# Improving Parallel Computing Platforms

## - programming, topology and routing -

Torsten Hoefler

Open Systems Lab

Indiana University

Bloomington, IN, USA

# State of the Art (what we all know)

- "Moores law" mandates parallelism
  - DMM (Clusters) and SMM (Manycores)
- DMM programming:
  - Message Passing (de facto standard MPI)
  - Task abstractions (CHARM++)
- SMM programming:
  - Task parallelism (Cilk, TBB, OpenMP 3.0)
  - Data (loop) parallelism (OpenMP, Cilk++)
- Languages:
  - UPC, Chapel, Fortress, X10

# Focus of this work

- Large-scale parallel computers
  - DMM Model
    - SMM obfuscates complexity (data distribution)!
    - Traditionally MPI
  - Large interconnection networks
    - Topology, Routing issues
  - Cluster computers
    - commodity components keep cost low
  - Scientific applications
    - Upcoming graph/Informatics applications

# Message Passing Interface

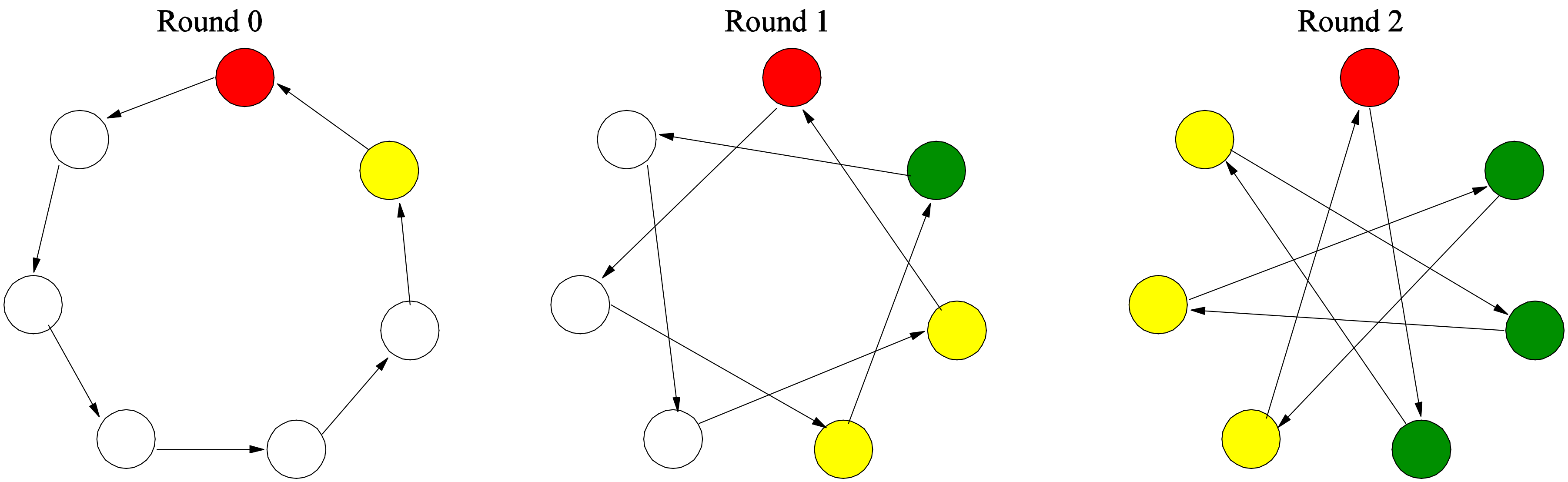
- 1998: MPI 2.0 - Well-known (no introduction needed)
- MPI Forum convenes since Jan 2008
  - Sep 2008: MPI 2.1 (merges and minimal changes)
  - Sep 2009: MPI 2.2 (bugfixes, API compatibility)
    - New scalable graph topology interface
    - Enhancements to collectives (MPI_IN_PLACE, Reduce_scatter_block)
    - Access restrictions to send buffers lifted
    - Better support for libraries (MPI_Reduce_local)
    - Deprecated C++ bindings (!)
- MPI 3 - Updates for the future
  - Better interoperability, updated collective operations, fault tolerance

# Focus on MPI Collective Operations

- High level of abstraction
  - Limited set of complex data movement operations
- Lifts MPI from "*the assembler of parallel programming*" to "*the C of parallel programming*"
- Enables standard-optimizations, such as tree structures for broadcast or reduce
- But **also** *network-specific* optimizations
  - Performance portability across specialized architectures is one of the *key points* of MPI
- Two examples: MPI_Barrier and MPI_Bcast on InfiniBand

# MPI_Barrier on InfiniBand

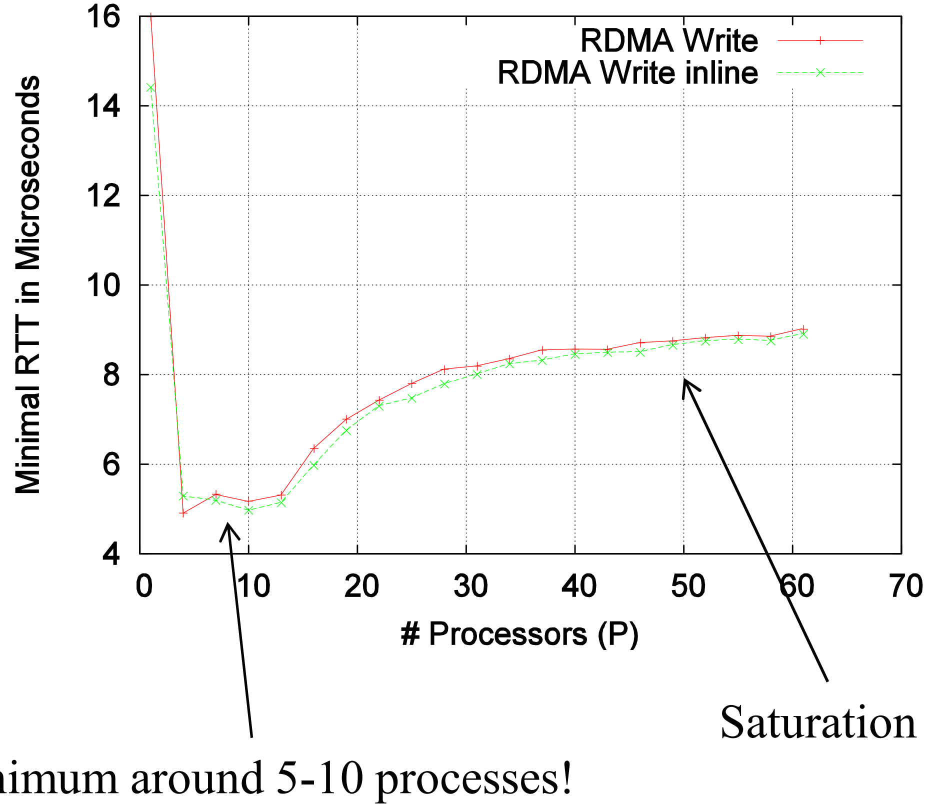
- Standard algorithm: Dissemination



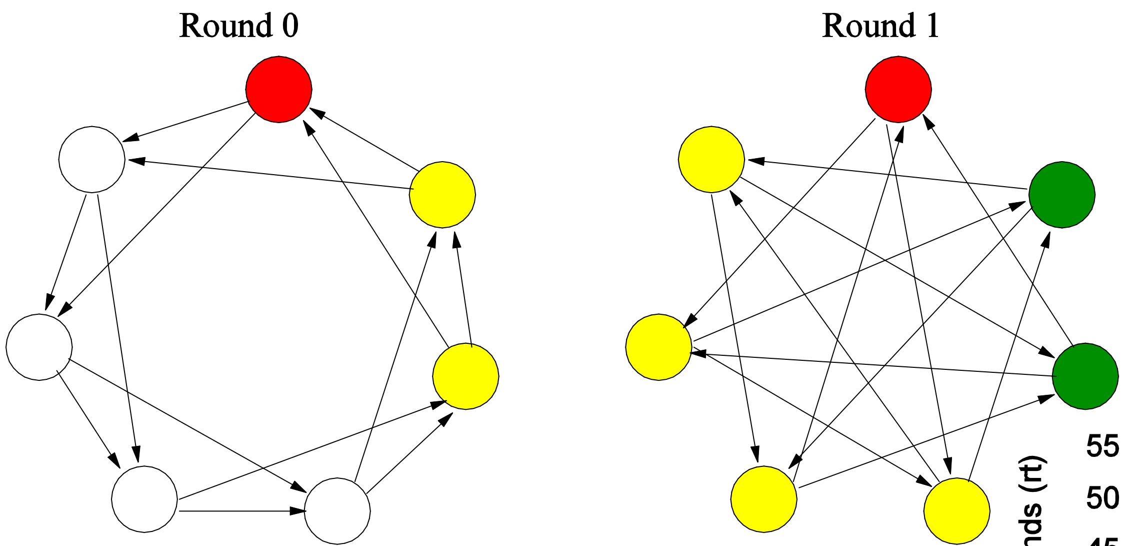

- Uses $log_2(P)$ rounds

# Is this optimal?

- No! Refine the model (1:N ping pong):



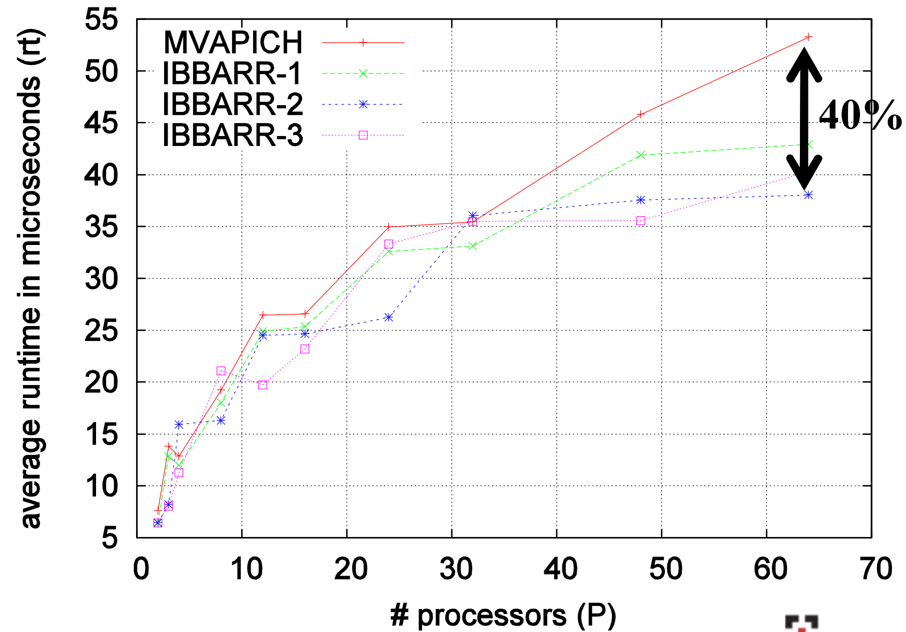
# N-way Dissemination




k peers per round!

$\Rightarrow \; log_k(P)$ rounds

Today's fastest MPI_Barrier on InfiniBand!

