IPAM Workshop

Fault-tolerance for the Internet via Self-Stabilization (or soft state ain't sufficient)

George Varghese
UCSD CSE
varghese@cs.ucsd.edu

June 2002

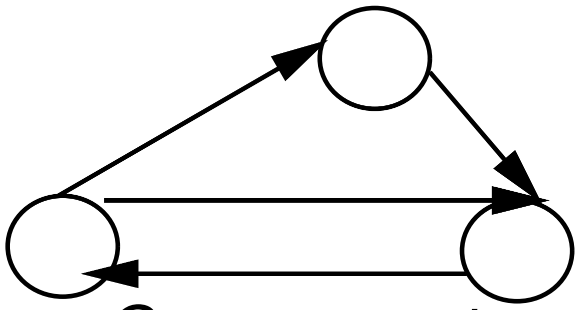
Talk goal: a perspective on fault-tolerance

- Several fault-tolerance viewpoints from *within* computer science (e.g., databases: atomic transactions; Byzantine tolerance: tolerating arbitrary faults) and *outside* computer science (coding theory: error-correcting codes; control theory: feedback control; statistics: identifying errors via sampling).
- Would like to describe a different viewpoint on fault tolerance : *self-stabilization*.
- Would like to make the somewhat bold claim that self-stabilization(SS) is a stronger notion than the popular notion of *soft state*.

Will try to illuminate the main thesis using practical examples from Token Passing, Link State Routing, Credit-based flow control, BGP, and RSVP

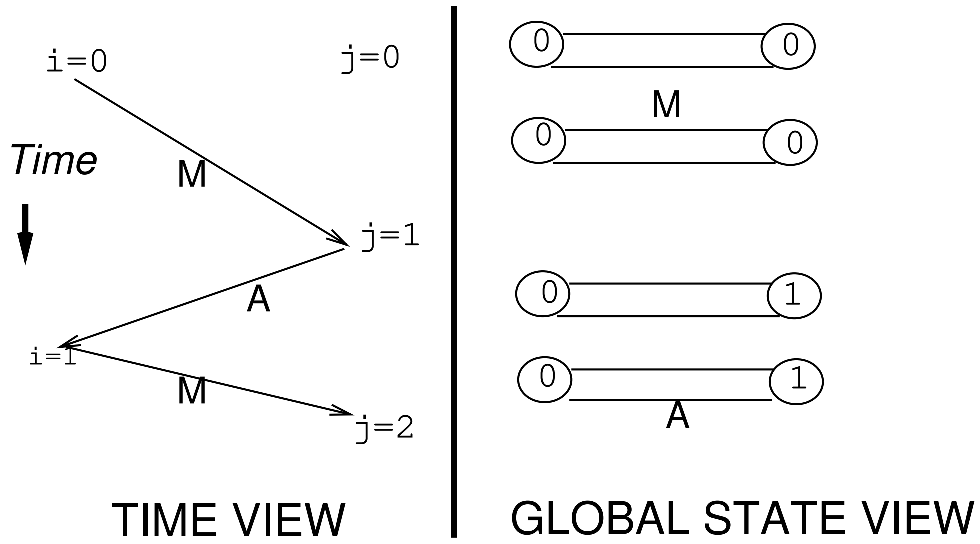
Outline

2.0 State Machine Model of Internet



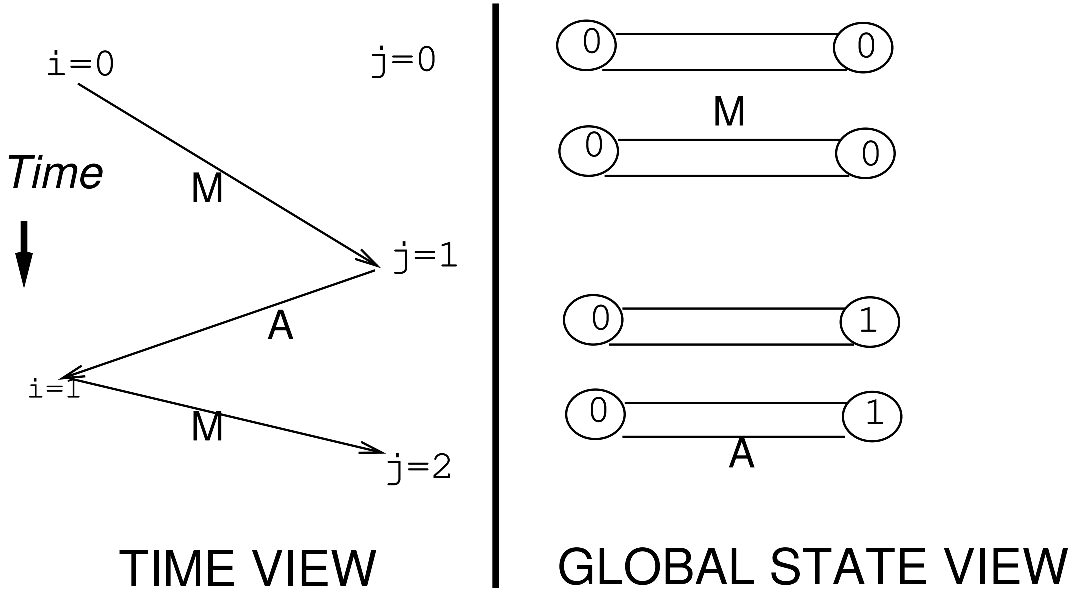
- Directed graph of nodes and links.
- Node state machine specifies: **1)** internal state change **2)** when packet sending is enabled **3)** state changes when packet is received on a link.
- Link state machine: a queue (FIFO) or a set (non-FIFO) of packets. Internal transitions to lose/corrupt packets.
- Sending (or receiving a packet) changes state of both sending node and outgoing link.

2.1 State and Time Views



- Execution: sequence of global states. For correctness all executions should satisfy specified properties expressed by a predicate.

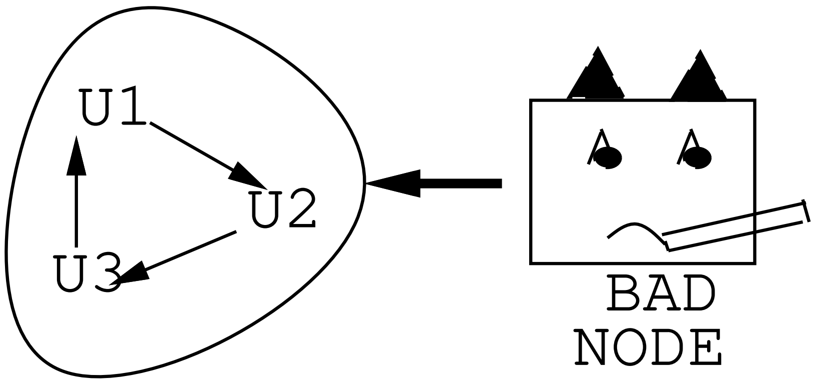
Invariants



- Invariants describe possible states of a protocol. In example:
  - **If** M in forward link or both links empty **then** $i = j$.
  - **Else if** A in reverse link . . .

Proofs by induction on the length of an execution with a manageable number of cases.

3.0 Self-stabilization



Protocol recovers by itself regardless of what state it starts in.

