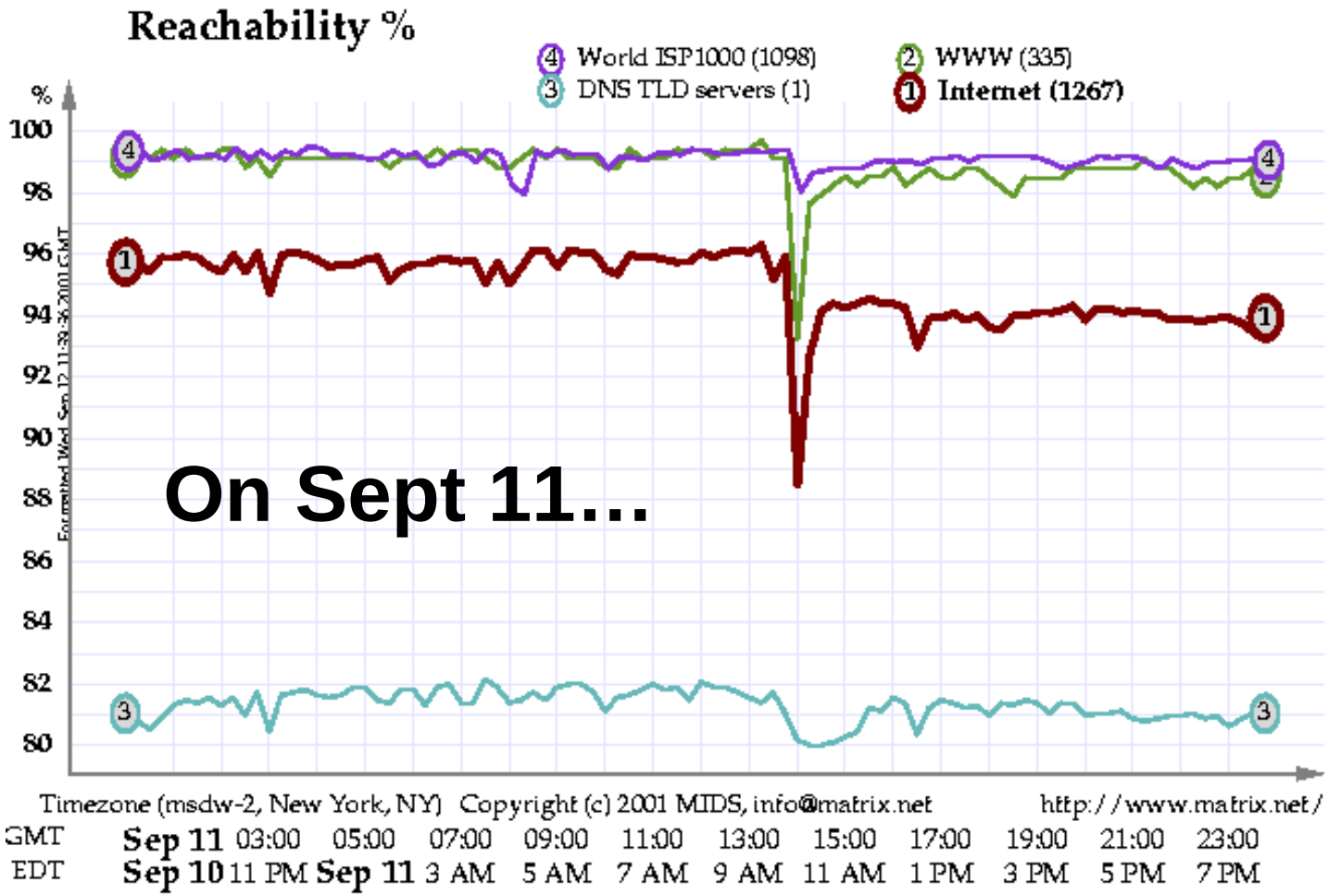
Towards More Error-Tolerant Networking Protocols

David Wetherall, University of Washington

First, a bunch of credits

- Joint work with Tom Anderson
- Collaboration with Stefan Savage, Scott Shenker, Ion Stoica
- Contributions by David Ely, Ratul Mahajan, Neil Spring

How robust is the Internet to failures?



