CS 268: Lecture 16

Robust Protocols

Isn’t the Internet Robust?

- Robustness was one of the Internet’s original design goals
- Adopted failure-oriented design style:
  - Hosts responsible for error recovery
  - Critical state refreshed periodically
  - Failure assumed to be the common case
- Internet has withstood some major outages with minimal service interruption
- Why the lecture on robust protocols?

Traditional Failure Models

- Most Internet protocols are design with (at most) two failure models in mind:
  - Participating nodes: fail-stop
  - Other nodes: malicious
    - Denial-of-service, spoofing, etc.
- They are usually vulnerable to participating nodes misbehaving:
  - Subverted nodes
  - Misconfigured nodes
  - Bug in software

Semantic vs Syntactic Failures

- Syntactic failures:
  - Node doesn’t respond, message ill-formed, etc.
- Semantic failure:
  - Node responds with well-formed message, that is semantically incorrect

Traditional Techniques

- Cryptographic authentication:
  - Verifies who is talking, but not what they say
- Formal verification:
  - Verifies that correct protocol operation leads to the desired result
- Fault-tolerance via consensus: (Byzantine techniques)
  - Requires that several nodes have enough information to do the required computation
  - In network routing, for instance, only the nodes at the end of a link know about its existence

Administrivia

- Solutions to midterm posted
  - Avg: 77
  - Std. Dev.: 9
  - Median: 75
- Change in lecture schedule: Four guest lectures
  - Vern Paxson: malware
  - Sean Rhea: OpenDHT
  - Phil Levis: Sensornets
  - Lakshmi: Making routing robust (relevant to today’s lecture)
Example: Arpanet Routing

- Link-state advertisement (LSA) came with sequence number with some maximal value M

- To determine if the sequence number has wrapped, a node compared the arriving number NA to the current number old NC:
  - NA > NC and NA-NC < NC+M-NA => no wrap, newer
  - NA > NC and NA-NC > NC+M-NA => wrap, older
  - NA < NC and NC-NA < NA+M-NC => no wrap, older
  - NA < NC and NC-NA > NA+M-NC => wrap, newer

- Normally, M >> all natural jumps in sequence numbers, so any large gap indicated wrap

Pathological Case

- M=100 and failing router emits LSAs w/ counters: 1, 33, 66

- If NC=1, then NA=33 looks new (and NA=66 looks old)
- If NC=33, then NA=66 looks new (and NA=1 looks old)
- If NC=66, then NA=1 looks new (and NA=33 looks old)

- Thus, these three LSAs live forever!

Fix

- Age LSAs (so they eventually die)

- Wraparound is done explicitly
  - Flush LSA with M, reinsert LSA with 1

Example 2: Routing Misconfiguration

- Router advertises short route to some prefix
- All other routers believe
- Several such instances have taken large portions of the Internet down

Example 3: Congestion Control

- Hosts can ignore TCP CC and send too fast
- Receiver can lie to sender about ECN bits

Traditional Techniques

- Don’t deal with any of these problems…
  - Parties could be authenticated
  - Protocols are correct
  - Can’t use consensus techniques
<table>
<thead>
<tr>
<th><strong>How Can We Avoid These Problems?</strong></th>
<th><strong>G1: Value Conceptual Simplicity</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• No single rule or algorithm</td>
<td>• Obvious, but often unheeded (e.g., BGP)</td>
</tr>
<tr>
<td>• Some general guidelines (presented next)</td>
<td>• Simplicity allows one to reason about behavior more easily</td>
</tr>
<tr>
<td>• Overall theme: design defensively</td>
<td>• Leads to better failure handling</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>G2: Minimize Your Dependencies</strong></th>
<th><strong>G3: Verify When Possible</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• The more nodes you depend on for correct information, the higher the chances for failure are</td>
<td>• Can’t use heavyweight Byzantine-style algorithms</td>
</tr>
<tr>
<td>• Example: Sender trusts receiver for congestion information</td>
<td>• But can try lightweight verification techniques</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>G4: Protect Your Resources</strong></th>
<th><strong>G5: Limit Scope of Vulnerability</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Example 1: SYN flood and SYN cookies</td>
<td>• If system is vulnerable to a failure anywhere else in system, then robustness is unlikely</td>
</tr>
<tr>
<td>• Example 2: Fair queueing in networks</td>
<td>• Route flap damping limits extent to which failures propagate</td>
</tr>
</tbody>
</table>
G6: Expose Errors

- To conflicting goals:
  - Automatically recover
  - Don’t let problems fester

Lightweight Verification

- ECN nonces
- Self-verifying CSFQ
- Listen and Whisper (Guest Lecture)

ENC

- Bit in IP header flipped when routers experience congestion
- Receiver returns this bit back to sender in TCP header
  - Keeps sending bit until sender returns CWR
  - CWR = congestion window reduced
- ECN: replaces packet drops with explicit signaling of congestion
  - Doesn’t require drops
  - No confusion between corruption losses and congestion losses

Problem

- ECN requires receiver to give information back to sender
- If receiver lies (doesn’t return bit), then sender keeps increasing window
- Lying receiver gets more bandwidth than truthful ones or non-ECN-enabled ones

Robust Congestion Signalling

- Use ECN bits in IP header to send to separate signals:
  - Congestion-bit: on or off
  - Nonce: randomly 0 or 1
- When congestion bit is set, nonce is erased
- Receiver must send back cumulative sum of nonces in ACK
- When congestion is signalled, receiver can’t see nonce, so must guess about it
  - If many nonce bits, this is very unlikely
  - With one nonce bit, improbable it can continue to guess right

Core-State Fair Queueing (CSFQ)

- A way to approximate fair queueing without state in core routers
  - Uses state in packets to replace state in router
- Uses probabilistic dropping on flows:
  - Set fair rate f
  - Incoming packets have rate r of flow
  - Drop packets with probability MAX[0, 1-fr]
Original CSFQ

- A contiguous and trusted region of network in which
  - Edge nodes – perform per flow operations
  - Core nodes – do not perform any per flow operations

Algorithm Outline

- Ingress nodes: estimate rate $r$ for each flow and insert it in the packets’ headers

Algorithm Outline

- Ingress nodes: estimate rate $r$ for each flow and insert it in the packets’ headers

Algorithm Outline

- Core node:
  - Compute fair rate $f$ on the output link
  - Enqueue packet with probability
    $$P = \min(1, \frac{f}{r})$$
  - Update packet label to $r = \min(r, f)$

Algorithm Outline

- Egress node: remove state from packet’s header

Problem with Design

- Single malfunctioning router (ingress or core) could lead to severe problems
  - Wrongly labeled $r$ will never be caught!

- Fix: self-verifying CSFQ
  - Pick flows at random
  - Measure their rate
  - If not consistent with marked rate, monitor and relabel flow
### SV-CSFQ

- Bad flows are soon detected somewhere, and bigger flows are detected sooner
- Point of detection moves near entrance point
- Little router state in core
- Can let hosts do their own estimation, since checking is so effective
  - If you have a self-verifying protocol, can then trust hosts....

### Listen and Whisper

- To be presented next week....