Using Multirail Networks in High-Performance Clusters

Salvador Coll, Eitan Frachtenberg, Fabrizio Petrini, Adolfy Hoisie and Leonid Gurvits

CCS-3 Modeling, Algorithms, and Informatics Group

Los Alamos National Laboratory

scoll,eitanf,fabrizio,hoisie,gurvits@lanl.gov

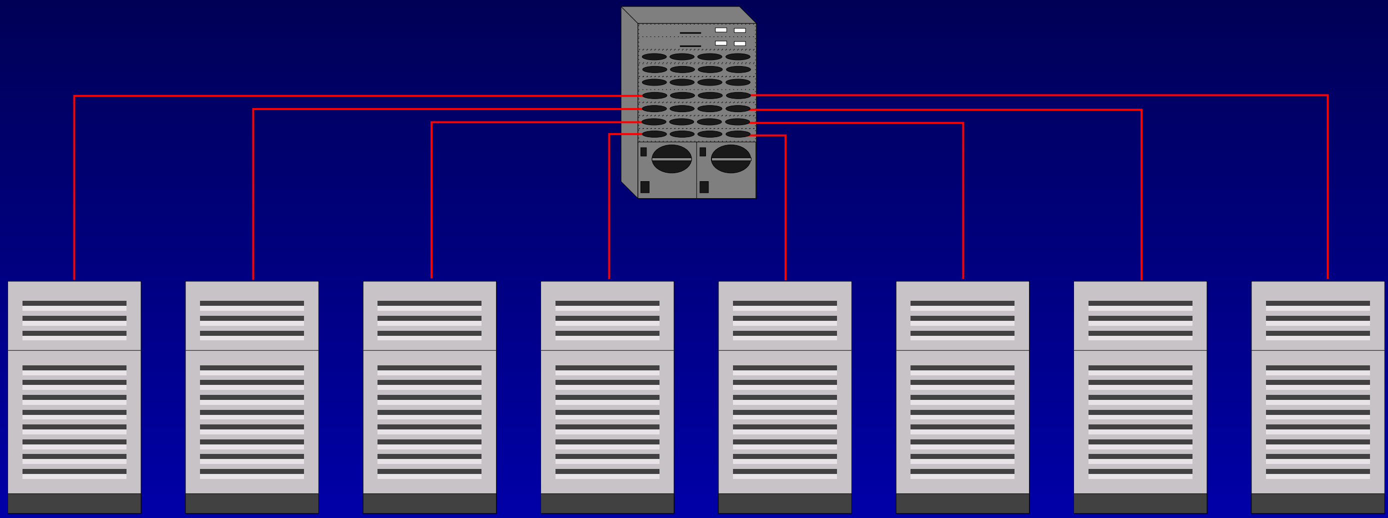
http://www.c3.lanl.gov/~fabrizio

Multiple Independent Network Rails

Using multiple independent networks is an emerging technique to (1) overcome bandwidth limitations and (2) enhance fault-tolerance.

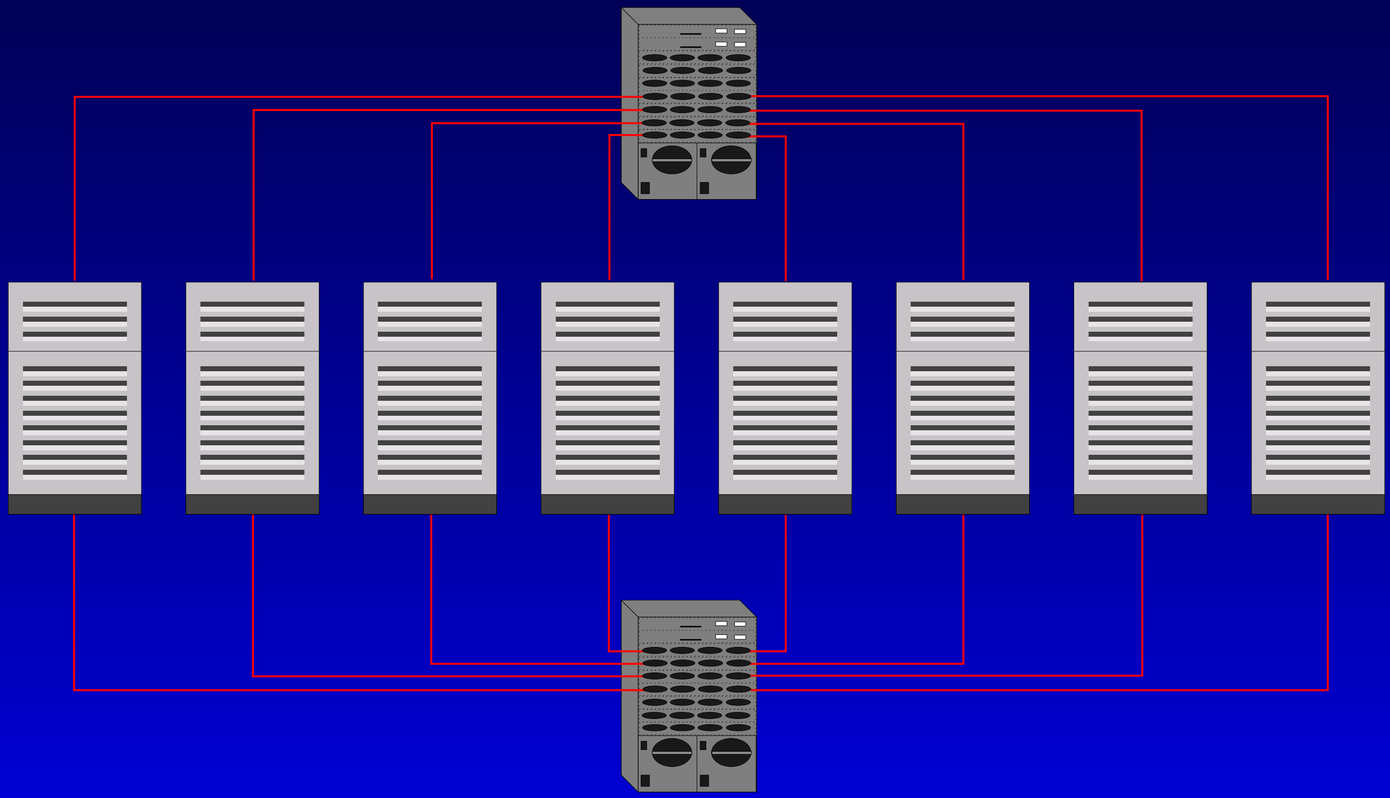
Multiple Independent Network Rails

Using multiple independent networks is an emerging technique to (1) overcome bandwidth limitations and (2) enhance fault-tolerance.



Multiple Independent Network Rails

Using multiple independent networks is an emerging technique to (1) overcome bandwidth limitations and (2) enhance fault-tolerance.



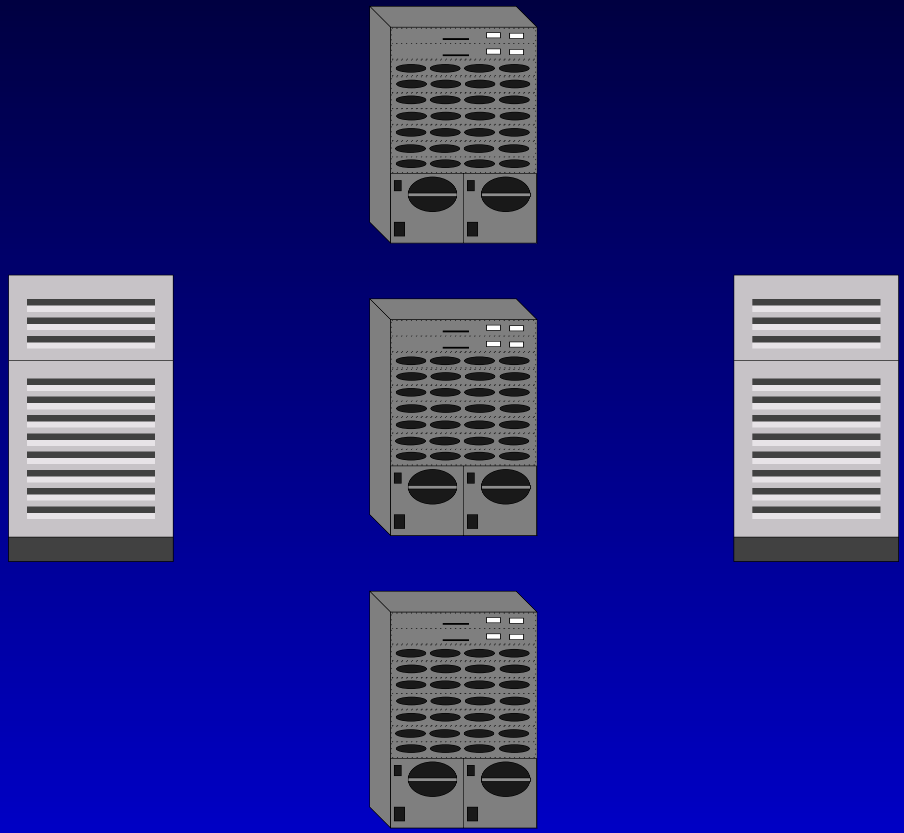
Examples of Multirailed Machines

- ASCI White at Lawrence Livermore National Laboratory – most powerful computer in the world, IBM SP
- The Terascale Computing System (TCS) at the Pittsburgh Supercomputing Center – the second most powerful computer in the world, Quadrics
- ASCI Q machine, currently under development at Los Alamos National Laboratory, Quadrics.
- Infiniband
- Experimental Linux clusters, Quadrics and Myrinet

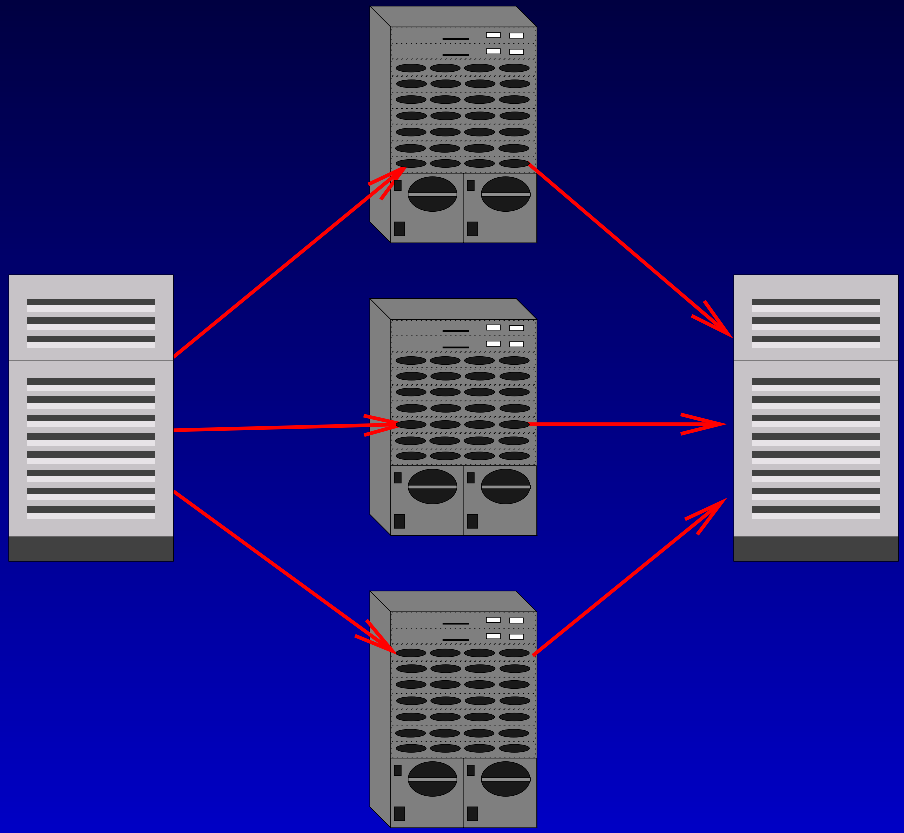
Open Problems

- Rail assignment
- Striping over multiple rails
- Implementation of communication libraries (e.g., MPI, Cray Shmem)
- Multiple rails and I/O interfaces
- Not much is known on how to use multiple rails