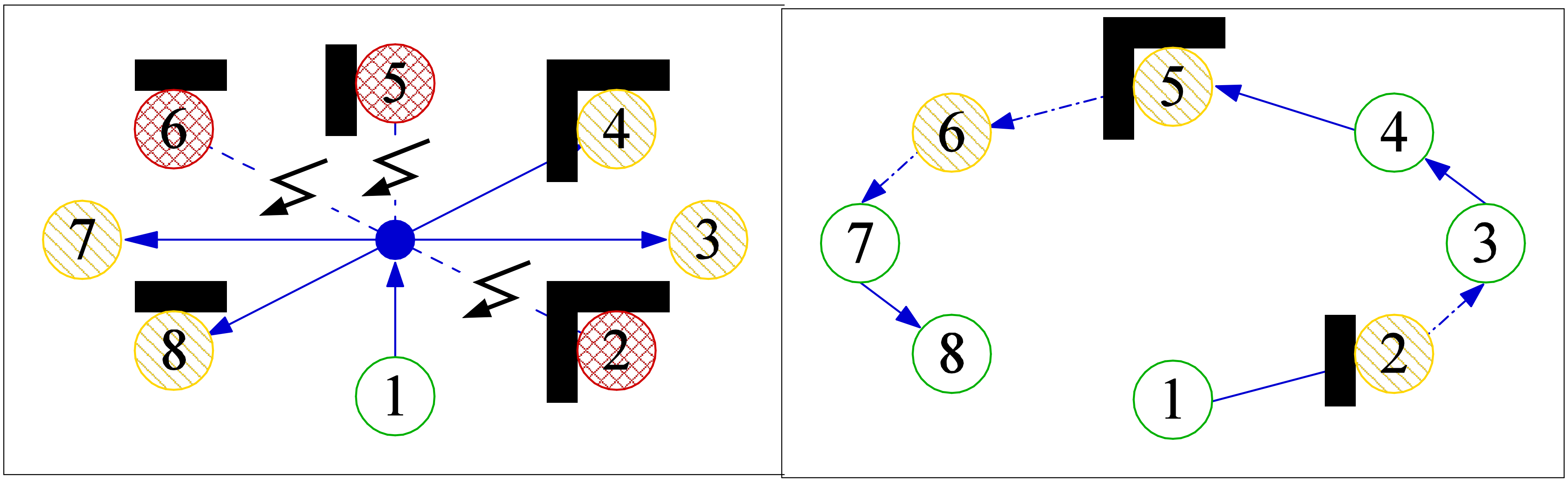



Refer to: Hoefler et al. "Fast Barrier Synchronization for InfiniBand" IPDPS 2006, CAC workshop

# 2nd Example: MPI_Bcast

- InfiniBand offers Multicast
  - It's unreliable, but runtime practically $\in \mathcal{O}(1)$

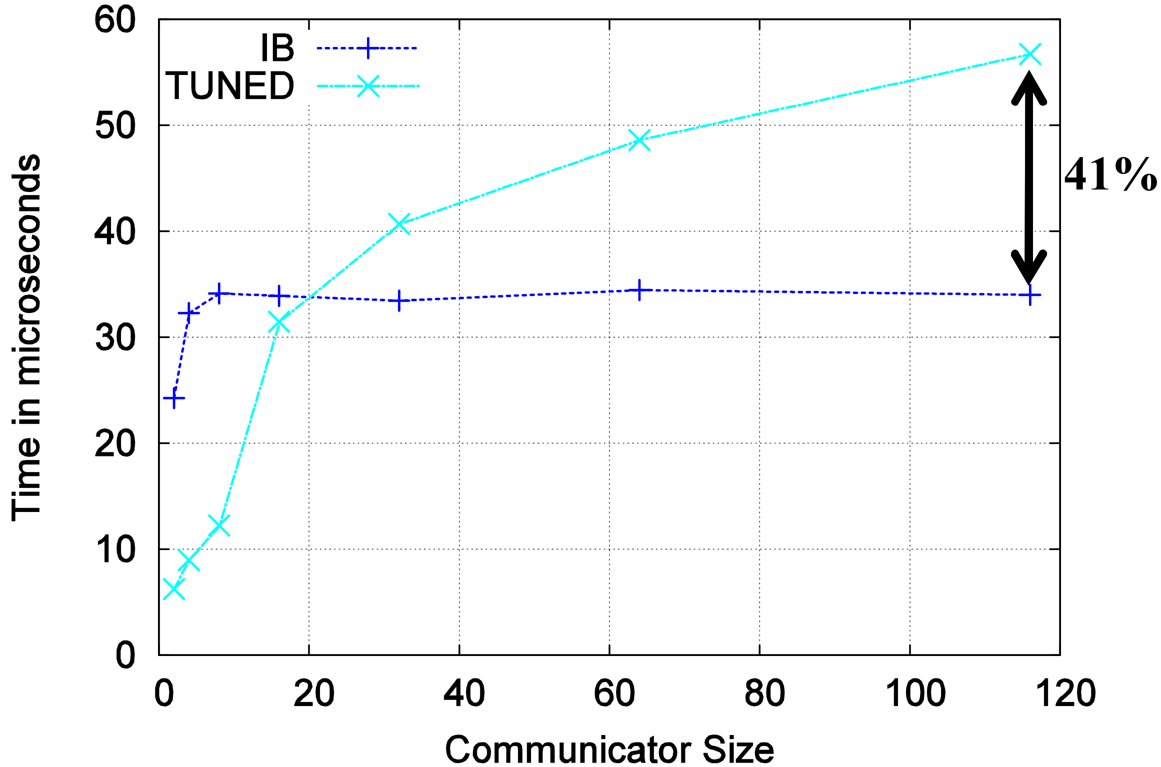
stage 1: multicast (unreliable) stage 2: chain broadcast (reliable)




  - Constant-time algorithm in low-loss network

# MPI_Bcast Results

- IMB 2-byte broadcast




Refer to: Hoefler, Siebert et al. “A practically constant-time MPI Broadcast Algorithm for large-scale InfiniBand Clusters with Multicast” IPDPS 2007, CAC workshop

# Intermediate Conclusions

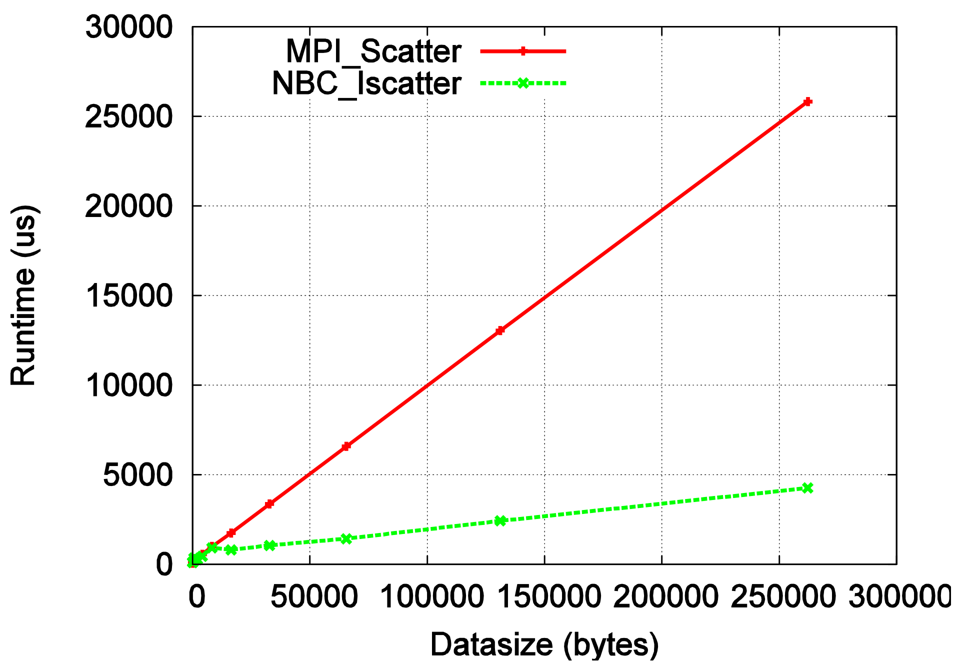
- High level of abstraction
  - Simplifies implementation
  - Offers optimization potential
  - Enables performance portability
- New directions
  - Nonblocking collective operations
  - Sparse collective operations

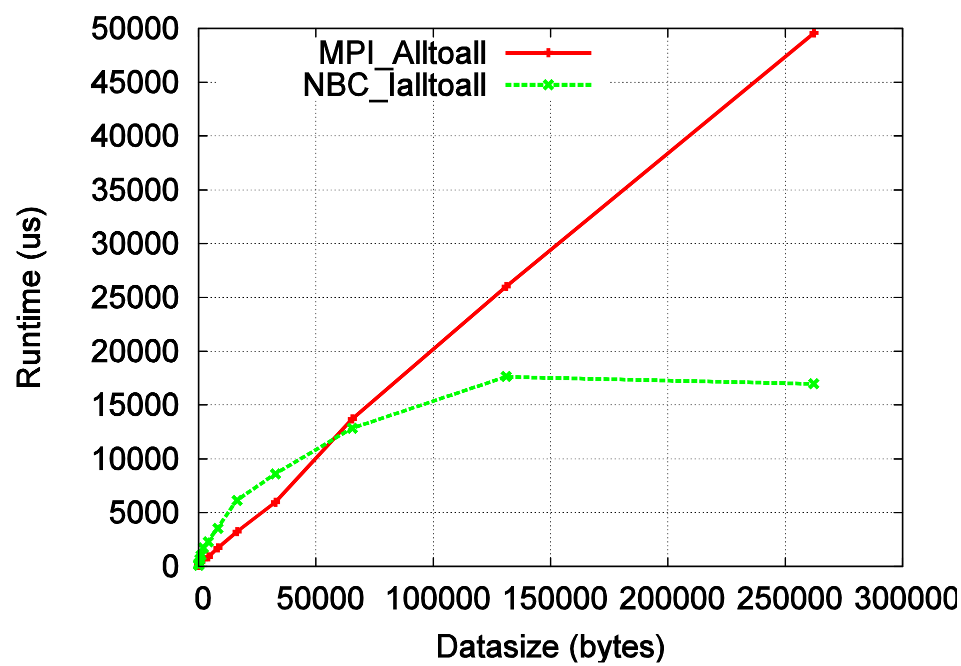
# Nonblocking Collective Operations

- Simple interface: MPI_Ibarrier()
  - Standard MPI semantics
- Enable new programming techniques
  - Decouple start from end (Hoefler et al. at SPAA'08)
  - Relax synchronization (Hoefler et al. at PPoPP'10)
- Enable communication/computation overlap
  - Hide latency (cf. Alexandrov's "early binding")
  - Should be a standard technique for point-to-point communications (is it yet?)

# Overlap potential

- Polling vs. threaded progression
- 64 InfiniBand nodes, MVAPICH vs. LibNBC
  - We assume ideal overlap (threaded has $2 \mu s$ constant overhead!)

