CSE/EE 461

Sliding Windows and ARQ
Slowly Spinning Wheels

Lockstep
Stop-and-Wait

- Only one outstanding packet at a time
- Also called alternating bit protocol
- Reliability
- Flow Control
Limitation of Stop-and-Wait

- Lousy performance if wire time $<<$ prop. delay
  - Max BW: $B$
  - Actual BW: $M/2D$
    - Example: $B = 100\text{Mb}/\text{s}$, $M=1500\text{Bytes}$, $D=50\text{ms}$
    - Actual BW = $1500\text{Bytes}/100\text{ms} \rightarrow 15000\text{ Bytes}/\text{s} \rightarrow 100\text{Kb}/\text{s}$
    - $100\text{Mb}$ vs $100\text{Kb}$?
More BW Please

• Want to utilize all available bandwidth
  – Need to keep more data “in flight”
  – How much? Remember the bandwidth-delay product?
• Leads to Sliding Window Protocol
• Window size says how much data can be sent without waiting for an acknowledgement
Sliding Window

- Sender can send a lot of data before waiting for an ack
  - Amount of data is the window size
    - #pkts, or #bytes, depending
- Sender tries not to send more data than the receiver can handle
  - Window size - sizeof(unacknowledged data)
- It supports multiple functions:
  - Reliable delivery
    - *If I hear you got it, I know you got it.*
    - ACK (Ack # is “next byte expected”)
  - In-order delivery
    - *If you get it, you get it in the right order.*
    - SEQ # (Seq # is “the byte this is in the sequence”)
  - Flow control
    - *If you don’t have room for it, I won’t send it.*
Sliding Window – Sender

- Window bounds outstanding data
  - Implies need for buffering at sender
    - Specifically, must buffer unacked data
- “Last” ACK applies to in-order data
  - Need not buffer acked data
- Sender maintains timers
  - Go-Back-N: one timer, send all unacknowledged on timeout
  - Selective Repeat: timer per packet, resend as needed
Sliding Window – Timeline

Receiver ACK design choices:
- Individual
  - Each packet acked
- Cumulative (TCP)
  - Ack says “got everything up to X-1…”
  - really, “my ack means that the next seq# I am expecting is X”
- Selective (newer TCP)
  - Acks says “I got X through Y”
  - Easier to keep pipe full with unreceived data
  - More complex
- Negative
  - Acks says “I did not get X”
  - Decrease retransmission time
Sliding Window – Receiver

- Receiver buffers too:
  - data may arrive out-of-order
  - or faster than can be consumed by receiving application
- Drop?
- No sense having more data on the wire than can be buffered at the receiver.
  - In other words, current receiver buffer size limits the window size
Flow Control

- Sender must transmit data no faster than it can be consumed by the receiver
  - Receiver might be a slow machine
  - App might consume data slowly

- Implement by adjusting the size of the sliding window used at the sender based on receiver feedback about available buffer space
  - Receiver “advertises” its receive window
  - Piggyback on the ack
One more thing…

- Decouple sending application from sending protocol
- Sender needs to buffer messages anyway in order to resend
- Each side maintains some local and remote buffer state and invariants
- Local buffer state is correct
- Remote buffer state is conservative
- Sending, receiving, reading and writing are allowed to perform according to the states of the buffers and the invariants
  - Invariants --> allowed behavior
Sender and Receiver Buffering

Sending application

Older bytes write Newer bytes

TCP

LastByteWritten

LastByteAced LastByteSent

= available buffer

Receiving application

Older bytes read Newer bytes

TCP

LastByteRead

NextByteExpected LastByteRcvd

= buffer in use

LastByteAced ≤ LastByteSent
LastByteSent ≤ LastByteWritten

LastByteRead < NextByteExpected
NextByteExpected ≤ LastByteRcvd+1
== if data arrives in order
else start of first gap.
Flow Control

**Last Byte Received - Last Byte Read \(\leq\) Max Receive Buffer**

**Advertised Window = Max Rcv Buffer - ((Next Byte Expected - 1) - Last Byte Read**

“All the buffer space minus the buffer space that’s in use.”

