Multistage Interconnection Networks are not Crossbars

- A Case Study with Infiniband -

Torsten Hoefler
Indiana University

talk at: Indiana University
Bloomington, IN, USA

22nd April 2009

Some questions that will be answered

1) How do large-scale HPC networks look like?
2) What is the ”effective bandwidth”?
3) How are real-world systems affected?
4) How are real-world applications affected?
5) How do we design better networks?

High Performance Computing

- large-scale networks are common in HPC
- growing application needs require bigger networks
- parallel applications depend on network performance
- it's often unclear how the network influences time to solution
- some network metrics are questionable (bisection bandwidth)

Networks in HPC

- huge variety of different technologies
- Ethernet, InfiniBand, Quadrics, Myrinet, SeaStar ...
- OS bypass
- offload vs. onload
- and topologies
- directed, undirected
- torus, ring, kautz network, hypercubes, different MINs ...

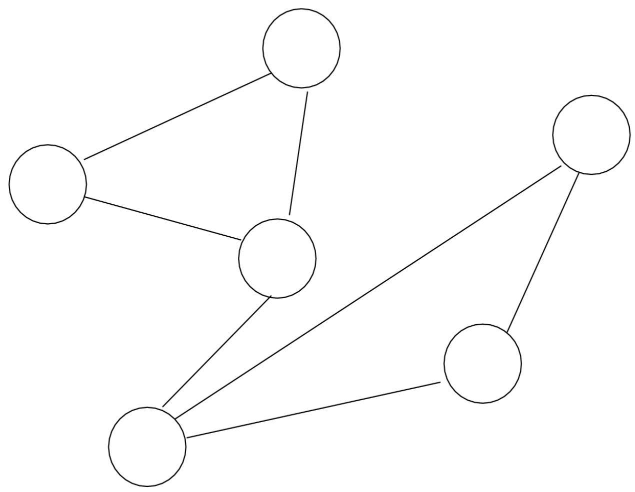
➔ we focus on topologies

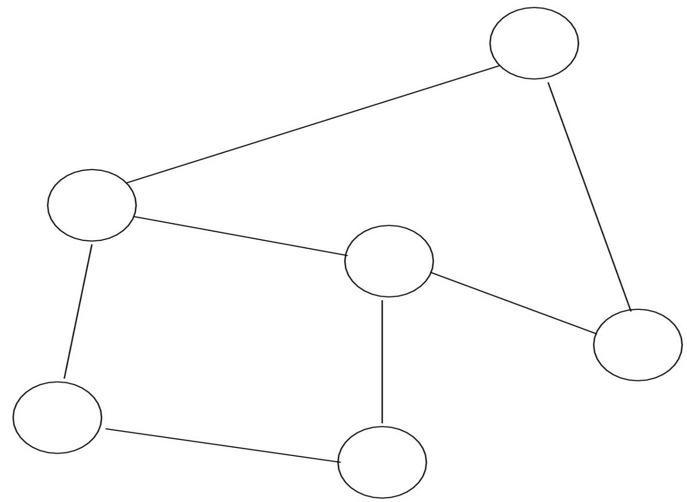
What Topology?

- Topology depends on expected communication patterns
  - e.g., BG/L network fits many HPC patterns well
  - impractical for irregular communication
  - impractical for dense patterns (transpose)
  - many applications are irregular (sparse matrixes, graph algorithms)
- We want to stay generic
  - fully connected not possible
  - must be able to embed many patterns efficiently
  - needs high bisection bandwidth
  - ➔ Multistage Interconnection Networks (MINs)

Bisection Bandwidth (BB)

Definition 1: For a general network with N endpoints, represented as a graph with a bandwidth of one on every edge, BB is defined as the minimum number of edges that have to be removed in order to split the graphs into two equally-sized unconnected parts.



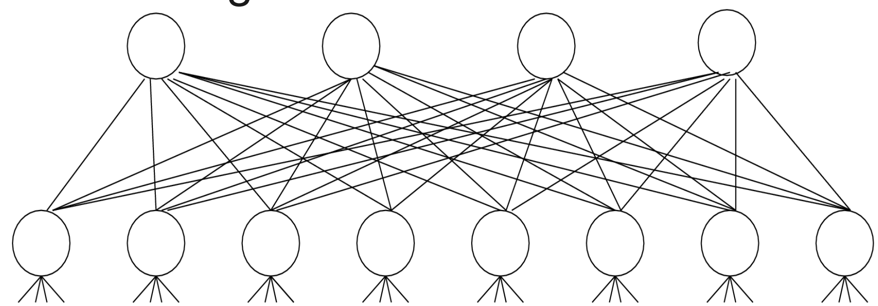


Definition 2: If the bisection bandwidth of a network is N/2, then the network has full bisection bandwidth (FBB).

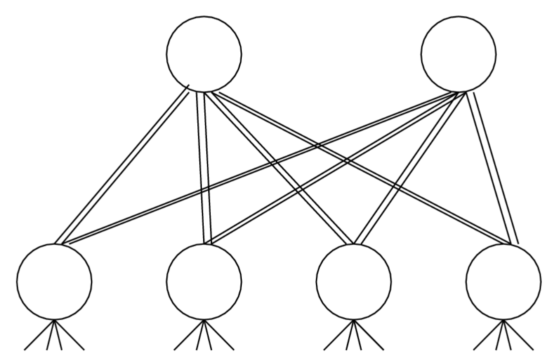
→MINs usually differentiate between terminal nodes and crossbars – next slide!

Properties of common MINs

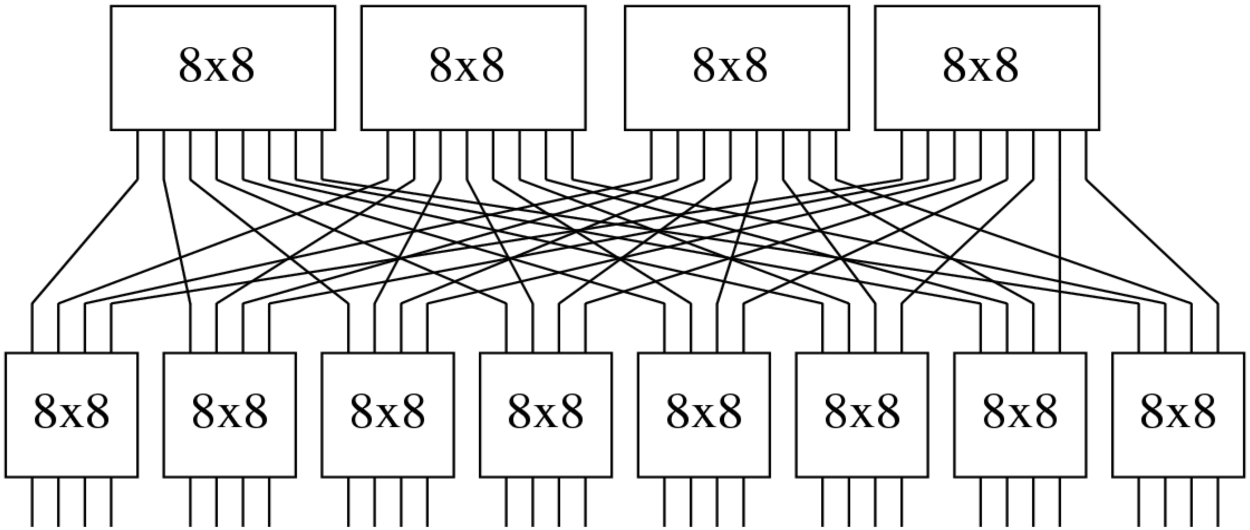
- Clos Networks [Clos'53]
- blocking, rearrangable non-blocking, strictly non-blocking
- focus on rearrangable non-blocking
- full bisection bandwidth
- $\frac{N}{2}+N$ NxN crossbar elements
- $\frac{N}{2}\times N$ endpoints
- $\frac{N}{2}\times N$ spine connections
- recursion possible



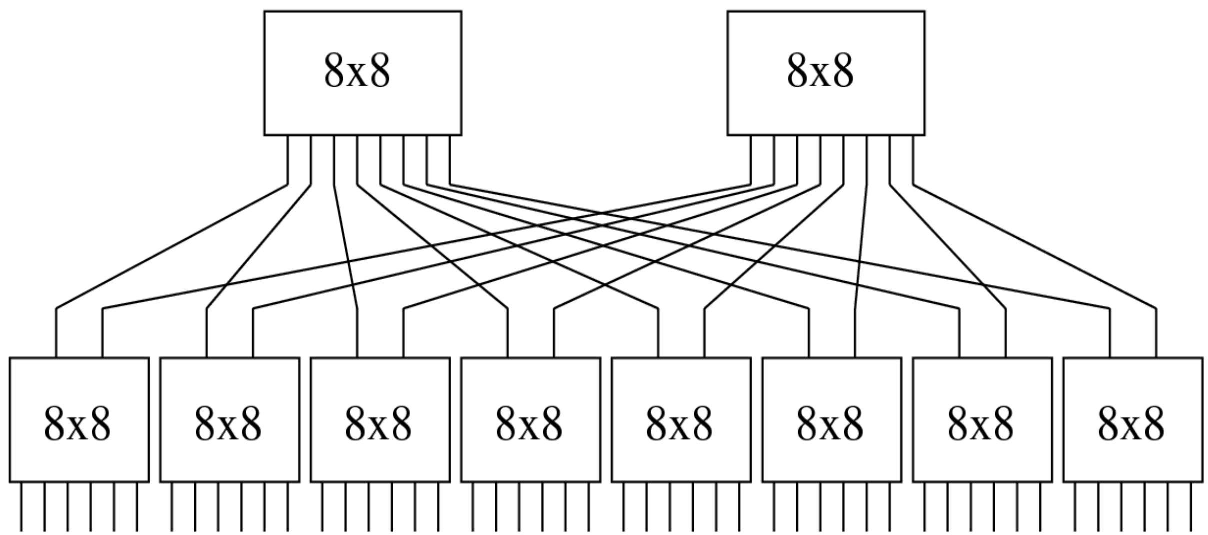
- Fat Tree Networks [Leiserson'90]
  - ”generalisation” of Clos networks
  - adds more flexibility to the number of endpoints
  - similar principles