Robustness is due to principled design

- Critical state is soft-state, not hard-state
- End hosts are responsible for error recovery
- Bad news propagates fast, good news slowly
- Use replicas or redundancy for single points of failure

- Every modern protocol design is scrutinized for its handling of failure cases

Bugs and mistakes can wreak havoc

- Implementation bugs and configuration mistakes aren't fail-stop. These errors can hurt parties not at fault and cascade

- AS7007 incident (April 1997)
  - Misconfigured routers at Virginia ISP send the wrong routing messages, taking down the Internet backbone for hours
  - This is just the most famous incident …
- RFC 2525, "Known TCP Implementation Problems", 1999.
  - 18 common bugs that are lurking out there, …

- There has been relatively little study of bugs and mistakes

Aside: a study of BGP mistakes (Ratul)

- How often do router configuration mistakes happen? How do they affect the Internet? Why do they occur? How can we prevent them?

- Take full routing traces from ~20 vantage points around the Internet, analyze them for likely mistakes, contact every implicated ISP to find out what happened.

- We find that:
  - Router configuration mistakes are pervasive (>100 per day)
  - Three out of four new origins for a prefix are mistakes
  - Router operators hijack connectivity ~15 times a day
  - Poor interfaces and tools make it really easy to screw up

Few principles for bugs and mistakes

- Earlier principles don't help us tolerate these errors. There are even few examples of error-tolerant protocols!
  - Perlman's "Byzantine Robust Routing" design (1988)
- One widely known principle:
  - *"Be liberal in what you accept and conservative in what you send"*–Postel
- Other relevant work in security protocols:
  - Abadi/Needham "Prudent Engineering Practices for Crypto Protocols"

- We don't systematically consider how well or poorly designs will respond to bugs or mistakes today.

The rest of this talk

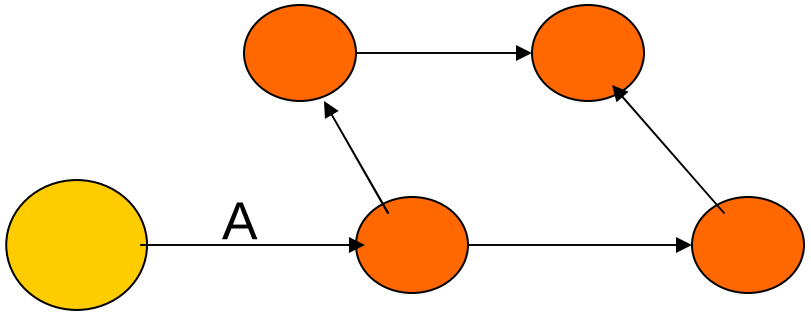
- Thesis of a work in progress:
  - Some protocols are more tolerant of bugs/mistakes than others
  - We can produce these protocols by design

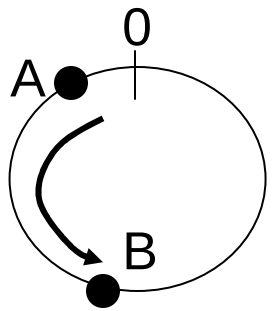
- First, a closer look at examples of flawed protocols
  - What went wrong and how the protocol was fixed
- Second, speculation on how to design better protocols
  - Towards design principles and techniques for coping

1. ARPANET Link-state Flooding

- In link state routing, routers exchange updates with their neighbors. These are flooded so they reach everyone. Then they are used to calculate routes.

