unreliable, unordered delivery: UDP**
  - no-frills extension of “best-effort” IP
- **Connection-oriented reliable, in-order delivery (TCP)**
  - connection setup
  - flow control
  - congestion control
- **services not available:**
  - delay guarantees
  - bandwidth guarantees

Reliable Protocol Development

- **Key issue in computer networking:**
  - Implementing reliable connection over an unreliable **network**
- **“Unreliable” can mean:**
  - Lost packets
  - Damaged packets
  - Duplicate packets
  - Out of order packets
  - Delayed packets
Building reliability

- We will develop a protocol
  - one step at a time
  - start with set of unrealistic assumptions
  - Remove them one at a time
  - Correspondingly adapt the protocol
- Result: protocol that delivers reliability over an unreliable network.
- For now, let us assume the sender and receiver are directly connected (link layer)
- Could just as easily be a network (transport layer)

Unrestricted Simplex Protocol

- Assumptions
  - Data transmitted in only one direction
  - Transmitting and receiving network layers always ready
  - Negligible processing time
  - Infinite buffer space
  - No lost or damaged packets
- Of course, this is unrealistic.
- What “protocol” is needed?
Unrestricted Simplex protocol

Simplex Stop-and-Wait Protocol

- **Assumptions**
  - Data transmitted in only one direction
  - No lost or damaged packets

- **Main concern: sender flooding receiver**
  - Solution: receiver sends dummy packet as acknowledgement (ACK)
  - Sender must ACK before sending next packet
  - Packets travel both directions, information only one
  - Half-duplex physical channel suffices
**Simplex Stop-and-Wait Protocol**

![Diagram of Simplex Stop-and-Wait Protocol]

**Simplex Protocol for Noisy Channel**

- **Assumptions**
  - Packets may be damaged or lost
  - Receiver can (almost always) detect damaged packets using checksum

- **We could use previous protocol**
  - Problem occurs when what happens?
  - Result?
  - Solution?
Simplex Protocol for Noisy Channel

Positive Acknowledgement with Retransmission

- Still simplex data transmission
  - 1-bit sequence number used
  - receiver expects particular sequence number next
  - packets with wrong number are rejected as duplicates
- Handles damaged and lost packets
- Potential problem?
Simplex Protocol for Noisy Channel

Case 1
Request
Timeout

Time

Frame 0

ACK 1

Arrive

Case 2
Request
Timeout

Request

Frame 0

Lost

Frame 0

ACK 1

Arrive
Protocol Improvements

- The resulting stop-and-wait protocol is reliable, that is, it will handle all problems except CRC failures (later).
- Nonetheless, this protocol has a serious potential problem. What?

Solution:

Performance problem

Stop and wait protocol (Automatic Repeat reQuest -- ARQ) works, but performance can be poor.

- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

\[ d_{trans} = \frac{L}{R} = \frac{8000\text{bits}}{10^9\text{bps}} = 8\text{microsec} \]

- \( U_{\text{sender}} \): utilization - fraction of time sender busy sending

\[ U_{\text{sender}} = \frac{L/R}{\text{RTT} + L/R} = \frac{.008}{30.008} = 0.00027 \]

- 1KB pkt every 30 msec \( \rightarrow \) 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!
Stop-and-wait timing diagram

\[ U_{\text{sender}} = \frac{L / R}{\text{RTT} + L / R} = \frac{0.008}{30.008} = 0.00027 \]

Other Improvements

- Use the same channel for data in both directions
- acks, naks, data, other control packets all transmitted in both directions.
- Sending short control packets is wasteful, however.
- The solution?
- Advantages?
- Other way to improve performance?
Sliding Window Protocols

Sender allows multiple, “in-flight”, yet-to-be-acknowledged packets, before stopping and waiting

- range of sequence numbers must be increased
- buffering at sender and/or receiver

- Sometimes called “pipelined protocols” (Kurose)

![Diagram: a stop-and-wait protocol in operation](a)

![Diagram: a pipelined protocol in operation](b)

Pipeline size = 3

First packet bit transmitted, \( t = 0 \)

Last bit transmitted, \( t = \frac{L}{R} \)

RTT

First packet bit arrives

Last bit of 2nd packet arrives, send ACK

Last bit of 3rd packet arrives, send ACK

Increase utilization by a factor of 3!

\[
U_{\text{sender}} = \frac{3 \times \frac{L}{R}}{\text{RTT} + \frac{L}{R}} = \frac{0.024}{30.008} = 0.0008
\]
Sliding Window Protocols

- Note: For simplicity, we number packets. Later we will see that TCP numbers bytes.