Real-World MINs



Clos Network
1:1



Fat Tree Network
1:3

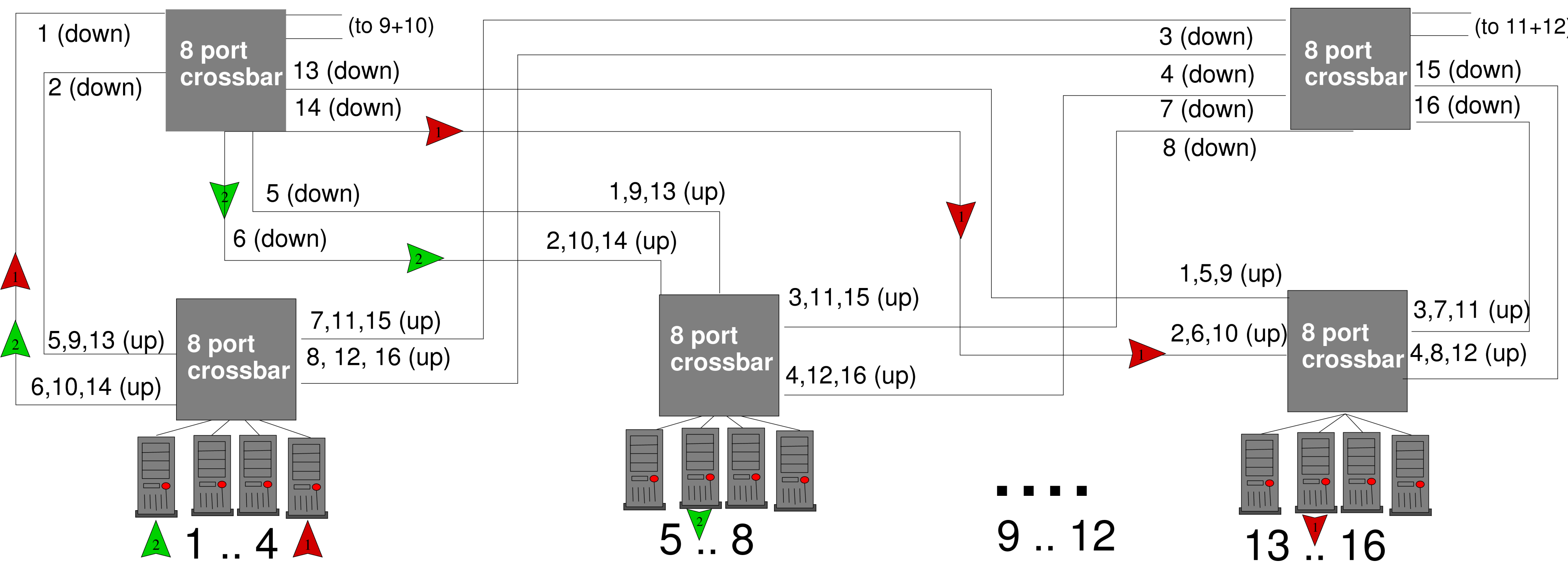
Routing Issues

- Many networks are routed statically (oblivious)
  - i.e., routes change very slowly or not at all
  - e.g., Ethernet, InfiniBand, IP, ...
- Many networks have distributed routing tables
  - even worse (see later on)
  - network-based routing vs. host-based routing
- Some networks route adaptively
  - Myrinet, Quadrics, ...
  - there are theoretical constraints
  - fast changing comm-patterns with small packets are a problem
  - very expensive (globally vs. locally optimal)

Case-Study: InfiniBand

- Statically distributed routing:
  - Subnet Manager (SM) discovers network topology with source-routed packets
  - SM assigns Local Identifiers (cf. IP Address) to each endpoint
  - SM computes N(N-1) routes
  - each crossbar has a linear forwarding table (FTP -> destination, port)
  - SM programs each crossbar in the network
- Practical data:
  - Crossbar-size: 24 (32 in the future)
  - Clos network: 288 ports (biggest switch sold for a long time)
  - 1 level recursive Clos network: 41472 ports (859 Mio with 2 levels)
  - biggest existing chassis: 3456 ports (fat tree)
    - I would build it with 32 288 port Clos switches

A FBB Network and a Pattern

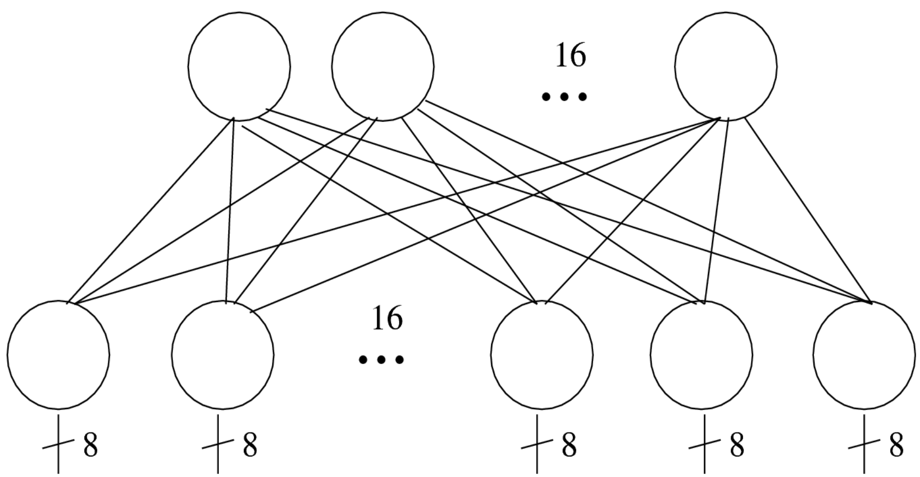


- This network has full bisection bandwidth!
- We send two messages from/to two distinct hosts and get half ½ bandwidth
- (1 to 7 and 4 to 8) D'oh!

Source: "MINs are not Crossbars", T. Hoefler, T. Schneider, A. Lumsdaine (to Appear in Cluster 2008)

Quantifying and Preventing Congestion

- quantifying congestion (link-oversubscription) in Clos/Fat Tree networks:
  - best-case: 0
  - worst-case: N-1
  - average-case: ??? (good question)
- lower congestion:
  - build strictly non-blocking Clos networks ($m \geq 2n-1$)
  - example InfiniBand (m+n=24; n=8; m=16)



288 port example

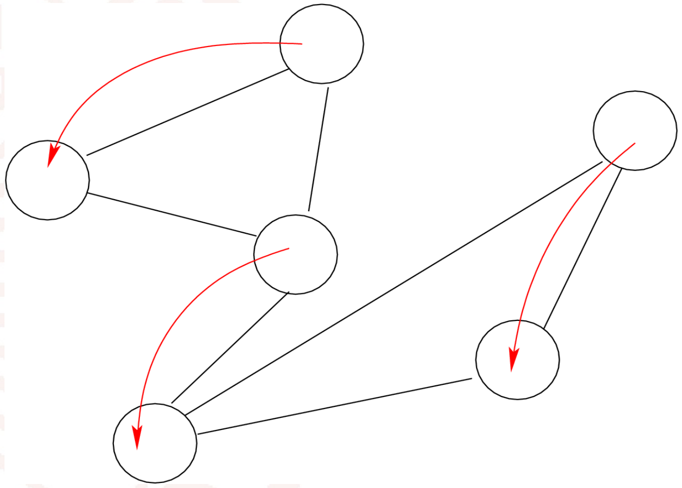
- many more cables and cbs per port
  - 16+16 cbs, 8*16 ports
  - 0.25 cb/port
- original rearrangable nb Clos network:
  - 24+12 cbs, 24*12 ports
  - 0.125 cb/port
- not a viable option

What does BB tell us in this Case?

- both networks have FBB!
- real bandwidth's are different!

- is BB a lower bound to real BW?
  - no, see example – FBB, but less real BW
- is BB an upper bound to real BW?
  - no, see example (red arrows are messages)
- is BB the average real BW?
  - will see (will analyze average BW)

- what's wrong with BB then?
  - it's ignoring the routing information



Effective Bisection Bandwidth (eBB)

