CS 194: Distributed Systems
Robust Protocols

Scott Shenker and Ion Stoica
Computer Science Division
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley
Berkeley, CA 94720-1776

Course Overview

- Traditional distributed systems material *(done)*
  - With an Internet emphasis
- New kinds of distributed systems *(done)*
  - P2P and DHTs
  - Sensornets
- New issues in distributed systems *(next three lectures)*
  - Protocol robustness and lightweight verification
  - Resource allocation
  - Incentive issues

What is Makes Sensornets/DHTs Different?

- Both structures are “data-centric”
  - Don’t care about identity of individual nodes
  - Care about name of data
- Both structures have very significant churn
  - Node failure is not a rare event
- Both must be self-organizing
- Sensornets: tied to physical reality
  - Relationship between data not dictated at the abstract level
  - Must be discovered through other means

What Makes These Issues Different?

- Robust Protocols:
  - Recognizing limitations of current techniques
  - Seeking new approaches
- Resource Allocation:
  - Most studies of distributed systems ignore how resources are allocated to different clients
  - They focus instead on correctness and performance
- Incentives:
  - Traditional computer science assumes cooperative clients
  - But why assume cooperation?

Back to Robustness

- Why do we need this lecture?

Don’t We Have Tools for Robustness?

- Formal verification:
- Cryptographic authentication:
- Fault-tolerance via consensus: (Byzantine techniques)

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
- Proof from experience: Internet has withstood some major outages with minimal service interruption
  - 9/11
  - Baltimore tunnel fire
  - etc.

Example: Arpanet Routing

- Early Arpanet used link-state routing
- Routers periodically flood the state of their connected links
  - link-state advertisements (LSAs)
- Each router then has map of entire network
- All routers compute shortest path routes on that map

Basic Challenge

- When receiving an LSA, a router needs to know if it is the latest such LSA
- Example:
  - Router sends “link down” followed a short while later by “link up”
  - If network re-orders packets, then receiving routers will think the link is down
- Challenge: ensure proper ordering, using limited state
  - Easy if given unlimited space for sequence numbers or timestamps
  - But if limited state, then have the “wrap-around” phenomenon
- How would you do it?

Early Arpanet Solution

- LSA had sequence number with some maximal value M
  - Any reordering introduced by network was only a small fraction of M
- To determine if the sequence number has wrapped, a node compared the arriving number NA to the current number NC
  - NA > NC ⇒ Arriving is either new, or an old one with the current message having wrapped
  - NA < NC ⇒ Arriving is either old, or a new one that has wrapped
- The ordering that resulted in the smallest gap was chosen

The Rules

- 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

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!
- Such an event took the Arpanet down...

Fix

- Age LSAs (so they eventually die)
- Wraparound is done explicitly
  - Flush LSA with M, reinsert LSA with 1

Why Didn't Traditional Tools Work?

- Formal verification:
- Cryptographic authentication:
- Fault-tolerance via consensus: (Byzantine techniques)

Why Didn't Traditional Tools Work?

- Formal verification:
  Verifies that correct protocol operation leads to the desired result
- Cryptographic authentication:
- Fault-tolerance via consensus: (Byzantine techniques)

Why Didn't Traditional Tools Work?

- Formal verification:
  Verifies that correct protocol operation leads to the desired result
- Cryptographic authentication:
  Verifies who is talking, but not what they say
- Fault-tolerance via consensus: (Byzantine techniques)

Why Didn't Traditional Tools Work?

- Formal verification:
  Verifies that correct protocol operation leads to the desired result
- Cryptographic authentication:
  Verifies who is talking, but not what they say
- 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

General Lesson

- 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
- Internet designed for syntactic failures, not semantic ones

Other Examples

- Router misconfigurations
- Congestion signaling ignored by receivers
- .....

Will be discussed in detail in 2nd half of lecture

How Can We Avoid These Problems?

- No single rule or algorithm
- Some general guidelines (presented next)
- Overall theme: *design defensively*

G1: Value Conceptual Simplicity

- Obvious, but often unheeded (e.g., BGP)
- Simplicity allows one to reason about behavior more easily
- Leads to better failure handling

G2: Minimize Your Dependencies

- The more nodes you depend on for correct information, the higher the chances for failure are
- Example: Sender trusts receiver for congestion information

G3: Verify When Possible

- Can't use heavyweight Byzantine-style algorithms
- But can try lightweight verification techniques
- Examples in 2nd half of lecture
- Active area of research

G4: Protect Your Resources

- Example 1: SYN flood and SYN cookies
    - Traditional TCP SYN packet requires server to establish state
    - Servers can support only a limited number of TCP connections
    - Sending a stream of bogus SYNs can tie up server
    - SYN cookies are used instead of state establishment
- Example 2: Fair queueing in networks
    - An aggressive flow can steal all the bandwidth on a link
    - Fair queueing ensures that all flows get their share
- Covered in next lecture

G5: Limit Scope of Vulnerability

- If system is vulnerable to a failure anywhere else in system, then robustness is unlikely
- BGP example:
    - Originally, every link event was sent everywhere
    - Route flap damping limits extent to which failures propagate

G6: Expose Errors

Two conflicting goals:

- Automatically recover
- Don’t let problems fester

Review

1. Value conceptual simplicity
2. Minimize your dependencies
3. Verify when possible
4. Protect your resources
5. Limit scope of vulnerability
6. Expose Errors

- Of these, #3 and #4 pose the most difficult technical challenge
    - #3 now
    - #4 next lecture