Update A floods to all nodes

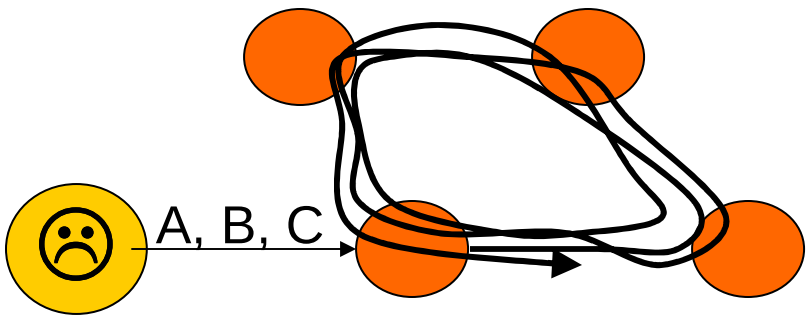


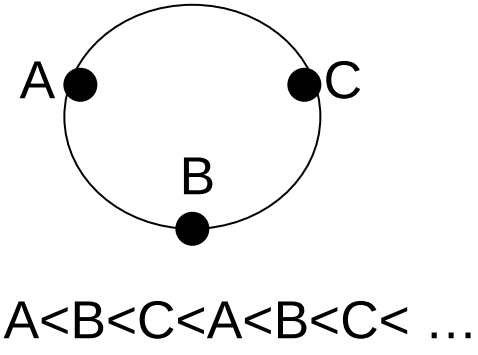


- Sequence numbers are used to order updates. ARPANET used modulo arithmetic to decide which update is new.

Problem – an endless flood

- One night the ARPANET stopped working. A corrupt router had injected messages that led to an endless sequence of updates …





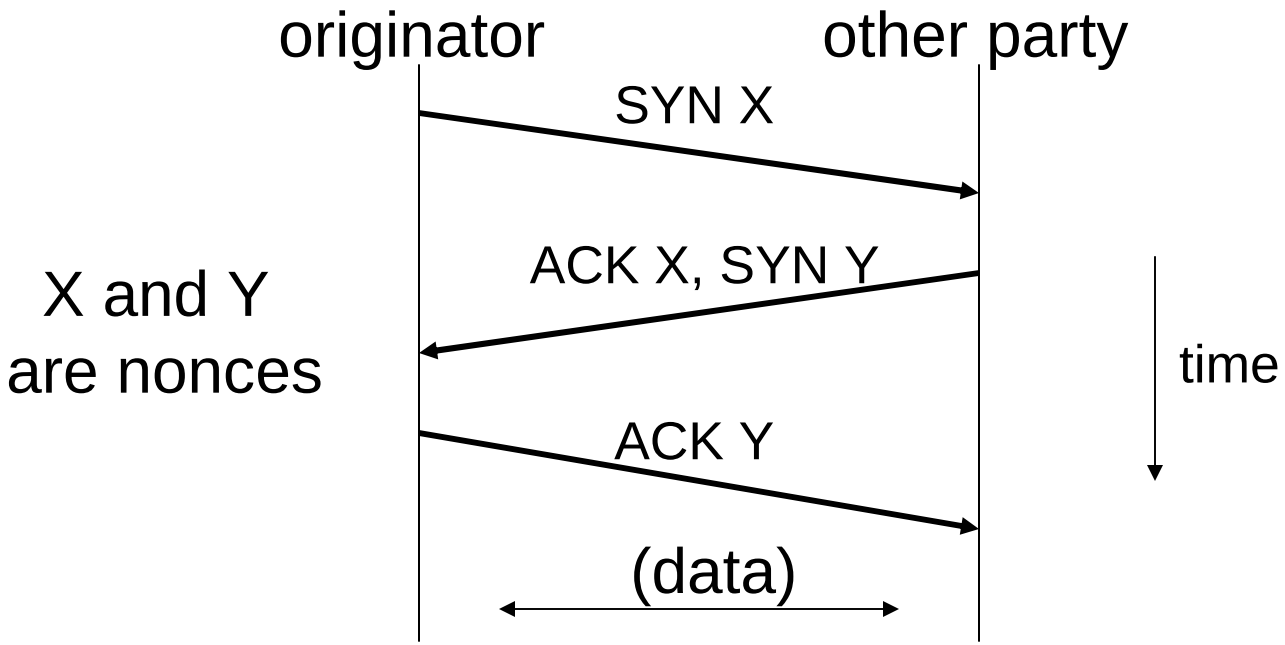
- This was hard to fix – purge entire network of bad data

Solution – reset, don't wrap numbers

- Sequence numbers taken from a large, linear space
- Now repeated updates in any order cannot be interpreted as new and cause an endless cycle
  - New work requires fresh messages to be injected by routers
- We use aging to purge an update with maximum sequence number, should that arise.