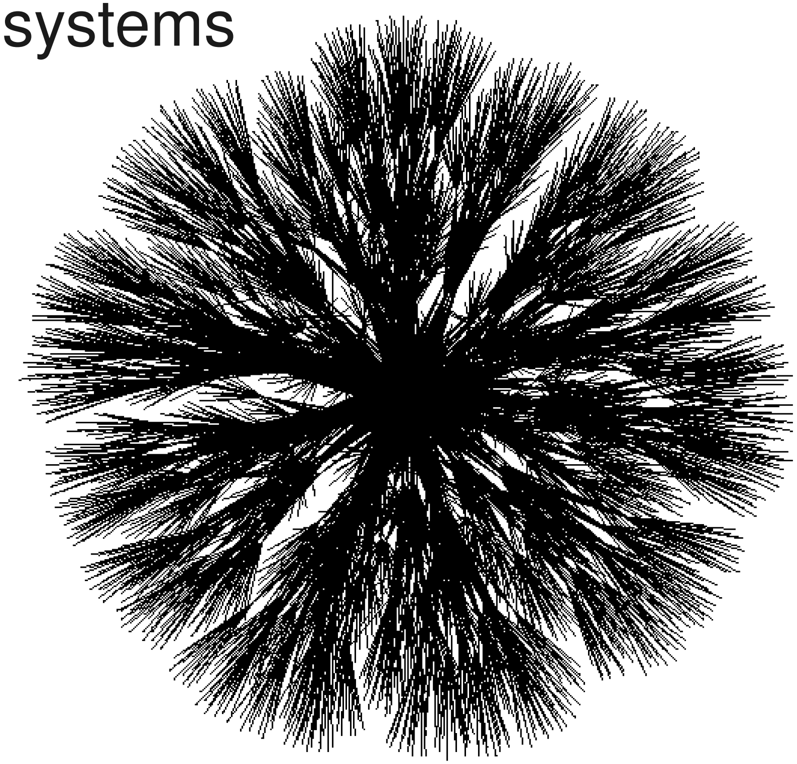
- eBB models real bandwidth
  - defined as the average bandwidth of a bisect pattern
- constructing a 'bisect' pattern:
  - divide network in two equal partitions A and B
  - find a peer in the other partition for every node such that every node has exactly one peer
- $\binom{N}{\frac{N}{2}}$ possible ways to divide N nodes
- $\frac{N}{2}!$ possible ways to pair 2 times N/2 nodes up
- huge number of patterns
  - at least one of them has FBB
  - many might have trivial FBB (see example from previous slide)
  - no closed form yet -> simulation

The Network Simulator

- model physical network as graph
  - routing tables as edge-properties
- construct a random bisect pattern
  - simulate packet routing and record edge-usage
  - compute maximum edge-usage (e) along each path
- bandwidth per path = 1/e
- compute average bandwidth
- repeat simulation with many patterns until average-bw reached confidence interval (e.g., 100000)
- report some other statistics

Simulated Real-World Networks

- retrieved physical network structure and routing of real-world systems (ibnetdiscover, ibdiagnet)
- Four large-scale InfiniBand systems
  - Thunderbird at SNL
  - Atlas at LLNL
  - Ranger at TACC
  - CHiC at TUC



Thunderbird @ SNL

- 4096 compute nodes
- dual Xeon EM64T 3.6 Ghz CPUs
- 6 GiB RAM
- ½ bisection bandwidth fat tree
- 4390 active LIDs while queried

source: http://www.cs.sandia.gov/platforms/Thunderbird.html

Atlas @ LLNL

- 1152 compute nodes
- dual 4-core 2.4 GHz Opteron
- 16 GiB RAM
- full bisection bandwidth fat tree
- 1142 active LIDs while queried

source: https://computing.llnl.gov/tutorials/linux_clusters/

Ranger @ TACC

- 3936 compute nodes
- quad 4-core 2.3 GHz Opteron
- 32 GiB RAM
- full bisection bandwidth fat tree
- 3908 active LIDs while queried

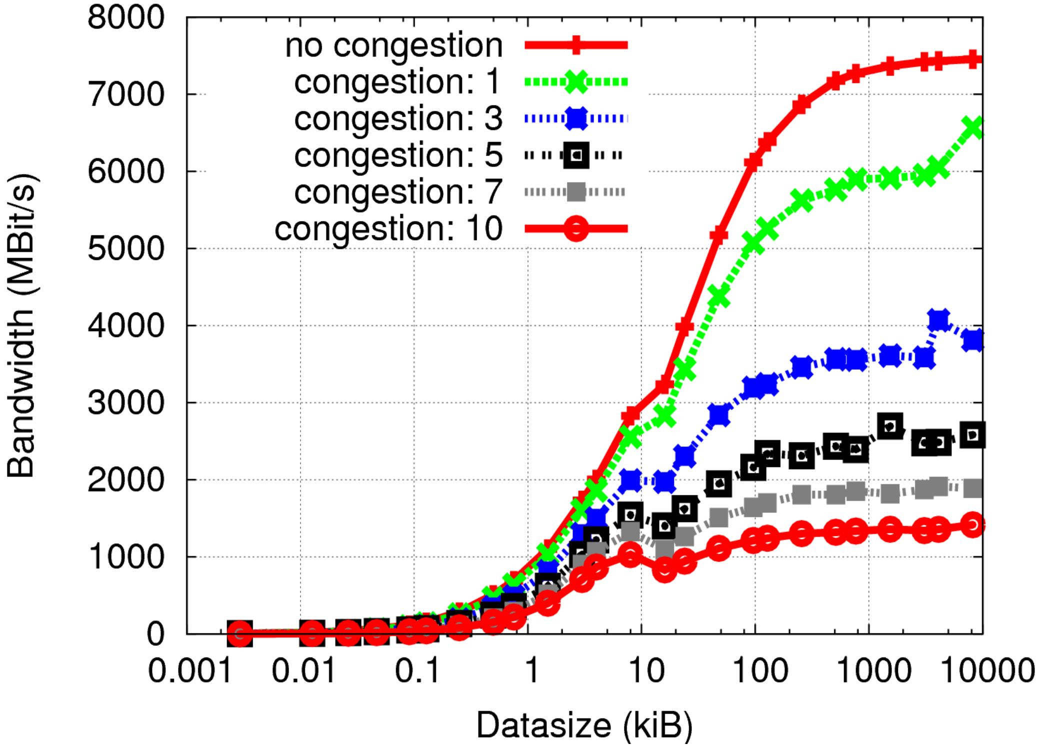
source: http://www.tacc.utexas.edu/resources/hpcsystems/

CHiC @ TUC

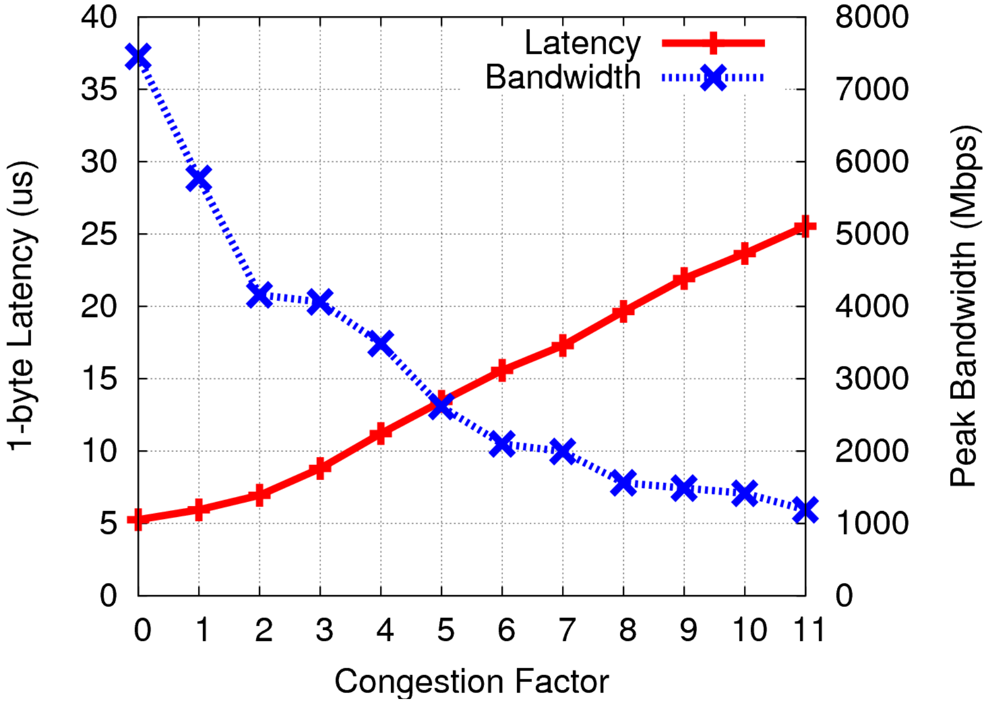
- 542 compute nodes
- dual 2-core 2.6 GHz Opteron
- 4 GiB RAM
- full bisection bandwidth fat tree
- 566 active LIDs while queried

source: http://www.tacc.utexas.edu/resources/hpcsystems/

Influence of Head-of-Line blocking



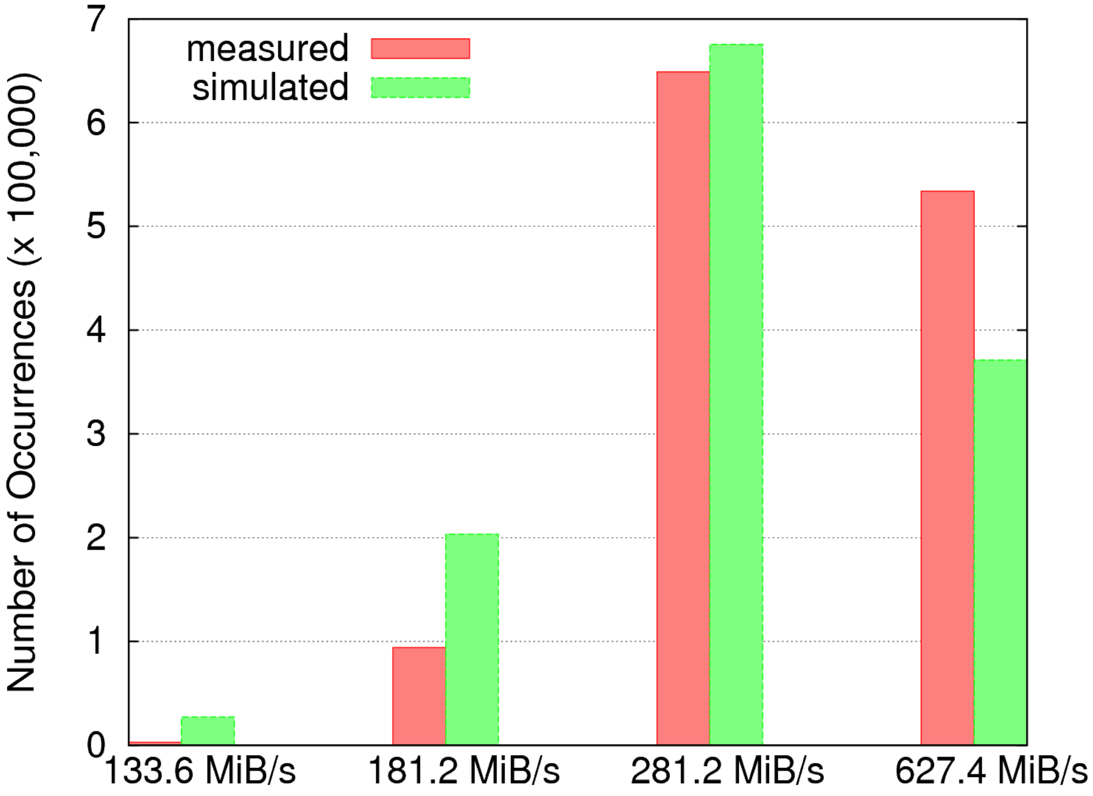
- communication between independent pairs (bisect)
- laid out to cause congestion