Lightweight Verification

- No general theory
- Will present 2.5 examples:
    - ECN nonces
    - BGP (listen and whisper)
    - SV-CSFQ

Explicit Congestion Notification (ECN)

- Bit in IP header flipped when routers experience congestion
    - Replaces packet drops with explicit signaling of congestion
- Receiver returns this bit back to sender in TCP header
    - Keeps sending bit until sender returns CWR
    - CWR = congestion window reduced
- ECN advantages:
    - 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

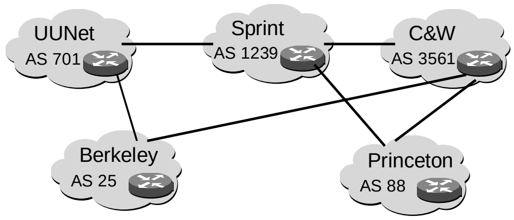
Robust Congestion Signaling (Ideal)

- Use bits in IP header to send two separate signals:
  - Congestion-bit: on or off
  - Nonce: large random number
- When congestion bit is set, nonce is erased
- Receiver must send back cumulative sum of nonces in ACK
- When congestion is signaled, receiver can’t see nonce, so must guess about it
  - If many nonce bits, this is very unlikely

Robust Congestion Signaling (Real)

- Use ECN bits in IP header to send two 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 signaled, receiver can’t see nonce, so must guess about it
  - Improbable it can continue to guess right

Interdomain Routing



Internet is composed of *Autonomous Systems* (AS) which use the *Border Gateway Protocol* (BGP) to exchange routing information.

BGP Basics

- BGP operates at the AS level
- It is a path vector routing protocol:
  - Every router has a table showing, for each destination AS, the shortest path to it
- Routes computed in a distributed recursive fashion
  - Each router learns of the available paths from their neighbors and then chooses the shortest one (for each destination)
  - These paths are then sent to all its neighbors

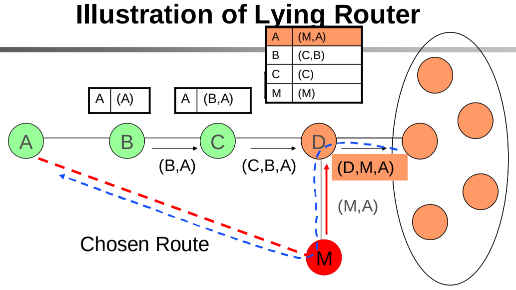
Routers Often Misbehave

- Misconfigurations
  - Major outages in 1997, 2001, 2003
  - 200-1200 misconfigurations/day
- Malice
  - Address space hijacking
  - Compromised routers

Problems

- If a router decides to arbitrarily drop packets, it can interfere with service
- If a router lies, routes can be disturbed
  - A malicious router can draw packets to it by claiming a short route
  - A single (well-placed) router can hijack 37% or Internet routes!

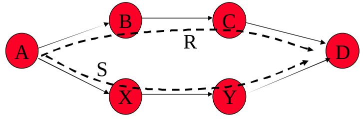
Illustration of Lying Router



How to Deal with Lying Routers

- Simple version (there is a more complex version)
- Source has secret x and inserts H(x) in its routing packets it originates
  - Call this the signature field
- Send route advertisements along two disjoint paths
- At each stage, routers apply h() to signature field, and increment path length
- At destination, compare signature fields and path lengths

Dealing with Lying Routers



R: signature s and path length k
S: signature t and path length l

Must have $h^{(k-l)}(t)=s$ to prove that both paths started with the same secret

If not, raise an alarm!

Using Consistency for Verification

- This is like standard Byzantine approaches, EXCEPT
- Consistency is not between independent calculations, but among different paths for sending the same information
- Lying router(s) have to interfere with every disjoint path in order to keep from raising an alarm
- Caveat: colluding routers can always create “false links”
- Addendum: more complex version verifies path, not just origin

Alarms, not Absolute Correctness

- This is a reasonable tradeoff for large systems
- There are ways to identify the cheaters (at least approximately)

Dealing with Dropping Routers

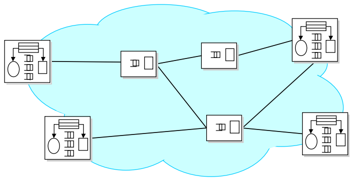
- Test if packets sent along this route arrive at destination
- Passive listening:
  - Listen for TCP SYN packet followed by a DATA packet
- Active dropping:
  - Drop some packets, wait for retransmissions

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-f/r]

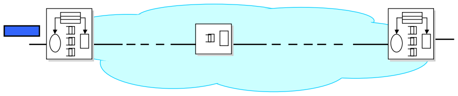
Original CSFQ

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



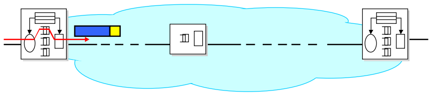
Algorithm Outline

- 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

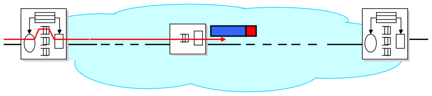


Algorithm Outline

- Core node:
  - Compute fair rate f on the output link
  - Enqueue packet with probability

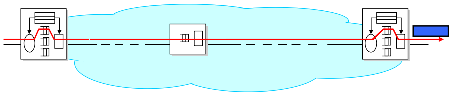
$$P = \min(1, f/r)$$

  - 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!

Self-Verifying CSFQ

- Fix: take meaurements!
  - 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….