As data arrives, receiver acknowledges it so long as all preceding bytes have also arrived. Advertised Window potentially shrinks depending on how fast receiving app is drawing out Data.
Flow Control On the Sender

**EffectiveWindow** = **AdvertisedWindow** - (**LastByteSent** - **LastByteAcked**)  

*OK to send that which there is room for, which is that which was advertised minus that which I’ve already sent since receiving the last advertisement.*
Sending Side -- One last detail

**Sender and Receiver Buffering**

- MaxSendBuffer
- MaxRcvBuffer

**LastByteWritten - LastByteAcked <= MaxSendBuffer**

_Can only hang on to unsent and unacked data if there’s room for it._

==> _BLOCK write(y) if_

(LastByteWritten - LastByteAcked) + y > MaxSendBuffer
Receiving Side -- One last detail

sender and receiver buffering

MaxSendBuffer

MaxRcvBuffer

Sender and Receiver Buffering

Sending application

TCP

LastByteWritten

LastByteAcked

LastByteSent

Receiving application

TCP

LastByteRead

LastByteReceived

NextByteExpected

LastByteRcvd

\[ \text{LastByteRead} < \text{NextByteExpected} \]

Can’t read data if it hasn’t arrived.

\[ \Rightarrow \text{BLOCK read(y) if} \]

LastByteRead < NextByteExpected
Example – Exchange of Packets

Stall due to flow control here

Receiver has buffer of size 4 and application doesn’t read
Example – Buffer at Sender

<table>
<thead>
<tr>
<th>T=1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>T=2</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>T=3</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>T=4</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>T=5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>T=6</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
</tbody>
</table>

- **Green** = acked
- **Orange** = sent
- **Yellow** = advertised
- **Brown** = queued
TCP Packet Format

16 bit window size gets cramped with large Bandwidth x delay

16 bits --> 64K
BD ethernet: 122KB
STS24 (1.2Gb/s): 14.8MB

32 bit sequence number must not wrap around faster than the maximum packet lifetime. (120 seconds)
-- 622Mb/s link: 55 seconds

What to do?
The IP Packet

<table>
<thead>
<tr>
<th>MAC header</th>
<th>IP header</th>
<th>Data</th>
</tr>
</thead>
</table>

**IP header:**

<table>
<thead>
<tr>
<th>00 00 00 00 00 00</th>
<th>00</th>
<th>00 00 00 00 00 00</th>
<th>00 00 00 00 00 00</th>
<th>00 00 00 00 00 00</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version</td>
<td>IHL</td>
<td>TOS</td>
<td>Total length</td>
<td>Identification</td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Header checksum</td>
<td>Source IP address</td>
<td>Destination IP address</td>
</tr>
</tbody>
</table>

**TCP Packet Format**

- 32 bits
- Src Port #
- Dest Port #
- Sequence #
- Acknowledgement #
- Urgent Ptr
- Flags
- Window Size
- Checksum

Options

Data

Camp 408 - Spring 2008
An Entire Ether Packet

The most common Ethernet Frame format, type II

IP header:

```
00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Version TOS Identification Flags Fragment offset
IHL Protocol Header checksum
0x0800: 0035 9367 3f30 000a 95c9 340a 0800 4500 ...
```

TCP Packet Format

```
Source Port #                  Destination Port #
32 bits                        32 bits
Acknowledgement #              Sequence #
Flags                          Window Size
Checksum                       Urgent Ptr
Options                        Data
```

tcpdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on en0, link-type EN10MB (Ethernet), capture size 96 bytes
```
0x0000: 0003 9367 3f30 000a 95c9 340a 0800 4500 ...
0x0010: 0035 9367 3f30 000a 95c9 340a 0800 4500 ...
0x0020: 035f f874 0050 6a5f dc42 0000 0000 0a02 ...
0x0030: ffff 5b38 0000 0204 05b4 0103 0300 0101 ...
0x0040: 080a 6a78 b5d6 0000 0000 ...
```
# EtherTypes