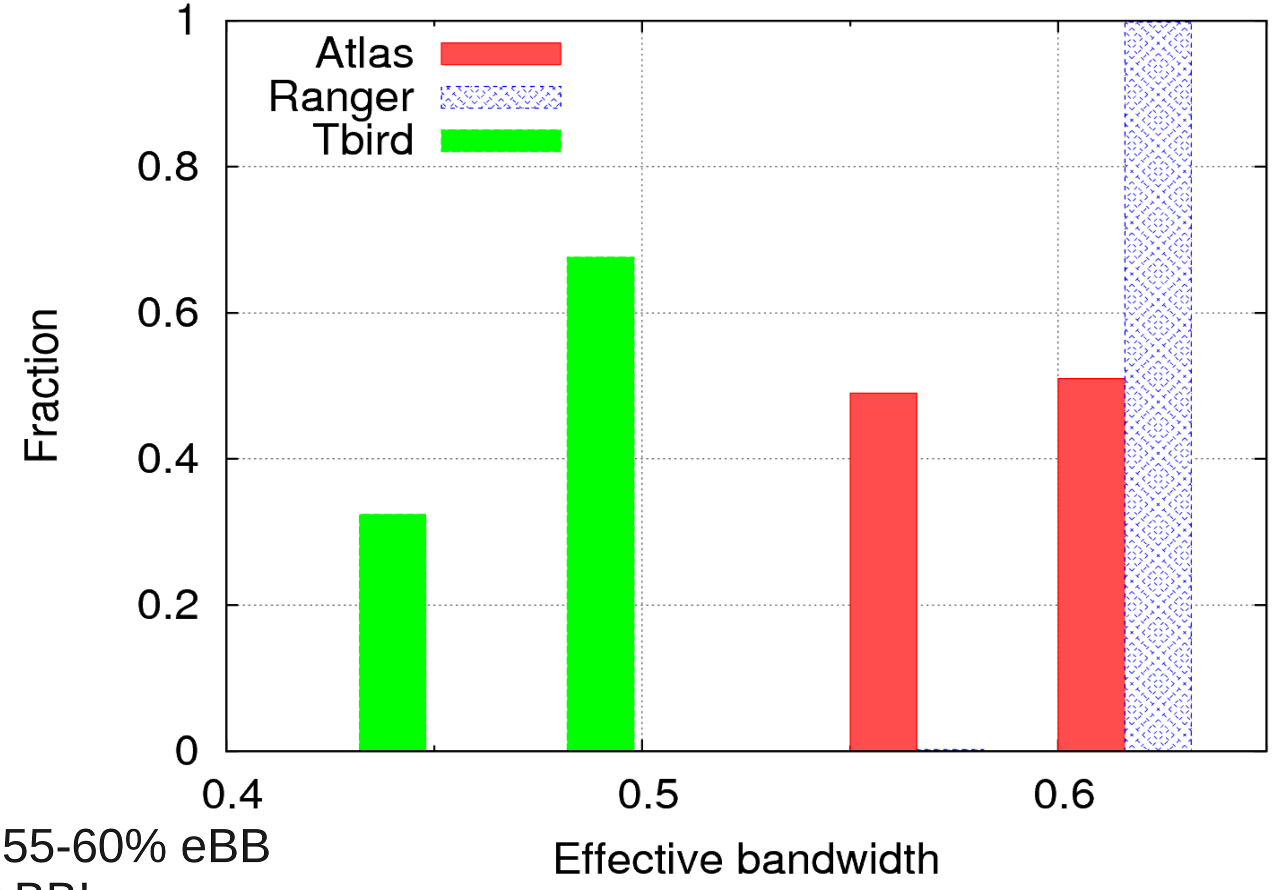
- maximum congestion: 11

Simulation and Reality

- compare 512 node CHiC full system run and 566 node simulation results
- random bisect patterns, bins of size 50 MiB/s
- measured and simulated >99.9% into 4 bins!



Simulating other Systems



- Ranger: 57.6%
- Atlas: 55.6%
- Thunderbird: 40.6%
  - FBB networks have 55-60% eBB
  - ½ BB still has 40% eBB!

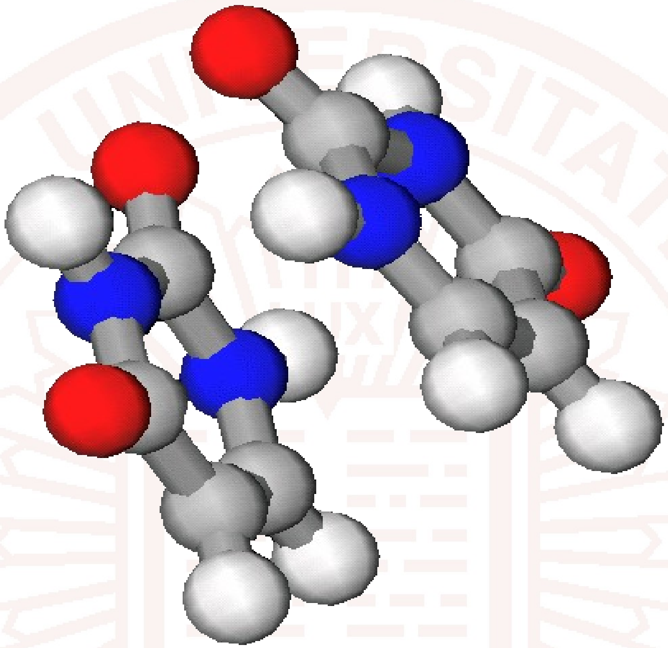
Other Effects of Contention

- not only reduced bandwidth, also:
  - the bandwidth varies with pattern and routing
  - not easy to model/predict
  - effects on latency are not trivial (buffering, ...)
  - buffering problems lead to message-jitter
  - leads to ”network skew” (will be a problem at large scale)

That's all Theory, what about Applications?

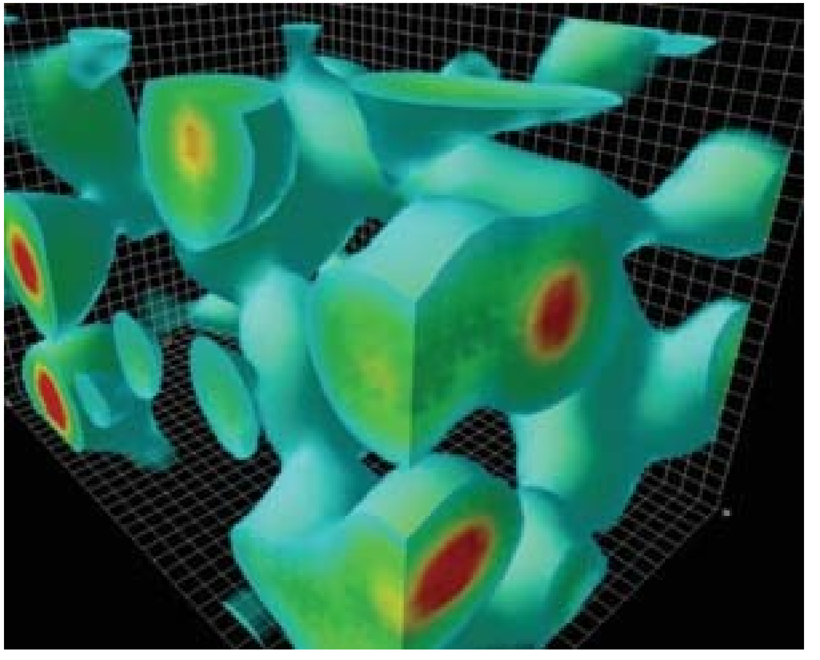
- analyzed four real-world applications
- traced their communication on 64-node runs
- HPC centric
- no data-center data
- more input-data is welcome!