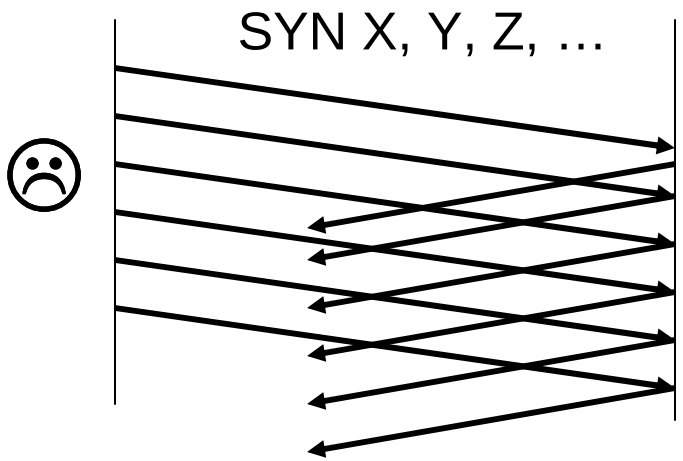
2. TCP Connection Establishment

- Two parties need to "SYNchronize" to form a connection
- TCP uses a three way handshake – for classic reliability!



Problem – SYN flooding

- If originator doesn't follow through it burdens the other party. Used for denial-of-service starting ~1996 through today.

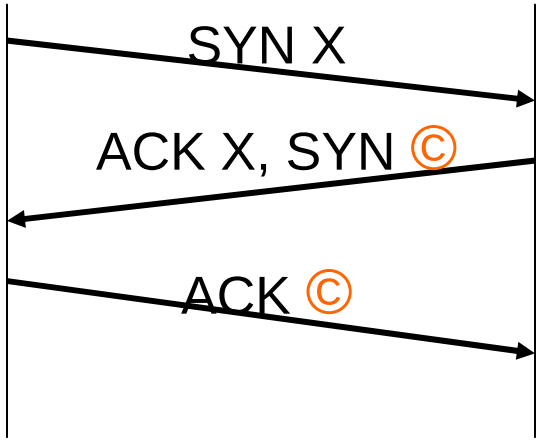


State kept for incomplete connections can exhaust resources …

- (Plus, if nonces are predictable then fake connections can be forged.)

Solution – offload state with cookies

- Don’t keep the state. Attach it to the packet and let the originator deal with it and return it to you later. But state must be an opaque, verifiable cookie so originator can’t mess with you …

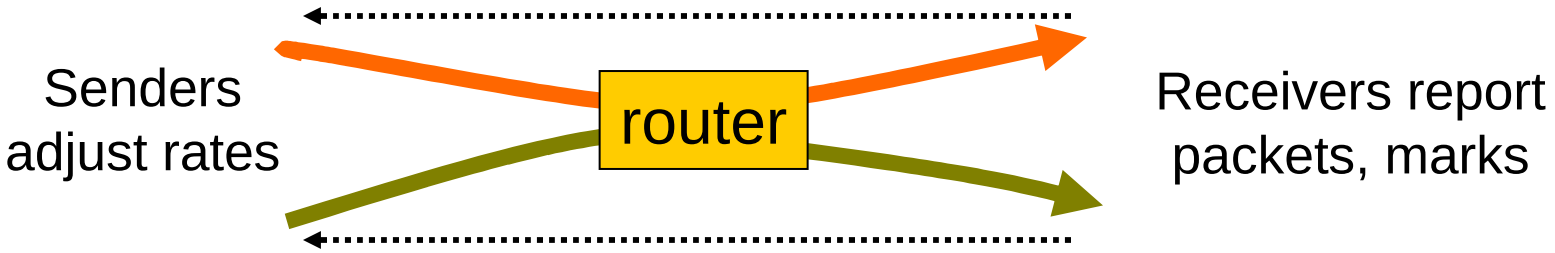


No need to keep per connection state initially

- Linux SYN cookies (‘97): reply nonce © is used to carry a cookie. It is securely hashed with a secret so it can be checked on return.