— Very robust: usual faults + spurious messages + corrupted state.

— Uniform Approach to Fault-tolerance.

— No initialization.
— Same goals as soft-state only somewhat older and more general.

3.1 Clarifications

- Code can't be corrupted.
- Orthogonal to Byzantine fault-tolerance (*all* nodes can have bad initial states in this model) while in Byzantine model *some* nodes can have arbitrary code and arbitrary initial state. In practice, much cheaper to do stabilizing routing over Byzantine routing.
- — Must tolerate Initial errors. Thus likely uses are: Routing, Scheduling, Resource Allocation. Not databases, nuclear reactors. OK for Internet infrastructure protocols like BGP not say FTP.

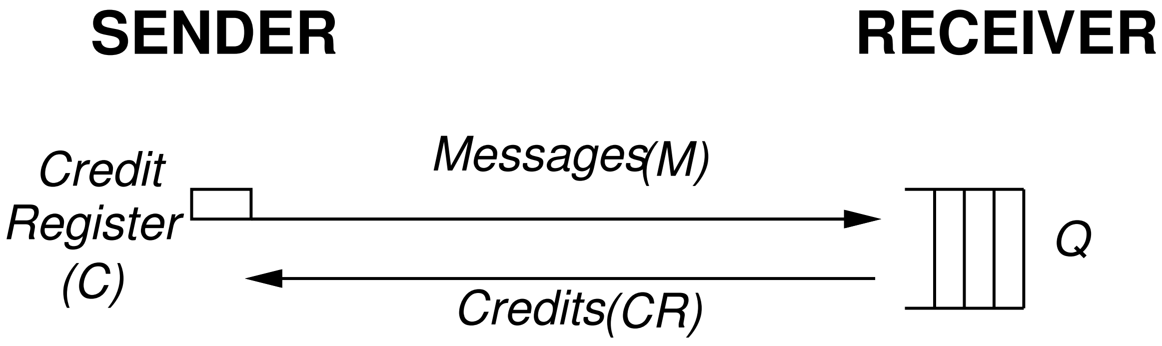
3.2 Some history

- Dijkstra 1974, 1 page paper with 3 algorithms, no proofs. Deconstruction of first two algorithms on my web page.
- Tony Lauck, then head of DECNET, recognized this as a seminal paper and pushed for it to be a goal in DECNET protocols. Two Lauckisms:
  - Systems should *definitely* self-stabilize before the customer logs a service call and *ideally* before the customer notices
  - Systems should autoconfigure without management intervention.
- Lamport concurrently championed the notion in the distributed systems community. In 2002, second paper chosen as the most influential DC paper of the last 10 years.
- DEC produced a number of influential protocols that autoconfigured and stabilized like the bridge spanning tree protocol and OSPF-style routing. Seem to work reliably . . .

3.3 Techniques for Self-stabilization

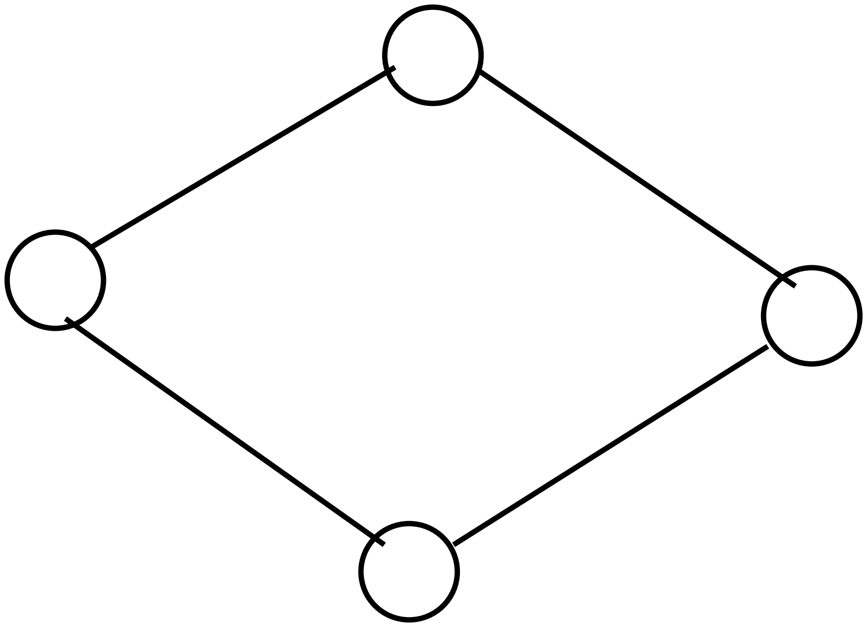
- After Dijsktra's paper, there were several examples of stabilizing protocols described by teams such as from Austin (*Gouda et al*) but they all seemed to used ad hoc methods.
- For my thesis, worked on general techniques to make protocols self-stabilizing. Here are two (no claim to completeness, there are others due to *Arora, Dolev, Katz and Perry etc.*)

3.3.1 Example: ATM flow control



INVARIANT: $M + C + Cr + Q = Max$

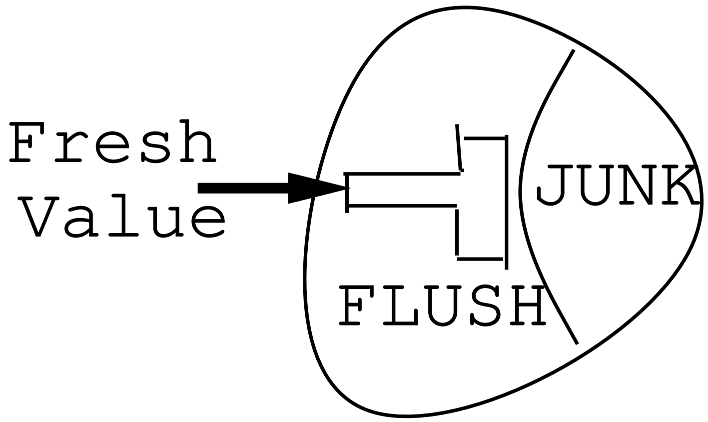
3.3.1 Local Checking and correction



- Local Checkability

- Local Correctability.