Application 1: MPQC



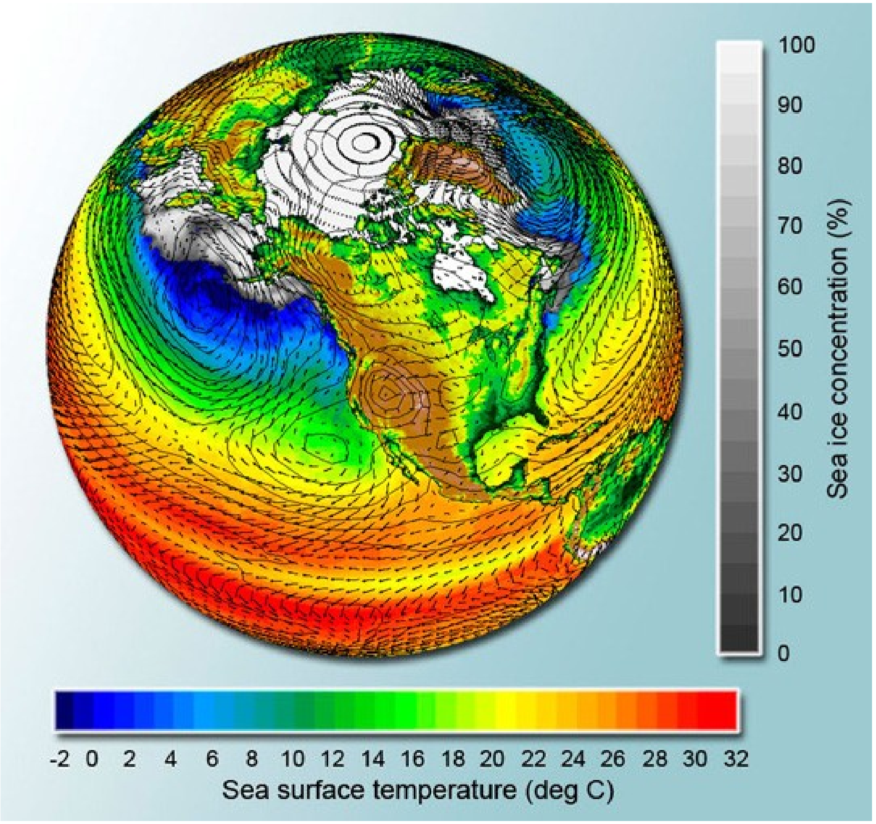
- Massively Parallel Quantum Chemistry Program (MPQC)
  - Thanks to Matt Leininger for the Input!
  - 9.2% communication overhead
  - MPI_Reduce: 67.4%; MPI_Bcast: 19.6%; MPI_Allreduce: 11.9%

Application 2: MIMD



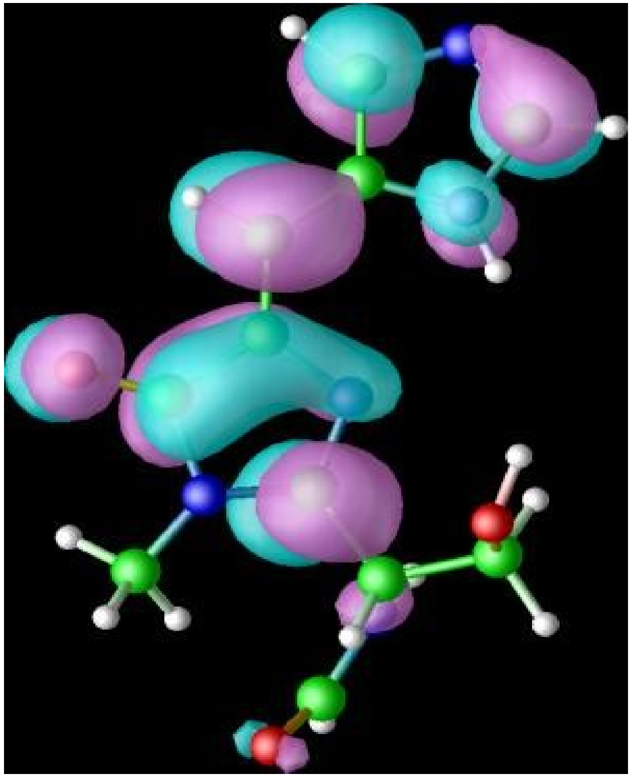
- MIMD Lattice Computation (MILC)
  - 9.4% communication overhead
  - P2P: 86%; MPI_Allreduce: 3.2%

Application 3: POP



- Parallel Ocean Program (POP)
  - 32.6% communication overhead
  - P2P: 84%; MPI_Allreduce: 14.1%

Application 4: Octopus



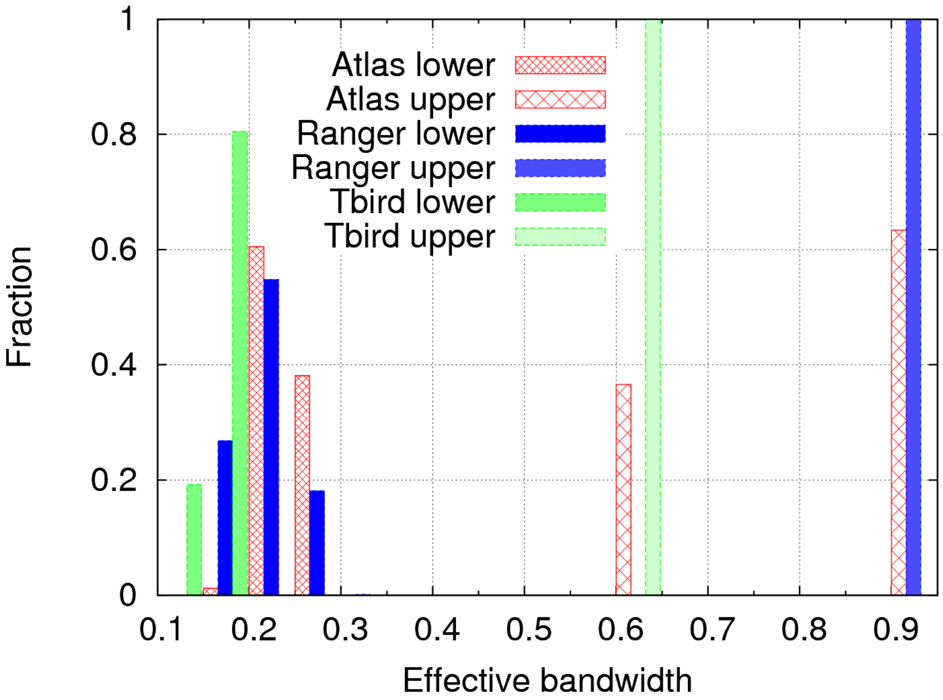
- Octopus (part of TDDFT package)
  - Thanks to Florian Lorenzen for the Input!
  - 10.5% communication overhead
  - MPI_Allreduce: 61.9%; MPI_Alltoallv: 21.9%

Conclusion: How do Applications Communicate?

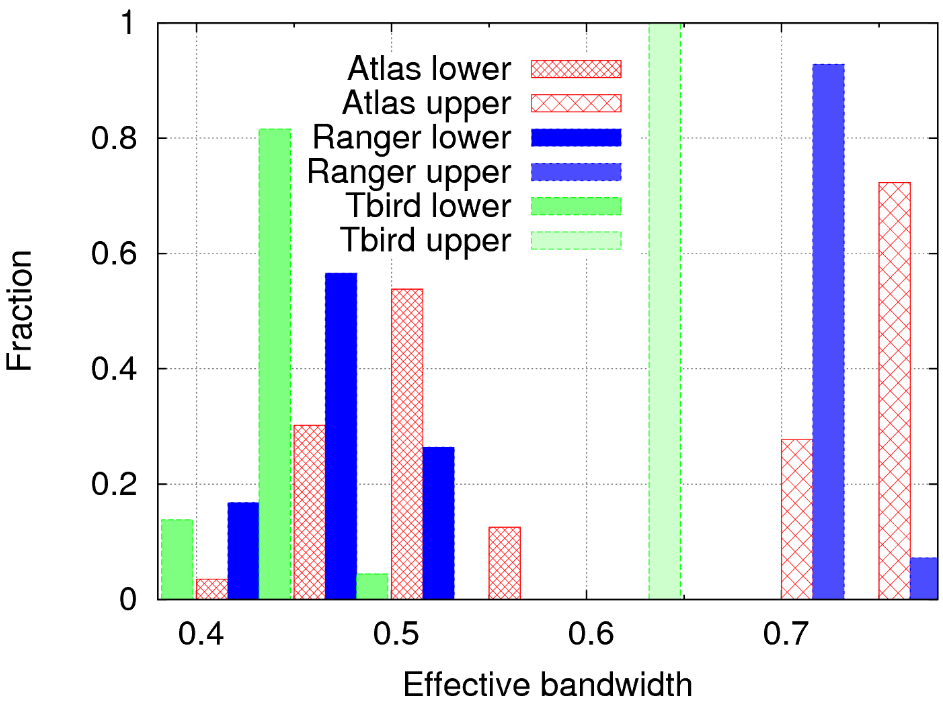
- Many applications use fixed communication patterns
- Collective communication is often used
- Nearest neighbor communication in all other cases

- how does that change the simulations?
- simulate different ”collective” patterns
  - tree
  - dissemination
  - nearest neighbor