3. Congestion Signaling with ECN

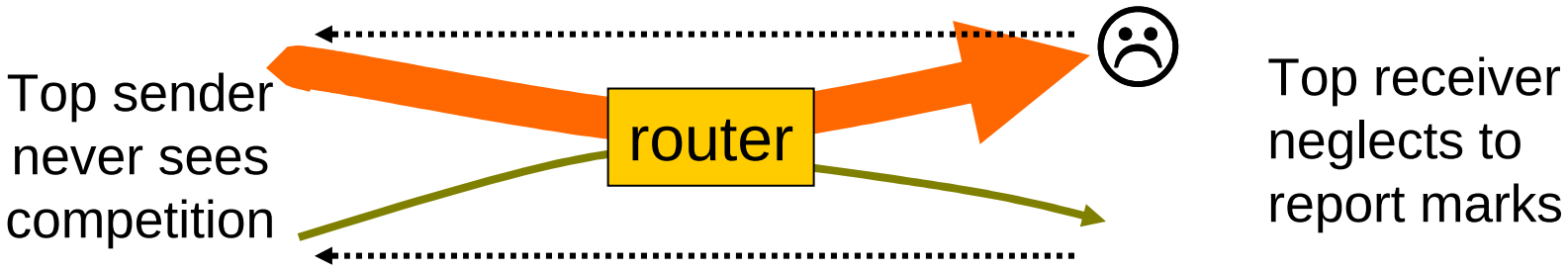
- Hosts adjust the rate of TCP connections to share network bandwidth based on router feedback (drops).



- With ECN, routers can mark packets to signal congestion, rather than drop them. Marks are returned from receiver to sender so it knows to slow down.

Problem – marks can be dropped

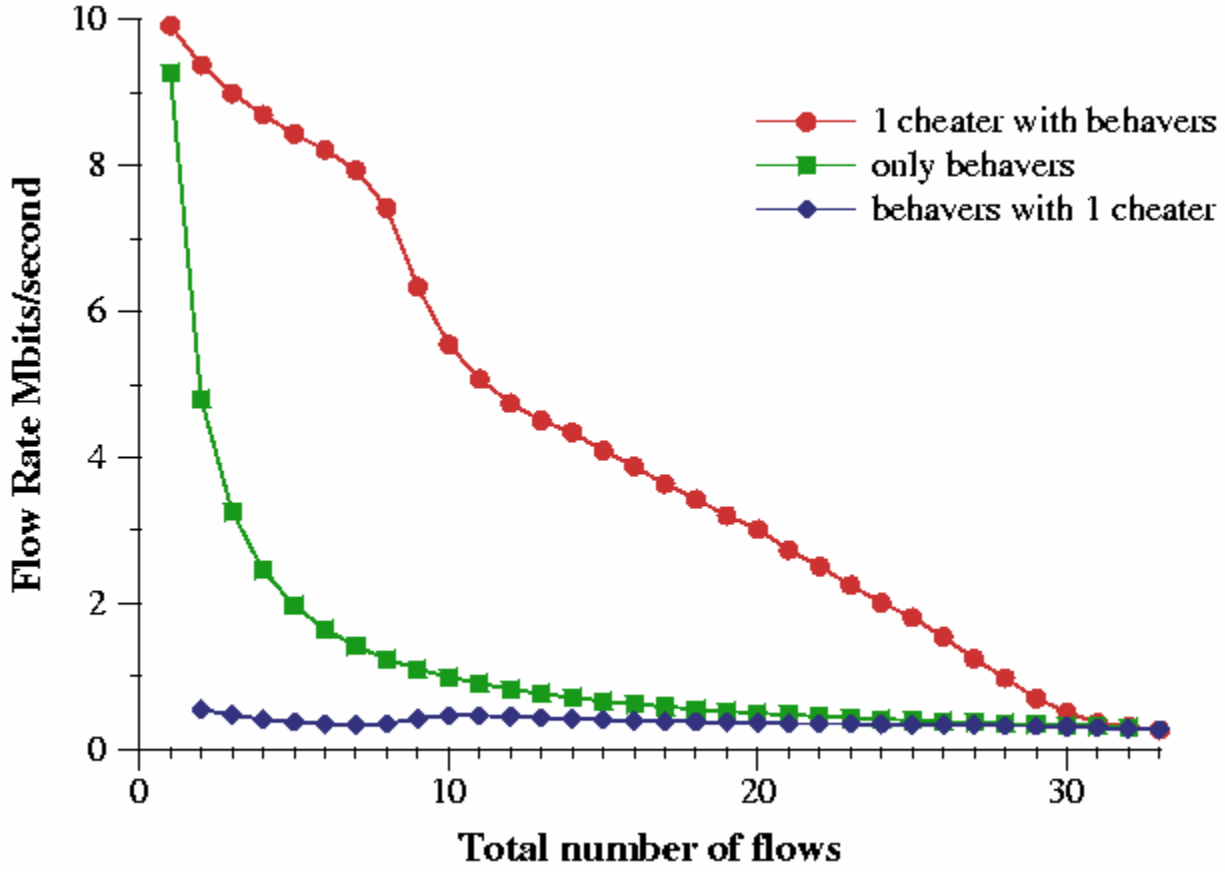
- Unlike dropped packets, which can't be undropped, marks on packets can be silently erased.