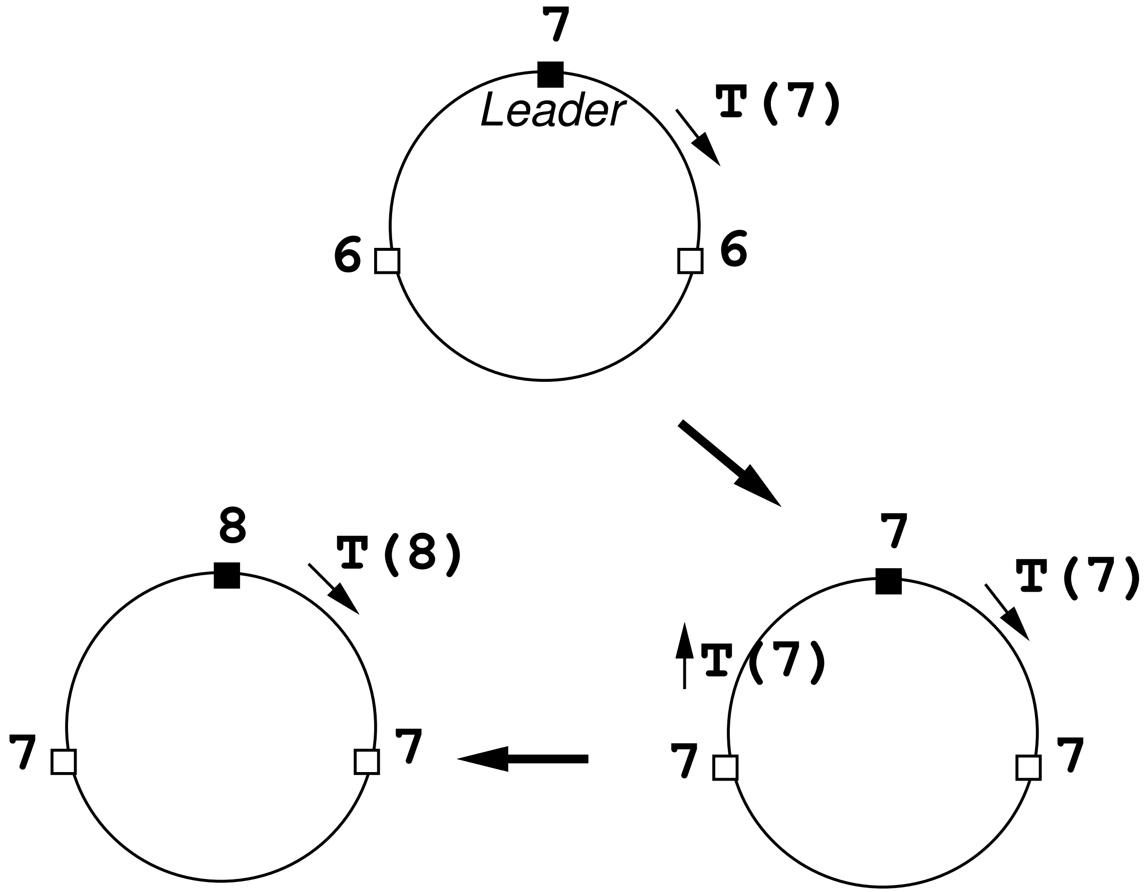
3.3.2 Counter flushing



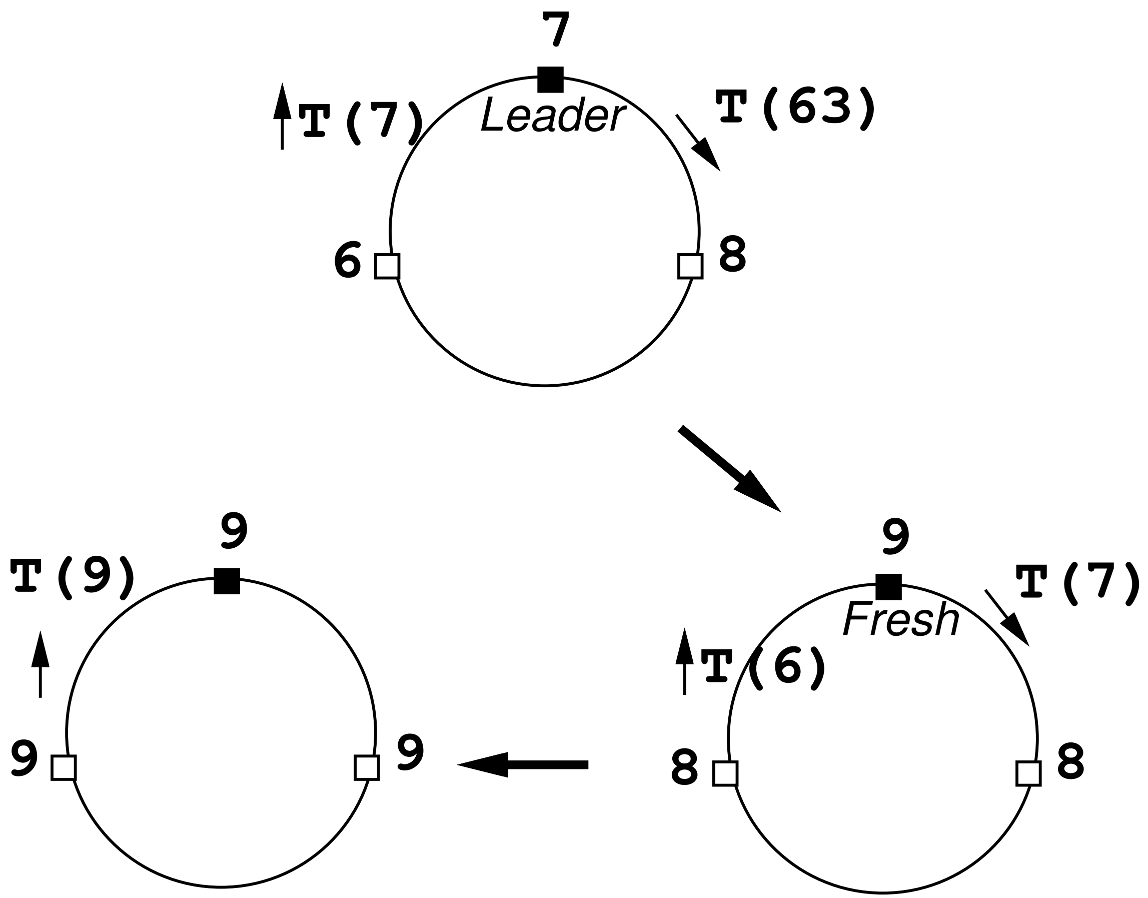
By adding a simple counter that is "large enough" and some checks can help stabilize a number of useful protocols:

- Request-response, Token Ring, Deadlock detection,
- Reset Protocols, virtual circuit set up.
- One-to-one and one-to-many sliding window protocols.

