Outline

- Recap
  - Packet processing algorithms (Chapter 5)
- Packet processing algorithms
- Summary and homework
IP Routing/Forwarding

- Used in hosts as well as routers
- Conceptual mapping: table driven
  - $f(\text{datagram, routing table}) \rightarrow (\text{next hop, interface})$
- IP routing/forwarding table
  - One entry per destination
  - Entry contains
    - 32-bit IP address of destination
    - 32-bit address mask
    - 32-bit next-hop address
    - N-bit interface number
### Example IP Routing/forwarding Table

- Values stored in binary
- Interface number is for internal use only
- Zero mask produces default route

<table>
<thead>
<tr>
<th>Destination Address</th>
<th>Address Mask</th>
<th>Next-Hop Address</th>
<th>Interface Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.5.48.0</td>
<td>255.255.255.0</td>
<td>128.210.30.5</td>
<td>2</td>
</tr>
<tr>
<td>128.10.0.0</td>
<td>255.255.0.0</td>
<td>128.210.141.12</td>
<td>1</td>
</tr>
<tr>
<td>0.0.0.0</td>
<td>0.0.0.0</td>
<td>128.210.30.5</td>
<td>2</td>
</tr>
</tbody>
</table>
IP Forwarding Algorithm

Given: destination address A and routing table R.
Find: a next hop and interface used to route datagrams to A.
For each entry in table R {
    Set MASK to the Address Mask in the entry;
    Set DEST to the Destination Address in the entry;
    If (A & MASK) == DEST {
        Stop; use the next hop and interface in the entry;
    }
}
If this point is reached, declare error: no route exists;
IP Fragmentation

- Needed when datagram larger than network MTU
- Divides IP datagram into fragments
- Uses FLAGS bits in datagram header
Fragmentation

Offset specifies 8-byte chunks of data rather than individual bytes
IP Fragmentation Algorithm

Given: an IP datagram, D, and a network MTU.
Produce: a set of fragments for D.
If the \textit{DO NOT FRAGMENT} bit is set \{
  \textit{Stop and report an error;}
\}
Compute the size of the datagram header, $H$;
Choose $N$ to be the largest multiple of 8 such that $H+N \leq \text{MTU}$;
Repeat until datagram empty \{
  Create a new fragment that has a copy of D's header;
  Extract up to the next $N$ octets of data from D and place the data in the fragment;
  Set the \textit{MORE FRAGMENTS} bit in fragment header;
  Set \textit{TOTAL LENGTH} field in fragment header to be $H+N$;
  Set \textit{FRAGMENT OFFSET} field in fragment header to $O$;
  Compute and set the \textit{CHECKSUM} field in fragment header;
  Increment $O$ by $N/8$;
\}
Reassembly

- Complement of fragmentation
- Uses IP SOURCE ADDRESS and IDENTIFICATION fields in datagram header to group related fragments
- Joins fragments to form original datagram

Given: a fragment, F, add to a partial reassembly.
Method: maintain a set of fragments for each datagram.
Extract the IP source address, S, and ID fields from F;
Combine S and ID to produce a lookup key, K;
Find the fragment set with key K or create a new set;
Insert F into the set;
If the set contains all the data for the datagram 
   
   Form a completely reassembled datagram and process it;


Instructor: Dr. Liang Cheng  
CSE398: Network Systems Design  
02/09/05
Data Structure For Reassembly

- Two parts
  - Buffer large enough to hold original datagram
  - Linked list of pieces that have arrived
TCP Connection

- Involves a pair of endpoints
- Started with SYN segment
- Terminated with FIN or RESET segment
- Identified by 4-tuple (src addr, dest addr, src port, dest port)
TCP Connection Recognition Algorithm (partially shown)

Given: a copy of traffic passing across a network.
Produce: a record of TCP connections present in the traffic.
Initialize a connection table, C, to empty;
For each IP datagram that carries a TCP segment {
    Extract the IP source, S, and destination, D, addresses;
    Extract the source, P1, and destination, P2, port numbers;
    Use (S,D,P1,P2) as a lookup key for table C and
    If the segment has the RESET bit set, delete the entry;
    Else if the segment has the FIN bit set, mark the connection
        closed in one direction, removing the entry from C if
        the connection was previously closed in the other;
    Else if the segment has the SYN bit set, mark the connection as
        being established in one direction, making it completely
        established if it was previously marked as being
        established in the other;
}

Problem 11 of Chapter 5 (Page 64).
TCP Splicing

- Join two TCP connections
- Allow data to pass between them
- To avoid termination overhead translate segment header fields
  - Acknowledgement number
  - Sequence number

![Diagram showing TCP splicing between Host A and Host B via a splicer]

<table>
<thead>
<tr>
<th>Connection &amp; Direction</th>
<th>Sequence Number</th>
<th>Connection &amp; Direction</th>
<th>Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Incoming #1</td>
<td>200</td>
<td>Incoming #2</td>
<td>1200</td>
</tr>
<tr>
<td>Outgoing #2</td>
<td>860</td>
<td>Outgoing #1</td>
<td>50</td>
</tr>
<tr>
<td>Change</td>
<td>660</td>
<td>Change</td>
<td>-1150</td>
</tr>
</tbody>
</table>
TCP Splicing Algorithm

Given: two TCP connections.
Produce: sequence translations for splicing the connection.
Compute D1, the difference between the starting sequences
  on incoming connection 1 and outgoing connection 2;
Compute D2, the difference between the starting sequences
  on incoming connection 2 and outgoing connection 1;
For each segment {
  If segment arrived on connection 1 {
    Add D1 to sequence number;
    Subtract D2 from acknowledgement number;
  } else if segment arrived on connection 2 {
    Add D2 to sequence number;
    Subtract D1 from acknowledgement number;
  }
}
Outline

- Recap
- Packet processing algorithms
- Summary and homework
Homework (due on 02/14)

- HW5.1 and HW5.2 are posted at the Blackboard System, which are due on 02/14. No need to hand in the following two questions. However, we will discuss them in the class.
  - Problem 11 of Chapter 5 (Page 64).
  - Scenarios that “do not fragment”.