- Definitions
  - sequence numbers range from 0 to $2^n - 1$
  - at any instant of time, the sender maintains list of consecutive sequence numbers corresponding to packets it is permitted to send; these packets fall within the sending window
  - the receiver maintains a receiving window corresponding to packets it is permitted to accept
  - windows need not be same size

- 1-bit sliding window protocol is equivalent to?

Sender Operation

- sequence numbers within sender’s window represent packets sent but not yet acknowledged
- receipt of ack bumps lower edge of window
- sender keeps copies of all packets currently in the window, in case it needs to retransmit them
- when window grows to maximum size, sender cannot accept packets from network layer until window shrinks
**Receiver Operation**

- sequence numbers within receiver window represent packets that may be accepted
- packets with sequence numbers outside the window are dropped
- received packet (with seq. number equal to lower edge of window) is accepted and window is shifted
- window size remains fixed

**Sliding Window Variations**

- **Go-Back-N** - receiver discards all subsequent packets (subsequent sequence numbers) following an error, forcing the sender to go back to the damaged/lost packet, send it and all subsequent packets
- **Selective repeat** - receiver stores correct packets following bad one. sender retransmits only bad packet, receiver acks highest received in order.
- For go-back-n, receiver window size?
- For selective repeat, window size?
**GBN vs. SR**

- Which is preferred if:
  - Propagation delay is low and error rate is high? Why?
  - Propagation delay is high and error rate is high? Why?
  - Propagation delay is high and error rate is low? Why?

---

**Go-Back-N**

**Sender:**
- k-bit seq # in pkt header
- “window” of up to N, consecutive unACKed pkts allowed

- ACK(n): ACKs all pkts up to, including seq # n - “cumulative ACK”
  - may receive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- timeout(n): retransmit pkt n and all higher seq # pkts in window
**GBN in action**

```
sender
| send pkt0
| send pkt1
| send pkt2
| send pkt3 (wait)
| rcv ACK0
| send pkt4
| rcv ACK1
| send pkt5

receiver
| rcv pkt0
| send ACK0
| rcv pkt1
| send ACK1
| rcv pkt3, discard
| send ACK1
| rcv pkt4, discard
| send ACK1
| rcv pkt5, discard
| send ACK1
| rcv pkt2, deliver
| send ACK2
| rcv pkt3, deliver
| send ACK3
```

**Go-Back-N Simulation**

https://media.pearsoncmg.com/aw/ecs_kurose_compnetwork_7/cw/content/interactiveanimations/go-back-n-protocol/index.html
Selective Repeat

- Packets arriving out of order are buffered:
  - as long as they are within receive window
  - eventually delivered in-order to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt

Selective Repeat (cont)

- Sender window
  - Starts at 0 and grows to X
  - Contains N consecutive seq #'s
  - again limits seq #'s of sent, unACKed pkts
- Receiver window
  - Fixed at size X, buffer for each seq # in window
  - empty/full bit flag for each buffer
  - packet arrives, seq number checked to see if in window, check buffer to see if we already have it; if not, store
Selective Repeat

- **Other features**
  - after in-sequence data packet arrives, start timer. send explicit ack if no reverse traffic before timer fires
  - send negative ack (nak) if packet arrives damaged or out of order; send only one nak per lost packet

- **Are naks most useful when variance of round-trip delay is large or small?**

- **Why?**
Selective repeat in action

Window Constraints for SR?
Selective repeat: dilemma

example:
- seq #’s: 0, 1, 2, 3
- window size=3
  - receiver sees no difference in two scenarios!
  - duplicate data accepted as new in (b)

Q: what relationship between seq # size and window size to avoid problem in (b)?

Next up...

- We now understand the basics of:
  - reliable communication
  - stop and wait protocol
  - sliding window protocols

- Next, let us apply this knowledge in order to estimate network performance under various
  - protocol parameter settings
  - situations (e.g., propagation delay, error rate)