CSE 461 - Module 4: Dealing with Errors

Errors

- Bit level:
  - Errors
    - Inverted (corrupted) bits
    - Missing bits / extra bits
  - Mechanisms
    - Error detection / Error correction
    - Clock synchronization
- Network level
  - Errors
    - (Corrupted frames)
    - Missing frames
    - Duplicate frames
    - Delayed frames
    - Out of order frames
  - Mechanisms
    - Redundancy
      - Timeout and retransmit
        - ARQ
      - Proactively / selectively retransmit
    - Duplicate detection
      - Naming
        - Sequence numbers
    - Judicious use of names (sequence numbers)
      - Time-to-live (TTL)
    - Re-order buffers
- Protocol level
  - Errors
    - malformed messages
    - disallowed responses
  - Mechanisms
    - ignore

        - send indication and then ignore
- Malicious agents
    - Issues
        - Privacy (intercept messages)
        - Integrity (change message contents)
        - Authenticity (who generated the message?)
        - Authorization (is that who allowed to do what they're trying to do?)
        - Denial of service (DOS)
    - Attacks
        - Man in the middle
        - Buffer overrun
        - Spoofing
        - DNS tainting
        - …
    - Heartbleed exploit
        - http://www.theregister.co.uk/2014/04/09/heartbleed_explained/

Network Errors::Missing Frames::Timeout and Retransmit

- Tanenbaum & Wetherall: Section 3.3
- We'll assume a simple request-response protocol
- There are two potential issues:
    - How does client know the server received the request?
    - When the server receives a request, how does it know that it hasn't missed any earlier ones?
        - Does it care?
- Client side solution: **Automatic Repeat Request (ARQ)**
    - Positive acknowledgements (ACKs)
        - Server sends an ACK only when it hears a request
        - When I hear an ACK, I know the server heard a request
        - When I don't hear an ACK, I know that I don't know whether or not the server heard the request
    - What parts of this are essential to correctness and what parts are “just performance”?
        - Consider leeway in the just performance choices
        - Determining a good set of choices requires extensive experimentation / exper
- Basic ARQ Works for 1st message, doesn't work for second message. Why?
- Naming messages

- UIDs
- Sequence numbers

- Client scheme (artist's rendition):

```
msg = new Message(seqno++);
for (attempt=0; attempt < MAX_ATTEMPTS; attempt++ ) {
   if (<process exiting>) break;
   response = sendBytes(msg);
   if (<response indicates IO error>) throw exception;
   if (<response indicates timeout>) continue;
   if ( response.seqno == msg.seqno ) { <process response>; break; }
   // ignore responses with unexpected sequence numbers
}
if ( attempt >= MAX_ATTEMPTS ) throw exception;
```

- Server's scheme (similarly sketchy):

```
while (1) {
   if (<process exiting>) break;
   msg = readMsg();
   if (<response indicates IO error>) throw exception;
   if (<response indicates timeout>) continue;
   <process message – includes sending appropriate response>
}
```

- Note: we're assuming each client has only one outstanding message in the above
  - Haven't worried about multi-threaded clients
  - Haven't yet implemented “sliding window”
- In more realistic situations, we might want to put buffers between the client and this code
  - Sending
    - Client invokes sendMessage(), which puts message in a queue and then returns
    - A sending thread in an infinite loop removes messages from the queue and sends them
  - Receiving
    - Server loop puts messages in a queue
    - Client calls readMessage(), which removes a message or blocks

- How does client thread wait for response message?