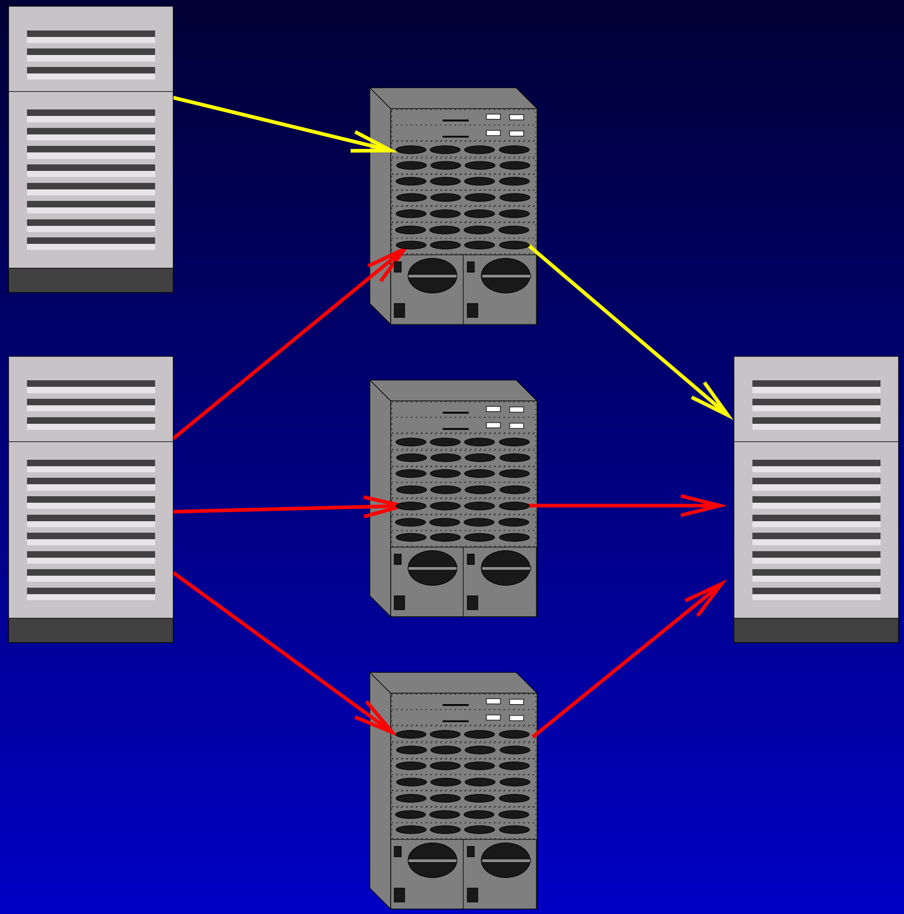
Rail Allocation



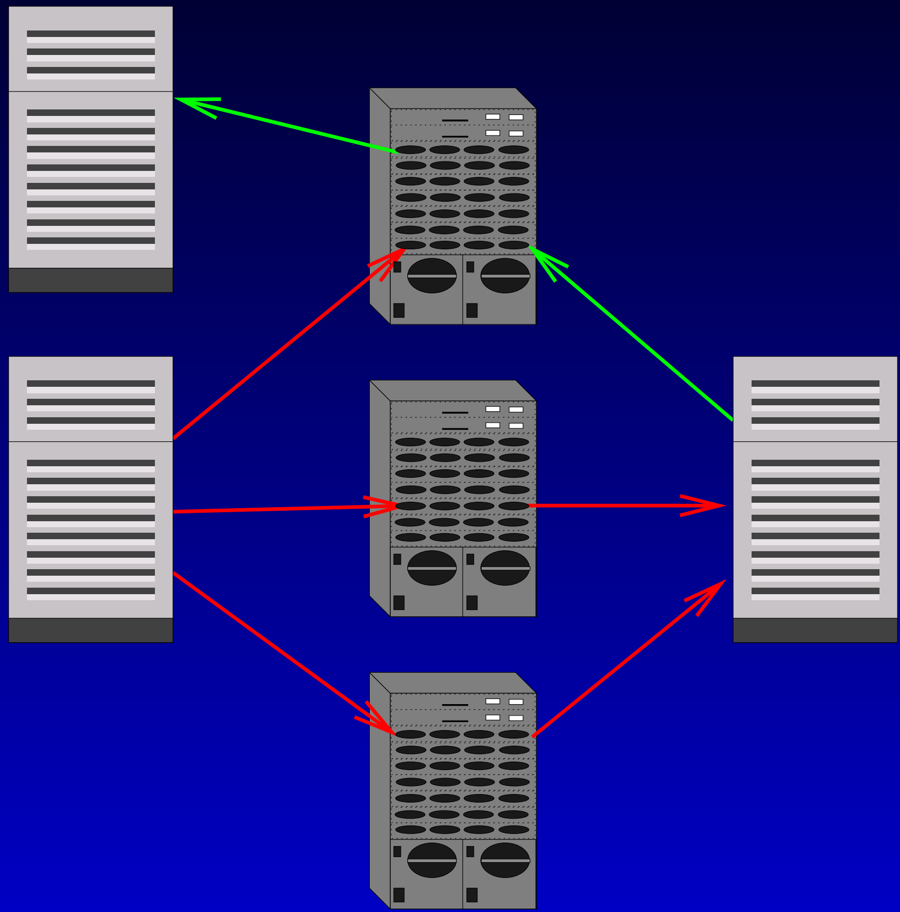
Rail Allocation



Rail Allocation



Rail Allocation



Bidirectional Traffic on the I/O bus

- Most PCI busses cannot efficiently handle bidirectional traffic with high performance networks
- Typically, aggregate bidirectional bandwidth is only 80% of the unidirectional one (Intel 840, Serverworks HE, Compaq Wildfire)
- The same problem is likely to appear in the first Infiniband and PCI-X implementations (e.g., those based on the Intel 870)
- Bidirectional traffic is very common in ASCI applications

State of the Art on Rail Allocation

- A common algorithm to allocate messages to rails is to choose the rail based on the process id of the destination process ($rail = destination\_id \bmod RAILS$)
- Multiple processes can compete for the same rail even if other rails are available
- No message striping
- No attempt to minimize bidirectional traffic

Los Alamos NATIONAL LABORATORY

Outline

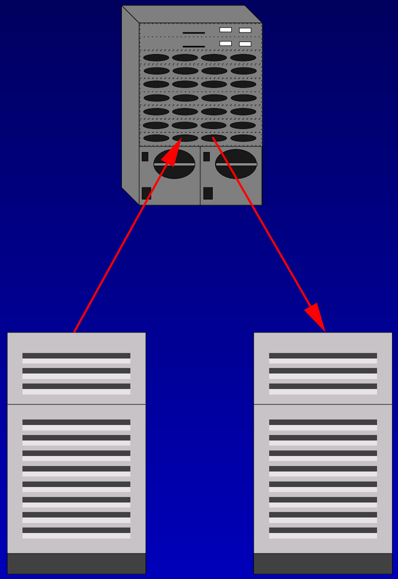
- Basic Algorithm
- Static rail allocation
- Dynamic rail allocation with local information
- Dynamic rail allocation with global information
- Hybrid algorithm
- Experimental evaluation
- Conclusion

Basic Algorithm

- The basic algorithm doesn't use any communication protocol
- Whenever a node needs to send a message,it send it on one rail,choosing it in round-robin fashion
- This base case can serve to illustrate the effects of both the overhead of the other protocols and the penalties of the bidirectional traffic

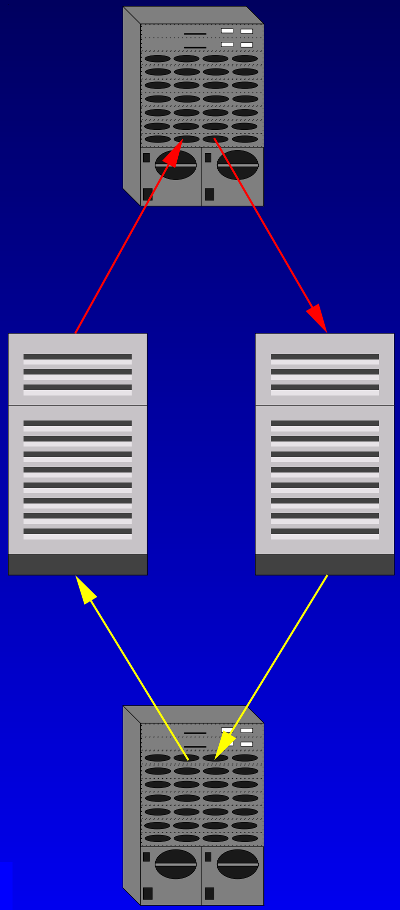
Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



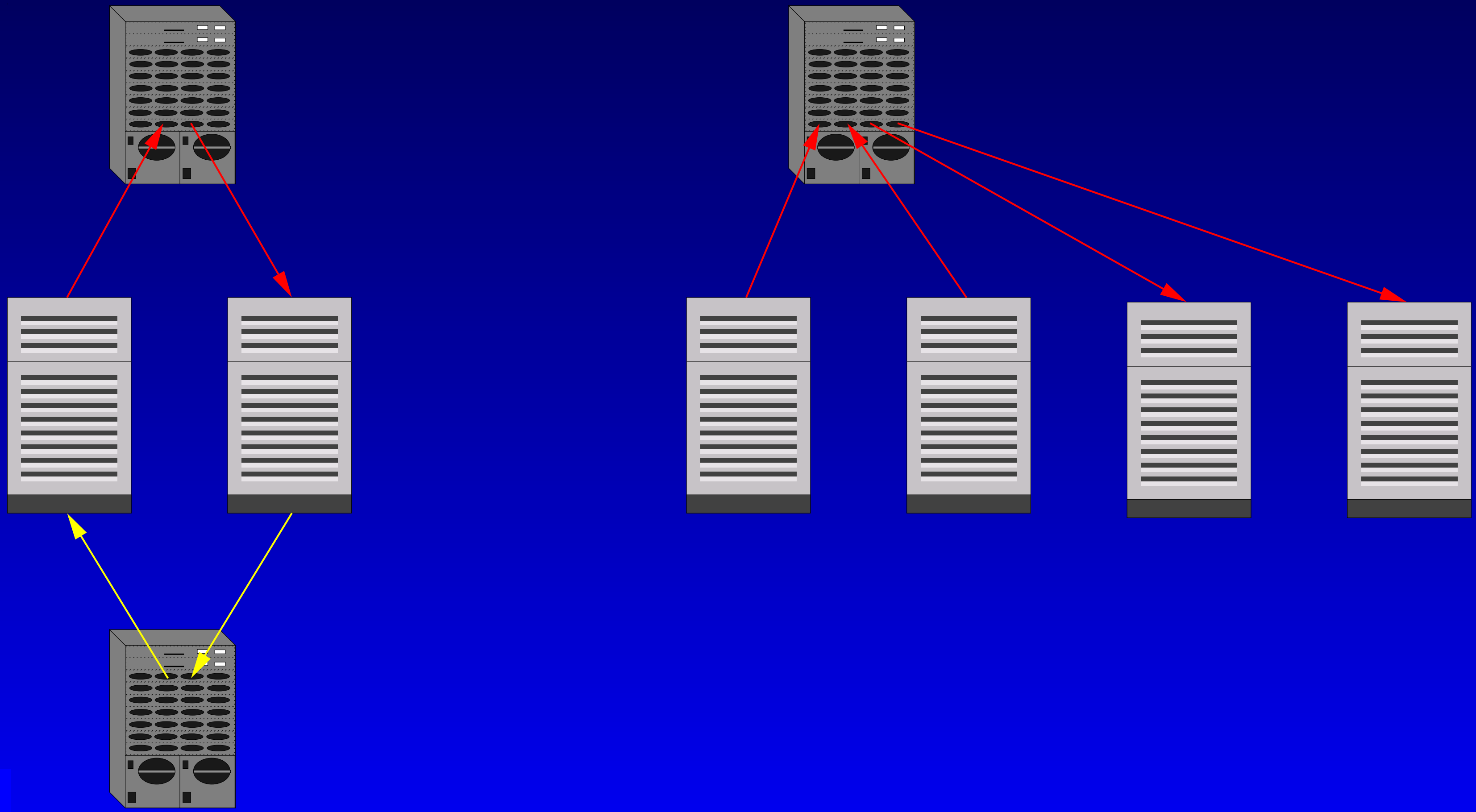
Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



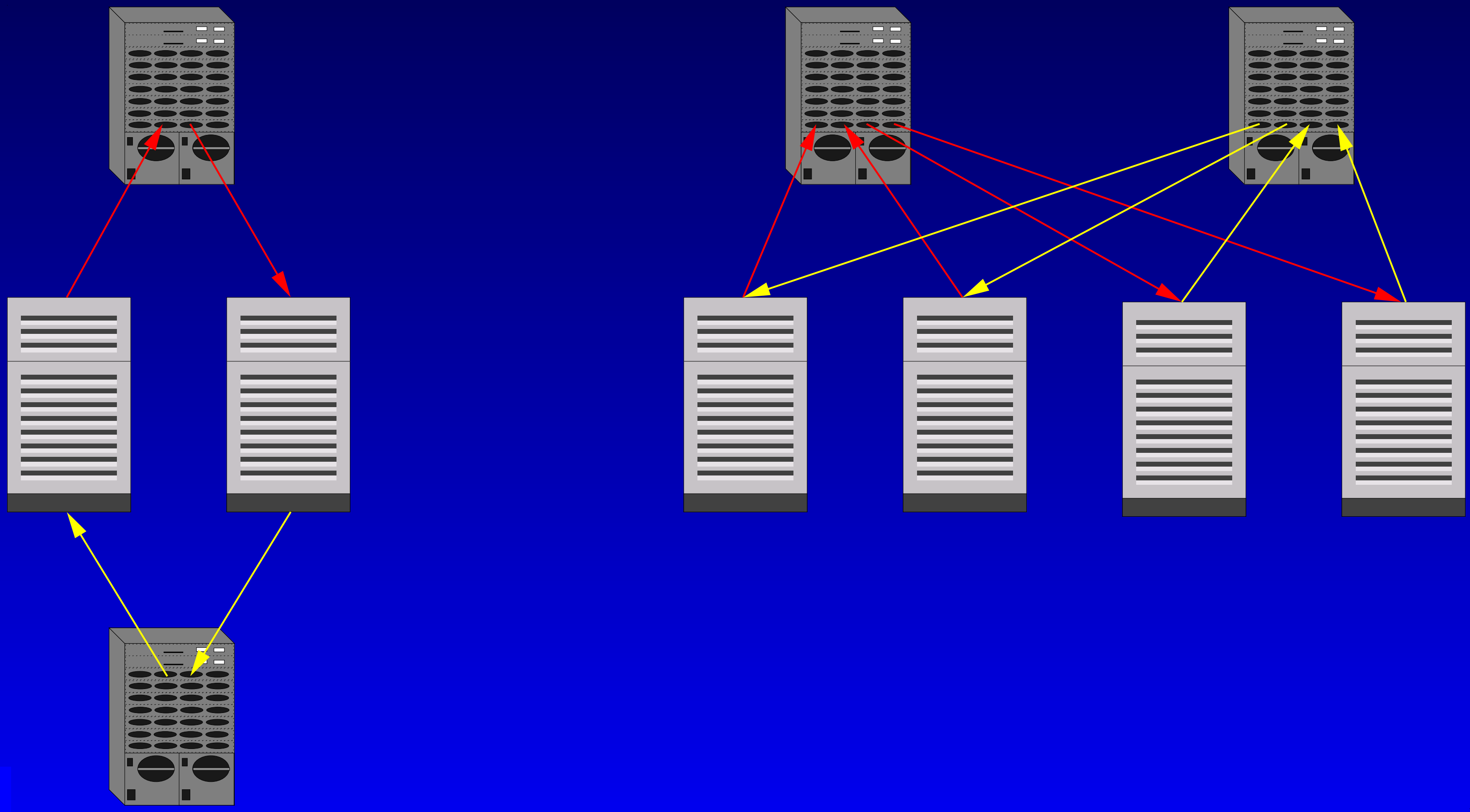
Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



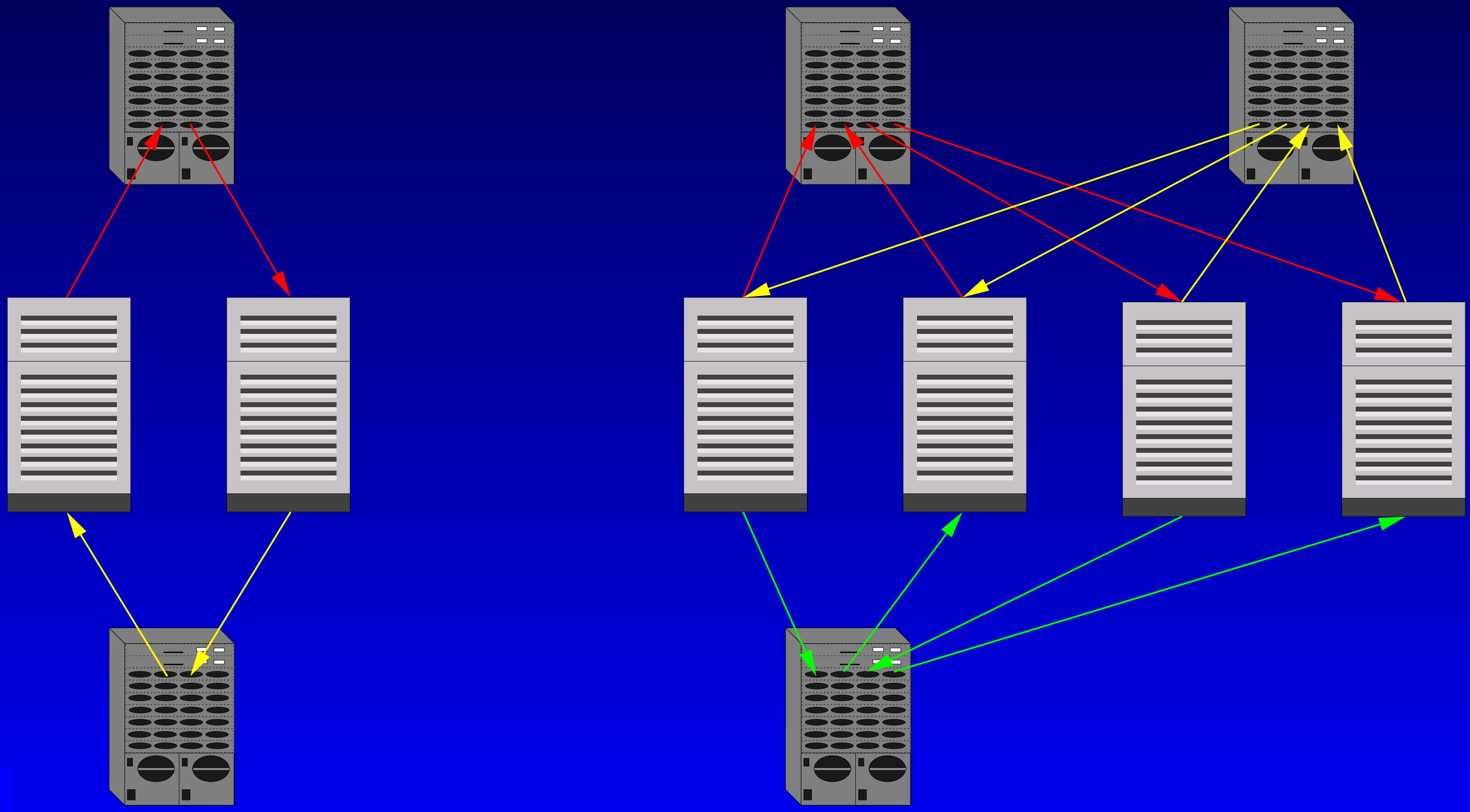
Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



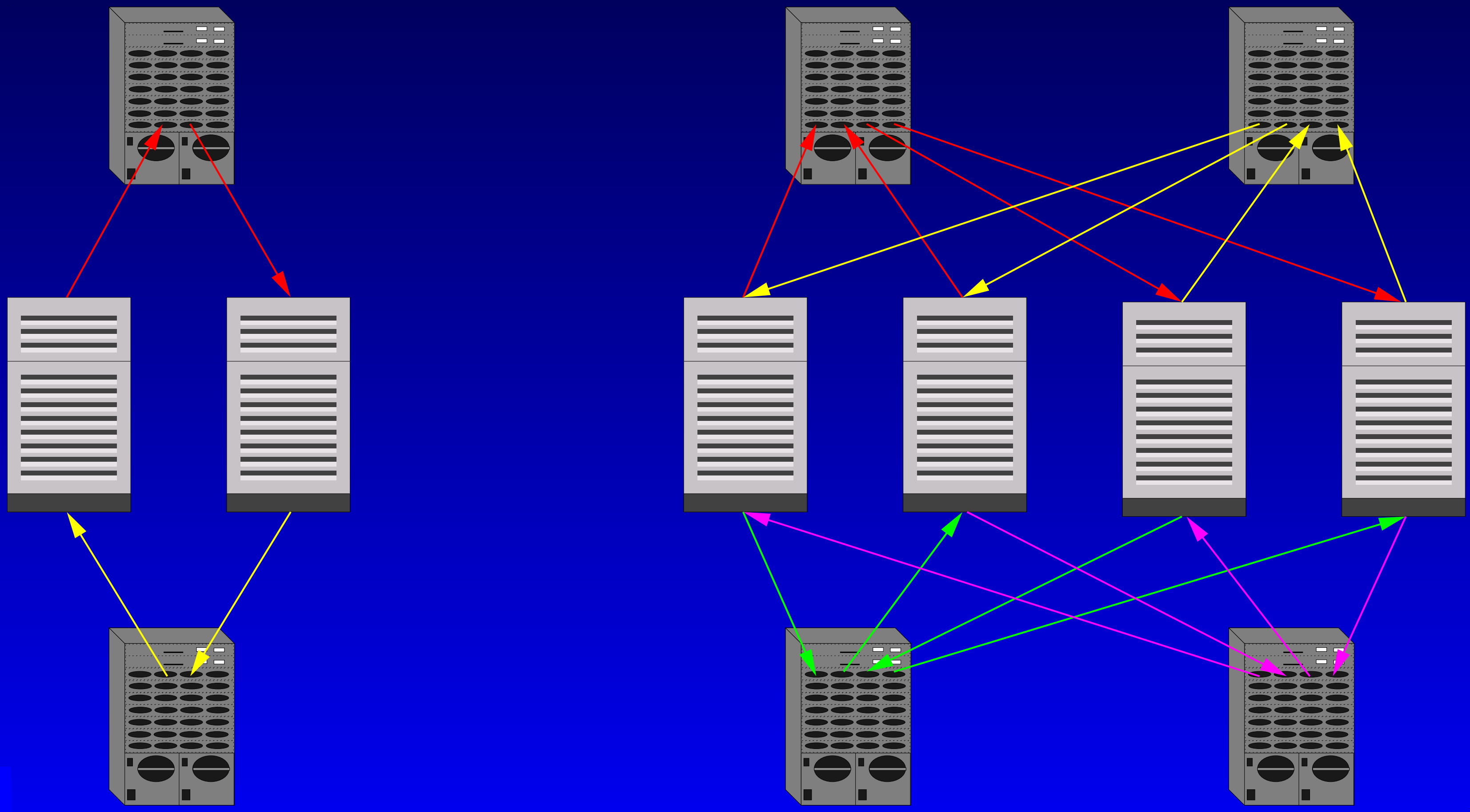
Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



Static Rail Allocation

- With static rail allocation each network interface can either send or receive messages, and the direction is defined at initialization time.



Lower Bound with Static Rail Allocation

A high number of rails is required for statically allocated unidirectional traffic.

A network with r rails can support no more than n nodes, where

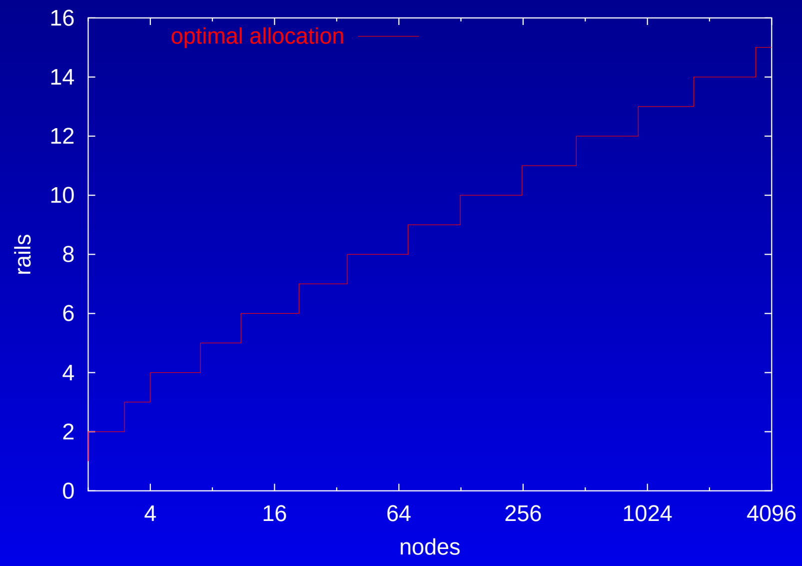
$$n \leq \binom{r}{\left\lfloor \frac{r}{2} \right\rfloor}$$

Lower Bound with Static Rail Allocation

A high number of rails is required for statically allocated unidirectional traffic.

A network with *r* rails can support no more than *n* nodes, where

$$n \leq \binom{r}{\lfloor \frac{r}{2} \rfloor}$$



Dynamic Algorithm with Local Information

- With the dynamic algorithms the direction in which each network interface is used can change over time
- The *local-dynamic* algorithm allocates the rails in both directions, using local information available on the sender side
- Messages are sent over rails that not sending or receiving other messages
- Messages can be striped over multiple rails
- There is no guarantee that traffic will be unidirectional

Los Alamos
NATIONAL LABORATORY

Dynamic Algorithm with Global Information

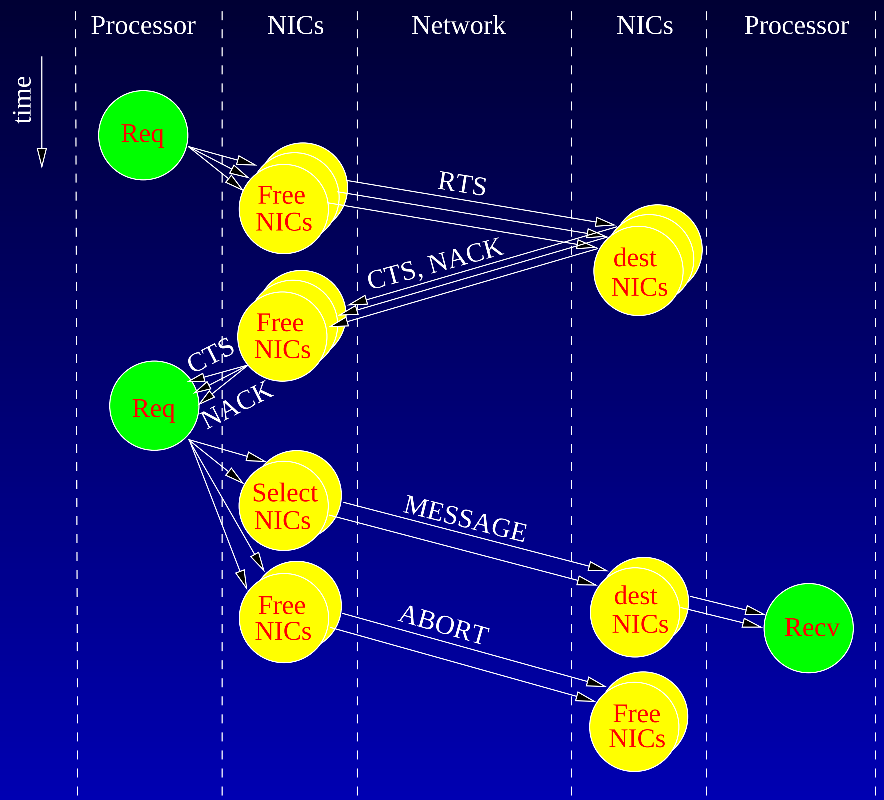
- The *dynamic* algorithm tries to reserve both end-points before sending a message
- In its core there is a sophisticated distributed algorithm that (1) ensures unidirectional traffic at both ends and (2) avoids deadlocks, potentially generated by multiple requests with a cyclic dependency

Dynamic Algorithm: Implementation Issues

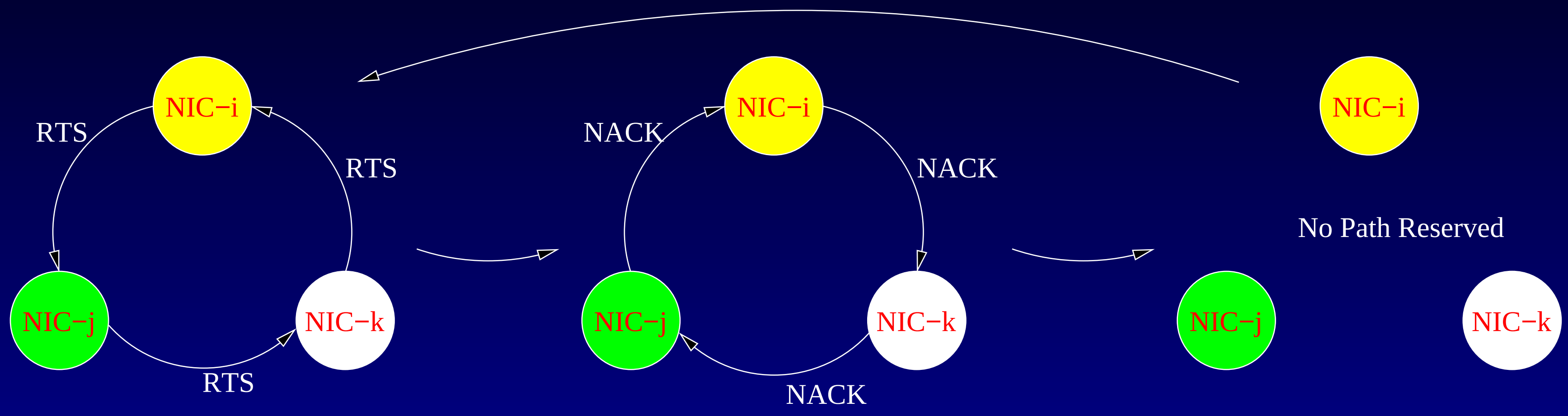
- The efficient implementation of this algorithm requires some processing power in the network interface, which needs to process control packets and perform the reservation protocol without interfering with the host
- For example, the Quadrics network interface is equipped with a thread processor that can process an incoming packet, do some basic processing and send a reply in as few as 2μs

Los Alamos NATIONAL LABORATORY

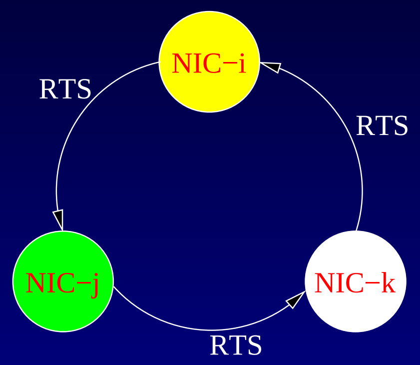
Dynamic Algorithm



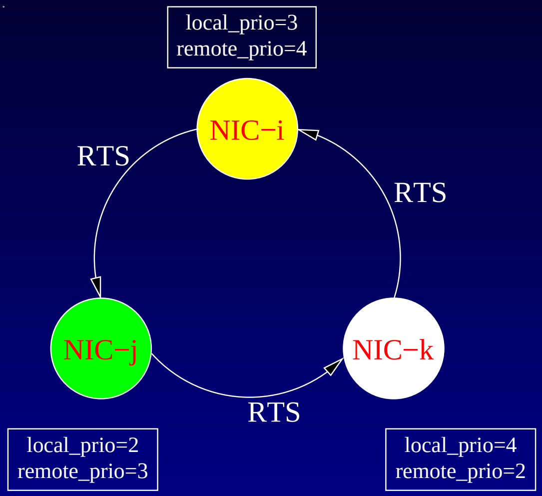
Deadlock (Livelock) in the Dynamic Algorithm



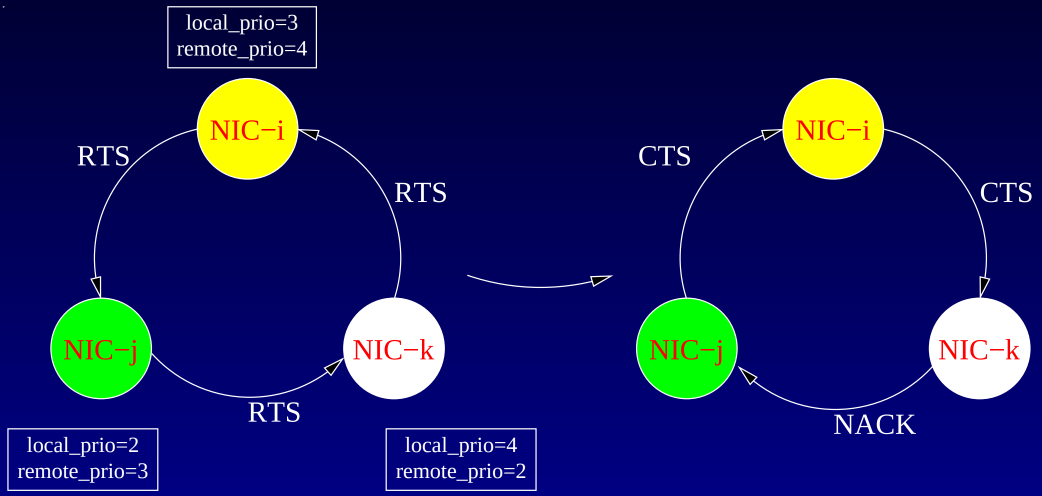
Deadlock Avoidance in the Dynamic Algorithm



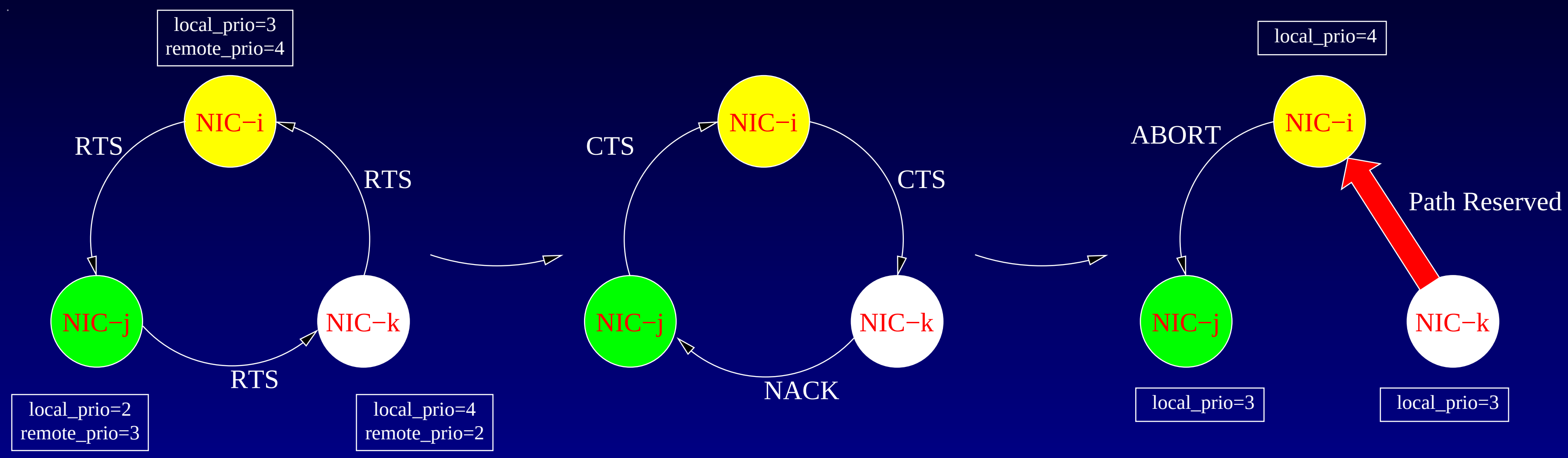
Deadlock Avoidance in the Dynamic Algorithm



Deadlock Avoidance in the Dynamic Algorithm



Deadlock Avoidance in the Dynamic Algorithm



Hybrid Algorithm

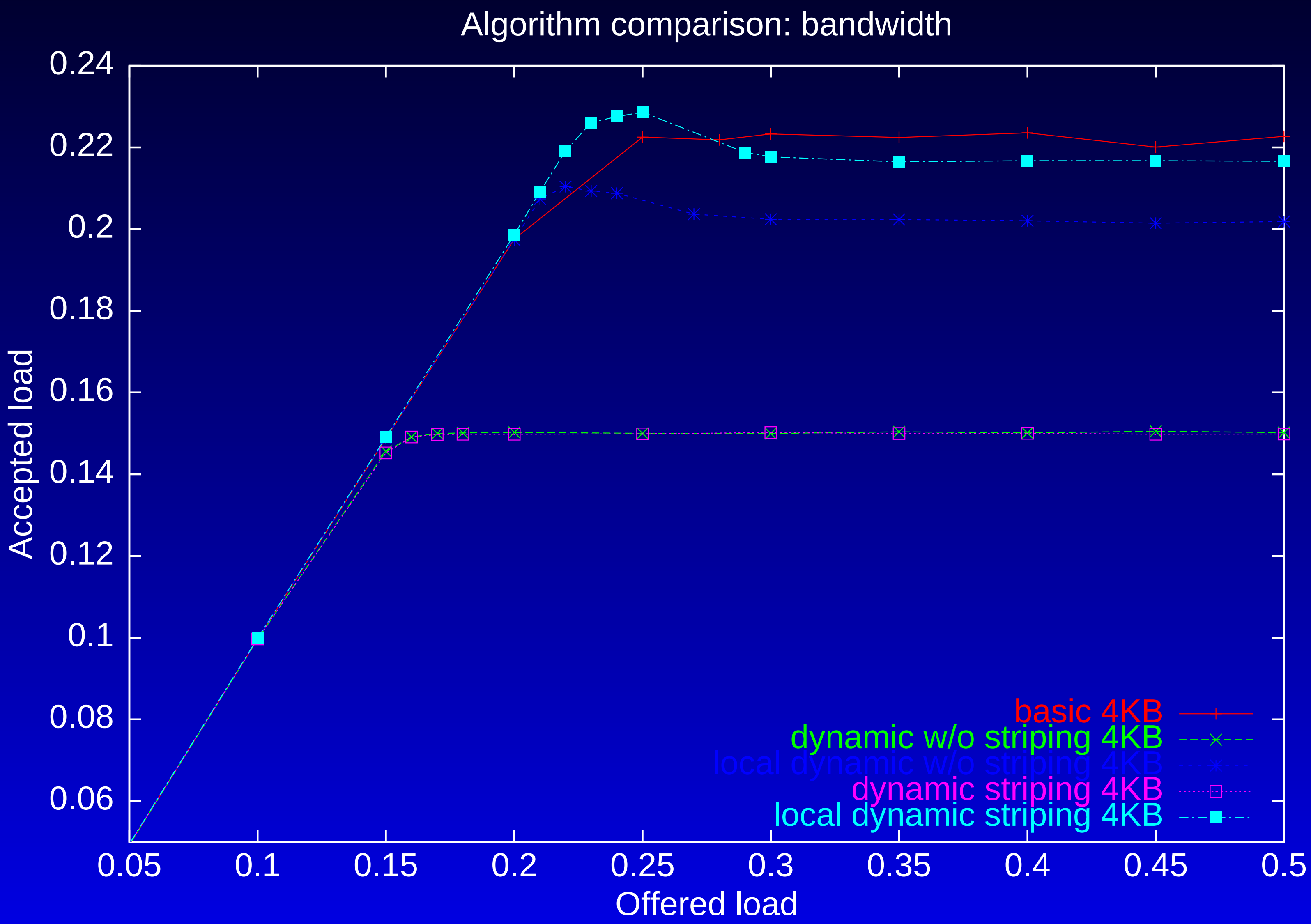
- The dynamic algorithm incurs a substantial overhead, for every message size.
- The hybrid algorithm sends short message without a reservation protocol
- Short messages are not striped
- It can cause bidirectional traffic for a short time

Experimental Framework

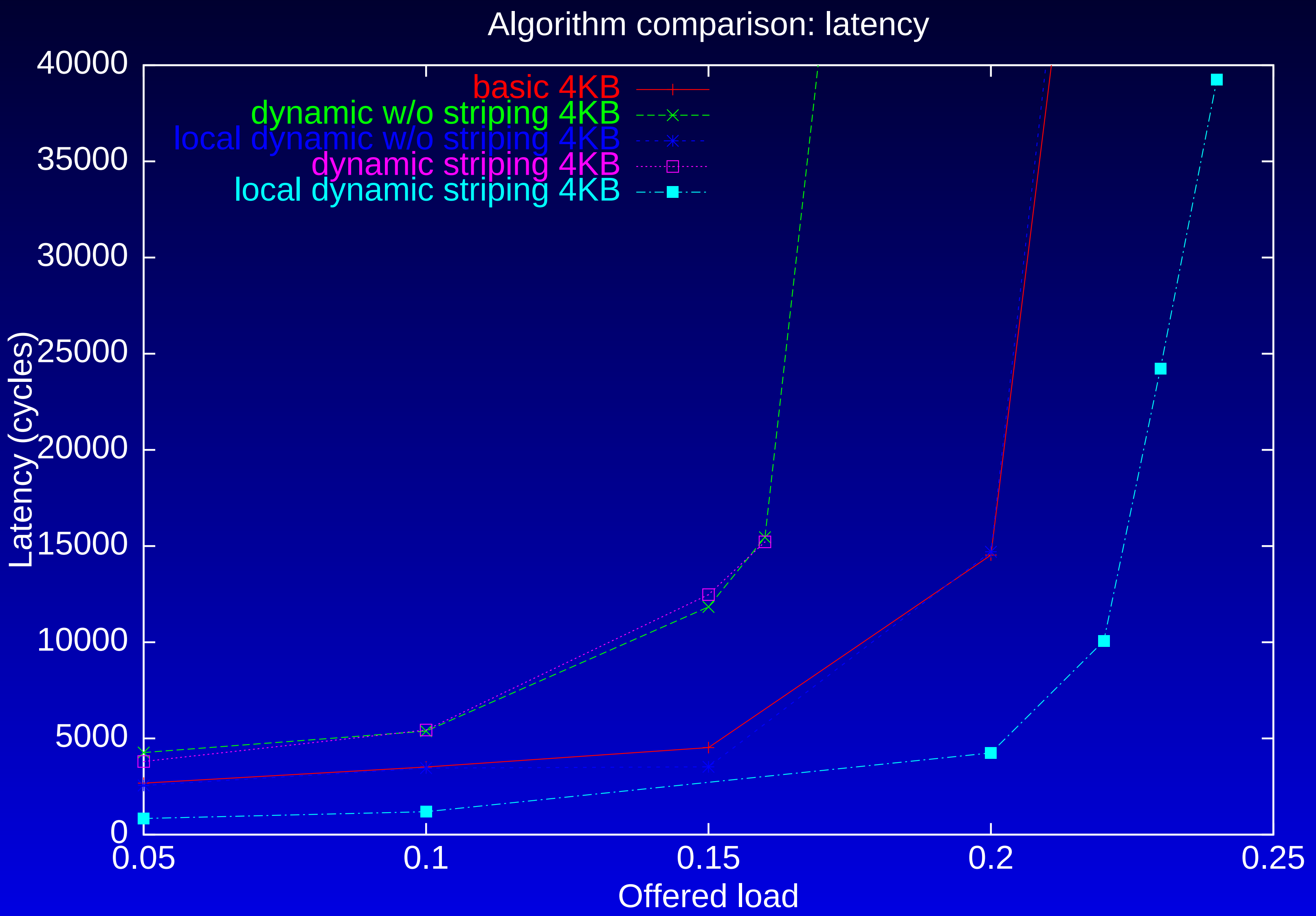
- We focus our attention on a family of fat-trees interconnection networks, ranging from 32 to 128 processing nodes
- Up to 8 independent rails
- Low level simulation of the network (network model based on the Quadrics network)
- We simulate the communication processor in the network interface
- We test the network using a synthetic communication benchmark
- We assess the network performance by using to parameters, the overall *accepted bandwidth* and the message *latency*

Los Alamos
NATIONAL LABORATORY

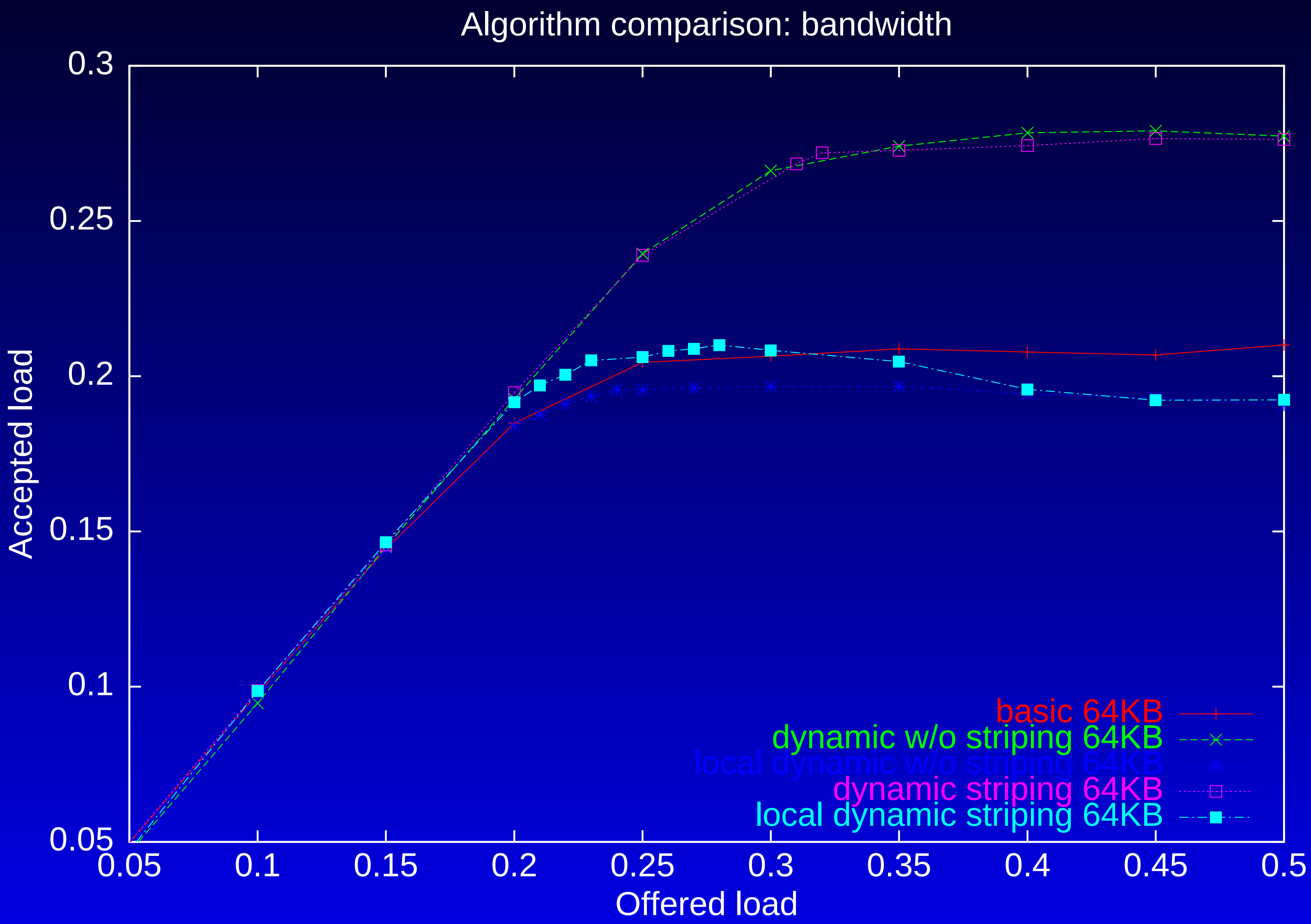
Results - Bandwidth, 4KB messages



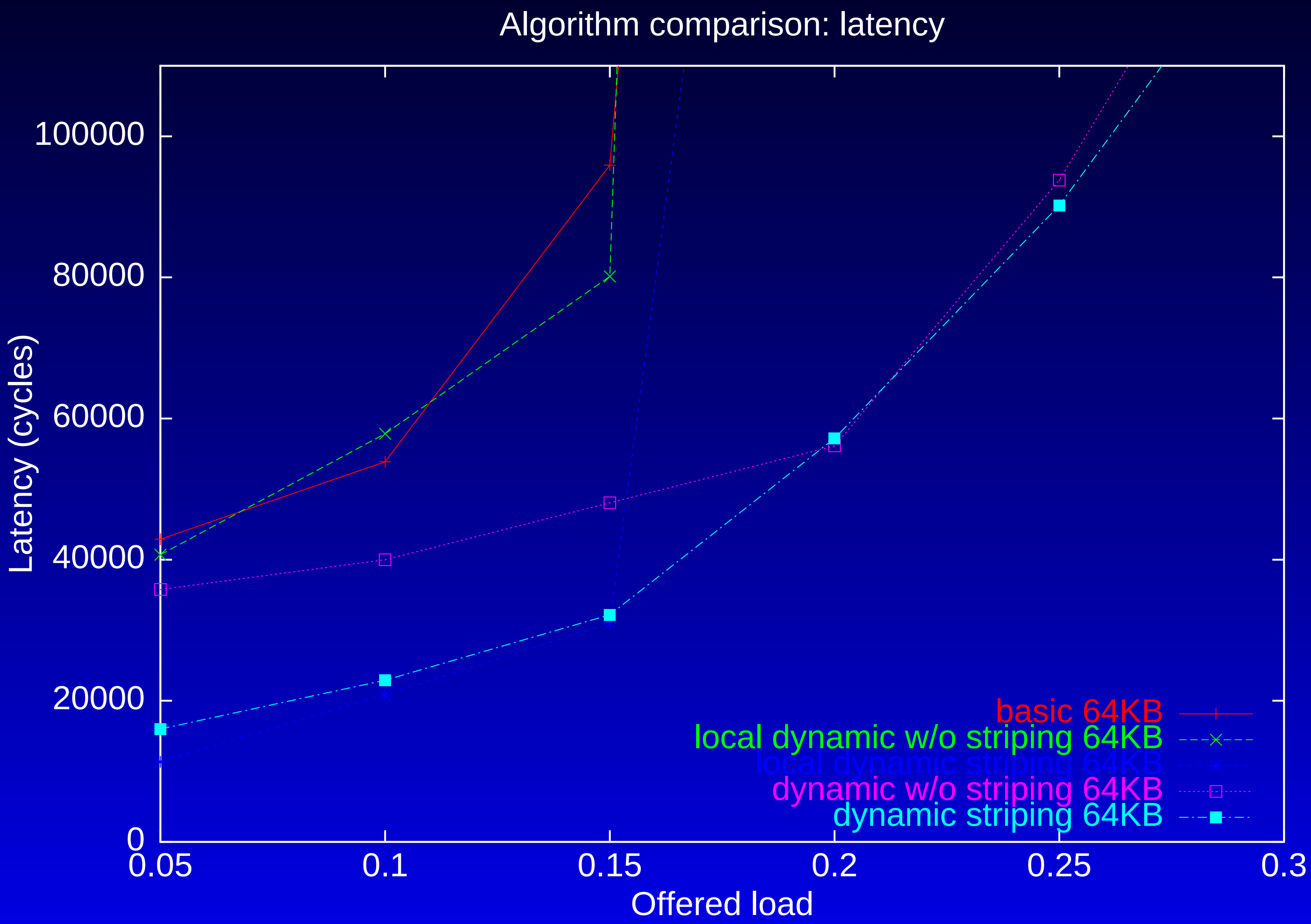
Results - Latency, 4KB messages



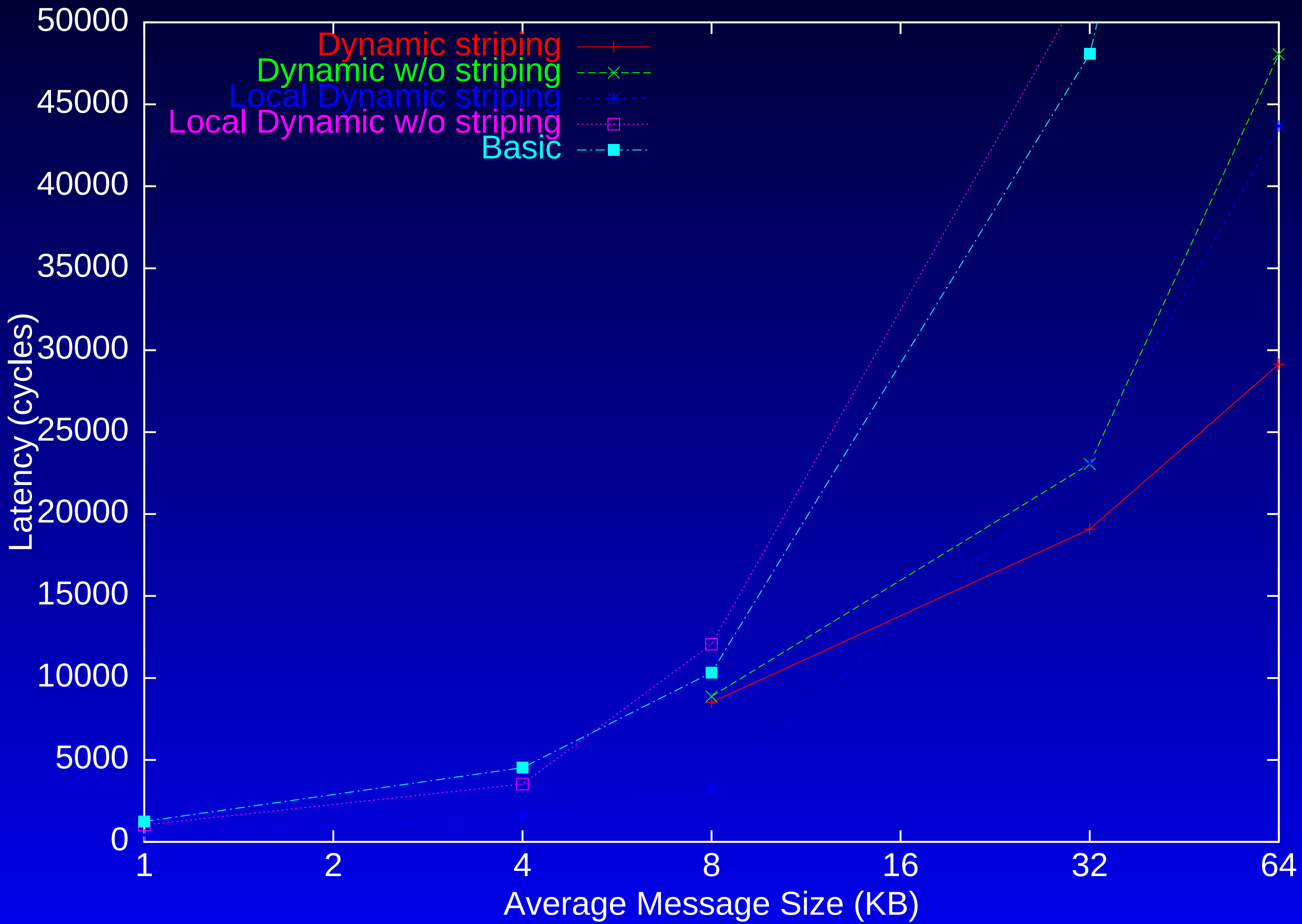
Results - Bandwidth, 64KB messages



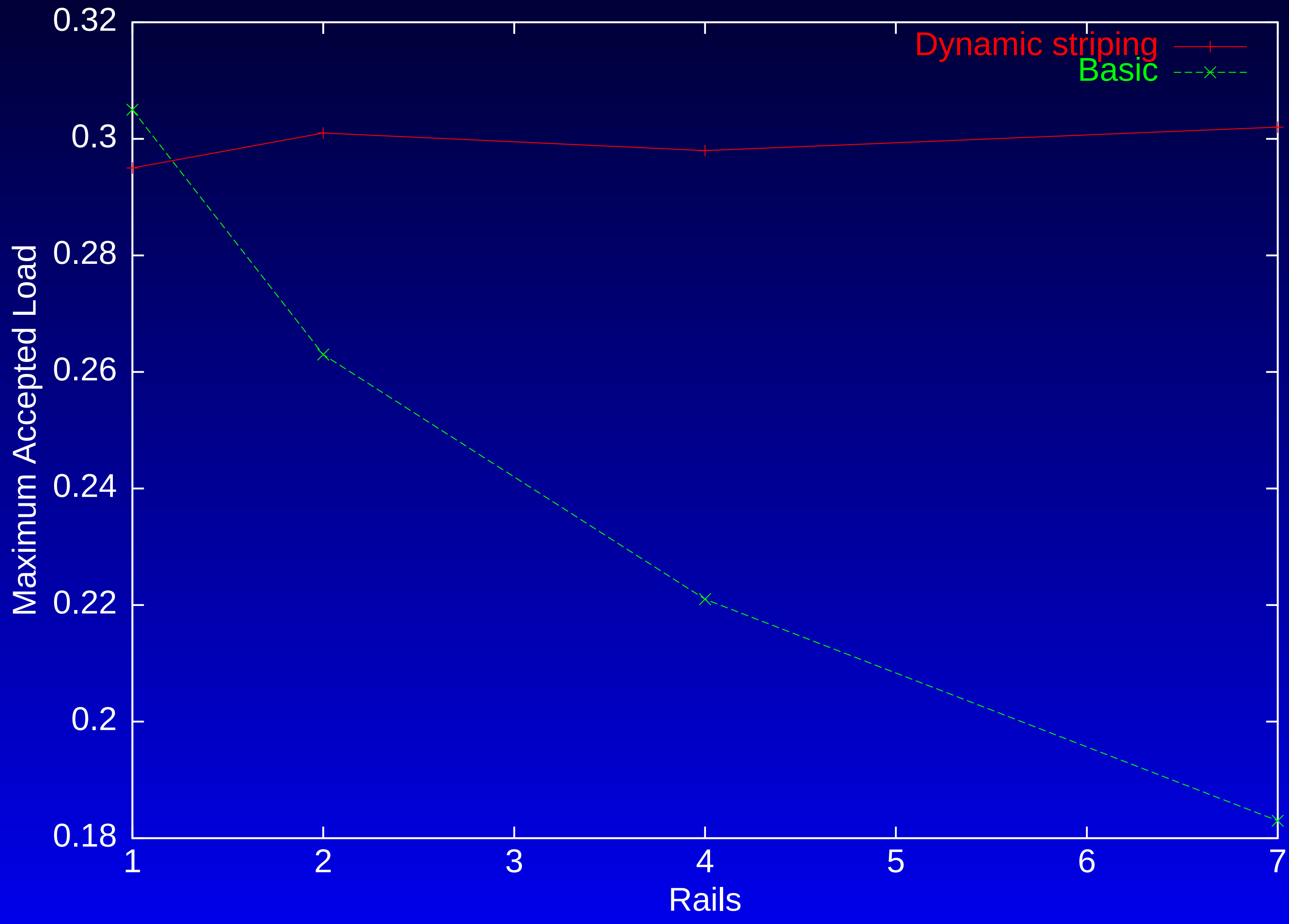
Results - Latency, 64KB messages



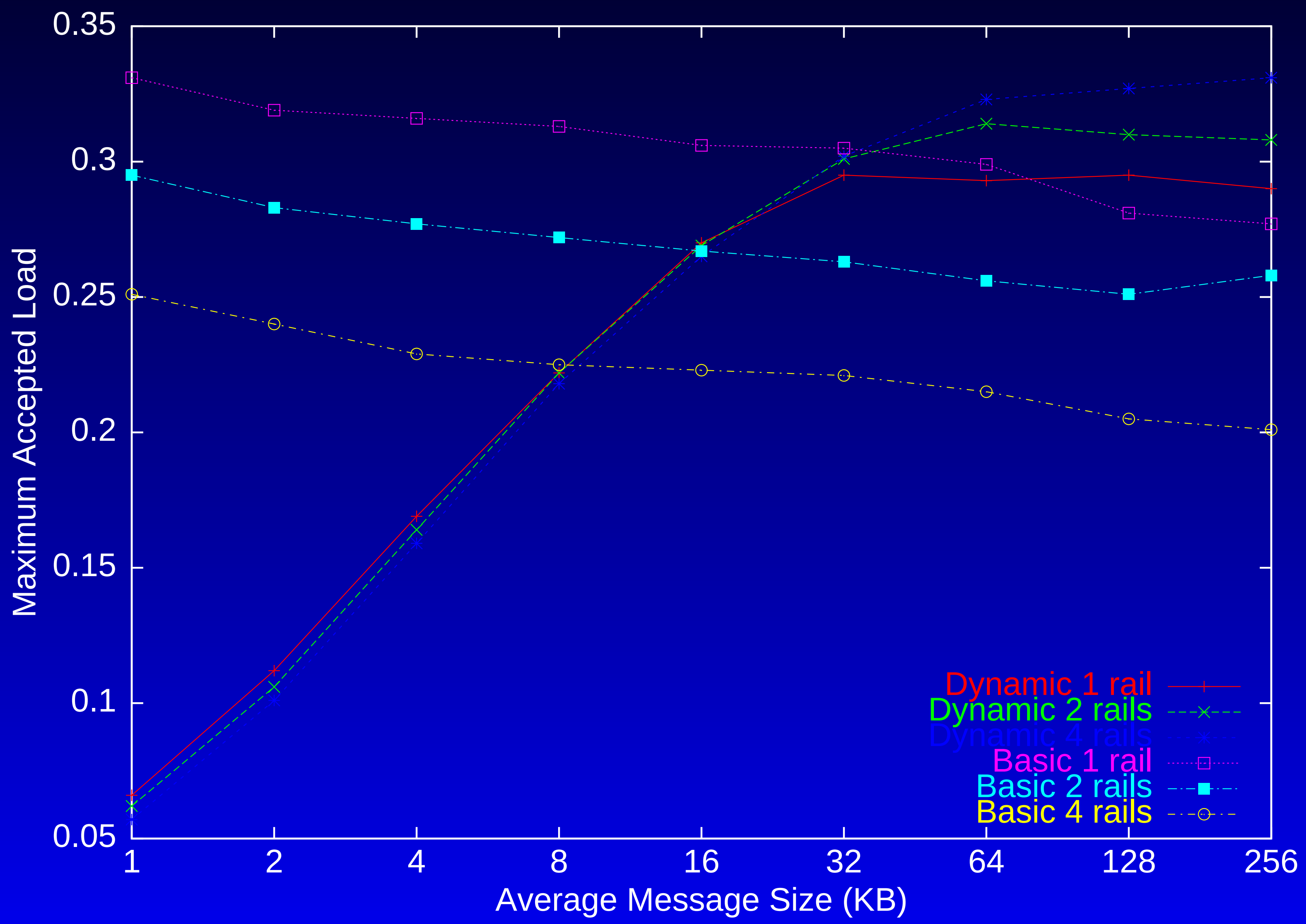
Results - Latency vs Message Size



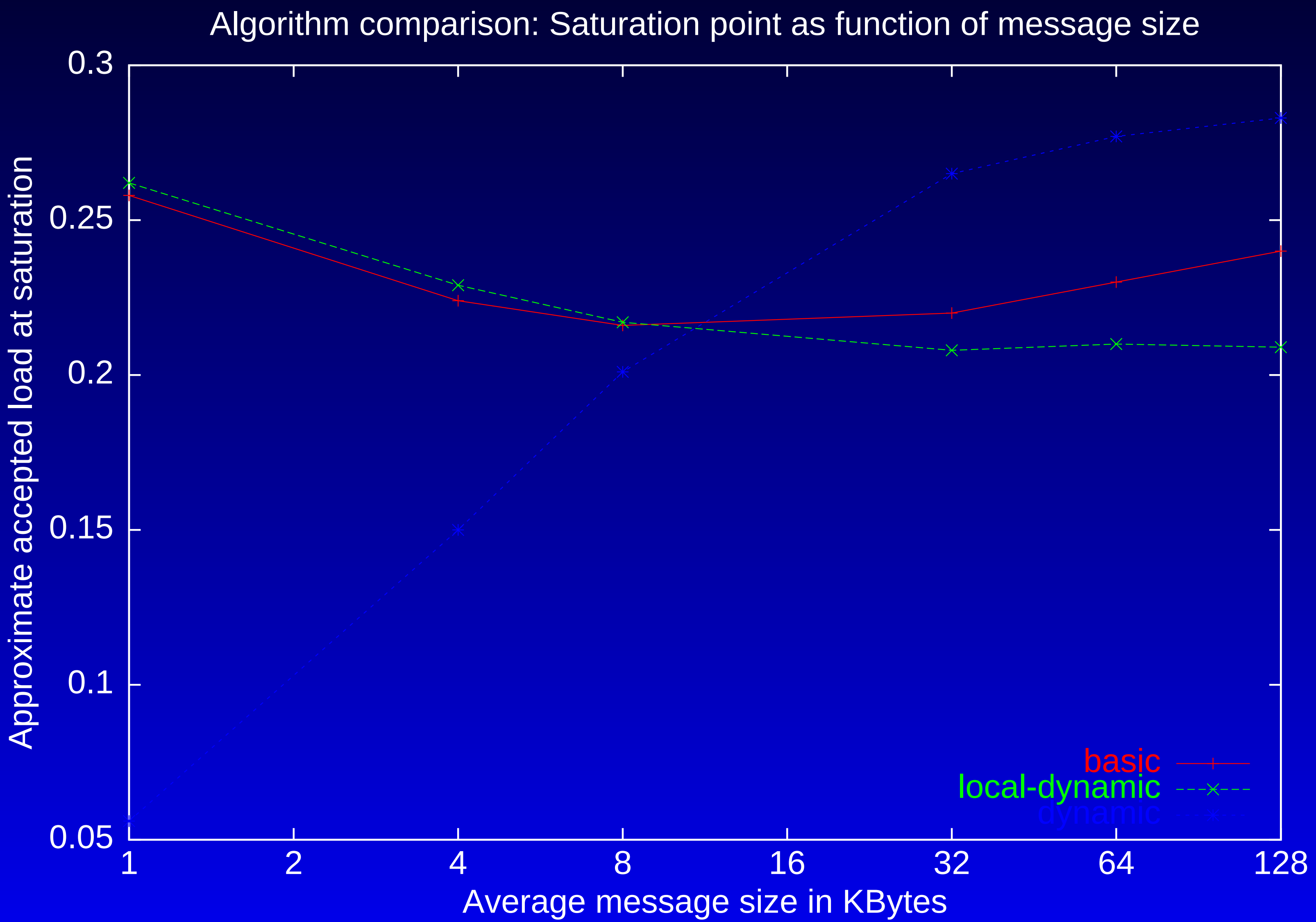
Results - Bandwidth vs Number of Rails



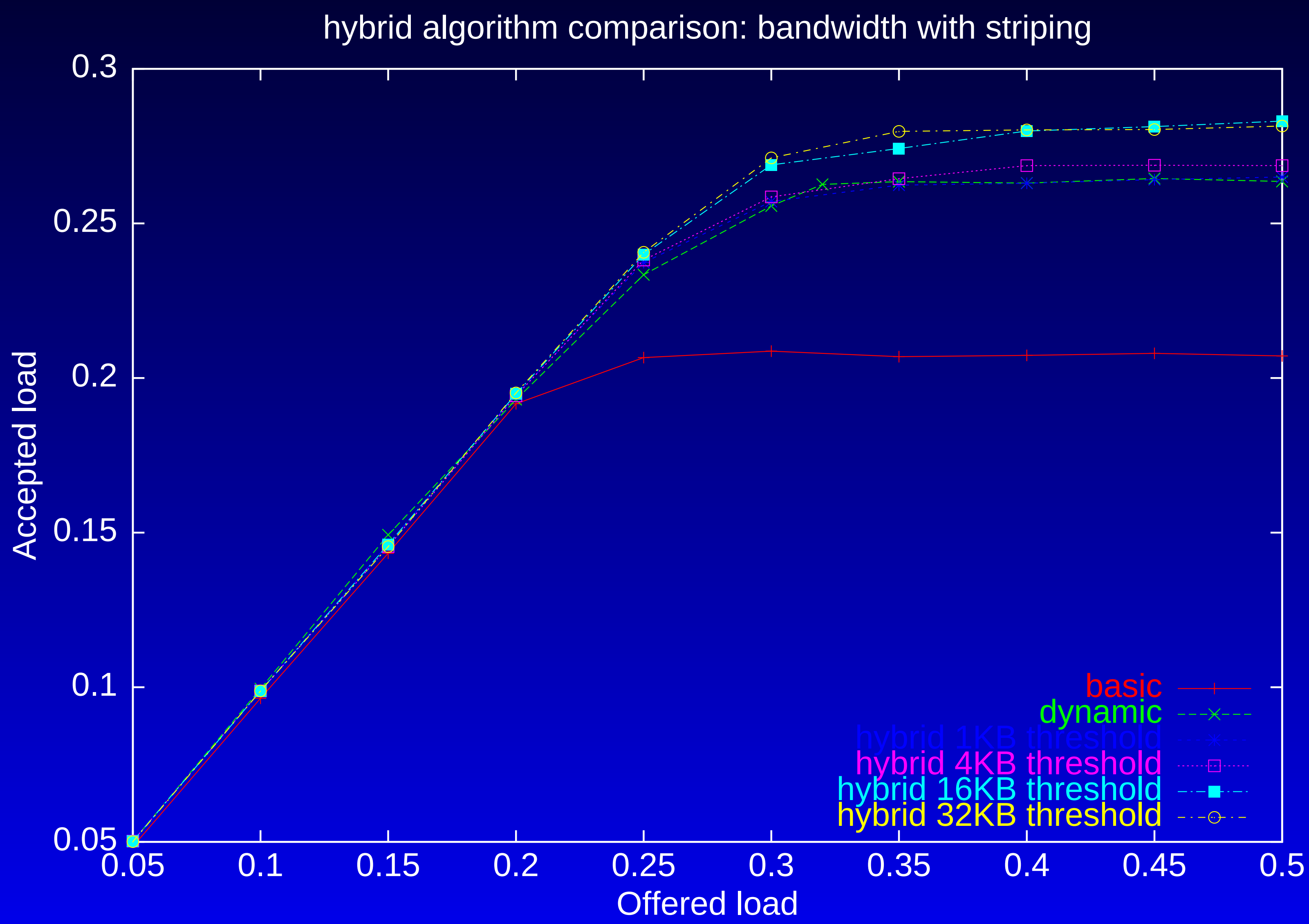
Results - Rail Scalability vs Message Size



Results - Saturation Points vs Message Size

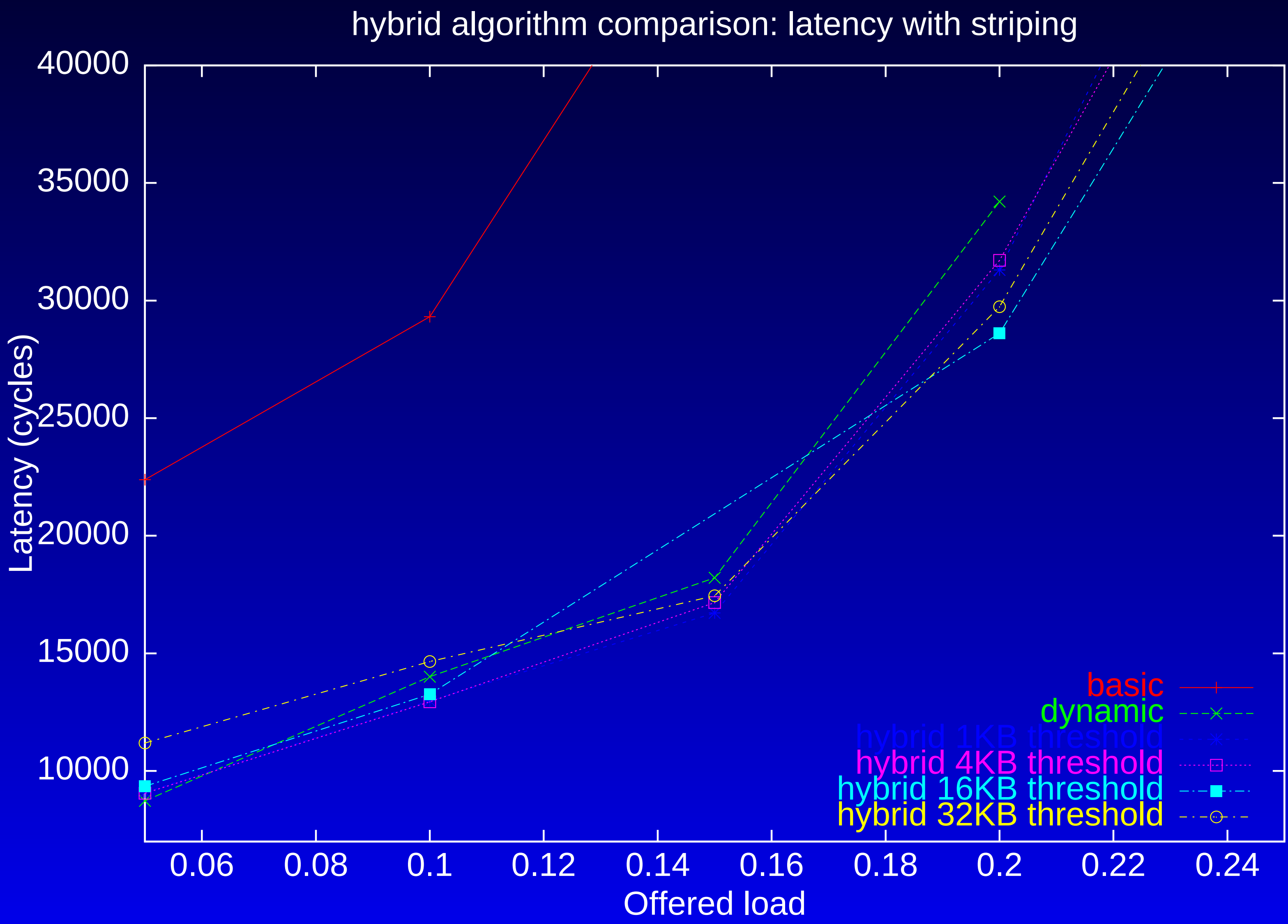


Results - Hybrid, Bandwidth with Striping



Los Alamos NATIONAL LABORATORY

Results - Hybrid, Latency with Striping



Conclusion

- We presented several algorithms to allocate multiple rails, static, dynamic and hybrid

Conclusion

- We presented several algorithms to allocate multiple rails, static, dynamic and hybrid
- The static algorithm requires a high number of rails, we proved a theoretical bound with constructive algorithm

Conclusion

- We presented several algorithms to allocate multiple rails, static, dynamic and hybrid
- The static algorithm requires a high number of rails, we proved a theoretical bound with constructive algorithm
- The dynamic algorithm performs relatively well for relatively large message sizes

Los Alamos
NATIONAL LABORATORY

Conclusion

- We presented several algorithms to allocate multiple rails, static, dynamic and hybrid
- The static algorithm requires a high number of rails, we proved a theoretical bound with constructive algorithm
- The dynamic algorithm performs relatively well for relatively large message sizes
- This algorithm is scalable with the number of rails

Conclusion

- We presented several algorithms to allocate multiple rails, static, dynamic and hybrid
- The static algorithm requires a high number of rails, we proved a theoretical bound with constructive algorithm
- The dynamic algorithm performs relatively well for relatively large message sizes
- This algorithm is scalable with the number of rails
- Incorporating protocol-free short message handling in the hybrid algorithm furtherly increases performance of the dynamic algorithm

Resources

- More information can be found at or

http://www.c3.lanl.gov/~fabrizio

- Or by sending an email to

fabrizio@lanl.gov or **eitanf@lanl.gov**