- During testing, VPNs/firewalls were found to do this.

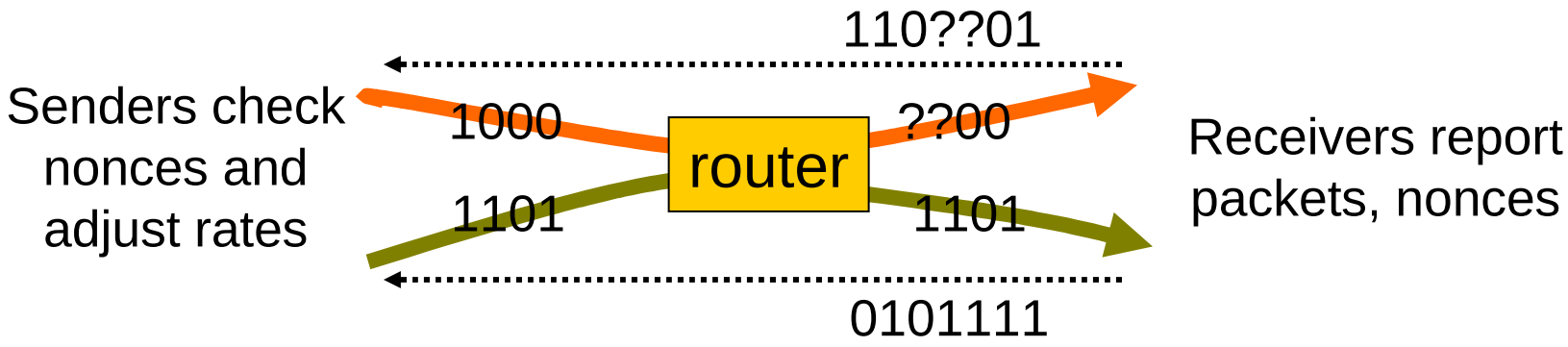
And it makes a big difference

- Buggy receiver gets 10X throughput at expense of others



Solution – check marks with nonces

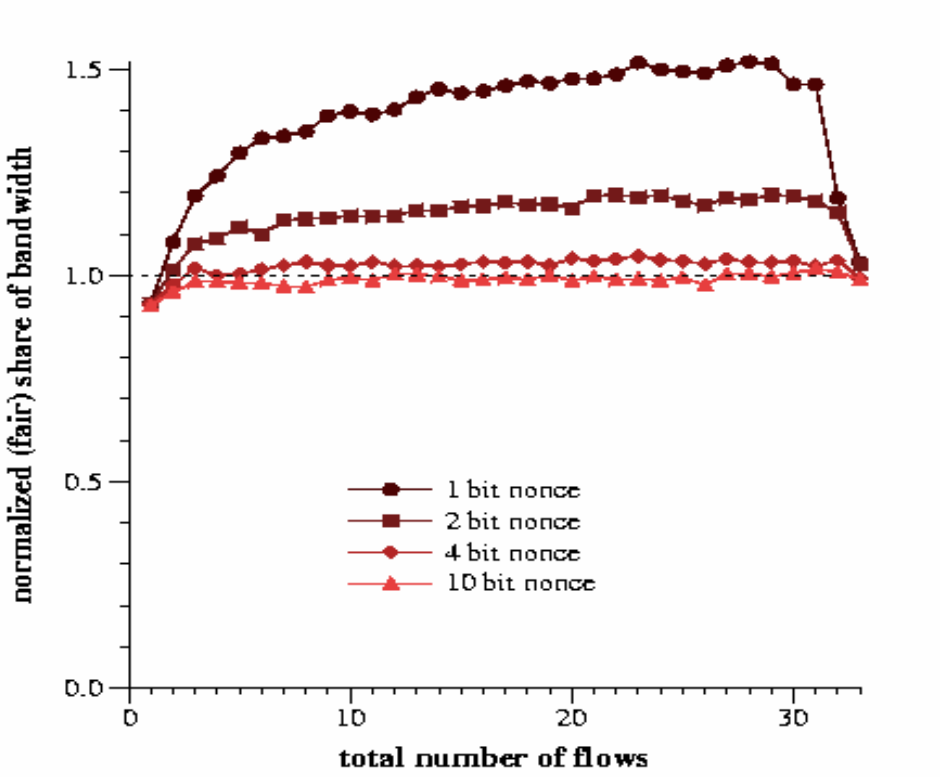
- Sender attaches nonces to packets that are erased to mark congestion. Receiver reports nonces as feedback. Now marks can't be undone without sender realizing the omission.



- This is the new IETF ECN design
  - See "Robust Congestion Signaling", Ely et.al., ICNP 2001