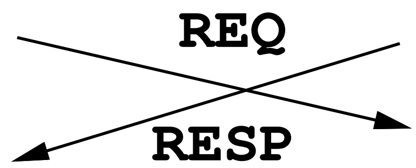
Token ring: good states

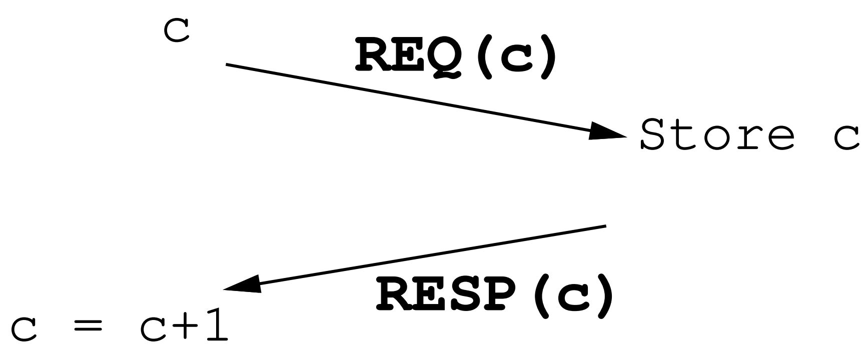


Token ring: stabilization



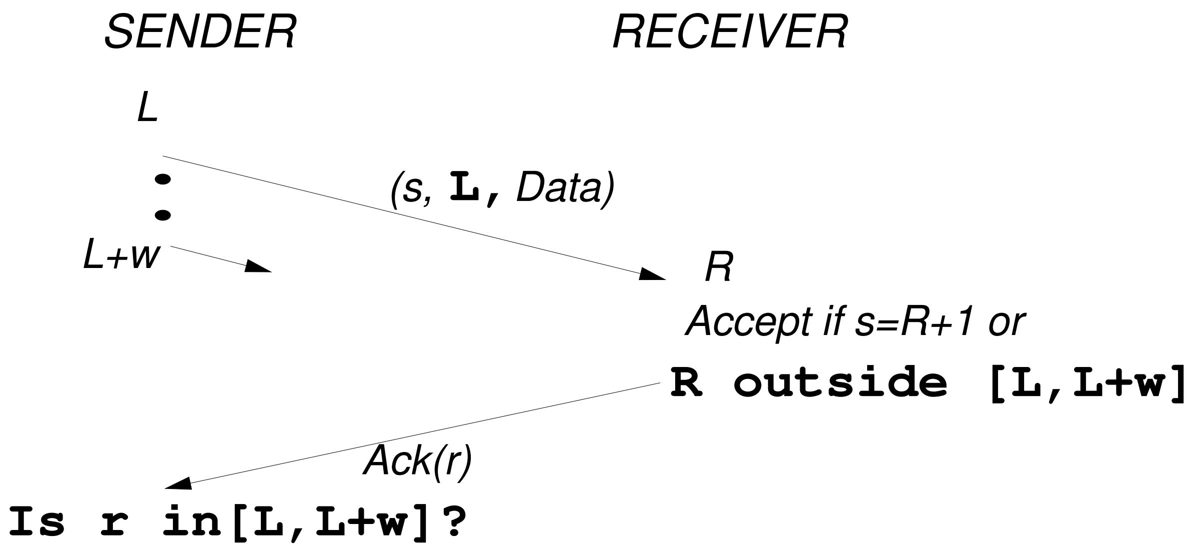
Request-response matching





Takes 2 *actual* round trip delays to stabilize by *counter flushing.*
Faster than *timer flushing*.

Sliding Window



- Need bolded checks at sender and receiver to avoid deadlock.
- Need sequence number modulus to be larger than window size times maximum number of stored packets.

3.4 Soft State Paradigm

- Not very precisely defined. General idea (*Clark*) is that hard-state (i.e., TCP connection state) is set up by explicit messages and is removed only after an explicit disconnect (e.g., TCP FIN?) *And yet most hard state protocols will also timeout state after sufficient inactivity such as a broken connection*

- Soft state is periodically timed out and so must be set up by some initial trigger and then refreshed at some periodic rate. No need for initialization.

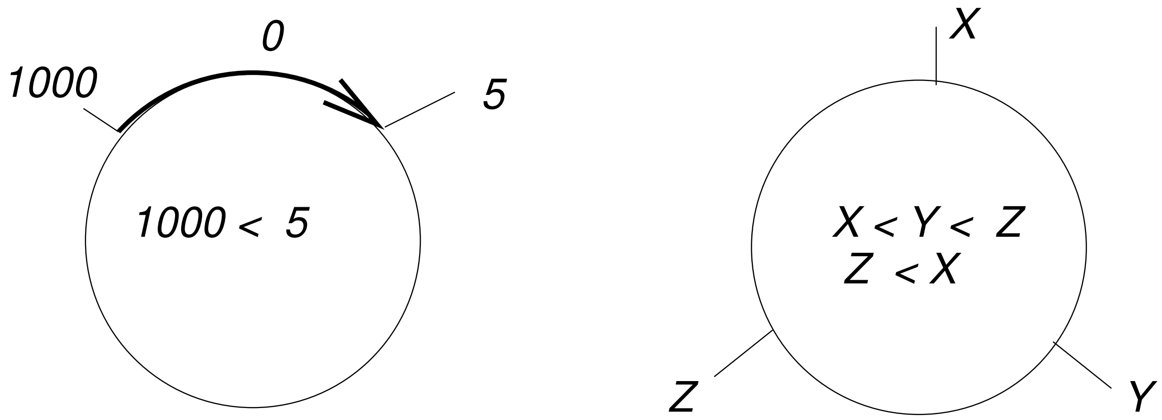
- Gone beyond Clark's initial trenchant observation to fairly dogmatic status: "soft state is good, hard state protocols are complex". "Periodic sending solves all problems".

Stabilization versus Soft-state

- Can make real and fairly complex protocols self-stabilizing. Using the counter flushing, Adam Costello designed a self-stabilizing version of a well known (FDDI) token ring protocol.
- Note that it is insufficient to just send state repeatedly and timeout as in the soft-state paradigm. Would not work in the ATM credit example or token rings as can stay in bad states without sufficient care.
- Periodic sending does work as long as the invariants of the system can be partioned into one-way invariants for each link in each direction.
- Making protocols self-stabilizing seems a better goal.

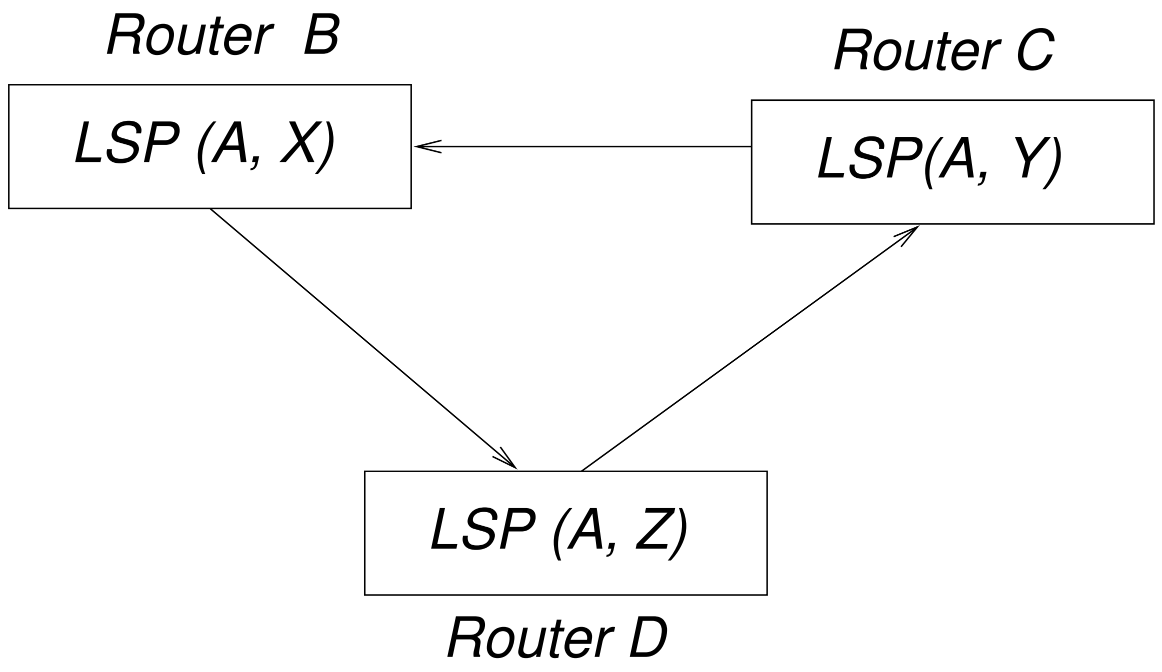
4 Examples from Networking

4.1 Link State Problem Context



- Routing updates, called link state packets in the ARPANET Routing protocol, carried a small sequence number, with larger numbers considered newer. (*Rosen et al*)
- Because sequence number spaces wrapped, they defined a circular comparison operator that is not well-ordered.