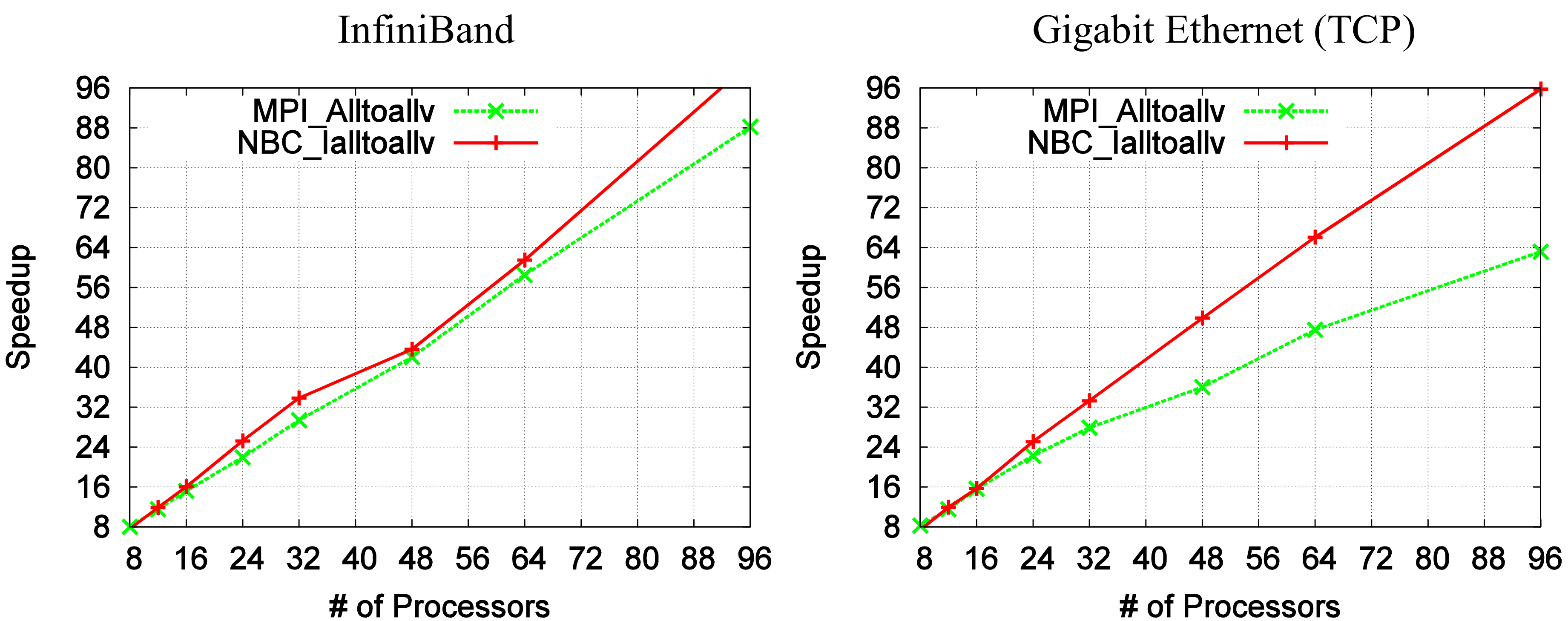



Refer to: Hoefler, et al. "Implementation and Performance Analysis of Non-Blocking Collective Operations for MPI" IEEE/ACM Supercomputing 2007 (SC07)

# Applications?

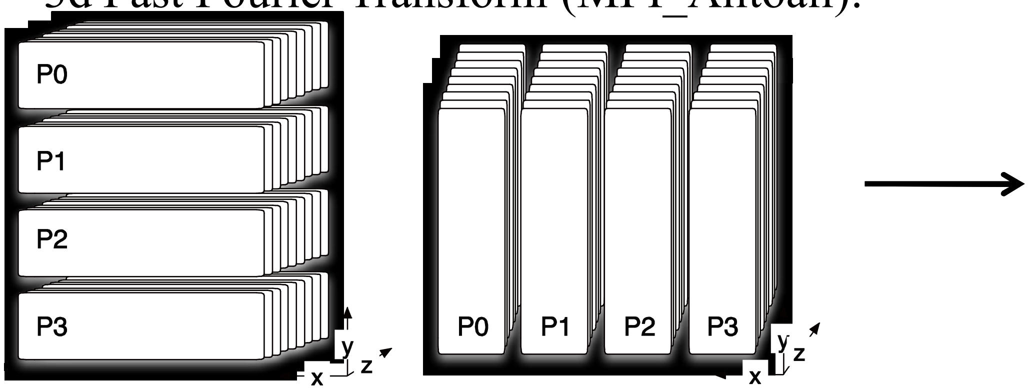
- Conjugate Gradient (3d Poisson, $800^3$ points)
  - Overlap boundary communication with local matrix product



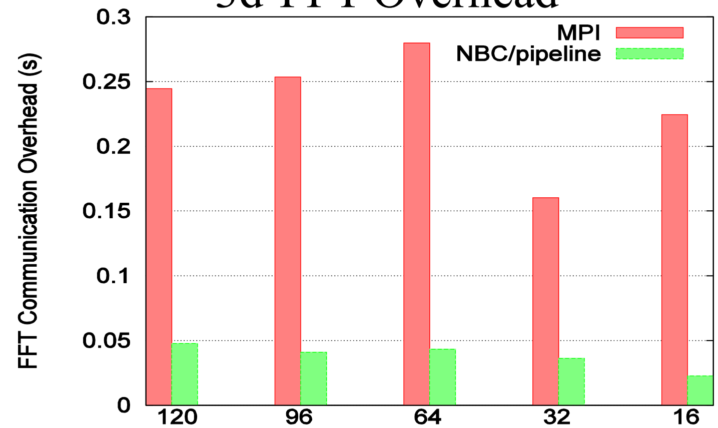

Refer to: Hoefler, Gottschling et al. "Optimizing a Conjugate Gradient Solver with Non-Blocking Collective Operations " Elsevier PARCO, Sept. 2007

# Some More Applications

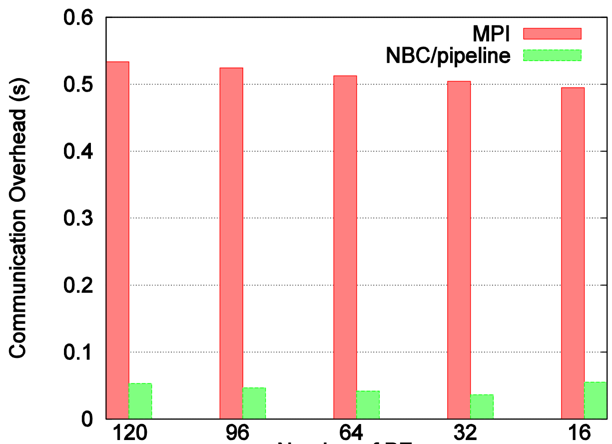
3d Fast Fourier Transform (MPI_Alltoall):



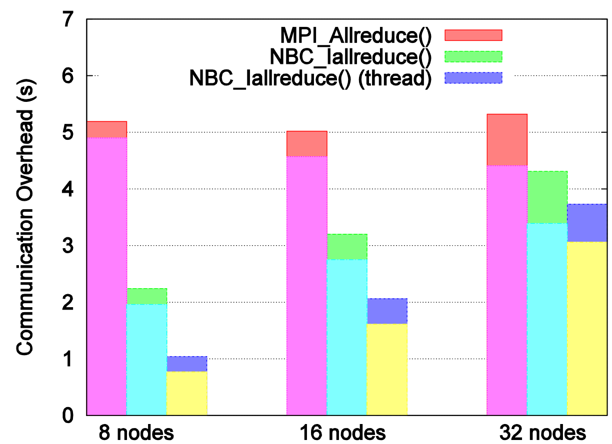

3d-FFT Overhead




Parallel bzip2 Overhead (MPI_Gather)



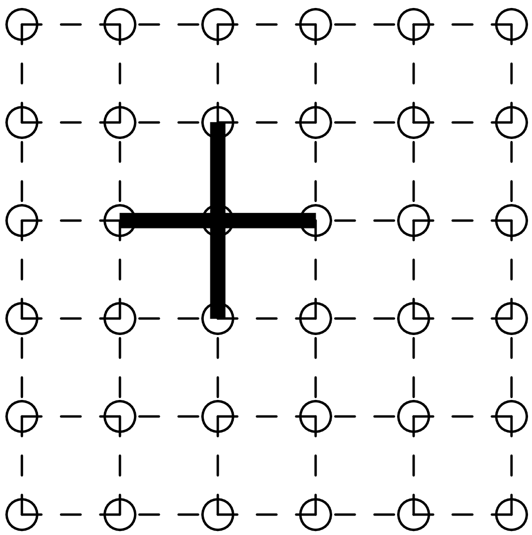

Medical Image Reconstruction Overhead

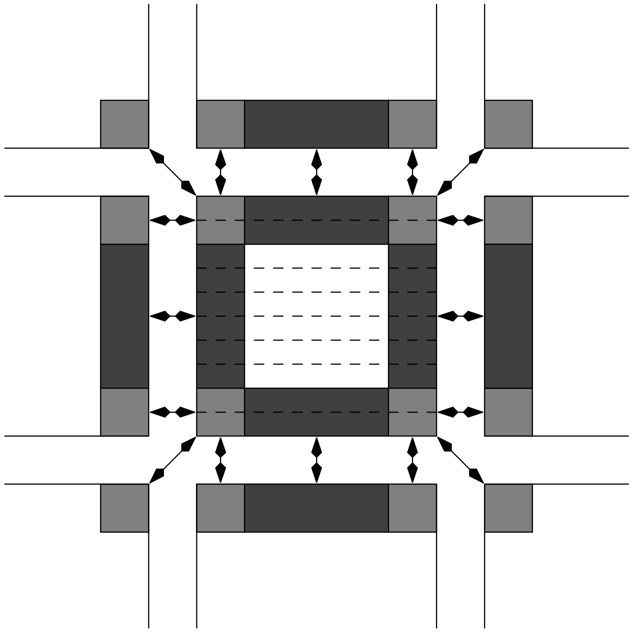



Refer to: Hoefler, Gottschling et al. “Leveraging Non-blocking Collective Communication in High-performance Applications ” ACM SPAA’08

# Sparse Collective Operations

- Now something completely different
  - More power to the users!
  - Specify arbitrary "flat" communication patterns



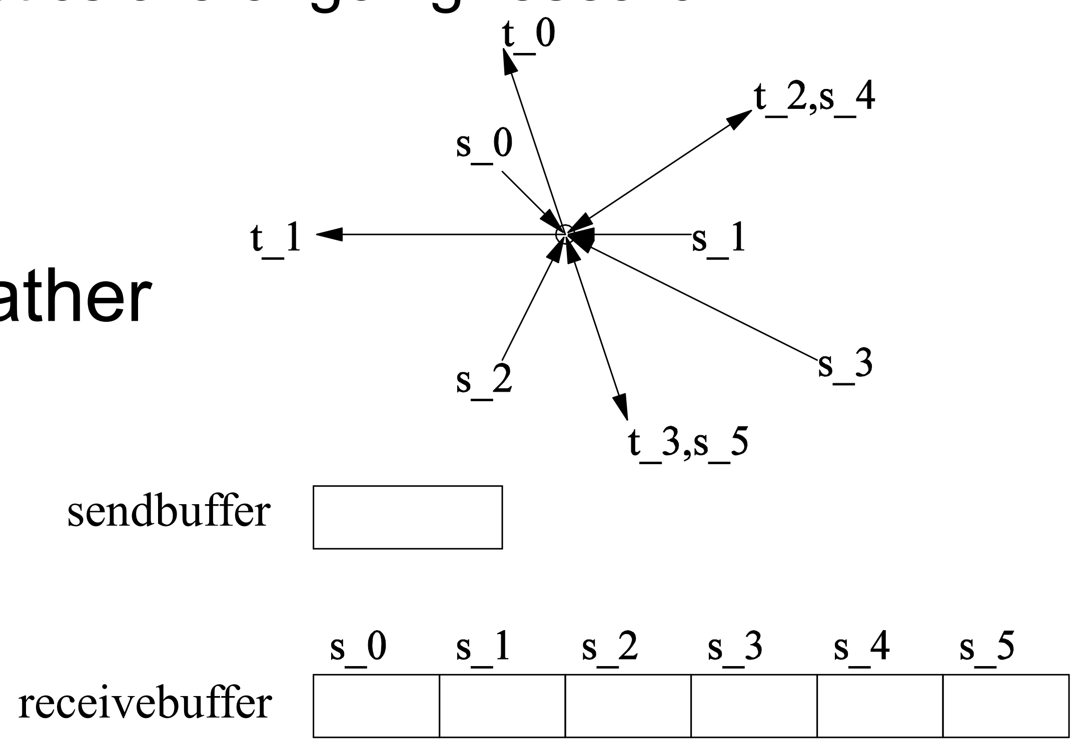


# MPI-2.2 New Topology Interface

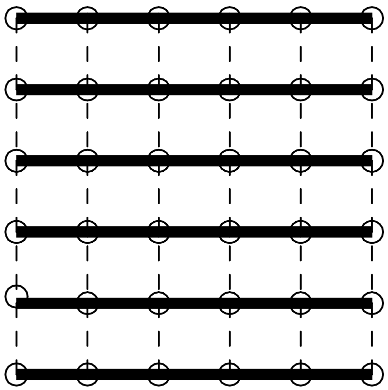
- MPI_GRAPH topology to specify communication
  - Usable (scalable) since MPI-2.2
  - Also added weights
- Enables intelligent process-to-node mapping
  - Of course NP-hard for general graphs
  - Discussed in literature (Träff, SC’02; Yu, SC’06)
- Scalable reference implementation available
  - Already implemented in MPICH-2!