*Links: Ethernet assigned numbers.*

<table>
<thead>
<tr>
<th>EtherType</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>IEEE 802.3 length.</td>
</tr>
<tr>
<td>0x0000</td>
<td>DLOG.</td>
</tr>
<tr>
<td>0x0060</td>
<td>XEROX NS IPD.</td>
</tr>
<tr>
<td>0x0061</td>
<td>X.75 Internet.</td>
</tr>
<tr>
<td>0x003D</td>
<td>ECMA Internet.</td>
</tr>
<tr>
<td>0x0044</td>
<td>Chaosnet.</td>
</tr>
<tr>
<td>0x0055</td>
<td>X 25 Level 3.</td>
</tr>
<tr>
<td>0x0010</td>
<td>ARP, Address Resolution Protocol.</td>
</tr>
<tr>
<td>0x0806</td>
<td>DRARP, Dynamic RARP.</td>
</tr>
<tr>
<td>0x0807</td>
<td>RARP, Reverse Address Resolution Protocol.</td>
</tr>
<tr>
<td>0x080F3</td>
<td>AppleTalk Address Resolution Protocol.</td>
</tr>
<tr>
<td>0x8100</td>
<td>EAPS, Ethernet Automatic Protection Switching.</td>
</tr>
<tr>
<td>0x8137</td>
<td>IPX, Internet Packet Exchange.</td>
</tr>
<tr>
<td>0x814C</td>
<td>SNMP, Simple Network Management Protocol.</td>
</tr>
<tr>
<td>0x86DD</td>
<td>IPX, Internet Protocol version 6.</td>
</tr>
<tr>
<td>0x880B</td>
<td>PPP, Point-to-Point Protocol.</td>
</tr>
<tr>
<td>0x880C</td>
<td>GSMP, General Switch Management Protocol.</td>
</tr>
<tr>
<td>0x8847</td>
<td>MPLS, Multi-Protocol Label Switching (unicast).</td>
</tr>
<tr>
<td>0x8849</td>
<td>MPLS, Multi-Protocol Label Switching (multicast).</td>
</tr>
<tr>
<td>0x8863</td>
<td>PPTP, PPP over Ethernet (Discovery Stage).</td>
</tr>
<tr>
<td>0x8864</td>
<td>PPPoE, PPP over Ethernet (PPP Session Stage).</td>
</tr>
<tr>
<td>0x88BB</td>
<td>LWAPP, Light Weight Access Point Protocol.</td>
</tr>
<tr>
<td>0x88CC</td>
<td>LLDP, Link Layer Discovery Protocol.</td>
</tr>
<tr>
<td>0x8E38</td>
<td>EAPOL, EAP over LAN.</td>
</tr>
<tr>
<td>0xFF</td>
<td>reserved.</td>
</tr>
</tbody>
</table>

---

TCPdump: verbose output suppressed, use -v or --verbose for full protocol decode
listening on eno, link-type EN10MB (Ethernet), capture size 96 bytes
30:16:03.830582 IP 10.0.1.4.63604 > www.cs.washington.edu HTTP/1795821890:1

```
0x0000: 0003 93e7 3f30 000a 95c9 340a 0300 4500 .?.0...4.E.
0x0010: 0003 35e4 4000 4006 75ac 0000 0104 8000 .-5.E.e.u.
0x0020: 0358 f874 0050 6af6 dc42 0000 0000 a002 .X.t.Pj.B...
0x0030: ffff 5b88 0000 0204 05b4 0103 0300 0101 ..[......
0x0040: 0000 6a78 b5d6 0000 0000 ..JX.....
```
Key Concepts

• Transport layer allows processes to communicate with stronger guarantees, e.g., reliability

• Basic reliability is provided by ARQ mechanisms
  – Stop-and-Wait through Sliding Window plus retransmissions