4.1 Link State Problem

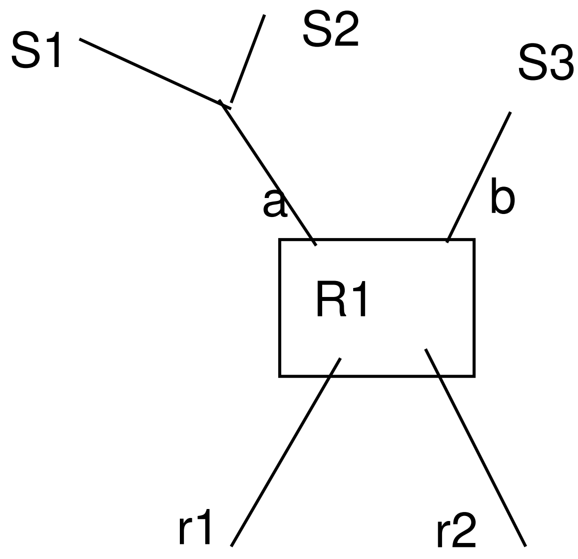


- Malfunctioning source injected three updates and crashed. Routers kept cycling between 3 updates X, Y, and Z.
- Days of coast-to-coast detective work for diagnosis and manual restart for fixing.

4.1 Link State Solution in OSPF

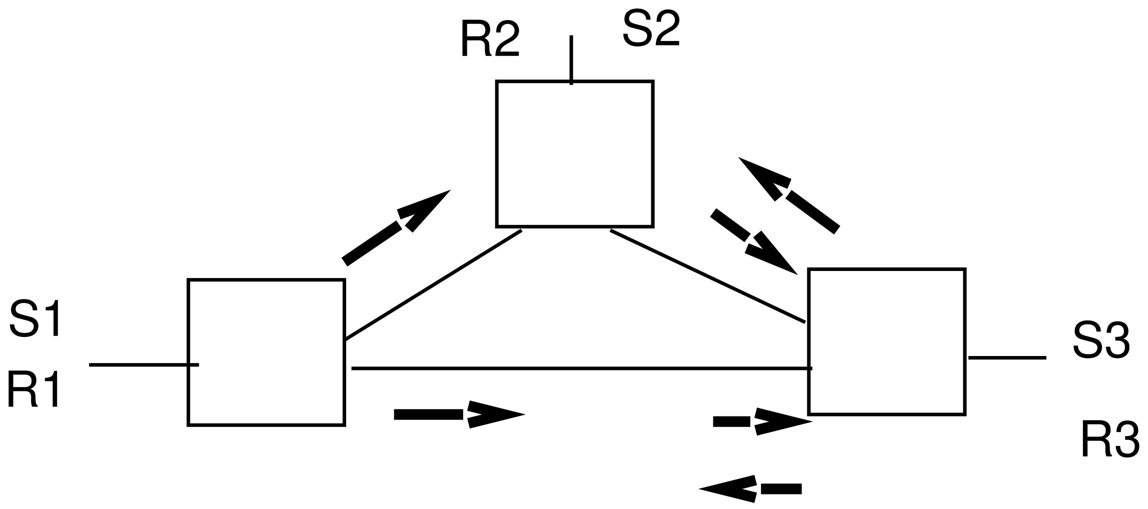
- OSPF came from DEC IS-IS. In early days Radia Perlman added a beatiful mechanism for stabilization where a router A that gets an old packet from neighbor B sends back the latest version to B.
- However, Perlman's original paper and design had a lollipop (initial line plus circle) sequence space that we realized did not add any extra resilience.
- Simplified by adding a large 64-bit linear sequence number space that *is* well-ordered. *Perlman, Lauck, Varghese*
- In unlikely event that source reaches max of space, source router R goes offline for half-hour after which all other routers are guaranteed to time-out all LSPs from R.
- An even more pragmatic trick is for R to come up as another router R' which it can since every router has multiple IP addresses for each link and has to choose one for its LSP source address!

4.2 RSVP Problem Context



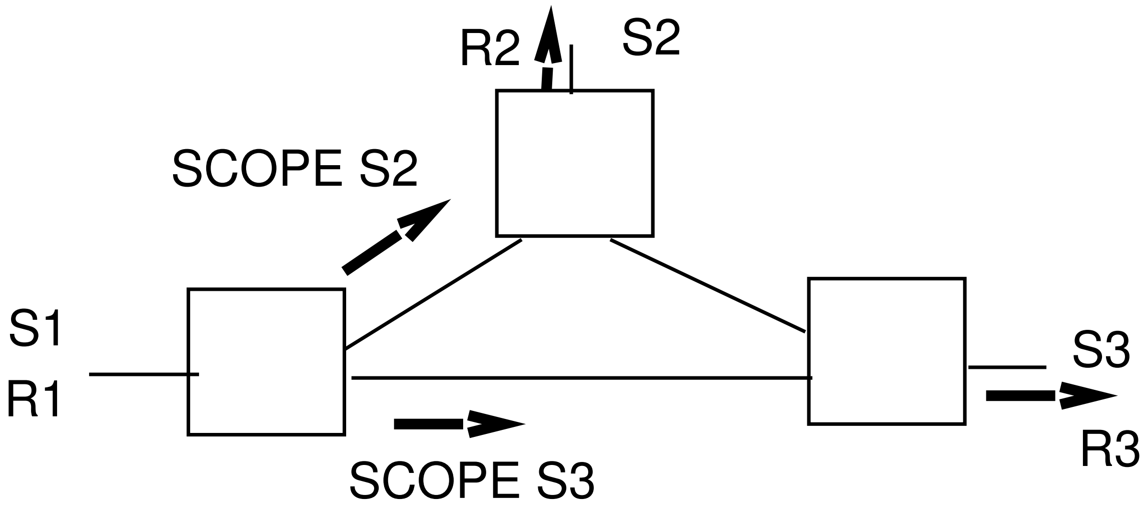
- For wildcard reservations in RSVP, we might as well aggregate sources S1 and S2 from point of view of R1 as " some sources on this group are reachable on Port a. Then forward reservation on all such ports.
- Alas, it does not work in when a group can have diff per-source trees whose union forms a cycle.