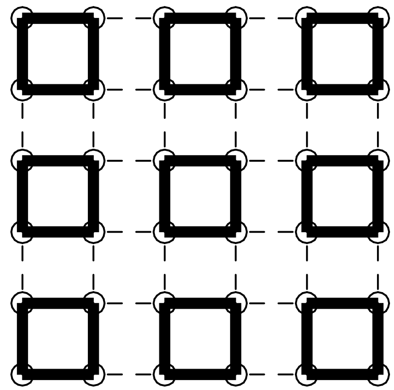
# MPI-3 (?) Sparse Collective Interface

- Enables optimized communication schedules
  - Message scheduling equivalent to graph coloring
  - Again NP-hard in the general case
  - Good heuristics are ongoing research
- Example:
  - Sparse Gather



# Scheduling Example

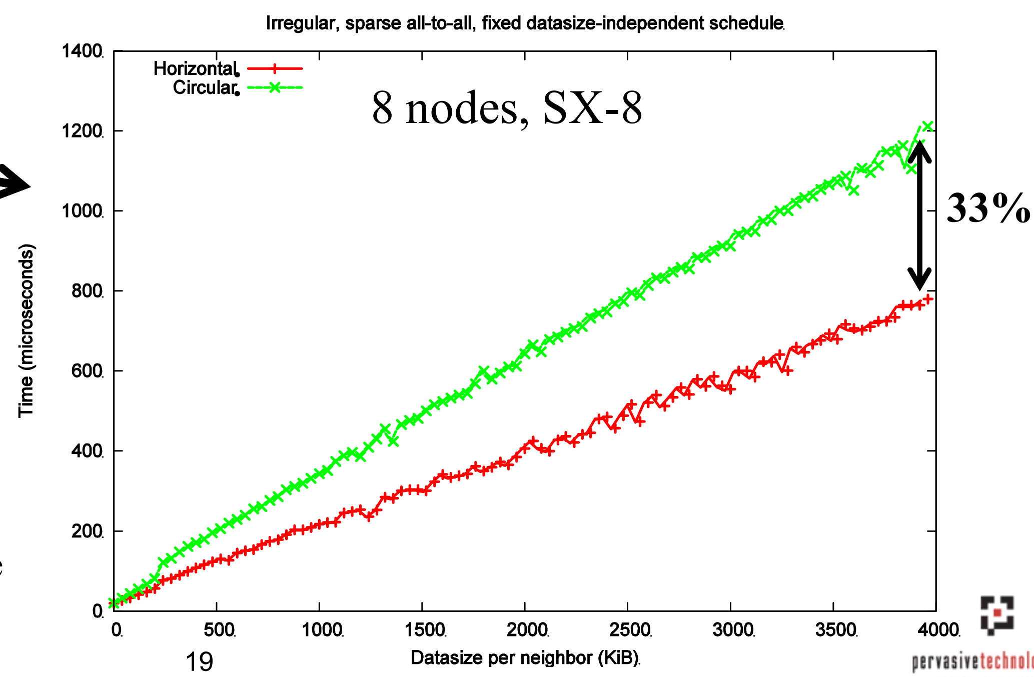




Two “heavy” rounds in each topology

Cost depends on scheduling.

Dimension-order scheduling

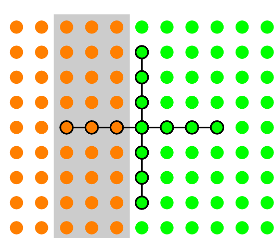



Refer to: Hoefler, Träff et al. “Sparse Collective Operations for MPI” IPDPS/HIPS 2009

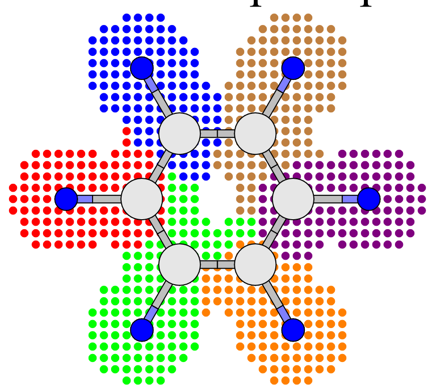
# Applications?

- Sparse collectives are implemented in LibNBC
  - Trivial scheduling / usability study
- TDDFT/Octopus – trivial change (simpler than before)

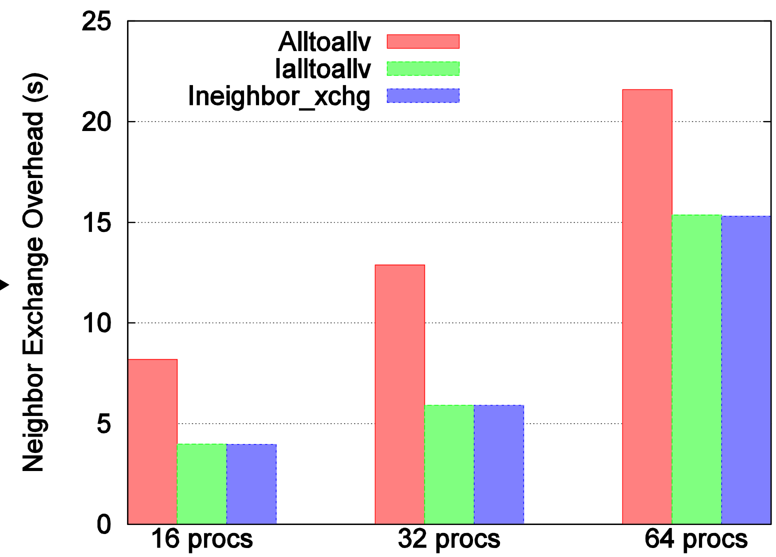
Standard stencil:



Benzene example topology:



6 MPI processes
(METIS decomposition)




Refer to: Hoefler, Lorenzen et al. "Sparse Non-Blocking Collectives in Quantum Mechanical Calculations" EuroPVM/MPI 2008

# Intermediate Conclusions

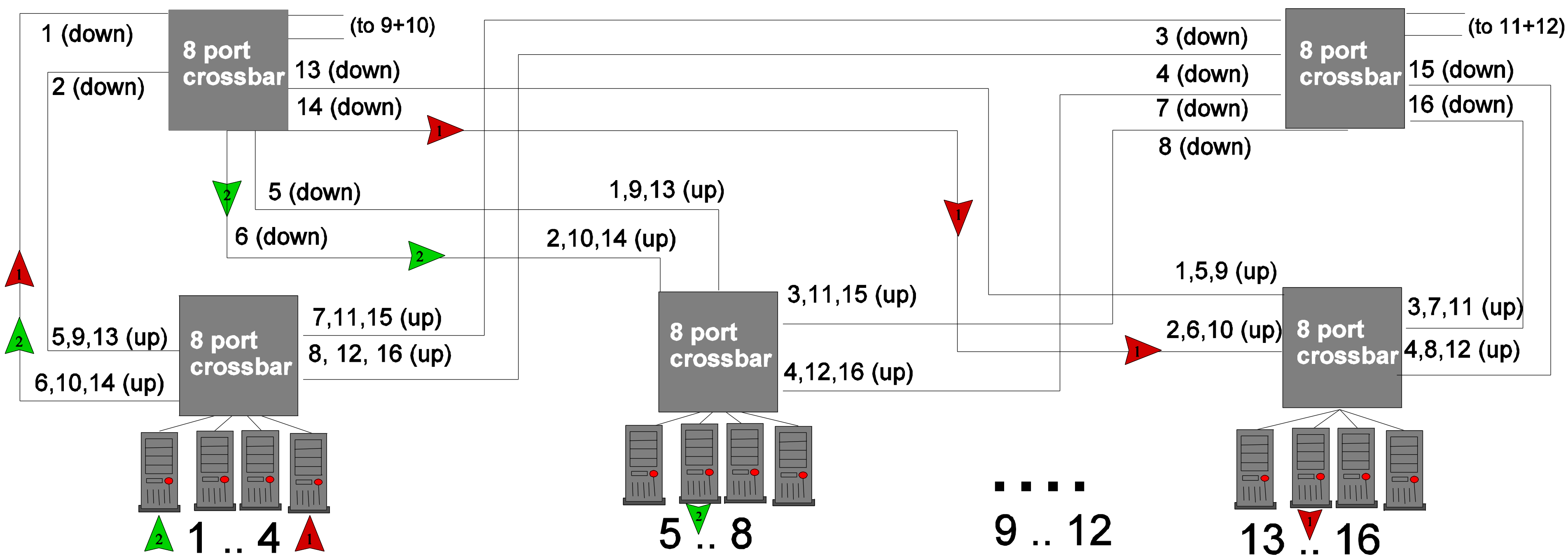
- Collective operations are a good abstraction!
  - Easy to use
  - High-level problem specification
  - Sparse collectives are even more powerful
- Overlapping computation and communication can be beneficial
  - Relatively hard to get right
  - Depends on support in communication middleware
  - Depends on the application or algorithm
- Process mapping seems important
  - Is it?

# Optimizing Collectives and Mappings

- Network is the most vital part
  - Mandates collective algorithms and topology mappings
- Network is defined by:
  - Topology (Torus, Hypercube, Fat-Tree, …)
  - Endpoint technology (Myrinet, InfiniBand, Portals, …)
- LogGP models most networks well
  - Ignores congestion in the network
  - Assumes full bisection bandwidth (FBB) (?)
- Do FBB networks solve all problems?
  - No! (why?)