Pattern Simulation Results

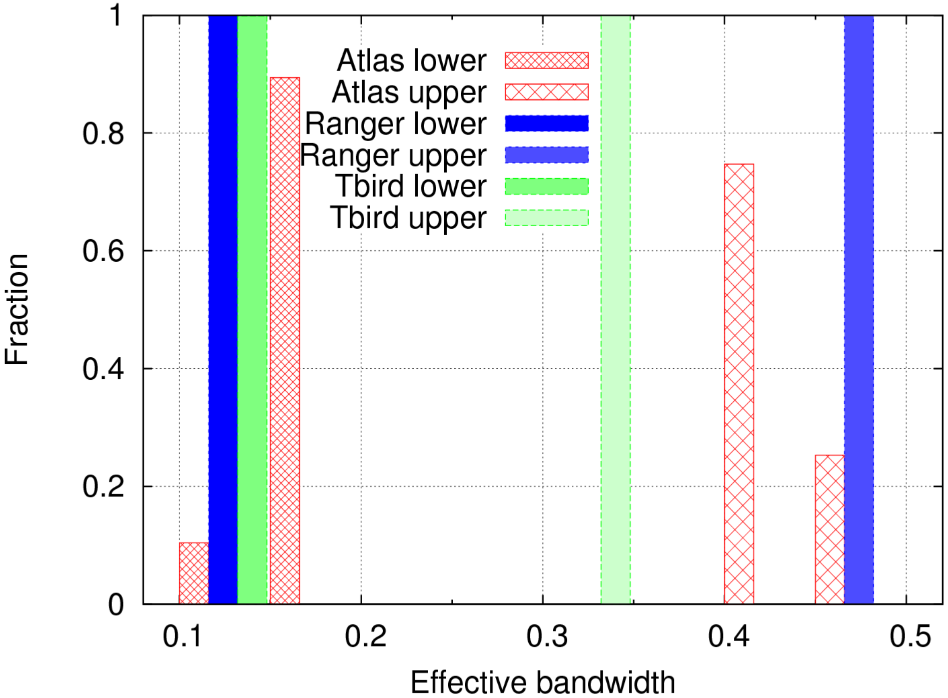


Tree simulation:
- Ranger: 69.9%
- Atlas: 71.3%
- Thunderbird: 57.4%



Six Neighbor (3d) simulation:
- Ranger: 62.4%
- Atlas: 60.7%
- Thunderbird: 37.4%

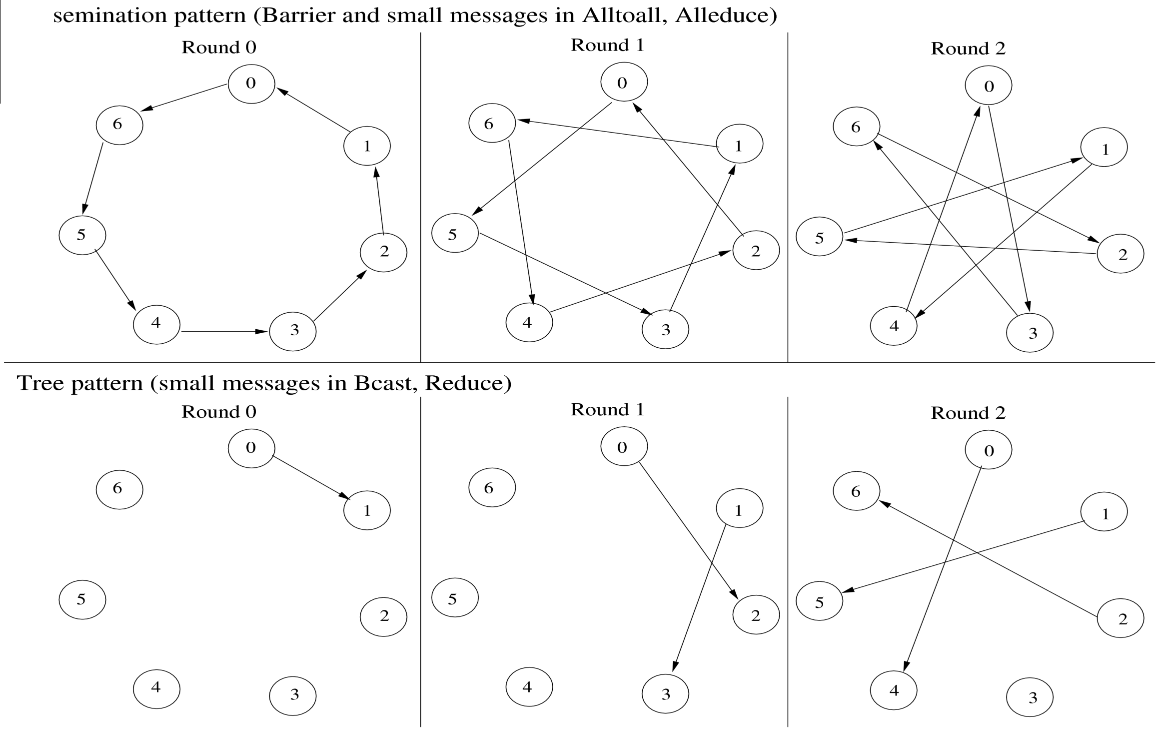
Pattern Simulation Results



Dissemination simulation:

- Ranger: 41.9%
- Atlas: 40.2%
- Thunderbird: 27.4%

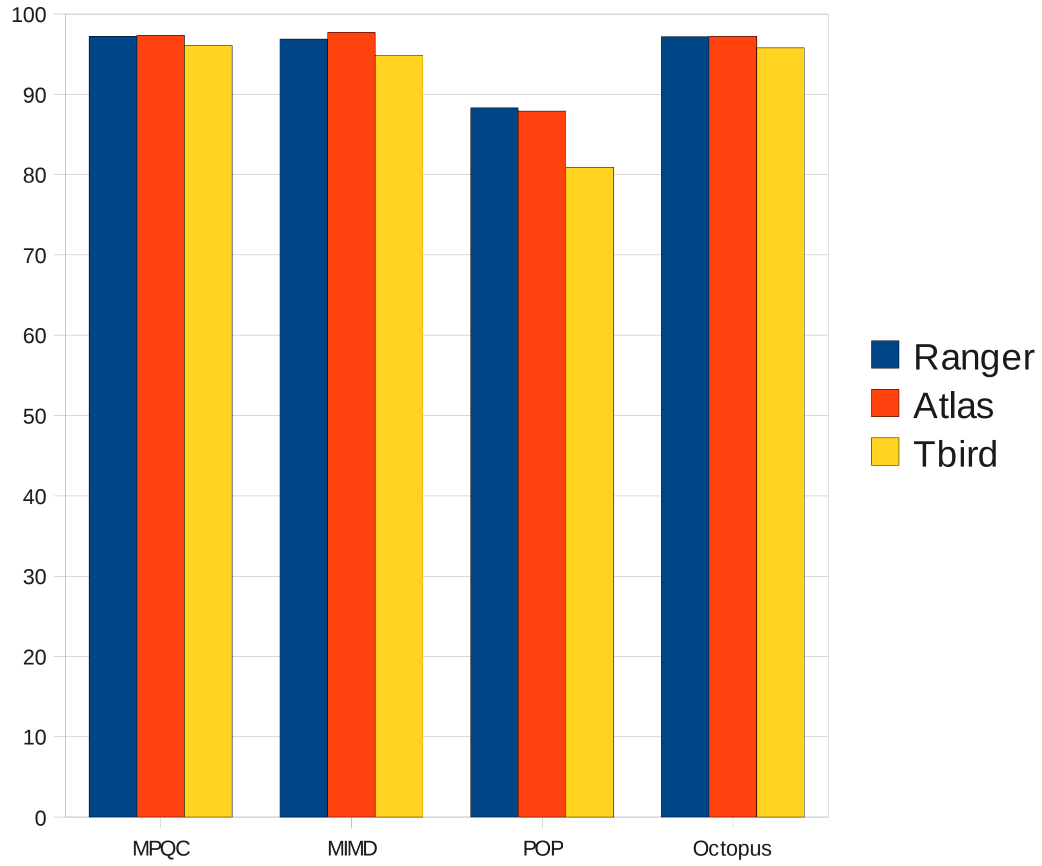
Comparison of Communication density (why is Dissemination so bad?)



What if there were no congestion?

this data is a guess! It provides only a rough estimation!

- percentage of application running-time if applications run at full-scale and the communication overhead remains constant (ideal weak scaling)



Those are all the Problems – Are there Solutions?

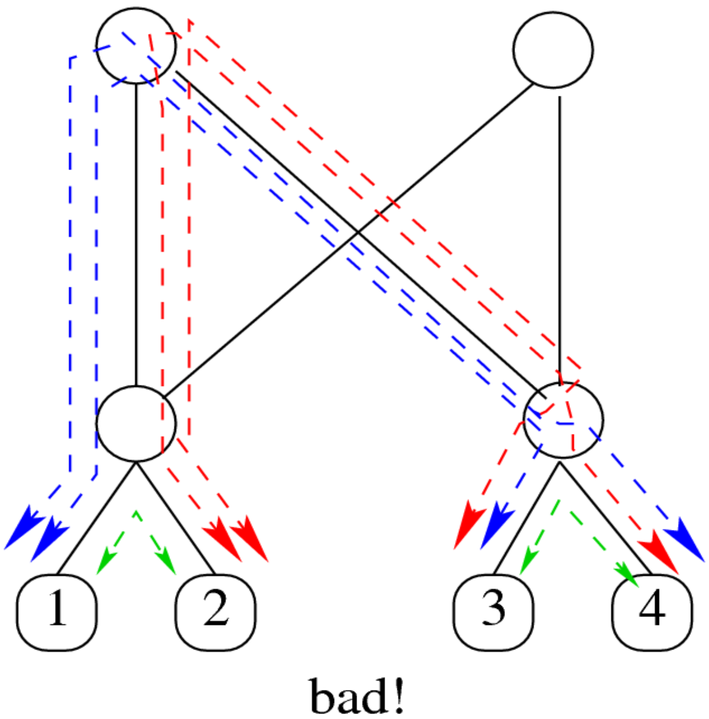
- yes, just too many (topologies, routing, technologies ...)
- we analyze Fat Trees and similar topologies
- simulate eBB for different network topologies and sizes
- guided by real-world system design (if recursive, then FBB in smallest parts)