4.2 RSVP Looping Problem



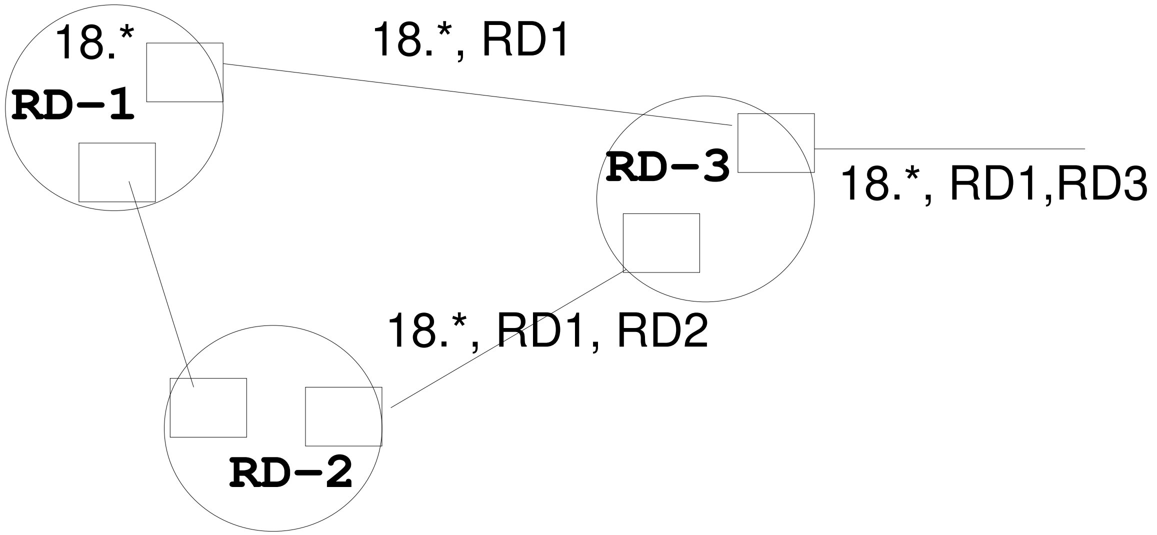
- Union of all trees can form a loop! When thinking from a self-stabilization perpective, makes it easier to imagine such scenarios.

4.2 RSVP Solution



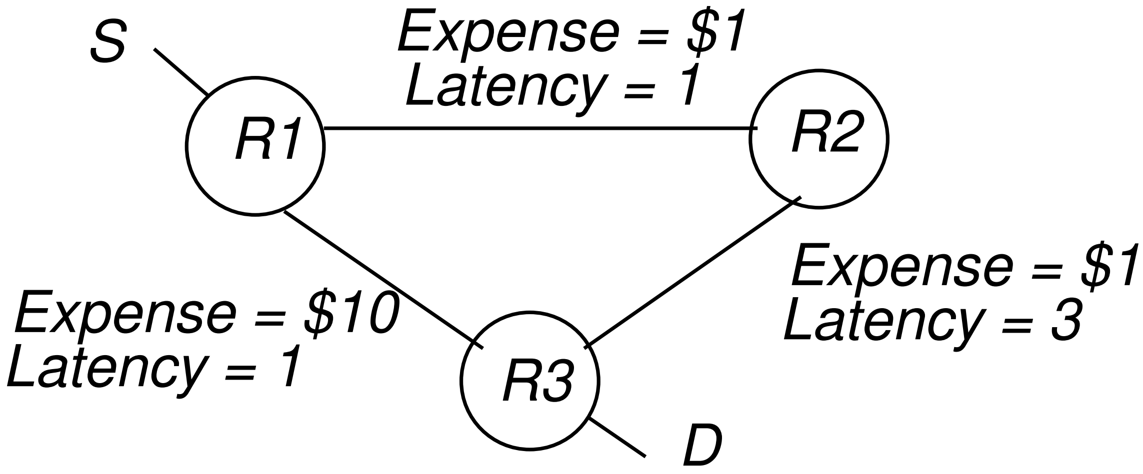
- Introduce a SCOPE variable. At junction points, break out RSV messages based on which sources are reachable on each link.

4.3 Border Gateway Protocol (BGP) Context



- Policy-based (e.g., no transit) routing protocol between routers in different ISP domains where certain links and routers are outlawed. Different from metric where all paths are possible but some are better
- Path vector scheme allows both policy flexibility and supposedly loop free routing.

4.3 BGP Looping Problem



Can we put BGP in bad states where it may never stabilize. Yes, easy to imagine:

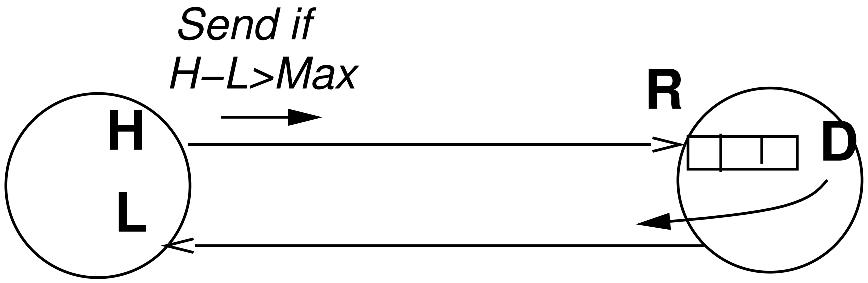
- Examples galore of policies where BGP keeps switching routes (see *Varadhan, Govindan, Estrin* and *Griffin-Wilfong*
- To add fuel to the fire, examples of BGP implementation instability (*Labowitz et al, SIGCOMM 98*)

4.3 BGP Solution

- Nice paper by *Gao and Rexford* that shows that by somewhat restricting policy (ordered chaos instead of complete chaos) one can get stable and loop-free routes that will converge.
- As is common in commercial world, routes come to a router from links that are either peer-to-peer, customer-provider, or backup.
- Gao-Rexford rule asks that every router never prefer a peer or provider route to a customer route, and backup links have lowest priority.
- Would be nice to have a complete proof of self-stabilization of BGP besides the more limited stabilization results they prove.

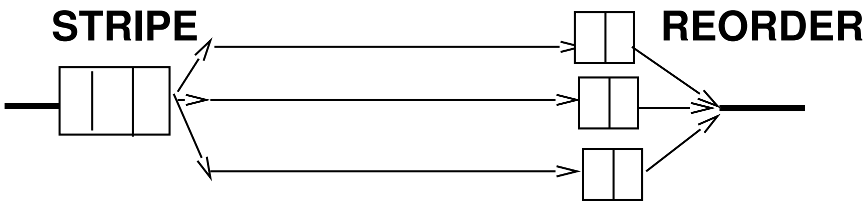
5 Weakening from Stabilization to Message Loss

5.1 Message Loss Counters



- FCVC: sliding window, no retransmissions, fixed size cells. (Kung, Chapman)
- Sender periodically sends count of cells sent (H). Receiver keeps count of cells received (R). Can detect loss and adjust cells dequeued (D).
- Example of 1-way invariants $H = R + cellsintransit$. If $L \neq D$ then there should be a credit D in transit.