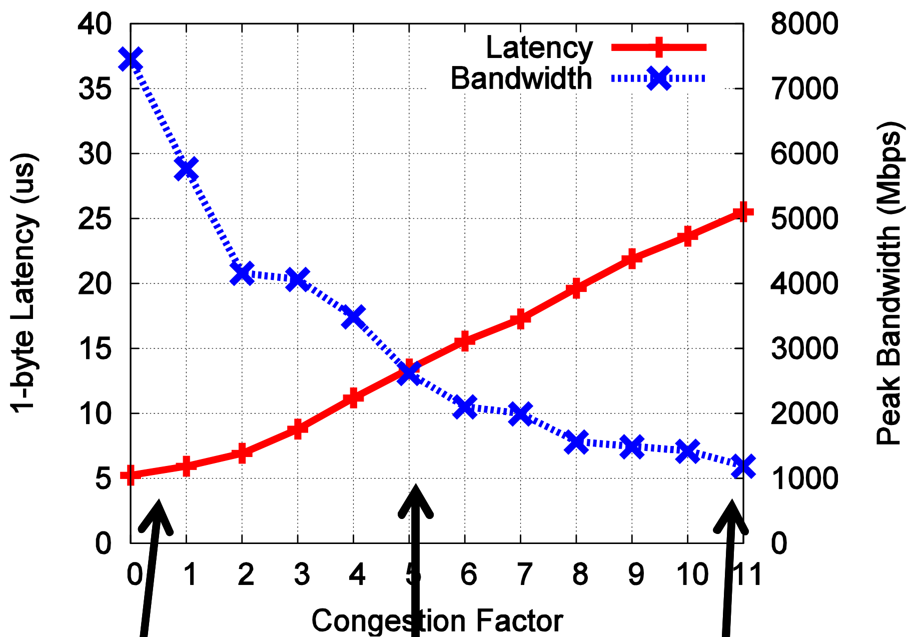
# Example: InfiniBand

- 30.2 % of Top500 (Jun 2009)
- Static routing (1→5, 4 → 14):




➢ No full bandwidth (cf. Valiant's bound)

# Quantifying Congestion

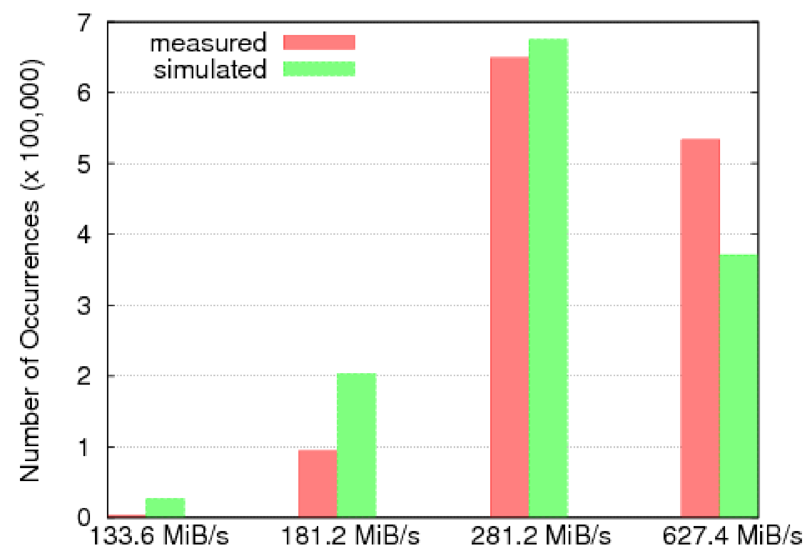



Microbenchmarks
(NetPIPE, IMB ping pong
Netgauge one_one)

Reality?

Lower Bound!

CHiC Supercomputer:
- 566 nodes, full bisection IB fat-tree
- effective Bisection Bandwidth: 0.699

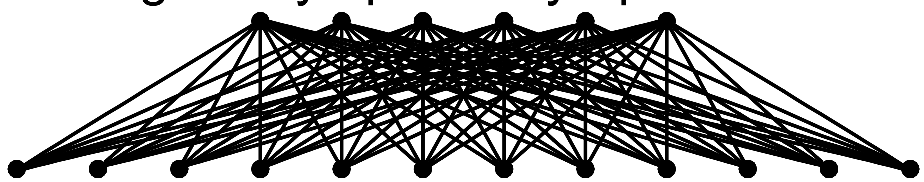



3 2 1 0
Congestion Factor

Refer to: Hoefler, Schneider et al. "Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks " IEEE Cluster 2008

# Full Bisection Bandwidth != Full Bandwidth

- expensive topologies do not guarantee high bandwidth
- deterministic oblivious routing cannot reach full bandwidth!
  - see Valiant's lower bound
  - random routing is asymptotically optimal but looses locality



- but deterministic routing has many advantages
  - completely distributed
  - very simple implementation
- InfiniBand routing:
  - deterministic oblivious, destination-based
  - linear forwarding table (LFT) at each switch
  - lid mask control (LMC) enables multiple addresses per port

# InfiniBand Routing Continued

- offline route computation (OpenSM)
- different routing algorithms:
  - MINHOP (finds minimal paths, balances number of routes local at each switch)
  - UPDN (uses Up*/Down* turn-control, limits choice but routes contain no credit loops)
  - FTREE (fat-tree optimized routing, no credit loops)
  - DOR (dimension order routing for k-ary n-cubes, might generate credit loops)
  - LASH (uses DOR and breaks credit-loops with virtual lanes)

# Some Theoretical Background

- model network as $G=(V_P \cup V_C, E)$
- path r(u,v) is a path between $u,v \in V_P$
- routing $R$ consists of $P(P\text{-}1)$ paths
- edge load $l(e)$ = number of paths on $e \in E$
- edge forwarding index $\pi(G,R)=max_{e \in E}\ l(e)$
  - $\pi(G,R)$ is a trivial upper bound to congestion!

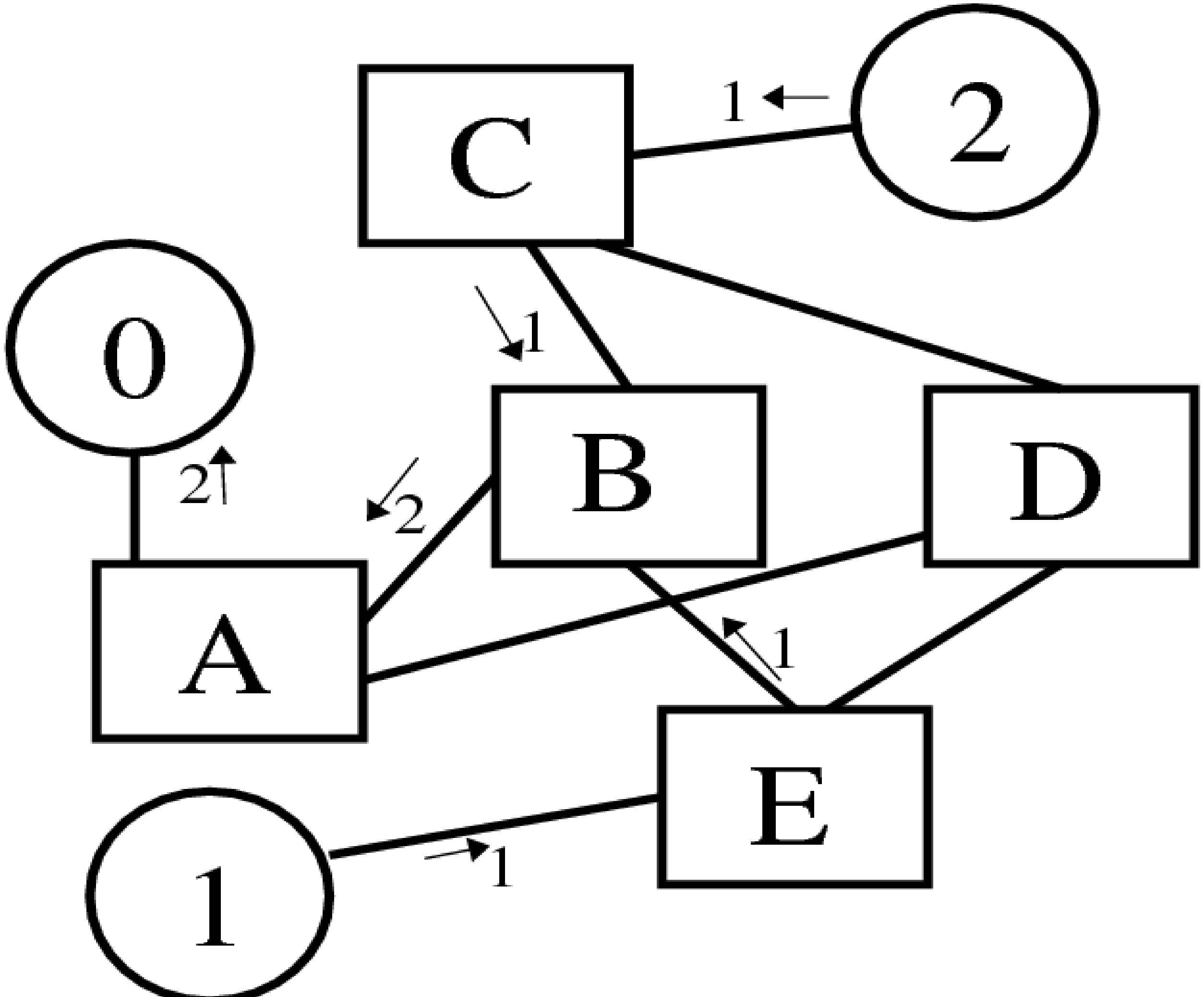
➢ goal is to find $R$ that minimizes $\pi(G,R)$
  - shown to be NP-hard in the general case

# Routing based on SSSP

- we propose P-SSSP routing
- P-SSSP starts a SSSP run at each node
  - finds paths with minimal edge-load $l(e)$
  - updates routing tables in reverse
    - essentially SDSP
  - updates $l(e)$ between runs
- let's discuss an example …

# P-SSSP Routing (1/3)

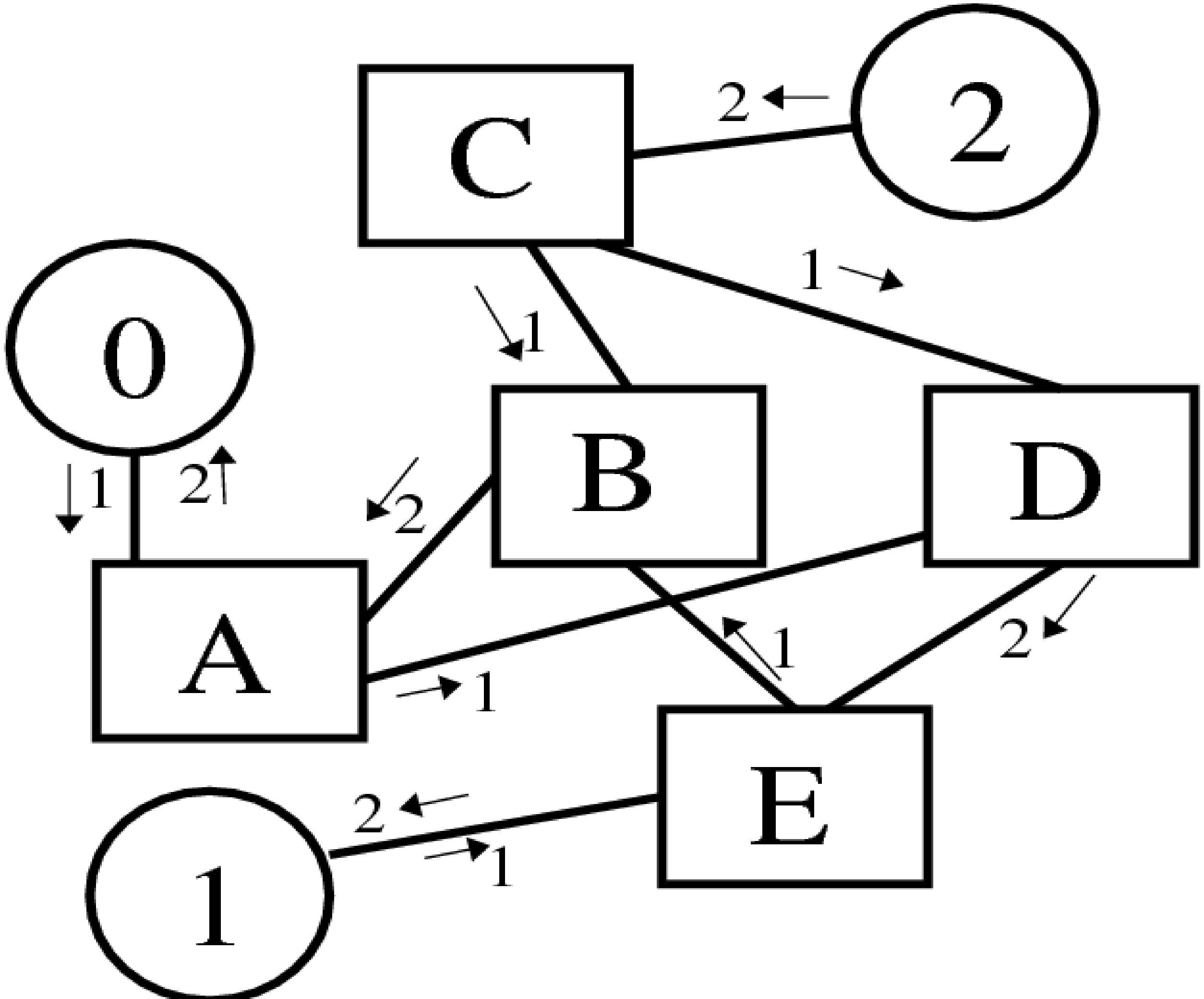
Step 1:
Source-node 0:



# P-SSSP Routing (2/3)

Step 2:

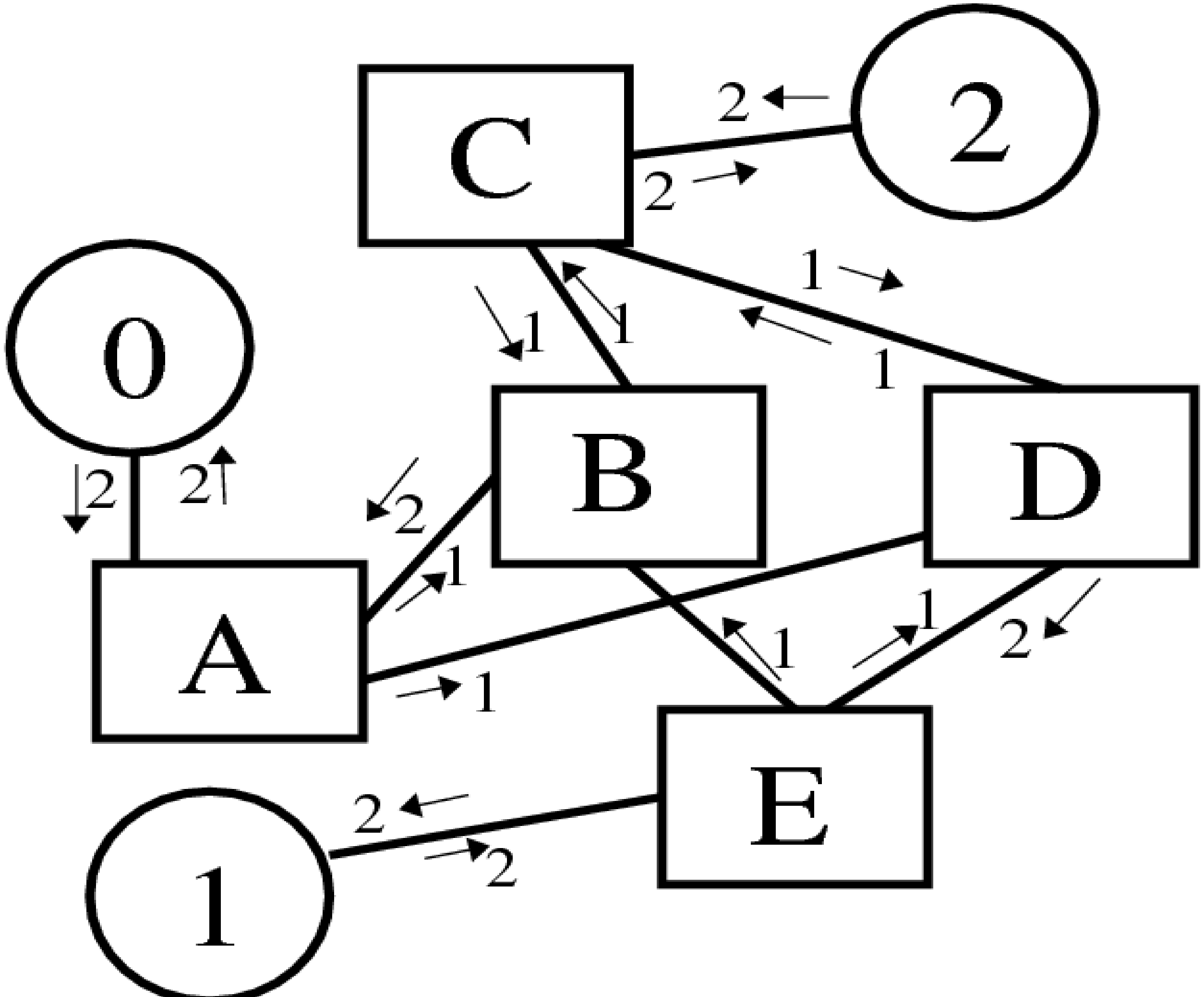
Source-node 1:



# P-SSSP Routing (3/3)

Step 3:
Source-node 2:

$\pi(G,R)=2$



# How to Assess a Routing?

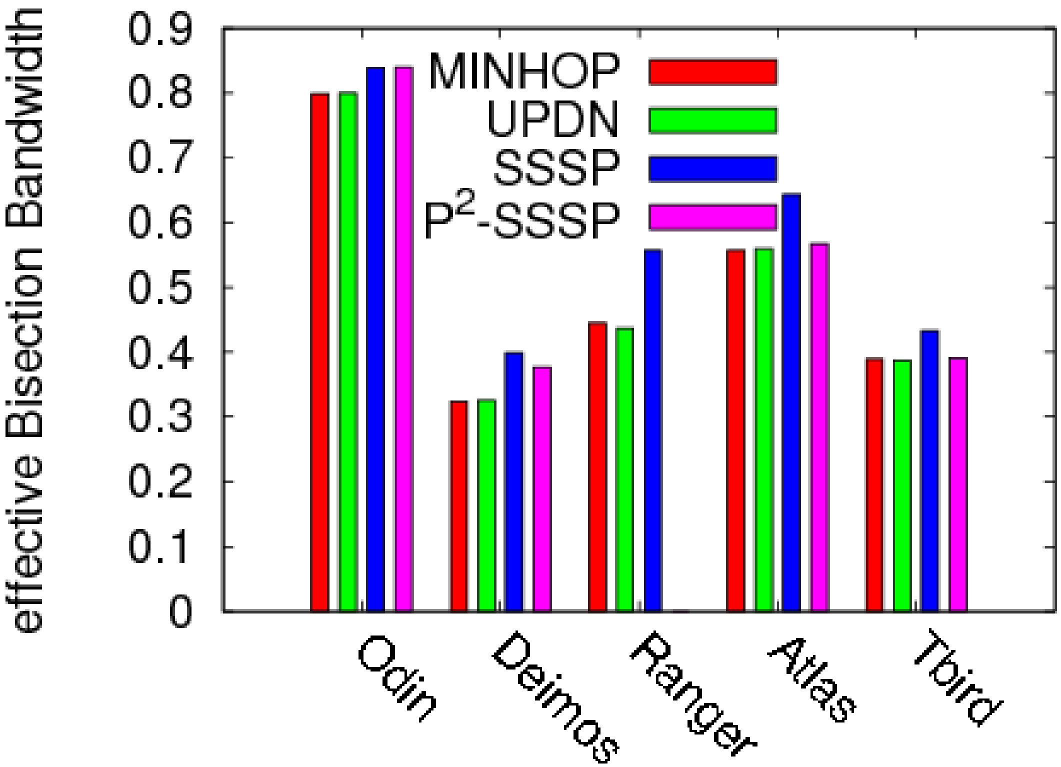
- edge forwarding index is a trivial upper bound
- ability to route permutations is more important
  - bisect P into two equally-sized partitions
  - choose exactly one random partner for each node
  - $\Theta(P!/(P/2)!)$ combinations!

- our simulation approach:
  - pick N (=5000) random bisections/matchings
  - compute average bandwidth
  - shown to be rather precise (Cluster'08)

# Comparison to Real Systems

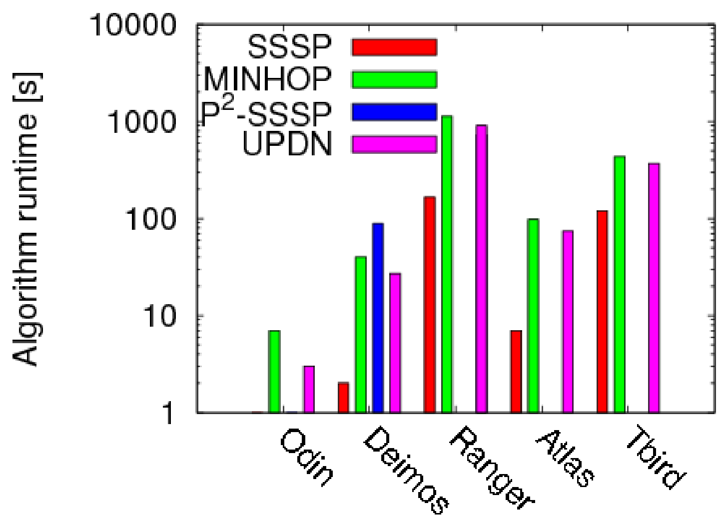
- `ibdiagnet`, `ibnetdiscover`, and `ibsim`
- we extracted topology and routing from:
  - Thunderbird (SNL) – 4390 LIDs
    - thanks to: Adam Moody & Ira Weiny
  - Ranger (TACC) – 4080 LIDs
    - thanks to: Christopher Maestas
  - Atlas (LLNL) – 1142 LIDs
    - thanks to: Len Wisniewsky
  - Deimos (TUD) – 724 LIDs
    - thanks to: Guido Juckeland and Michael Kluge
  - Odin (IU) – 128 LIDs