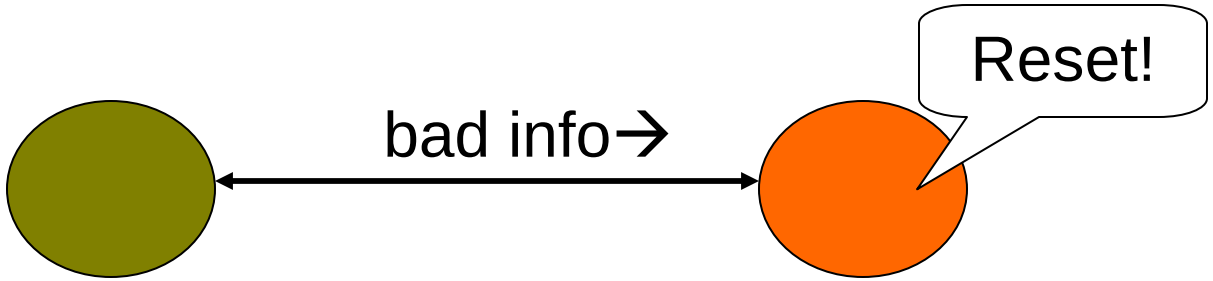
Probabilistic checking works well

- Buggy receiver gets 1.5X, and checking is efficient



4. BGP (Internet routing) Error Handling

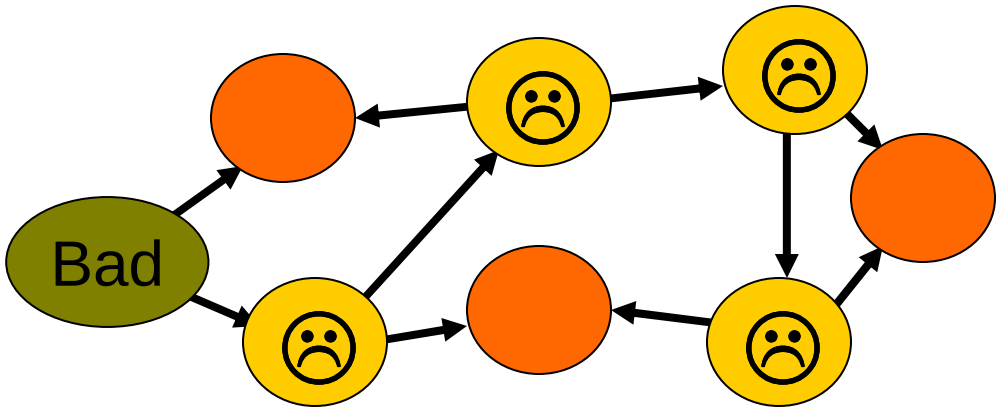
- In BGP routing, peers exchange announcements over a TCP connection and use them to select forwarding paths



- If bad information is received by a peer, which of course shouldn't happen, it resets the connection and retries.

Problem – errors can be magnified

- Some routers pass on bad info rather than reset (yellow)
- Bad info propagates much further than otherwise
- Many “correct” routers see the bad info and reset (orange)



- This caused a widespread outage in October 2001

Solution – weed out individual errors

- Add error checking at a finer granularity
  - Individual routes rather than whole peering sessions
- Correct behavior is then to drop individual errors
- Bad behavior, which passes errors, doesn't hurt as much

- BGP spec being revised in NANOG and IETF.

Key observations

- The improved designs don't have fewer bugs – they respond better in the presence of bugs
  - Innocent bystanders are less affected, and no cascades
- The downfall of fragile designs is obvious in hindsight
  - And maybe could be spotted beforehand with careful analysis

How do we tolerate errors by design?

- We can't prevent errors, so we must cope with them
  - Many different implementations
  - Many different parties with different goals
- Related approaches that aren't silver bullets:
  - Security via cryptographic techniques, e.g., authentication
  - Fault tolerance via replication and Byzantine agreement
- Bottom line: we need new principles and techniques

A possible methodology