5.2 Load Balancing

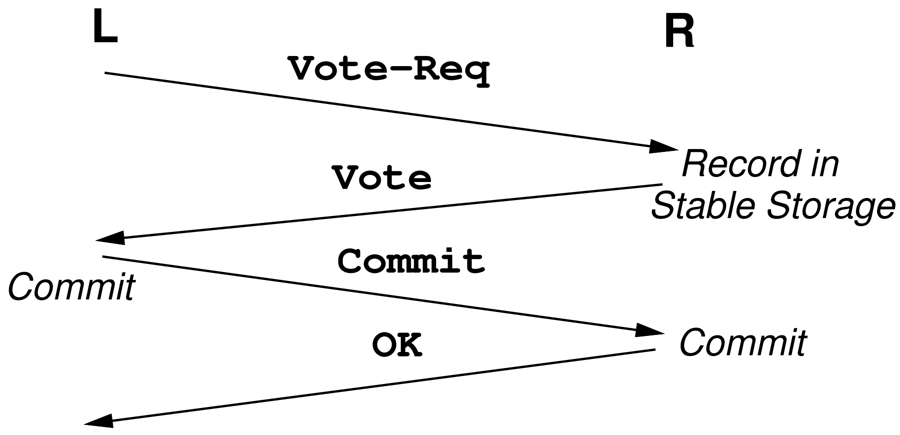


- Sender stripes cells in round-robin order. Receiver buffers cells and removes them in same order. *Adiseshu, Varghese, Parulkar*
- FIFO violated if cells are lost. Can detect (by loss counters) and adjust (by inserting ghost packets).

6.0 Other failure mechanisms

- *Stabilization:* All nodes can start in arbitrary states but work correctly thereafter and only *eventual* correctness is assured. e.g., Counter Flushing
- *Database:* Some nodes can fail in fault-stop fashion but *continuous* correctness is assured. e.g., 2-Phase Commits
- *Byzantine:* Some nodes can fail in *arbitrary* fashion but *continuous* correctness is assured. e.g., Authenticated Broadcast.

6.1 Two-phase Commits



- Safety always maintained but loss of liveness (*blocking*) caused by coordinator failure after Phase 1.

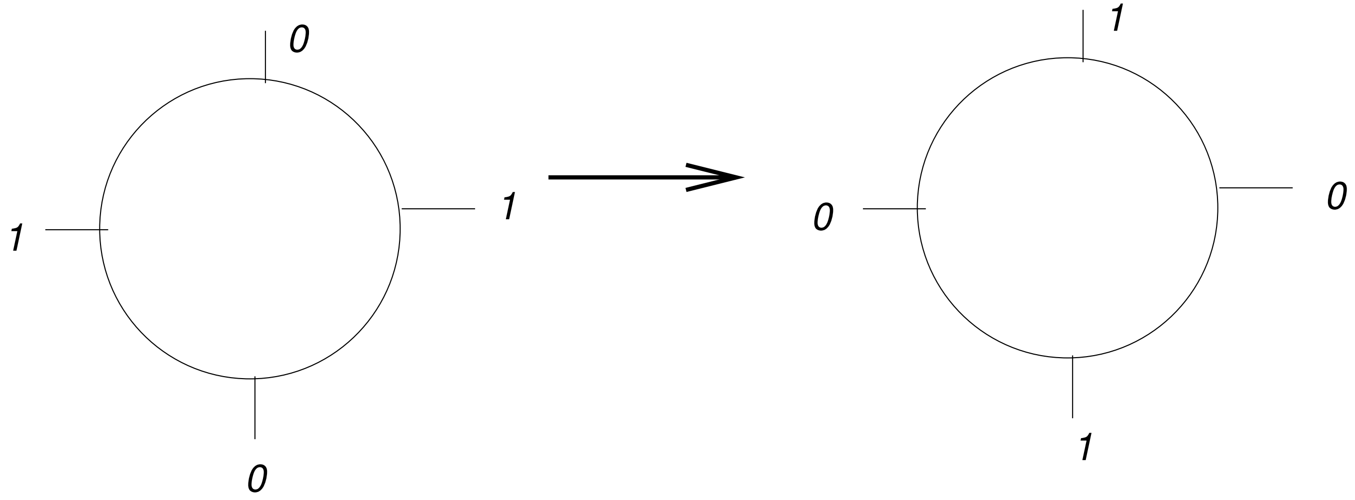
- Adding a *third phase* allows nodes to know votes after Phase 2. Can elect new coordinator if majority of nodes stay up. Majority rule needed to deal with *partitions.* More efficient state machine approaches in say ISIS.

6.2 Authenticated broadcast for Byzantine Agreement

- Nodes broadcast input values. *Goal*: good nodes agree on value set and use majority value.
- Each sender *signs* each message it sends or relays. One round broadcast sufficient for good nodes.
- Faulty node may send conflicting values to good nodes. To avoid this, good nodes rebroadcast values.
- Could receive through chain of faulty nodes; so rebroadcast for $f+1$ rounds.
- Complexity needed only if good nodes do not agree on input values. Not too expensive in common case if $f=1$. Used first in SIFT aircraft system.

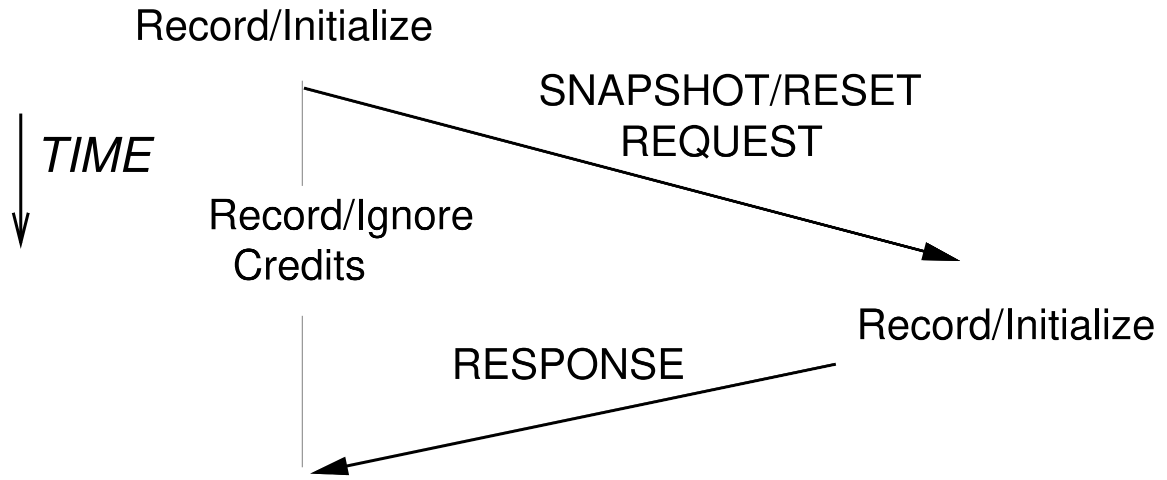
7.0 Some potential lessons from Self-stabilization

