Solving Hot Spot Contention Using InfiniBand Architecture Congestion Control

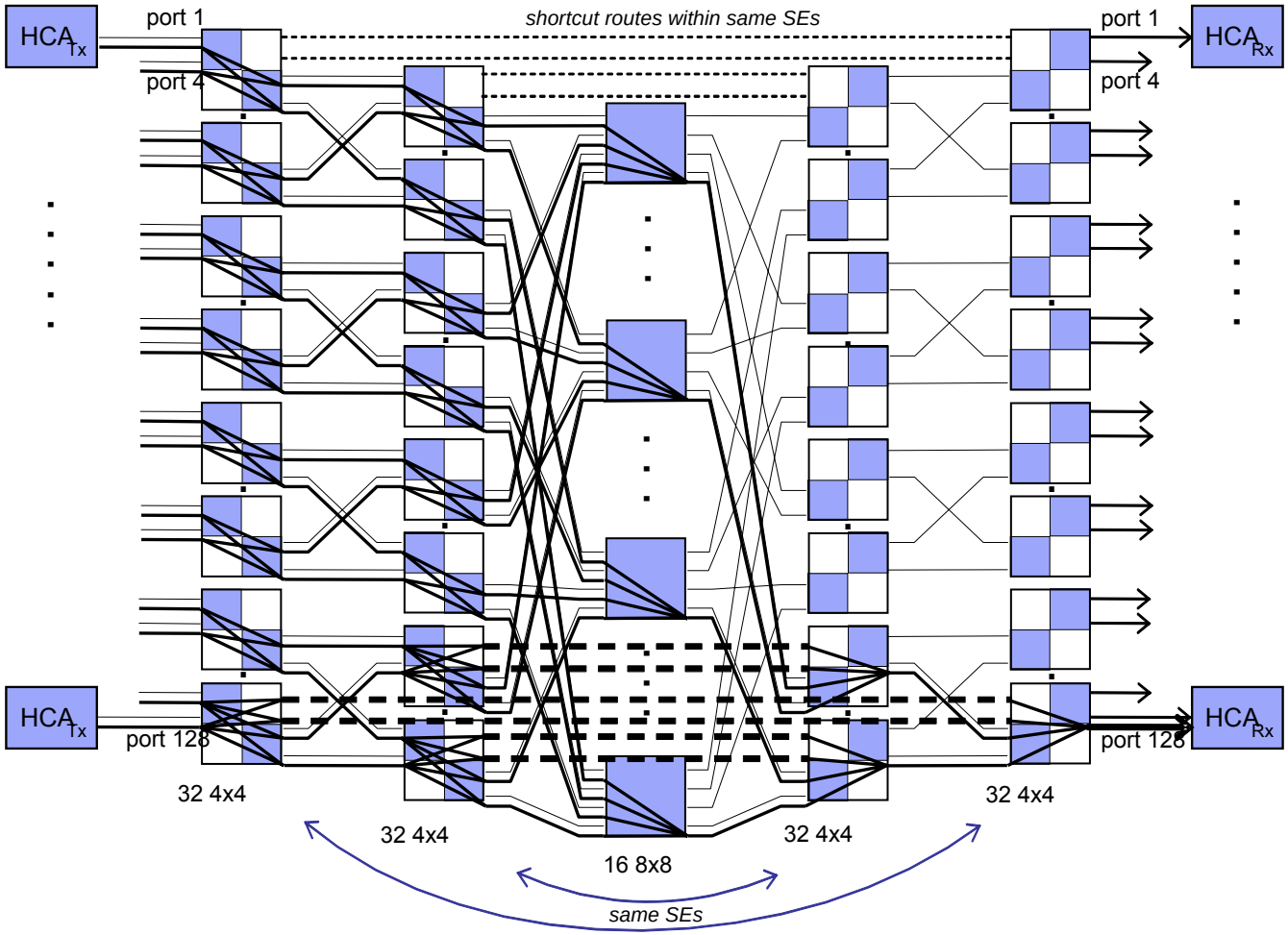
G. Pfister, M. Gusat, W. Denzel, D. Craddock, N. Ni, W. Rooney, T. Engbersen, R. Luijten, R. Krishnamurthy and J. Duato

G. Pfister (pfister@us.ibm.com) and N. Ni are with IBM Systems and Technology Group, Austin, TX USA. M. Gusat and W. Denzel are with IBM Research GmbH, Rüschlikon, Switzerland. D. Craddock and W. Rooney are with IBM Systems and Technology Group, Poughkeepsie, NY USA. J. Duato is with Technical University of Valencia, Spain.