# Real-world Results



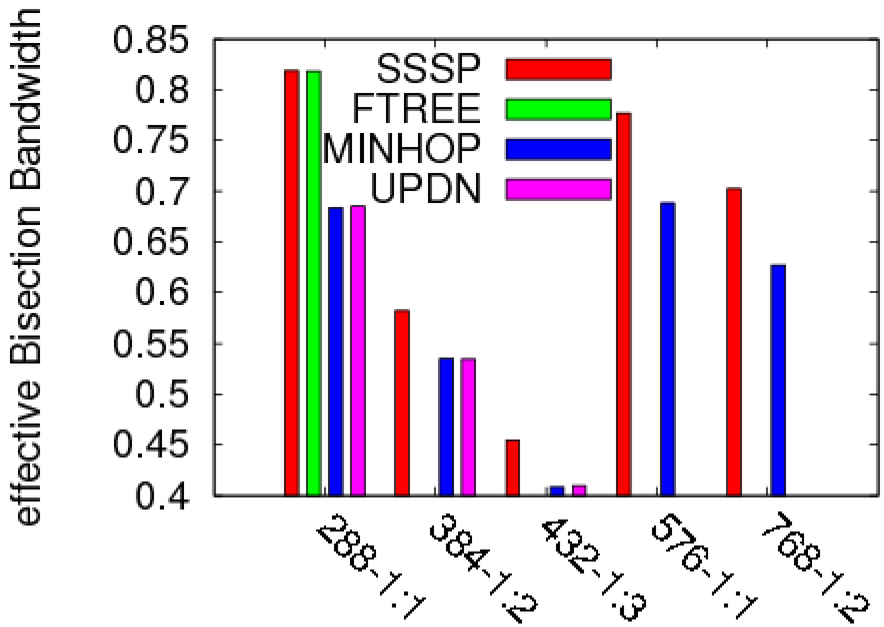

Real-World Bandwidth

Real-World Runtime



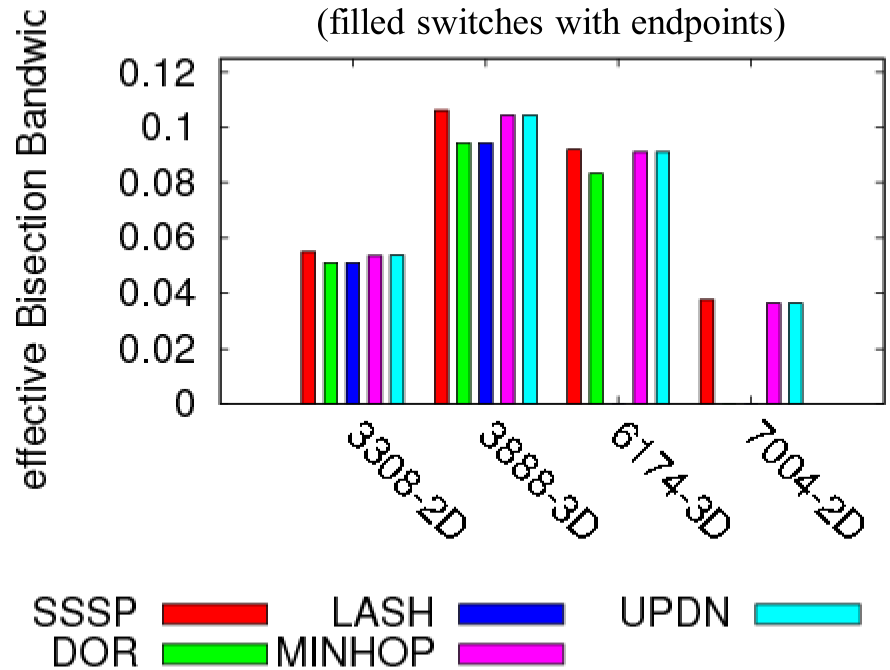
# Some more Topologies

Fat-tree topologies



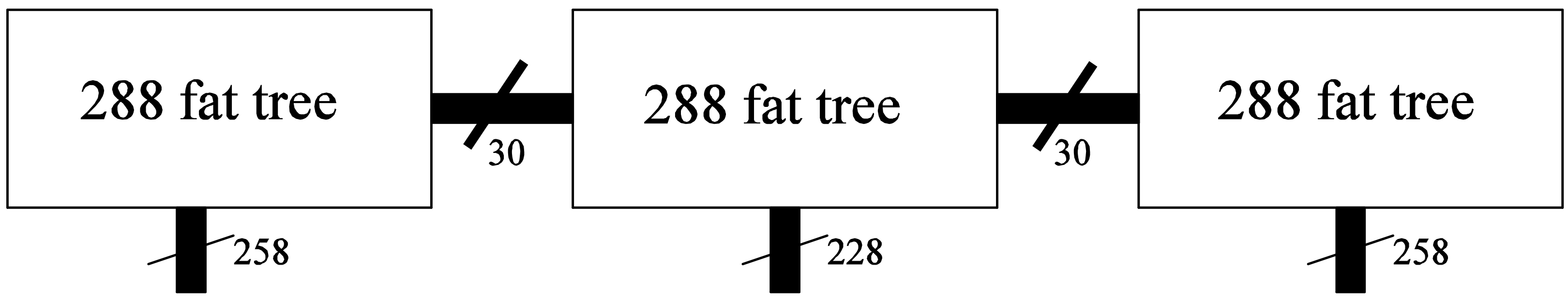

k-ary 2,3-cube topologies (torus)

(filled switches with endpoints)



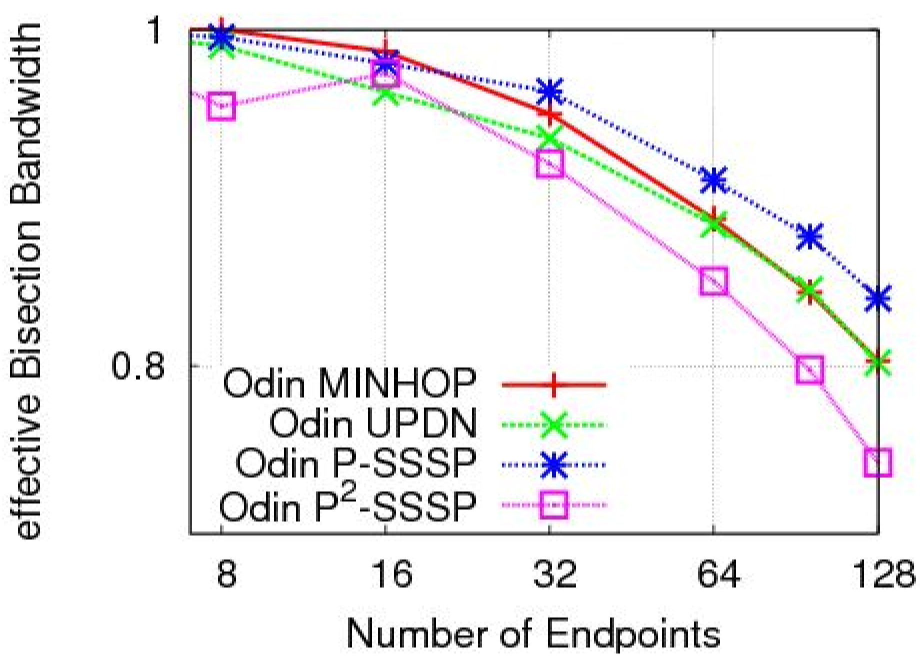
# Simulations are good, but still Simulations

- we implemented our routing with OpenSM's file method
- tested it on the Deimos and Odin clusters (needs exclusive admin access to whole machine – many thanks to Guido Juckeland)
- Odin is standard fat-tree, Deimos' topology:



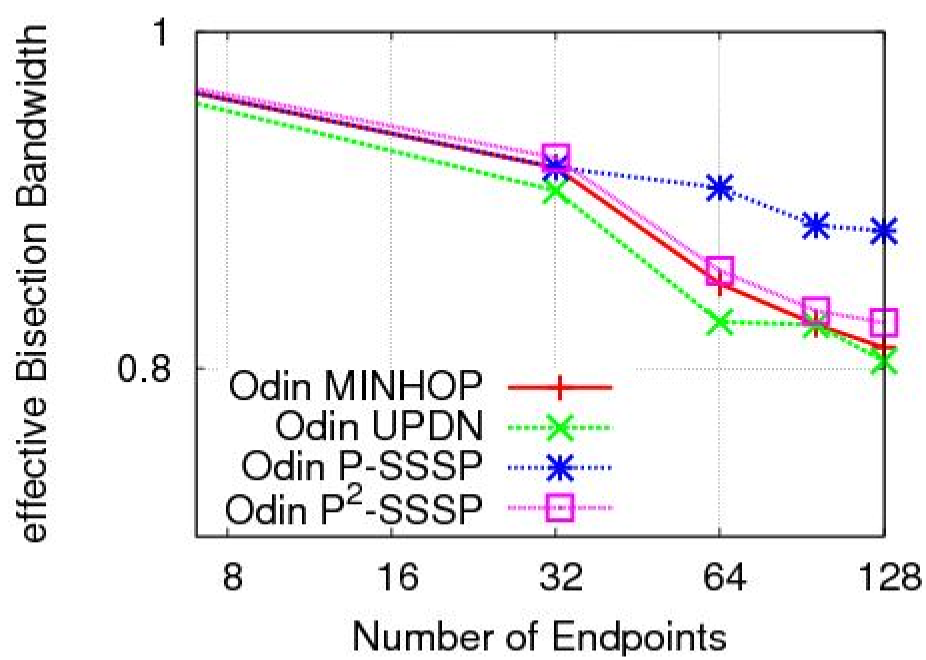
# Benchmark Results Odin

Simulation




Simulation predicts 5% improvement

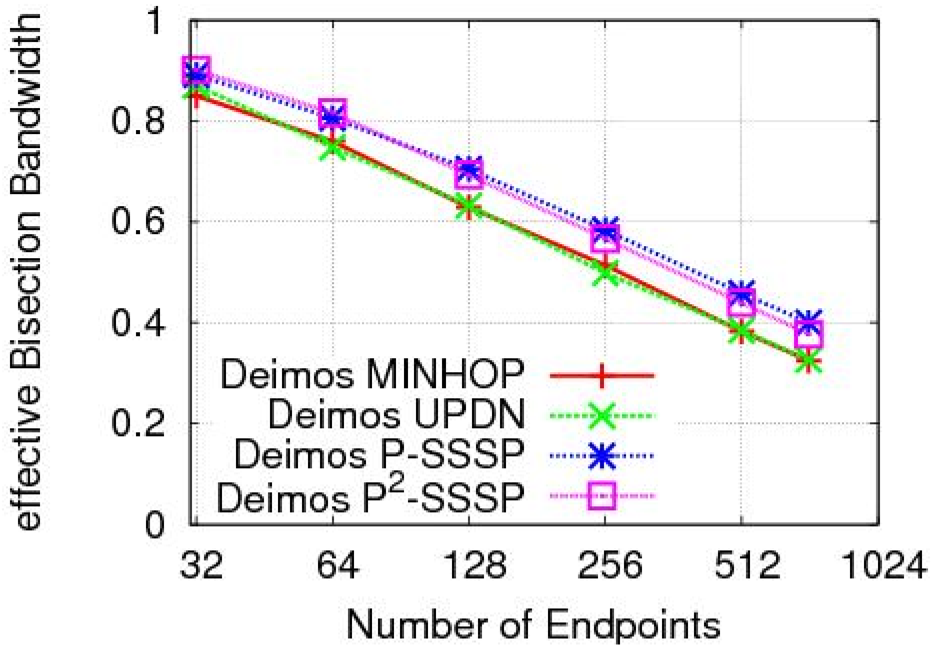
Benchmark
(Netgauge Pattern eBB)




Benchmark shows 18% improvement!

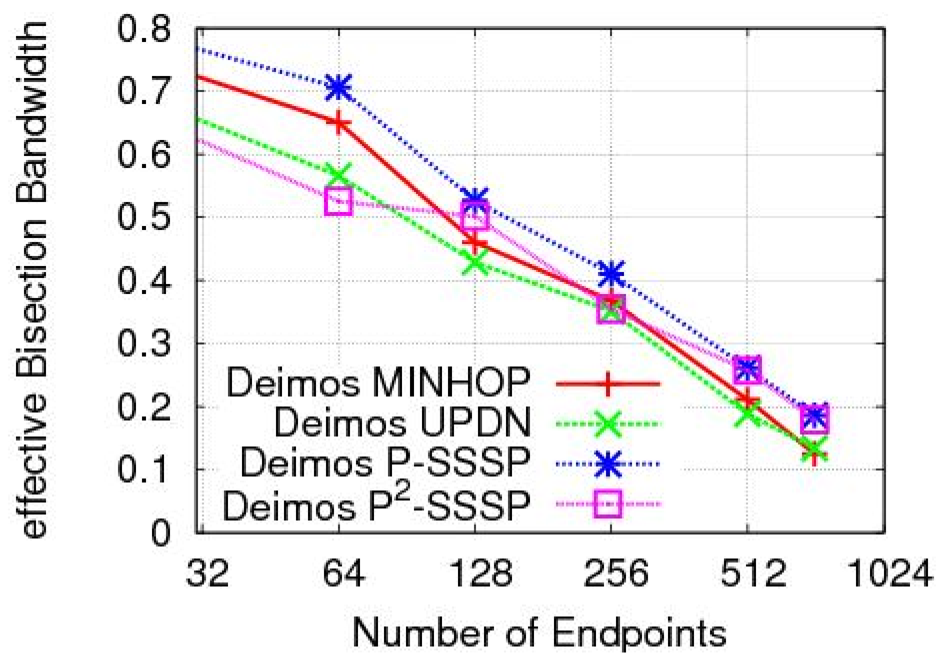
# Benchmark Results Deimos

Simulation




Simulation predicts 23% improvement


Benchmark
(Netgauge Pattern eBB)




Benchmark shows 40% improvement!