7.1 Self-stabilization appears to generalize soft-state



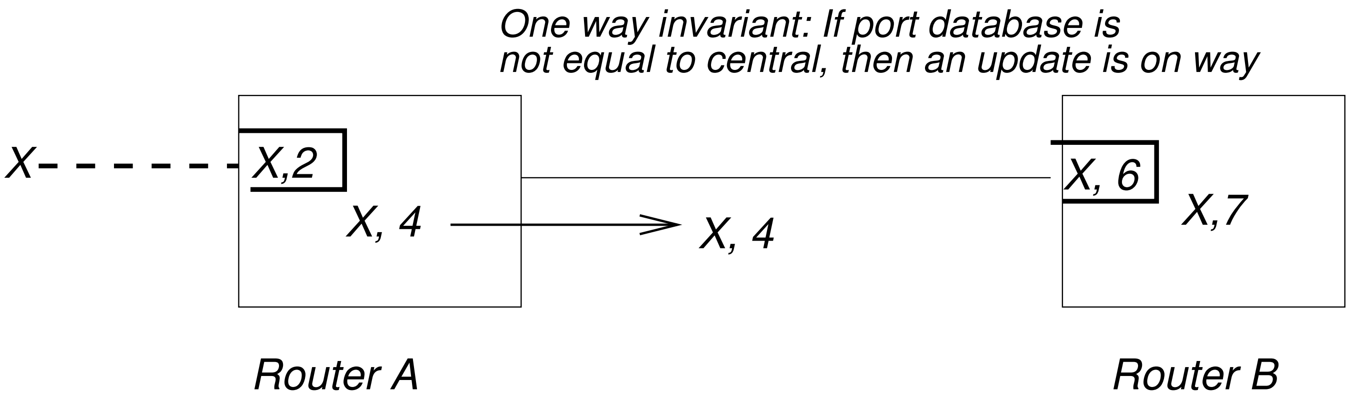
- Periodically sending state is necessary but insufficient in general. Ring can stay in bad states if counter size is too small and no leader.

7.2 General techniques for stabilization do exist



- For example, can run periodic distributed audits that can be made stabilizing to check state and correct if necessary. Such checks can help catch protocol and implementation bugs.

7.3 Periodic sending works with one-way invariants



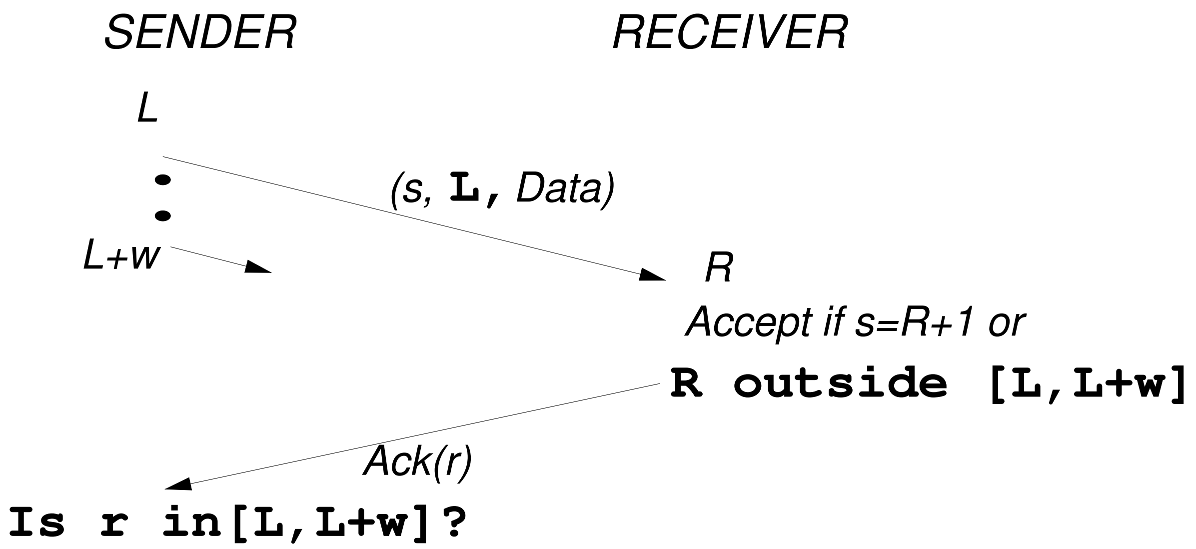
- Periodic sending as in RSVP and in say Distance Vector (e.g., RIP) works well if invariants are one way and only involve variables on one direction of link.

7.4 Proofs and Formal Methods are not Infeasible

Even proofs of abstractions of parts of a complex protocol help! Several generations:

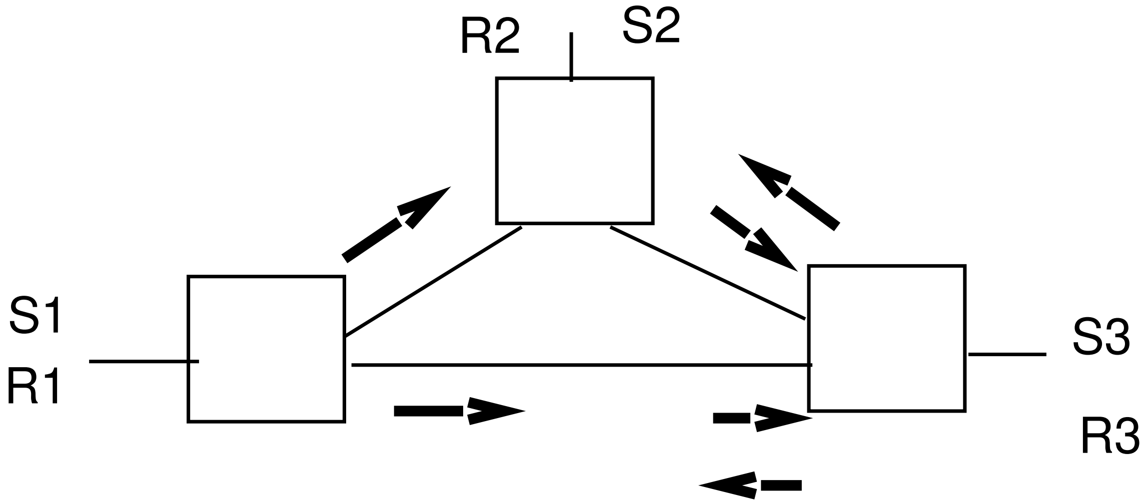
- **Old:** Tajnabis's proof of self-stabilization of distance vector gave Lauck the confidence to use distance vector in DECNET Phase IV
- **Middle-aged:** The AN-2 Reset algorithm to do routing (very different from Distance Vector or Link State) was proved correct by Leslie Lamport.
- **Baby:** Formal analysis of suitable abstraction of BGP by Griffin-Wilfong have captured policy aspects well and allowed Gao-Rexford to do a proof of stabilization under limited policies and limited initial conditions. Why not do similar proofs of other key protocols (e.g., MPLS label set up protocols?)

7.5 Stabilization Motivates Defensive Checks



- When receiving a message a common transformation in lots of stabilizing transformers is to weed out "obvious junk".
- More formally, when a message M is received in state s, accept M only if M, s can be extended to a legal global state. Generalizes obvious defensive checks and can help catch bugs not just errors.

7.6 Stabilization motivates "what if" thinking



- Stabilization frees us to think "what could happen if RSVP got stuck in a state where reservations form a loop". Attitude.
- Prevailing attitude among IP implementors and designers is more like "that's a corner case and can never happen". But corner cases do happen quite often in our lifetimes at Internet speeds.

8.0 Summing Up

- Self-stabilization appears to generalize soft-state.
- General techniques for stabilization do exist.
- Periodic sending works with one-way invariants
- Proofs and Formal Methods are not Infeasible.
- Stabilization Motivates Defensive Checks.
- Stabilization motivates "what if" thinking.

More papers, references on my web page at
http://www-cse.ucsd.edu/users/varghese/