- In a protocol, message are exchanged between black boxes that perform work and maintain state. The interaction accomplishes some function

- For error-tolerant design:
  1. Verify as much as you can (messages)
  2. Protect your resources (state)
  3. Contain the scope of problems (interaction)

Techniques for verification

- Idea is to question your assumptions about received messages

- Add information to allow for consistency tests
  - ECN with Nonces
- Get rid of unverified information
  - Link state flooding
- Use independent, redundant methods
  - One possibility: run link state and distance vector routing

- Other examples:
  - TCP handshake (nonces)
  - IP traceback (Savage, SIGCOMM 2000)
  - Self-verifying CSFQ (Stoica)

Techniques for protecting resources

- Idea is to only expend resources in response to verified or local information.

- Shift the resource burden
  - SYN cookies

- Other examples:
  - Lazy Receiver Processing (Druschel OSDI 1996)
  - Cookies in Photuris (Karn RFC2522)

Techniques for containment

- Idea is to localize the impact of the error so that others can continue, albeit in a degraded fashion.

- Tolerate inconsistency
  - BGP error handling

- Other examples:
  - BGP route flap dampening (RFC 2439, Nov. 1998)
  - Areas in OSPF (hierarchy in routing protocols)

Conclusions (Rhetoric)

- It's time to get systematic about bugs and mistakes
  - They are a significant, under-explored weakness
- Some protocols are more tolerant of bugs than others
  - It's not just that they have fewer bugs
- Protocol design can play a role in coping
  - If we can formulate principles that lead to error-tolerant designs
  - This is a research challenge. How far will we get?