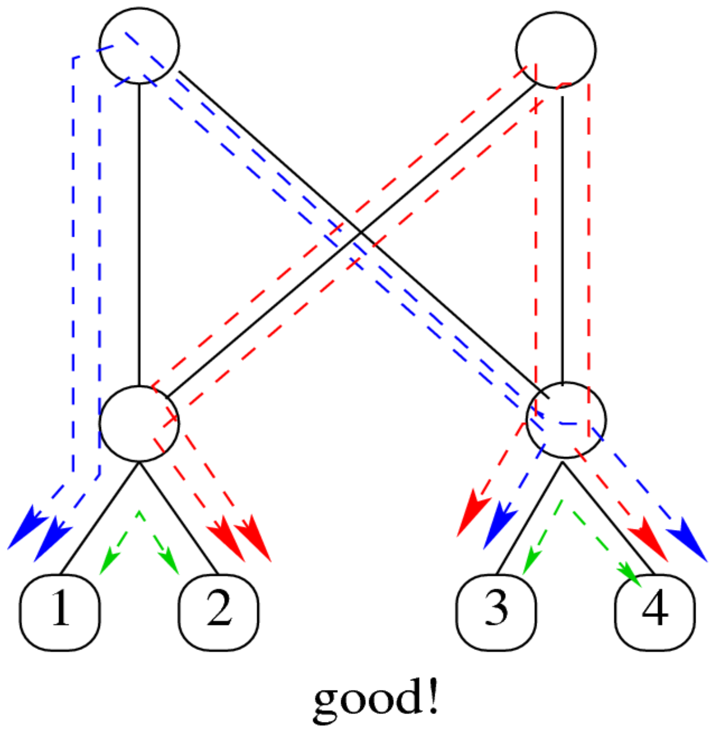
A new Problem – Routing Tables

- generating Fat Tree topologies is easy
- but we also need routing tables!
  - not trivial to generate
- OpenSM has several ways
  1) Min Hop (optimizes path length)
  2) Up*/Down* (constrained BFS)
  3) Fat-Tree (similar to Up*/Down*)
  4) LASH (uses SLs to distribute paths, [Skeie'92])
  5) load routes from a file
- step back to understand ...

What does a good Routing Table look like?

- differs from pattern to pattern, e.g., [Zahavi'07]
- we use bisect-pattern to stay general
  - can't say much for this generic pattern :-(
  - minimize the maximum number of paths through any given edge = increase ”balancedness”





Not trivial ... Let's step into (Graph) Theory

- physical network = graph with N terminals and |V| vertices
- routing R = set of N(N-1) paths between all terminals
- forwarding index of an edge/vertex = number of paths in R that lead through this edge/vertex
- forwarding index of a graph with routing R = maximum forwarding index in graph with R
  - → find a routing R that minimizes edge forwarding index
- NP-complete for vertex forwarding index [Saad'93]
- likely to be similarly hard for edge forwarding index
  - → find good heuristics/solutions
  - → analyze/evaluate real-world networks/routes

Evaluating a set of routes!

- simulator approach
- walk all N(N-1) paths and record edge-usage (takes a while)
- report maximum minimum and average forwarding index

Cluster	Nodes	E	σ	min	max	eBB
Odin	128	139	35	40	262	0.746
CHiC	566	646	152	58	1743	0.606
Atlas	1142	1807	670	1012	4211	0.556
Ranger	4081	7653	11140	184	90435	0.568
TBird	4391	10869	2878	7658	25169	0.406

- what does that mean?
  - hard to tell (need to solve the forwarding index problem)
  - but we can compare different routings R now!

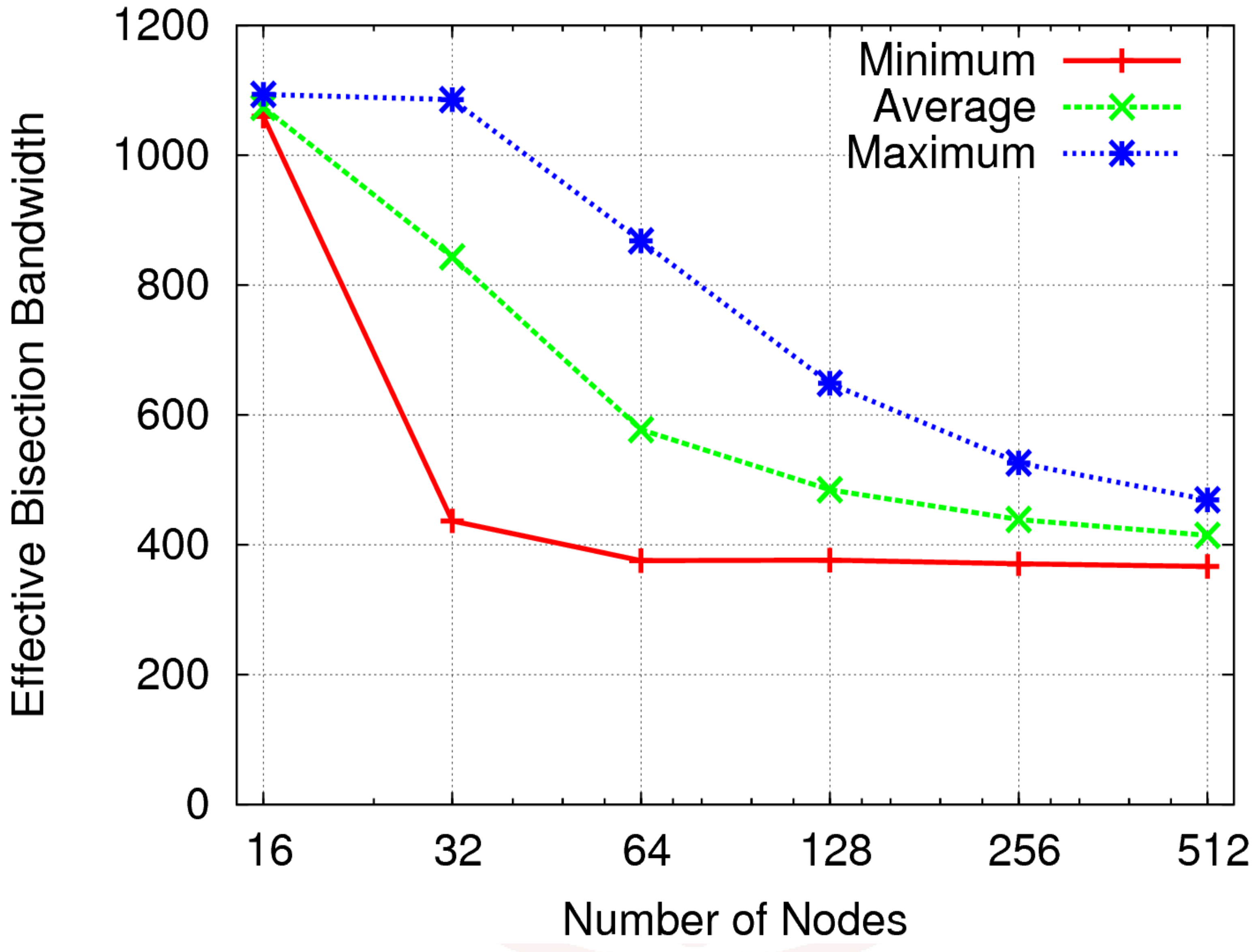
This is all ongoing Research

- more about forwarding indexes (bridge theory to practice)
- other communication patterns (e.g., tree, shift ...)
- more applications (analyze influence of jitter)
- different topologies (does it have to be Fat Tree?)
- evaluate adaptive routing strategies [Geoffray'08]
- ”fun” InfiniBand work (hope to do some Ethernet too)
- seeking for collaborations! (contact me!)

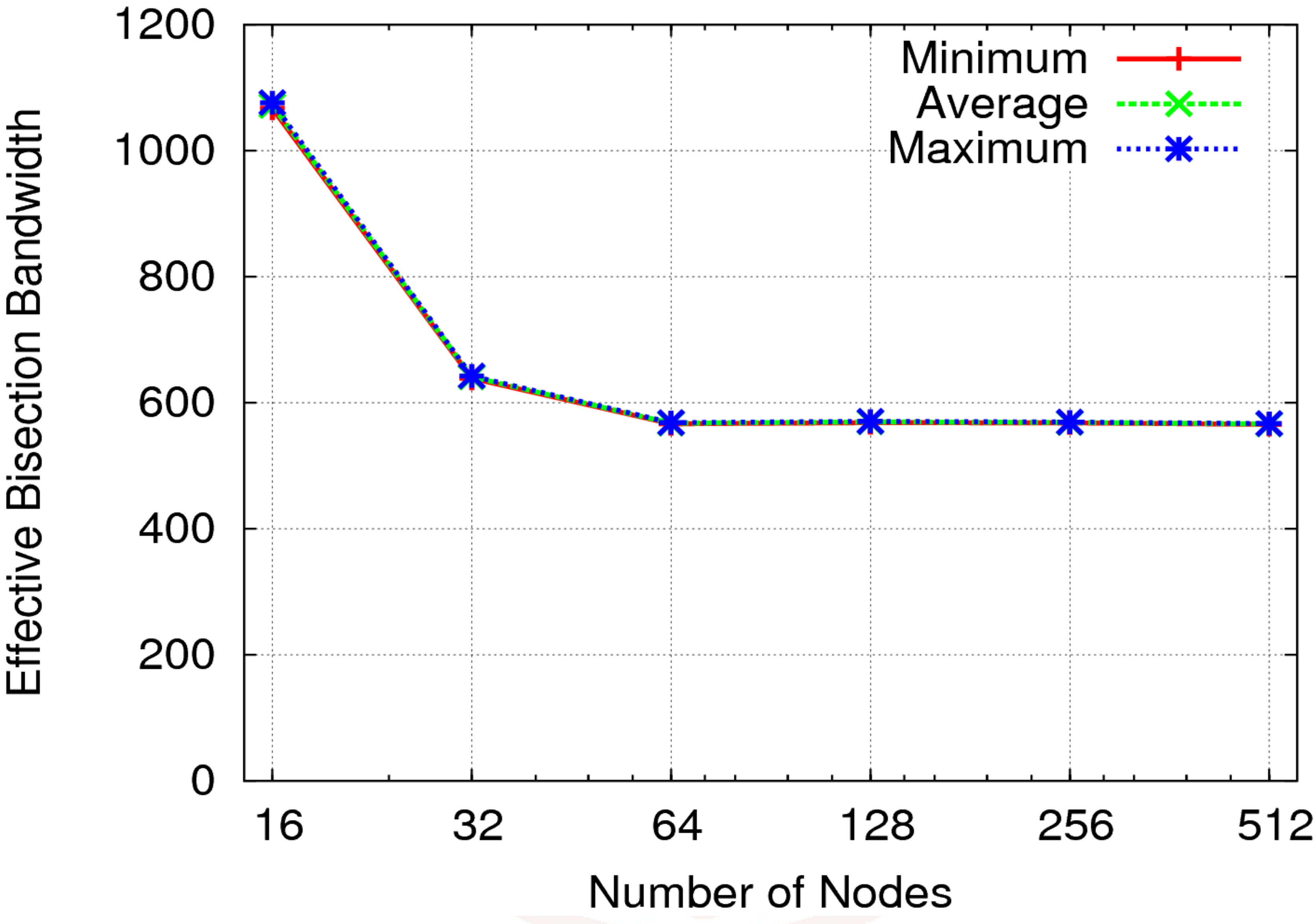
Special thanks to T. Schneider (TUC)