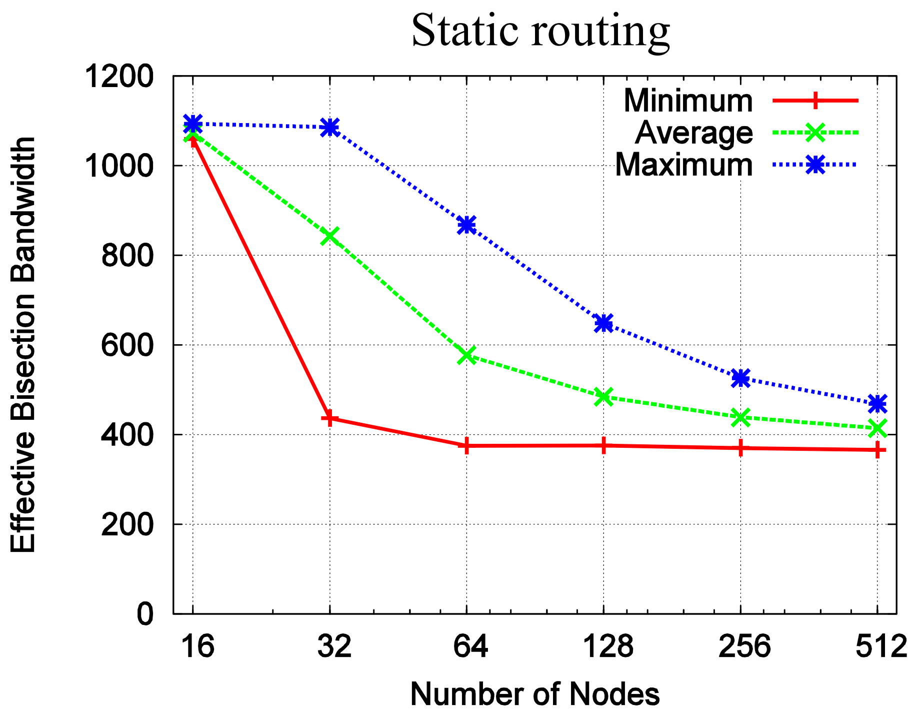
# Intermediate Conclusions

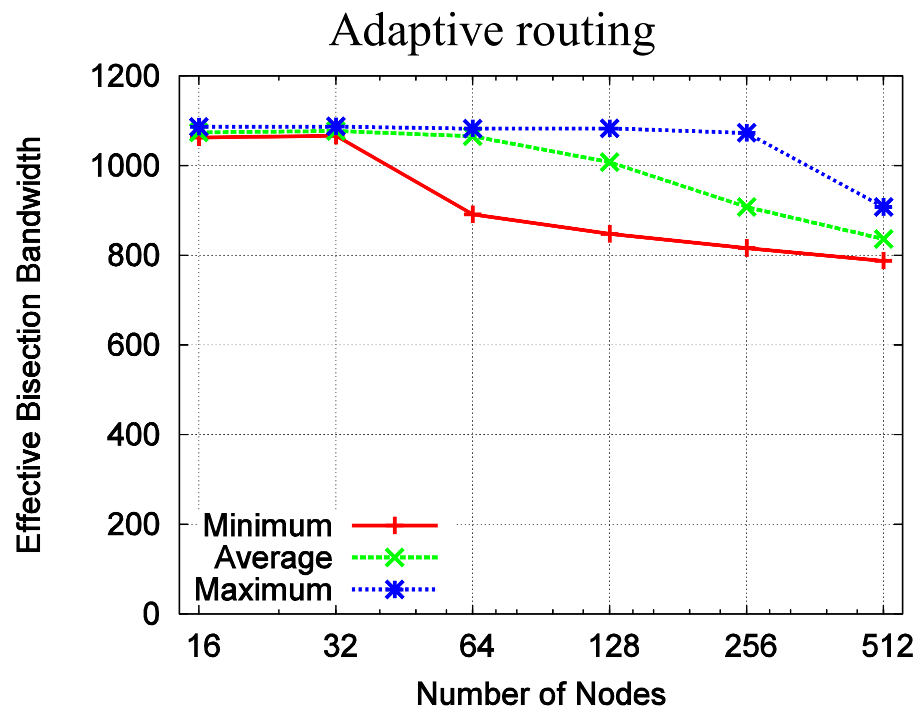
- P-SSSP routing for deterministic oblivious routing (IB) works better than established methods
- simulation shows increase in effective bisection bandwidth over standard OpenSM routing
  - e.g., Odin 5%, Deimos 23%, Atlas 15%, Thunderbird 6%
- benchmarks show even higher improvements
  - Odin 18%, Deimos 40%
- Oblivious routing seems suboptimal
  - Adaptive routing is hard
  - Random routing needs bandwidth (we have enough in fat-trees)

Refer to: Hoefler, Schneider et al. "Optimized Routing for Large-Scale InfiniBand Networks" IEEE Hot Interconnects 2009

# Adaptive Routing in Myrinet

- 512 nodes Myri 10G two-stage folded Clos network
- Netgauge, eBB with 50 MiB messages




Refer to: Geoffray, Hoefler “Adaptive Routing Strategies for Modern High Performance Networks” IEEE Hot Interconnects 2008

# Final Conclusions

- From a <u>programmers</u> perspective:
  - Specify communication at a high level
    - Communication pattern
    - Communication intensity
    - Process arrival pattern?
  - We aim to simplify and extend specification possibilities
- From a <u>system designer</u>'s perspective:
  - Optimize for applications
    - Choose model carefully (endpoint, pattern)
    - Design topology and routing accordingly
    - Provide hints to the upper layers?
- <u>Parallel systems need to be optimized as a whole</u>

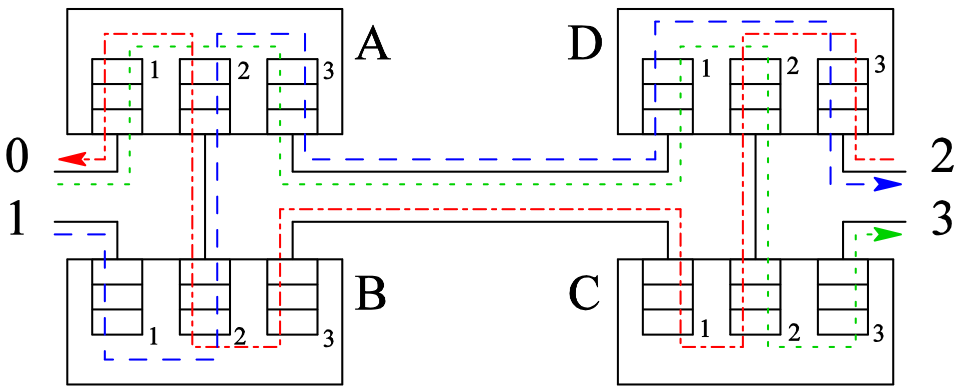
# Acknowledgments & Questions

- Thanks to:
  - Andrew Lumsdaine @IU (Ph.D., Postdoc Advisor)
  - Wolfgang Rehm @TUC (M.Sc. Advisor)
  - Timo Schneider @TUC (Student intern, Advisee)
  - Christian Siebert @NEC (M.Sc. Student, Advisee)
  - Jesper Larsson Traeff @NEC (Co-author)
  - … and all other co-authors and colleagues!
- Questions?

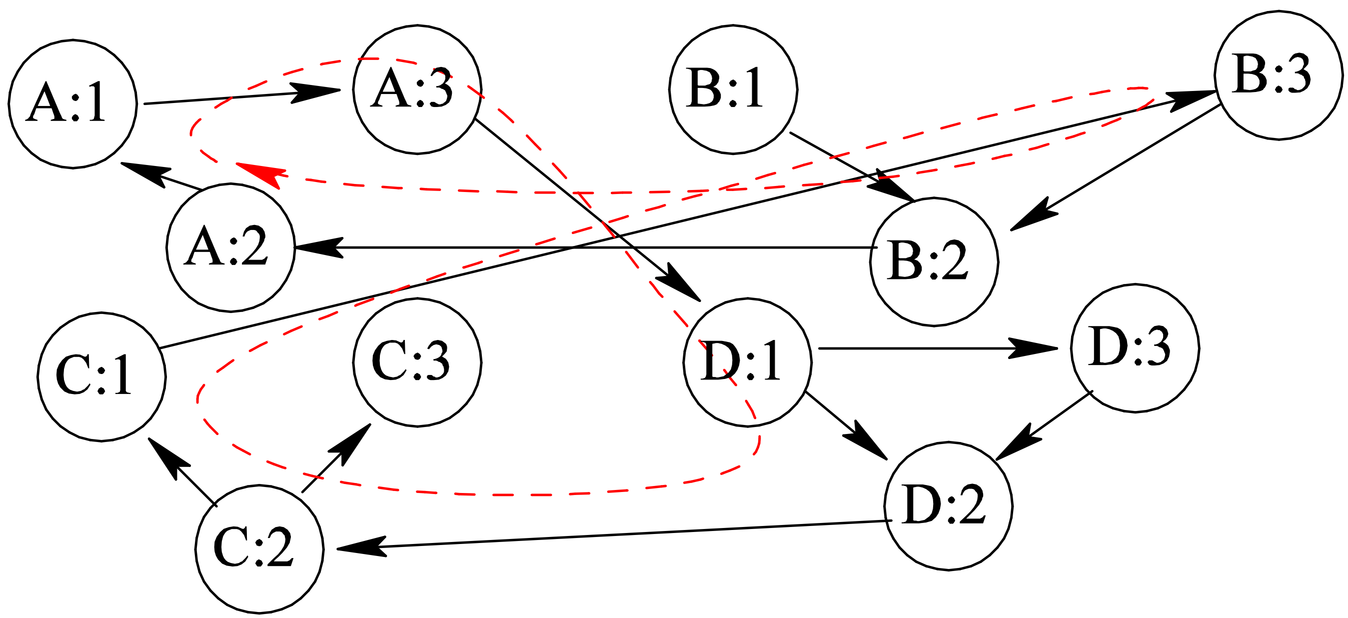
# Backup Slides

Backup Slides

# Credit Loops



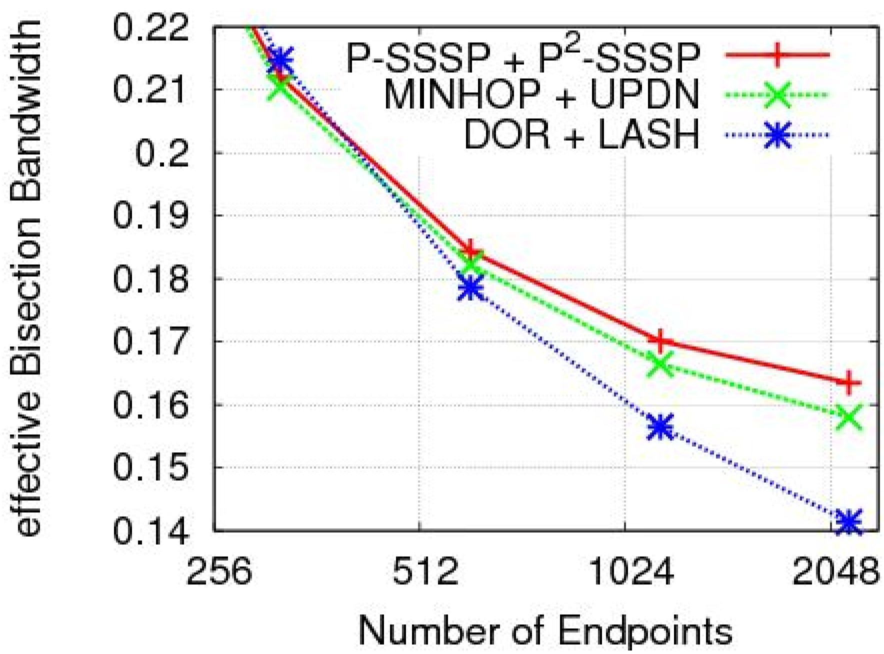

Source Network and Routes




Buffer Dependency Graph

# Even more Topologies

2-ary n-cube topologies (hypercube)
(filled switches with endpoints)




random topologies
(12 nodes per switch)