Outline

- The problem
  - Demonstrated solution
- Why the problem matters
- How the solution works
  - Networks behaving badly
  - Preliminary guidelines
- Conclusions and Future directions

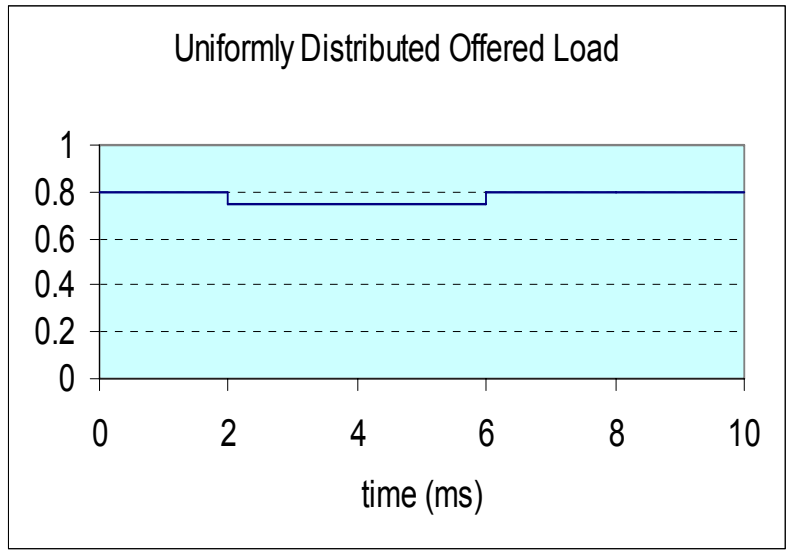
3-Stage 32-Port InfiniBand Fat Tree, Simulated

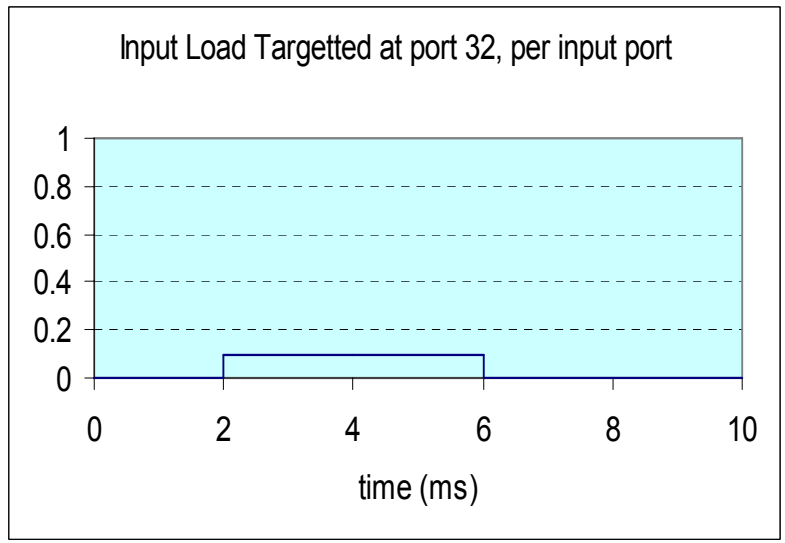


- Drawn unfolded: Up on left, Down on right.
- Dashes & dots are shortcut paths within switches

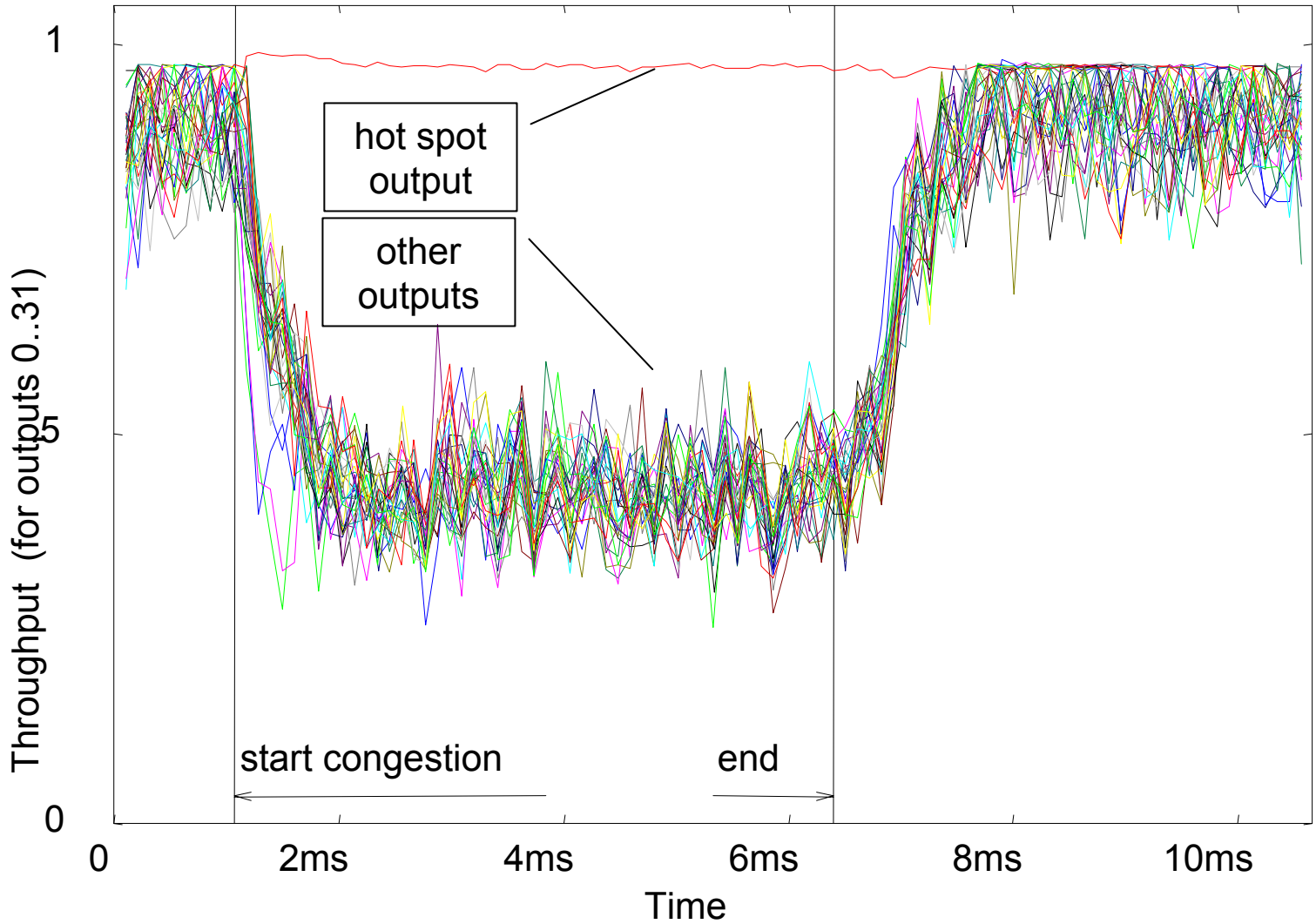
Input Traffic Pattern

- Run for a bit at 80% load, with destinations uniformly distributed from each source to each destination.
- 2 ms. into run, each input moves 9% of its load to port 32
  - Lower uniformly-distributed load to keep aggregate load constant.
- 4 ms. later, go back to original uniform load.



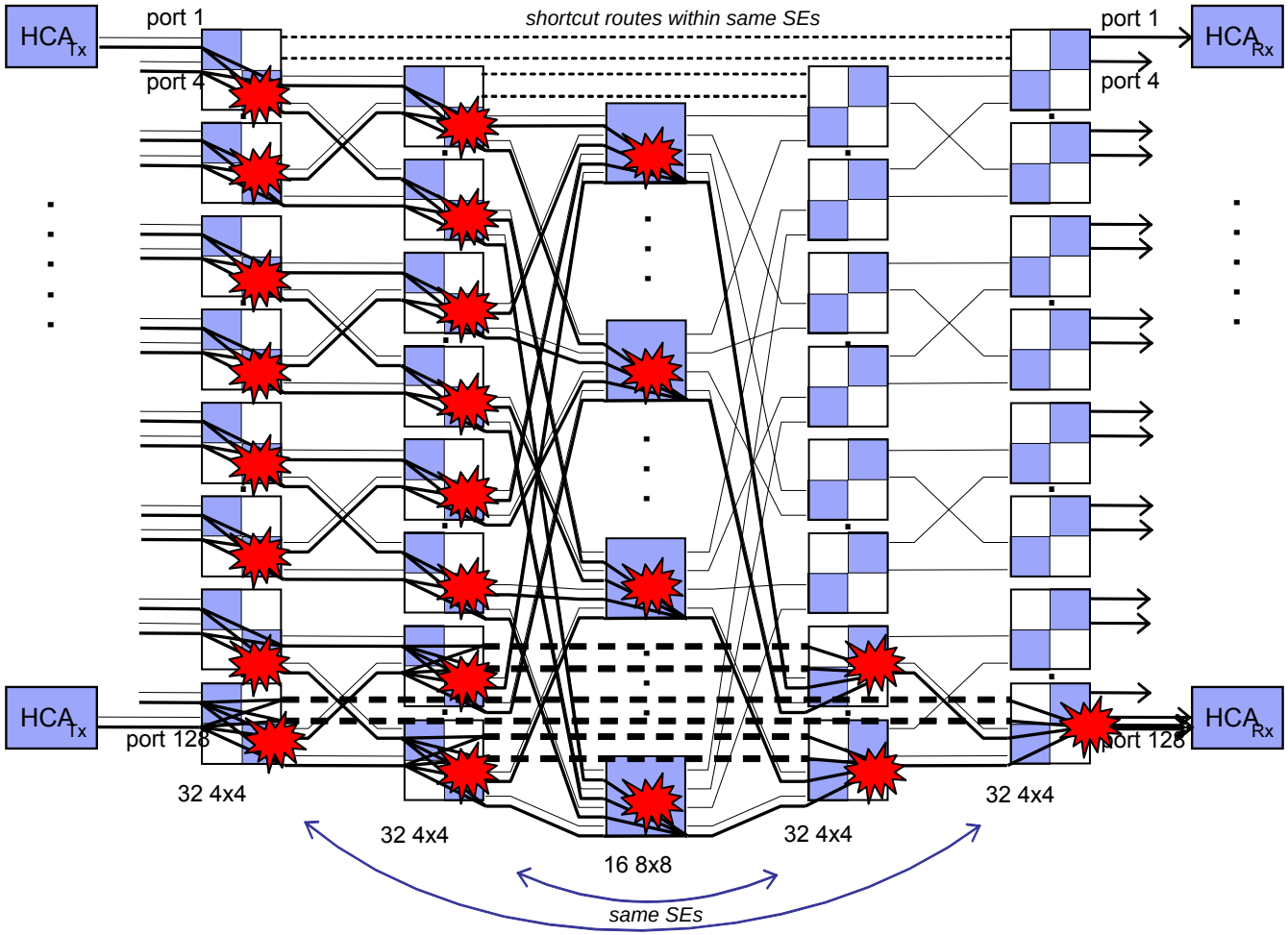


Result: Global Catastrophic Loss of Throughput



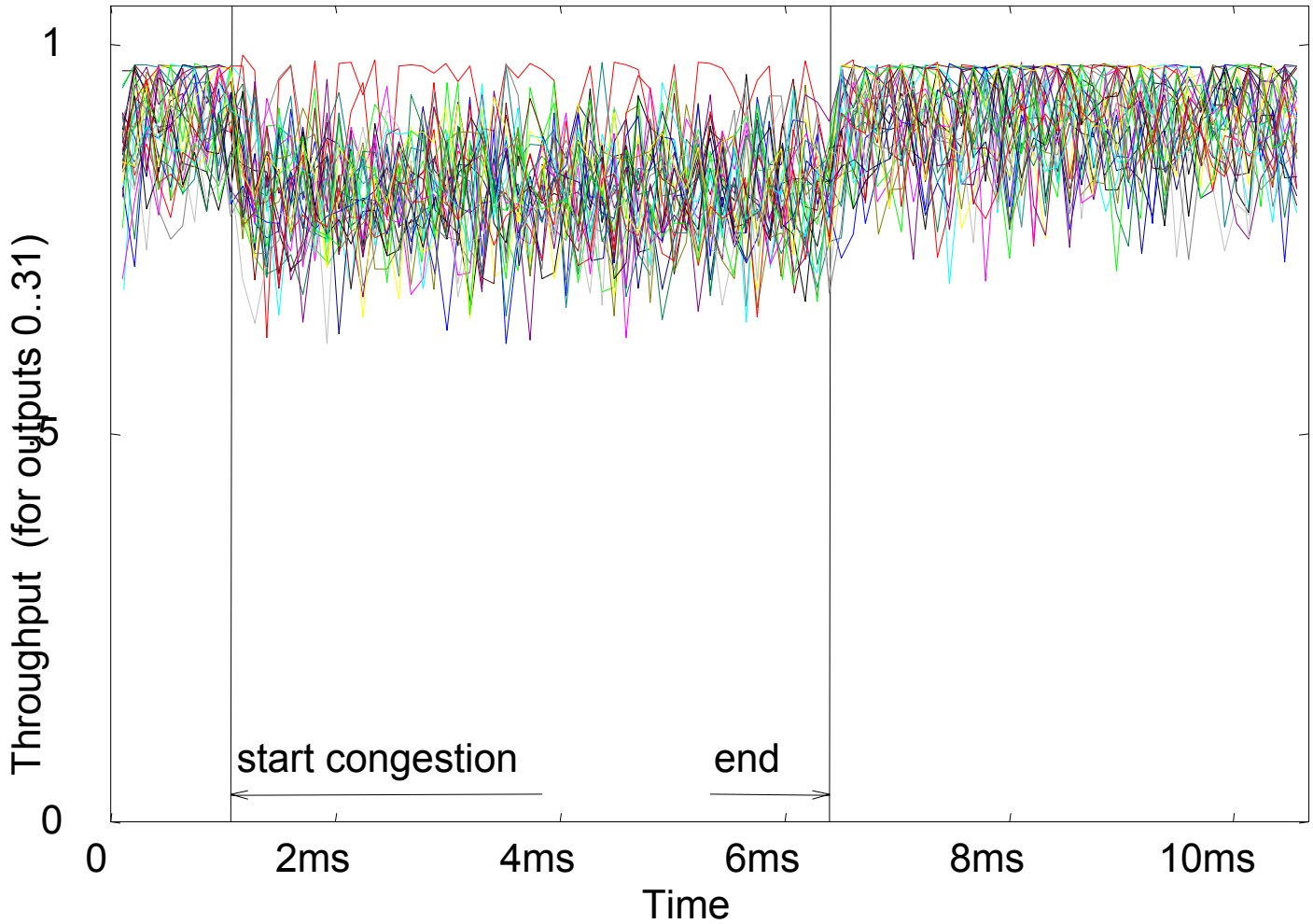
- Traffic to **one** port messes up everybody else.

Why: Tree Saturation / Congestion Spreading



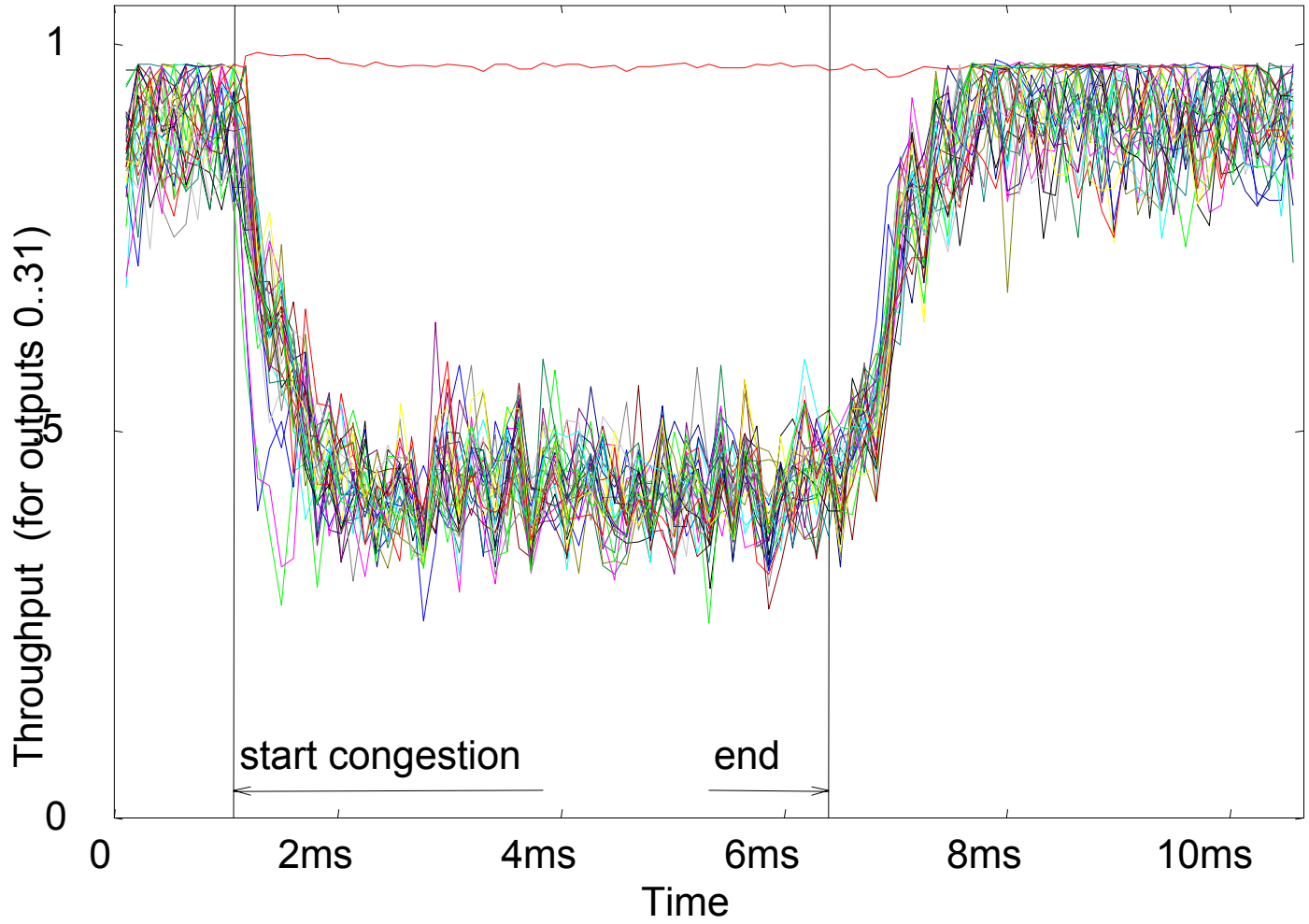
- Hot output link saturates; link-level FC fills queuing of next stage
- Exhausts all storage in switch; backs up to next stage; etc., until all traffic whacked.

Same, With InfiniBand Congestion Control



- Throughput drop = reduction in load keeping aggregate load constant.
- Simulations modeled product-purposed hardware designs as closely as possible.

Same, Without (Repeat)



Why Does This Matter?

- Solves a 20-year outstanding problem…
  - Pfister & Norton, “Hot Spot Contention and Combining in Multistage Interconnection Networks”, IEEE TC, 10/1985
- …for the first time.
  - not all co-authors agree with this; explanation later.
- Original paper spawned a mini-industry of efforts to solve the problem.
  - Eventually died out in the parallel arena because it didn’t seem to occur in practice. Why?

    (There are cases where it’s shown up in practice; has been treated as a bug.)

Why Aren't Networks Falling Over?

Factor	Current / Past	Emerging
Over-provisioning (mainly commercial)	Drown the problem in bandwidth; it's cheap.	Doesn't scale with # of nodes; node # scaling will be increasingly needed (Moore's law freq scaling KO). Msg sizes rising dramatically (XML)
Single use systems (mainly HPC)	One problem per machine → congestion is a bug	Virtualization on and of clusters, so communication not algorithmically predictable
Must have > 1 message outstanding	Early system limit; single use; little multiprogramming	All CPU chips multithread, multi-core; multiprogramming must be used

- Net: They'll start falling over pretty soon.

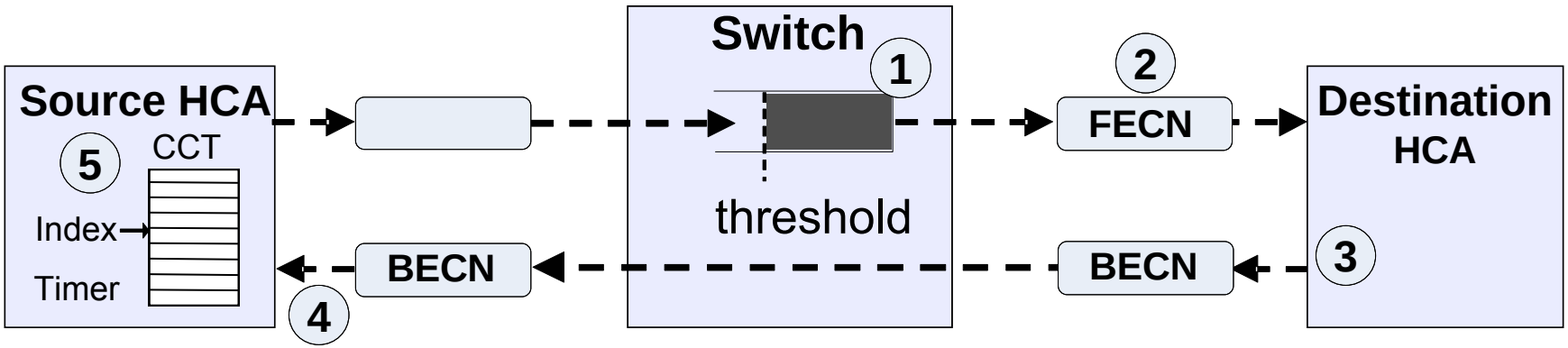
Why Prior Work Doesn't Fix It

- **Assumes hot flow pre-identification before running**
  - Divert hot flow to a separate lane, network, queue, or …
  - Can help, but it's hard to predict bugs. (ISP xmp)
  - Also: Predict → rearrange to avoid the problem
- **Works – if the hot flow stops soon enough**
  - Frantically reorganize switch element buffers to free up space
  - Only works for a limited time
  - IBA CC quenches the source.
- **Shuts down entire offending port**
  - May be many virtual systems behind one physical port.
  - IBA CC just targets the hot flow(s) of a port
- **Targets lossy networks**
  - Large amount of work on IP networks has little applicability: no link-level flow control (also other issues, like speed)

Not that these techniques aren't useful and good; they just aren't sufficient to solve this particular problem.

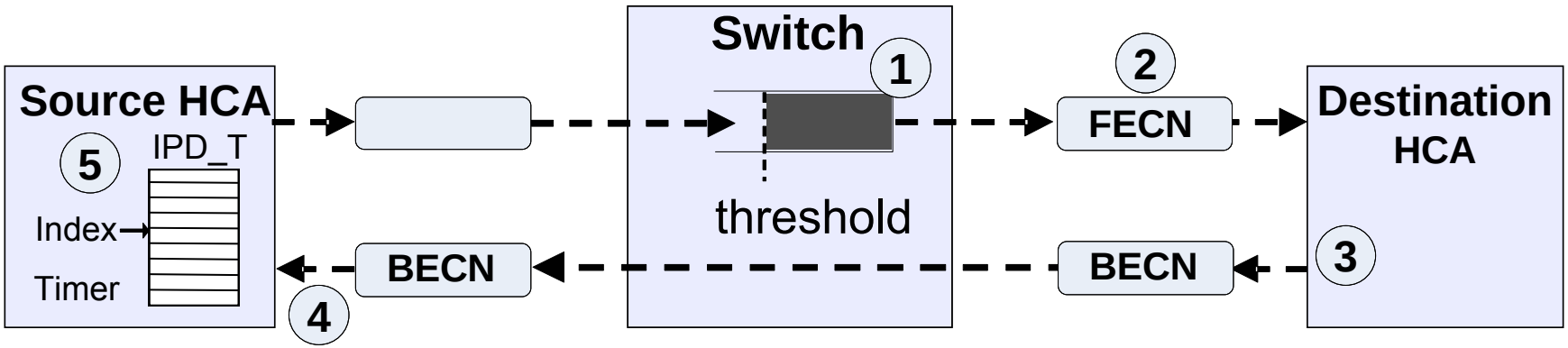
(Detailed references given in the printed paper.)

How It Works (1)



1) **Detect congestion**
   - Queue for VL > threshold
   - Optional: Mark only if root = output credits available
2) **Propagate out**: set Forward Explicit Congestion Notification bit on all packets as long as congestion exists
3) **Propagate back**: receive FECN, respond with Backward Explicit Congestion Notification bit
   - Part of normal response, or congestion notification packet
   - Sent back to source Queue Pair

How It Works (2): Responding



4) **Determine Added Delay**: Whenever receive a BECN for QP, increment index into Inter-Packet Delay Table
   - One index per QP
5) **Apply Delay**: When sending, wait for the Inter-Packet Delay specified in IPD Table entry
6) **Decay the Delay**: Whenever a timer pops, decrement all IPD Table indices on CA.

See IB Architecture Vol. 1 Rel 1.2 for details.

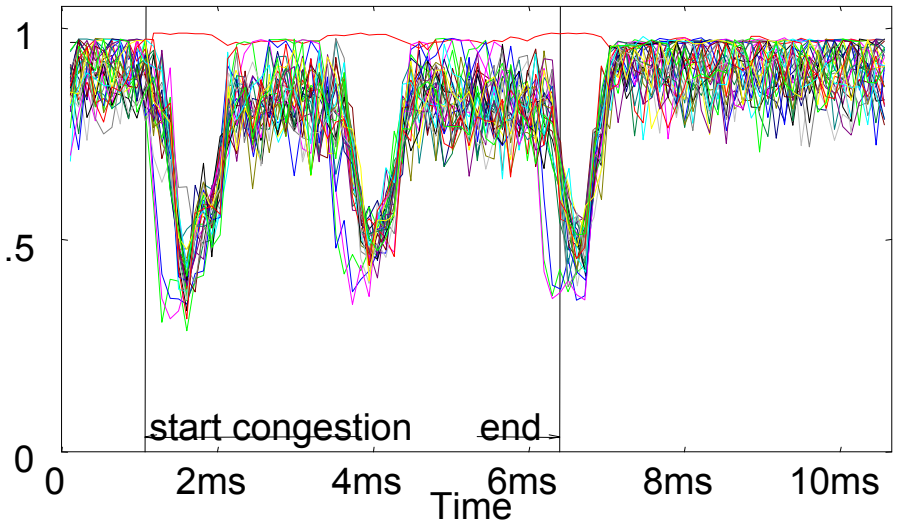
But Does That Really Work?

- Many parameters need to be chosen:
  - Length of IPD Table
  - IPD Table entry values
  - Timer period
  - Detection threshold
  - IPD Table index increment
- This is a feedback system with delay
  - Worse because of forward/back propagation, chosen to keep switches simple while targeting only the hot flows
- Can you get bad behavior?
- Yes.

Networks Behaving Badly

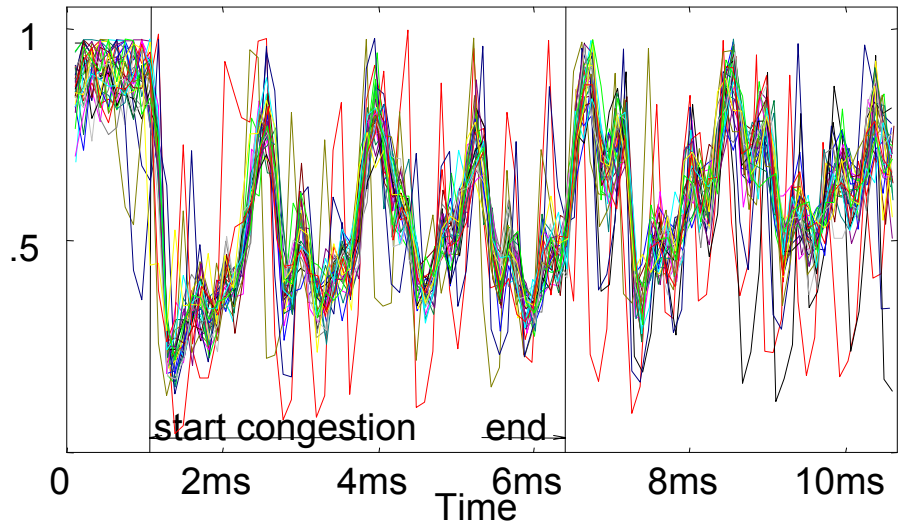
Traffic case 4 (a)
high load: 0.9
large hotspot degree: 32
hotspot severity: 300%

IPD index step too low (=2)



Traffic case 4 (b)
high load: 0.9
large hotspot degree: 32
hotspot severity: 300%

IPD index step too high (=40)

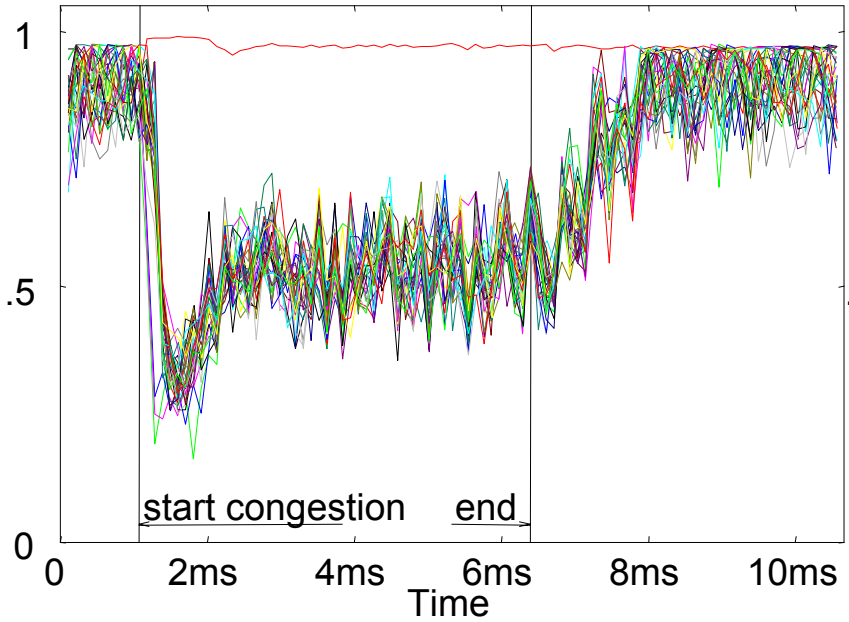


- Cannot chose value at one end of the scale; either end produces bad behavior.

Networks Behaving Badly, 2

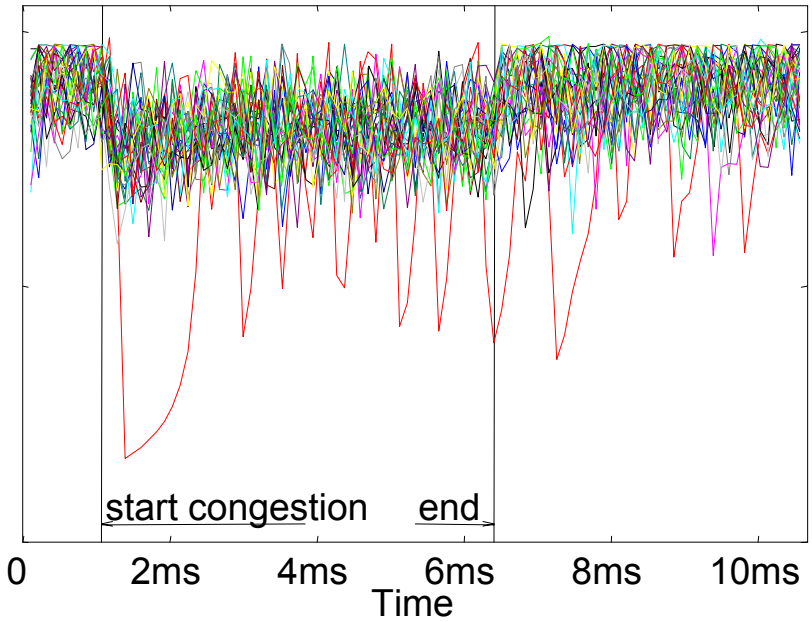
Traffic case 4 (c)
high load: 0.9
large hotspot degree: 32
hotspot severity: 300%

IPD recovery timer too fast (=2.6us)



Traffic case 4 (d)
high load: 0.9
large hotspot degree: 32
hotspot severity: 300%

IPD recovery timer too slow (=84us)



- Recovery timer can't be pushed to an extreme, either

But It Appears Tunable

- We have run literally 100s of simulation cases
  - Networks: 8, 32, 128, 432, 256, 512 ports; 2x2, 8x8, 16x16 switches; 2, 3, 8 stages of switching; fat tree and Omega (Banyan) network.
  - Traffic & hot-spot cases:
    - 50%-90% base background rate
    - All nodes contribute to hot spot, or just 3 (fire hose effect)
    - Hot spot very severe (300% link capacity) or just enough to cause trouble (100%+ε).
- Have ***always*** managed to find a set of parameters that squelch the problem.

Preliminary Guidelines

- N = network size, i.e., number of ports
- H = maximum hot spot degree
  - Worst case = N, but can be less if system partitioning known.
- Absolute time (μsec) values are referenced to simulated RTTs of 2-20 μsec. without congestion.

Item	**Value**
IPD Table size	128
Switch threshold for detection	90% of queue capacity, with some hysteresis (not critical)
IPD table index increment	Min(1/6•N, 1/2•H)
Max IPD value	2/3 H μsec.
Recovery timer	10 μsec.

Conclusions & Future Work

- The 20-year-old problem of tree saturation (congestion spreading) is reappearing.
- Automatic congestion control is necessary; congestion avoidance becoming difficult or impossible
  - cannot assume pre-knowledge or congestion time limits
  - application dynamics create traffic unpredictability
  - resilient self-healing systems running mission-critical applications – application migration, virtualization, etc.
  - peer-to-peer protocols increase difficulty rebalancing
  - Bugs!
- InfiniBand's Congestion Control offers a solution, which we have verified in a wide variety of cases.
- But it is a feedback system that can behave badly.
- We've offered preliminary guidelines for parameter setting
- Future:
  - More detailed guidelines
  - Other cases: output throttling, multiple hot spots
  - Investigate other marking and throttling mechanisms.