Backup Slides

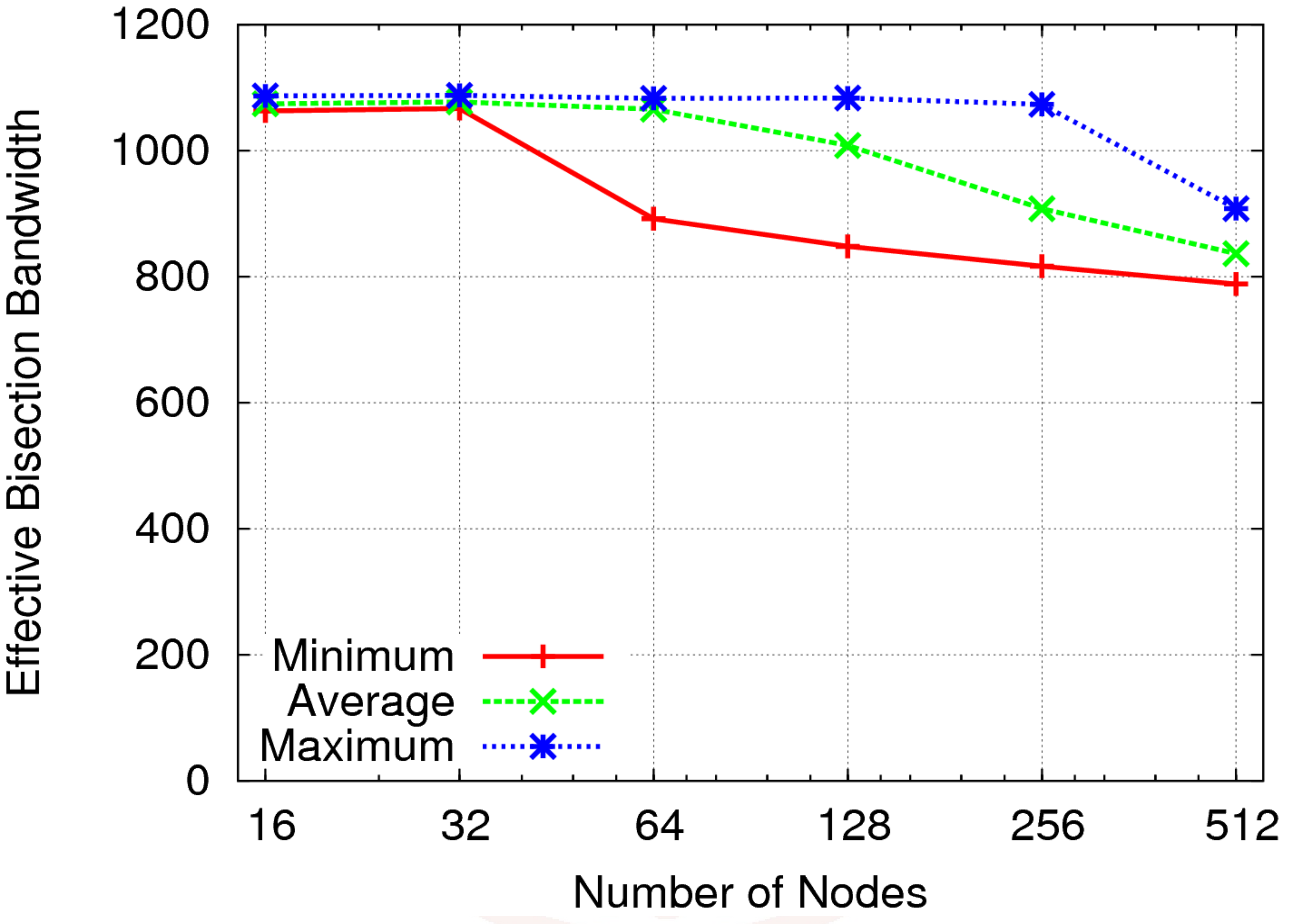
Static Routing in Myrinet



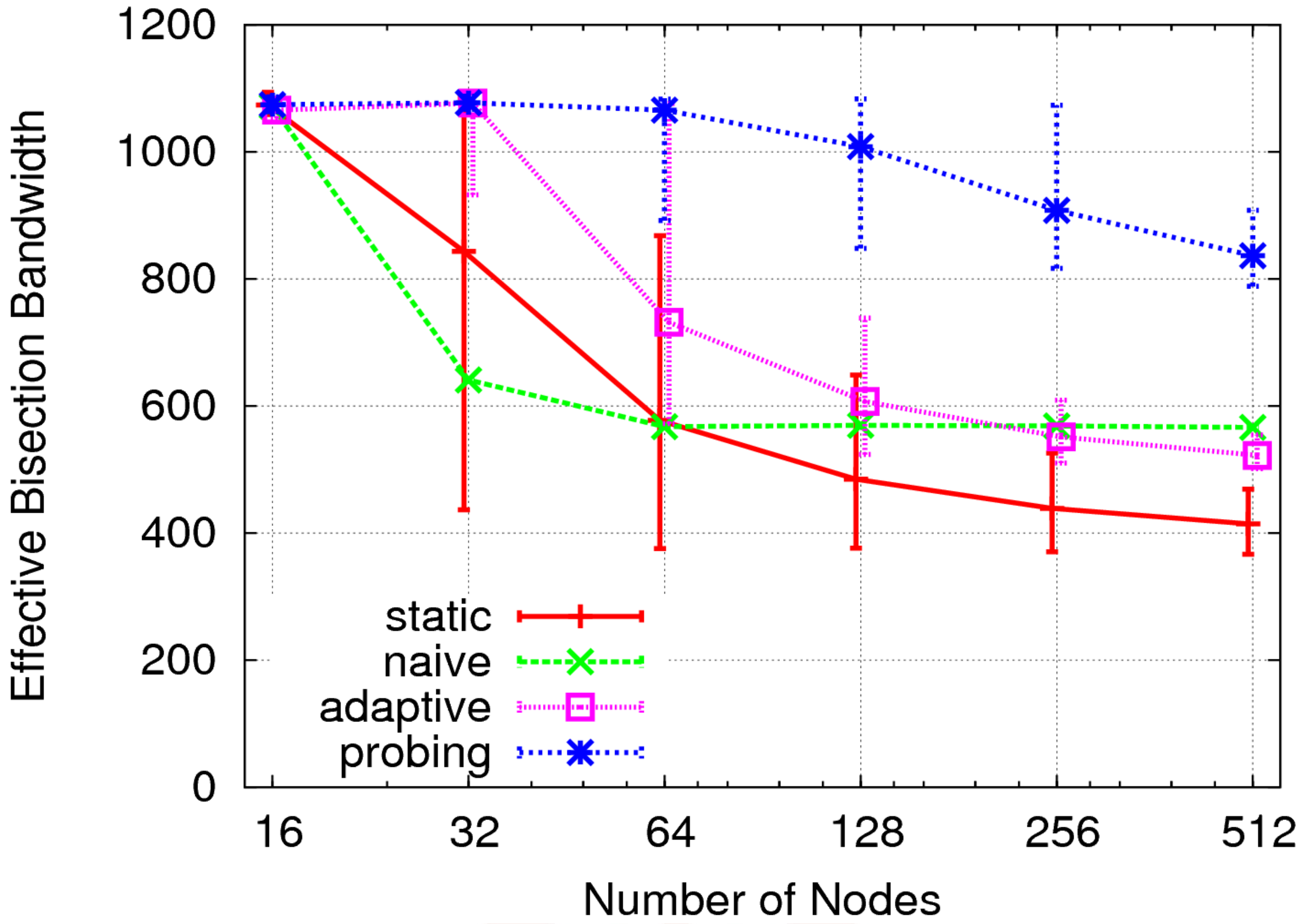
Random Routing in Myrinet



Probing Adaptive Routing in Myrinet



Routing Options in Myrinet



A graph-theoretical Routing Heuristic

- goal:
  - minimize forwarding index (of course)
  - minimize number of hops (latency)
- 1st Greedy Heuristic:
  - N(N-1) Dijkstra's with forwarding-indexes as weight
    - ➔ O(N^4) :-) ... too slow
- 2nd (weaker) Greedy Heuristic:
  - N Dijkstra's
    - ➔ O(N^3) ... works
  - forwarding indexes are ”better” than evaluated systems'
  - no benchmark data yet (are there volunteers? ;-)

Network Generation

- Now that we have routing – here the results: