Hewlett Packard
Enterprise

Path-synchronous Performance Monitoring in HPC Interconnection Networks with Source-Code Attribution

Adarsh Yoga and Milind Chabbi

13 November 2017

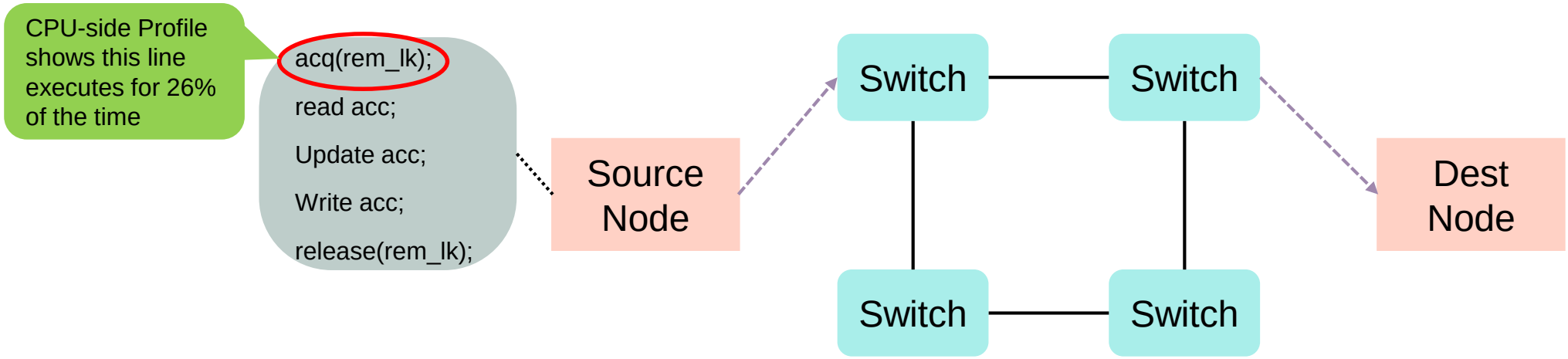
Motivation

- Inter-node data movement is a critical performance limiter

- Bottlenecks in interconnection networks can occur due to multiple factors
  - Application software design
  - Network components provisioning and health
  - Intra/Inter node job interference
  - Topology/routing algorithms

- Network remains a black-box from a developer's perspective

Bottleneck in NWChem



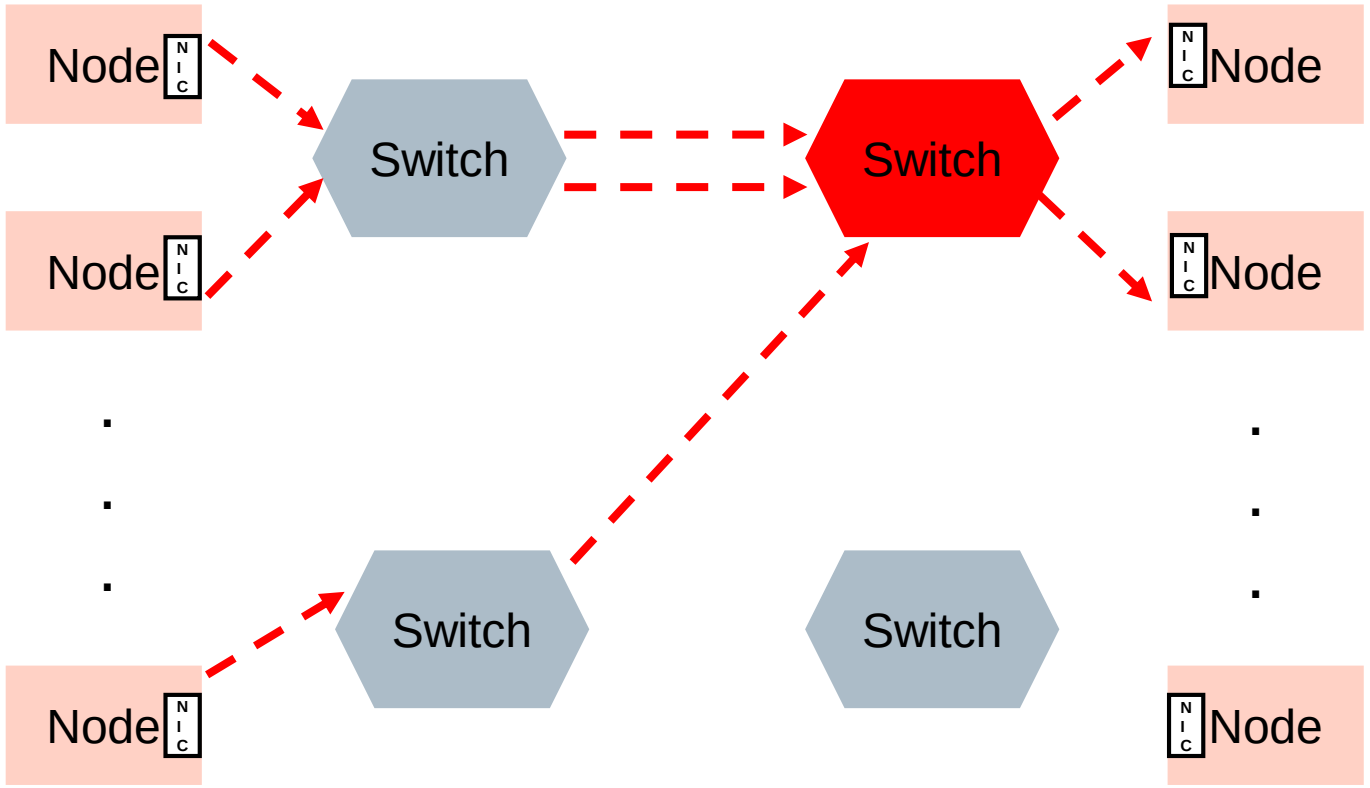
Cause not known!

- Is there load imbalance?
- Is it due to inefficient lock implementation?
- Is there interference for another job?
- Is it a network bandwidth problem?

Detailed Insight Necessary

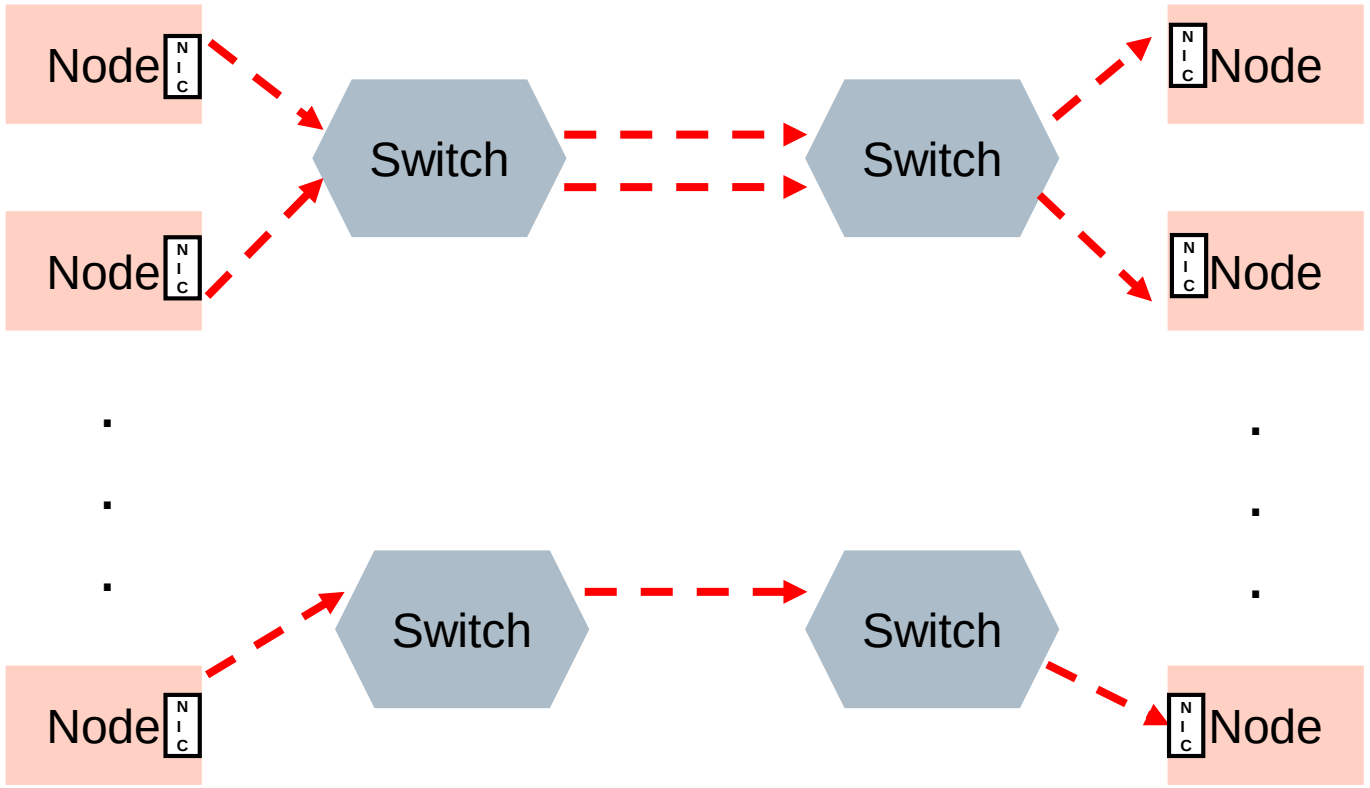
- State-of-the-art tools monitor the performance only within a node

- Detailed insight into the data movement and consequence on application source code necessary



Detailed Insight Necessary

- State-of-the-art tools monitor the performance only within a node
- Detailed insight into the data movement and consequence on application source code necessary
- Many performance problems can be fixed by simple refactor of application source code



Our Work

Goal: Identify performance problems that occur in the network and correlate to application source code.

Approach:

- Track the movement of packets through the network
- Collect performance metrics about the packets at every step
- Attribute metrics to application source code
  - Aid application developer with rich visualization and automatic data ingestion

Overview

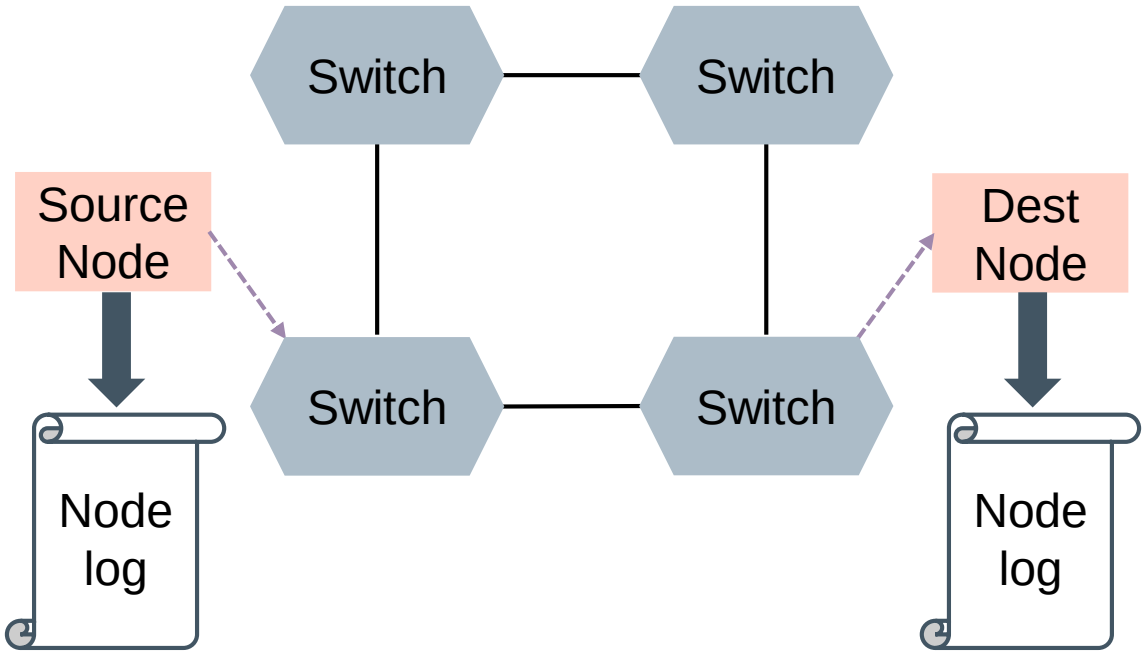

Input MPI program
Performance Stats Collection
Path Construction
Performance Visualization
Output graphs

Performance Stats Collection

- Tracking all messages/packets is expensive
  - Use sampling to randomly/smartly select messages to monitor
- A single bit (PM bit) in packet header used to track packets that are selected to be monitored (**marked packet**)
- Advantages:
  - Unsynchronized data collection in concurrent autonomous many-component systems
  - Hop-by-hop path synchronous metrics

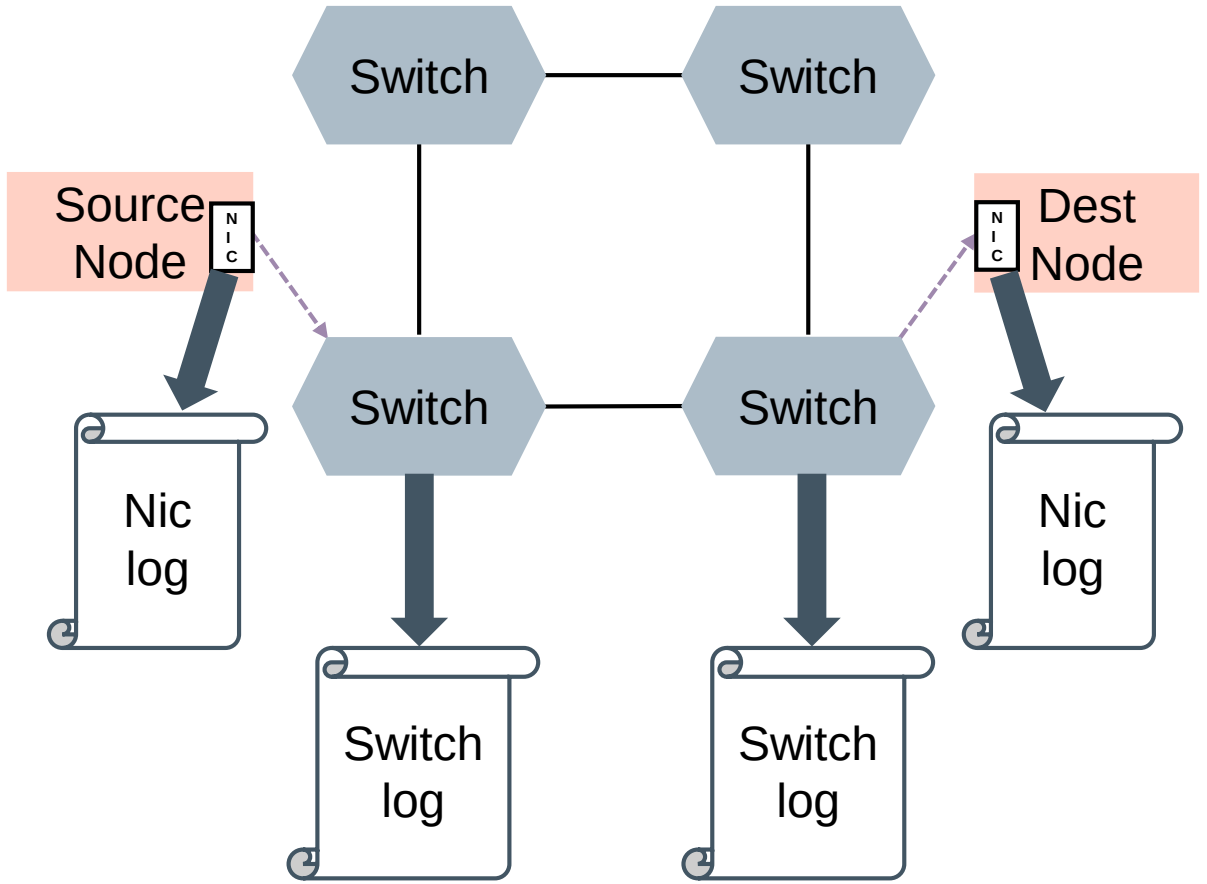
Node-side Stats Collection

- Software profiler selects a message at random and marks it to be tracked
- Both source and destination nodes collect stats for a marked message
- **Stats**: source node id, destination node id, message id, departure time/arrival time, application side calling context



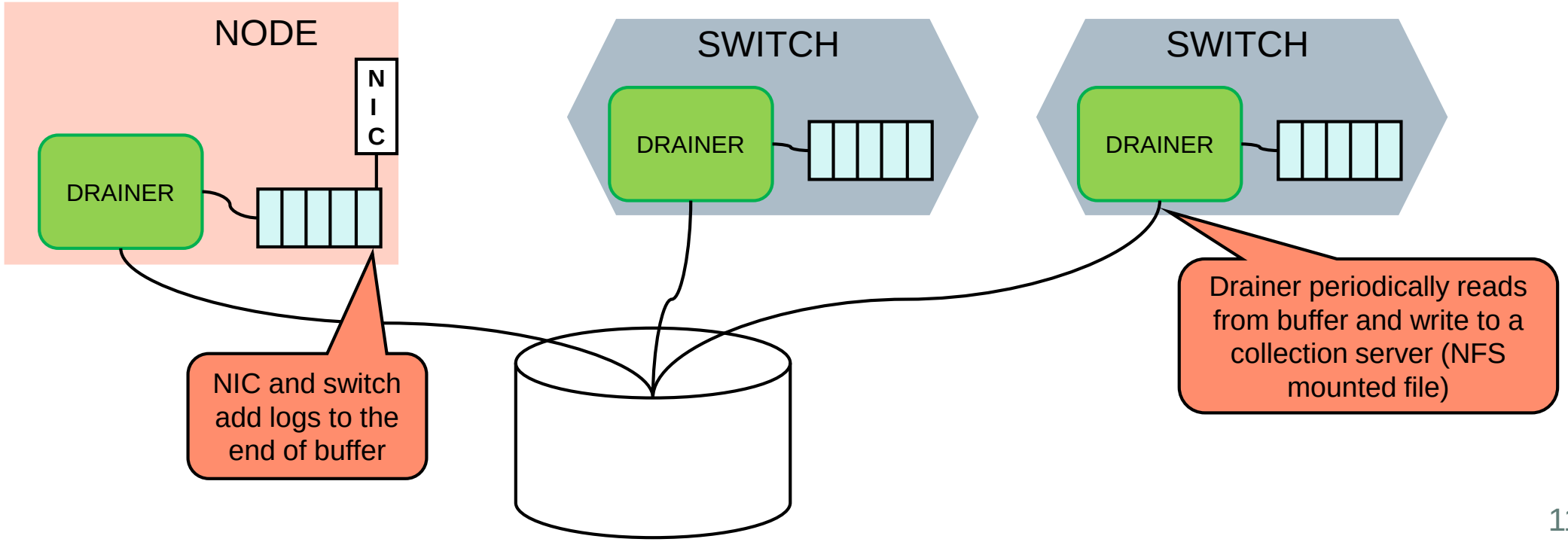
NIC and Network-side Stats

- For a marked message, NIC hardware selects one among N packets and sets PM bit in header
  - Could select more than one packet
- NIC and switch collect stats for the packet if PM bit set
- **Stats**: output port, arrival time, departure time, buffer size, credits



Hardware Extensions

- Goal is to minimize time spent in logging performance stats at NIC and Switch
- Log to a on-chip circular bounded buffer and a independent hardware component periodically drains logs from the buffer



Hewlett Packard Enterprise

Implementation – SST/Macro

- Simulator for large-scale interconnection networks
- Emphasize coarse-grain approximations over accuracy
- Pros:
  - Modular design; can be easily extended with additional network components
  - Perform analysis of varying network design parameters like machine models, packet flow models and topologies
  - Model realistic applications using *Skeletons* – MPI communication with limited computation

SST/Macro for Stats Collection

- Extended to *Node* module implementation to
  - randomly select messages to monitor
  - Log node side performance statistics
- Extended the Nic and the Switch module to collect performance statistics and write to the buffers
- Implemented the *Drainer* as sub-component of the *Node* and *Switch* module and added buffers to hold the logs

Output: Set of log files containing performance statistics for monitored packets

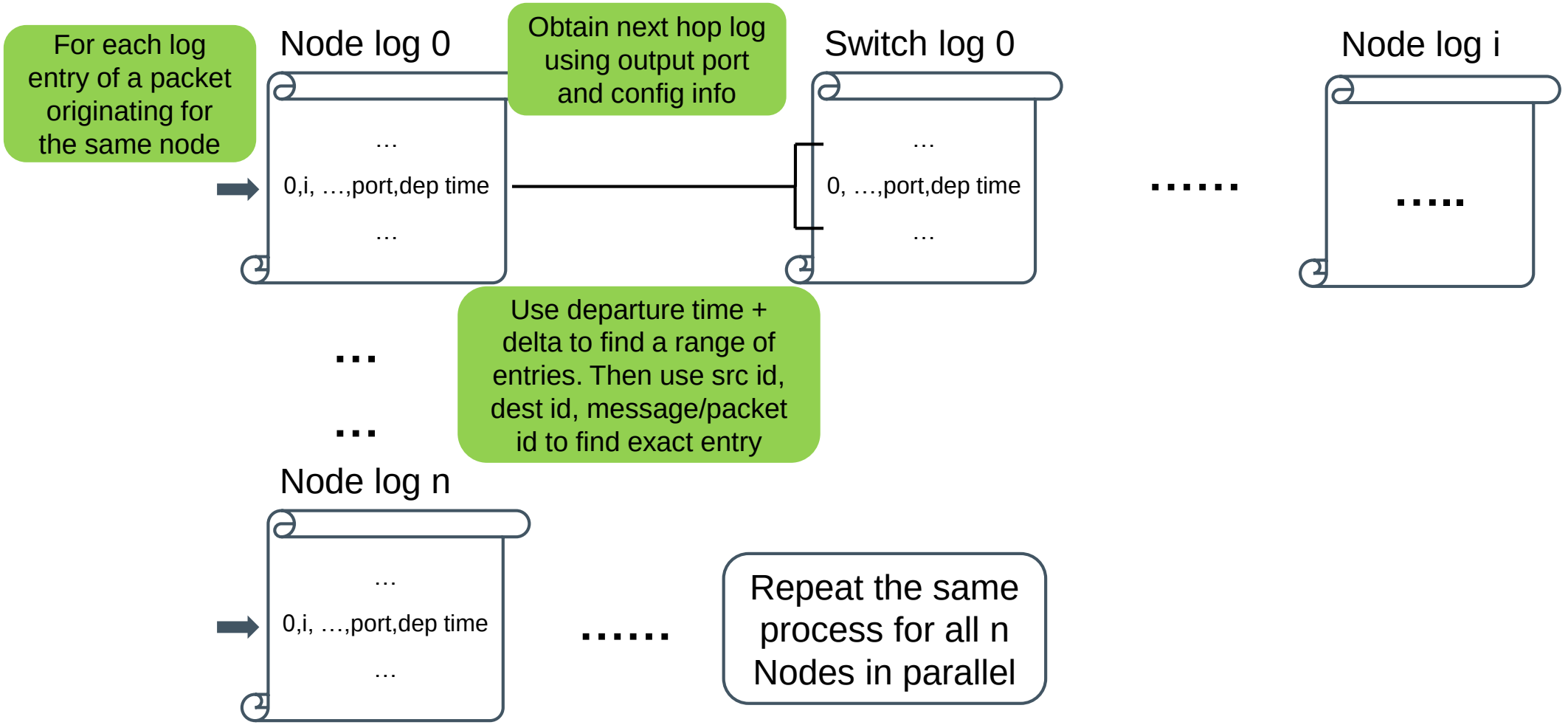
Overview


Input MPI program
Performance Stats Collection
Path Construction
Performance Visualization
Output graphs

Path Construction

- A software tool to reconstruct path taken by every marked packet
- Compute per-component performance statistics (e.g. delay) at each hop in the path
- Complete calling context with source code attribution at the end points

Path Construction Algorithm



Overview


Input MPI program
Performance Stats Collection
Path Construction
Performance Visualization
Output graphs

Performance Visualization

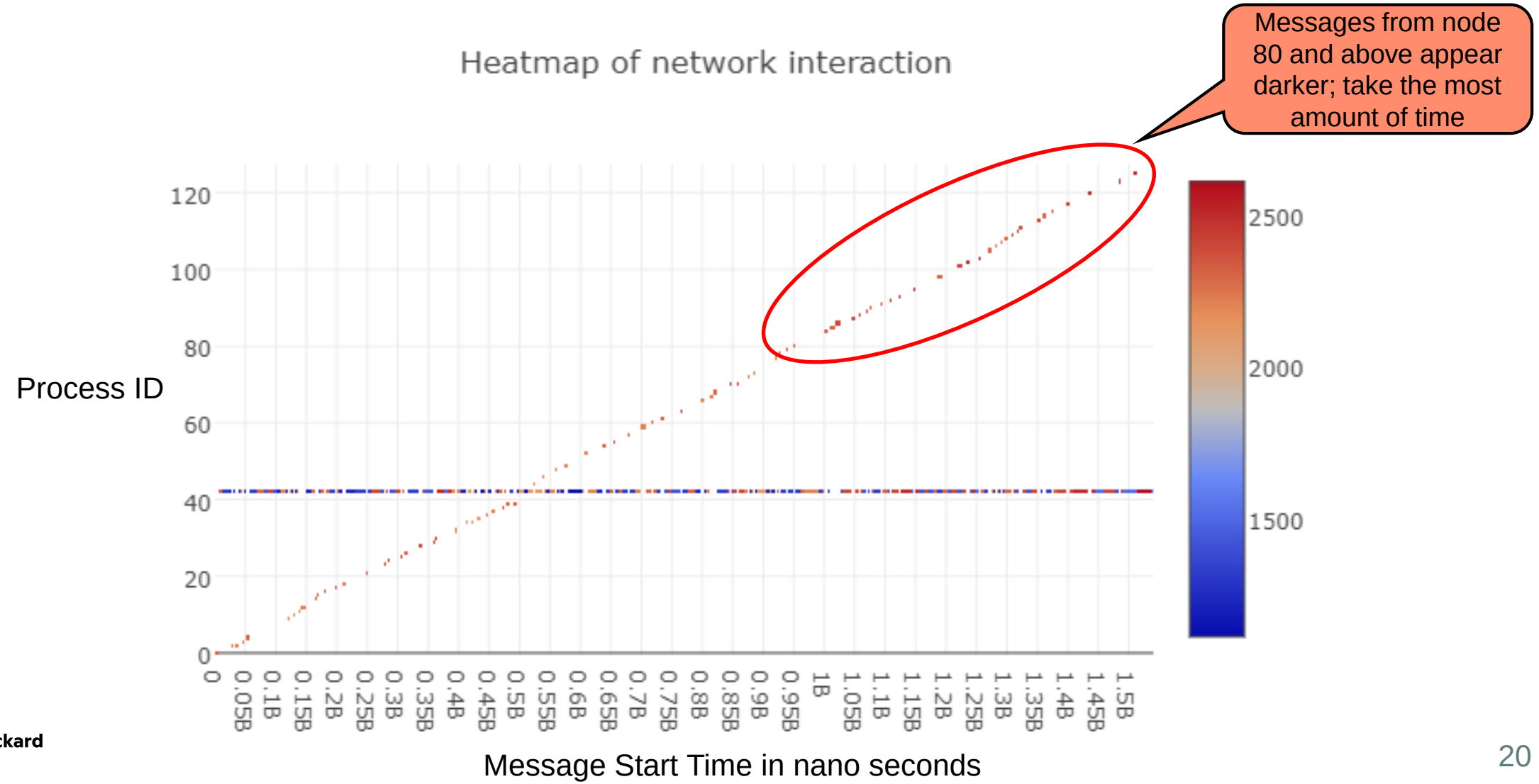
- Idea is to identify messages that took the most amount of time
  - Identify a pattern among the messages, e.g. all the slow messages happened in a small time window
  - Identify patterns for e.g. the slow messages were due to a bottleneck in a particular component
- Generate a heatmap to visualize entire network interaction
- Per-node stacked bar graphs to identify delays in specific components

Evaluation

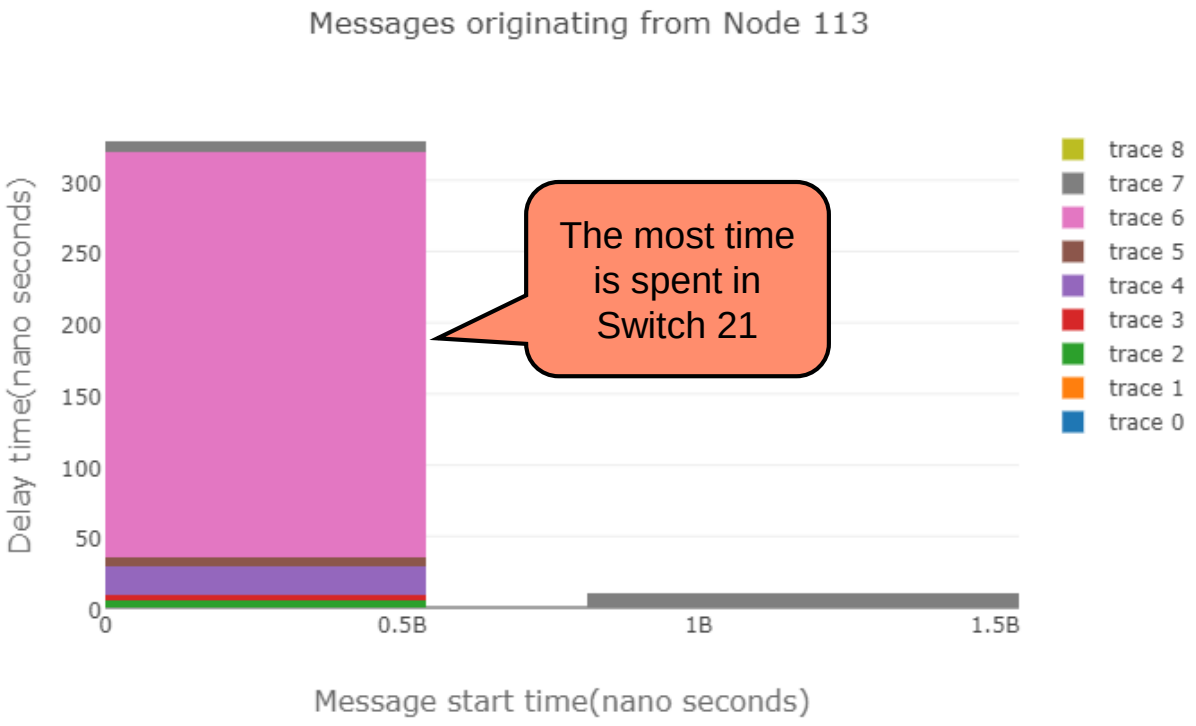
- Environment: 4 socket, 60 core Intel Xeon machine with 1 TB DRAM
- Simulated NERSC Edison system with Dragonfly topology containing 5586 nodes
- Set of 3 MPI skeleton applications – Ncast, Broadcast and MultiApp

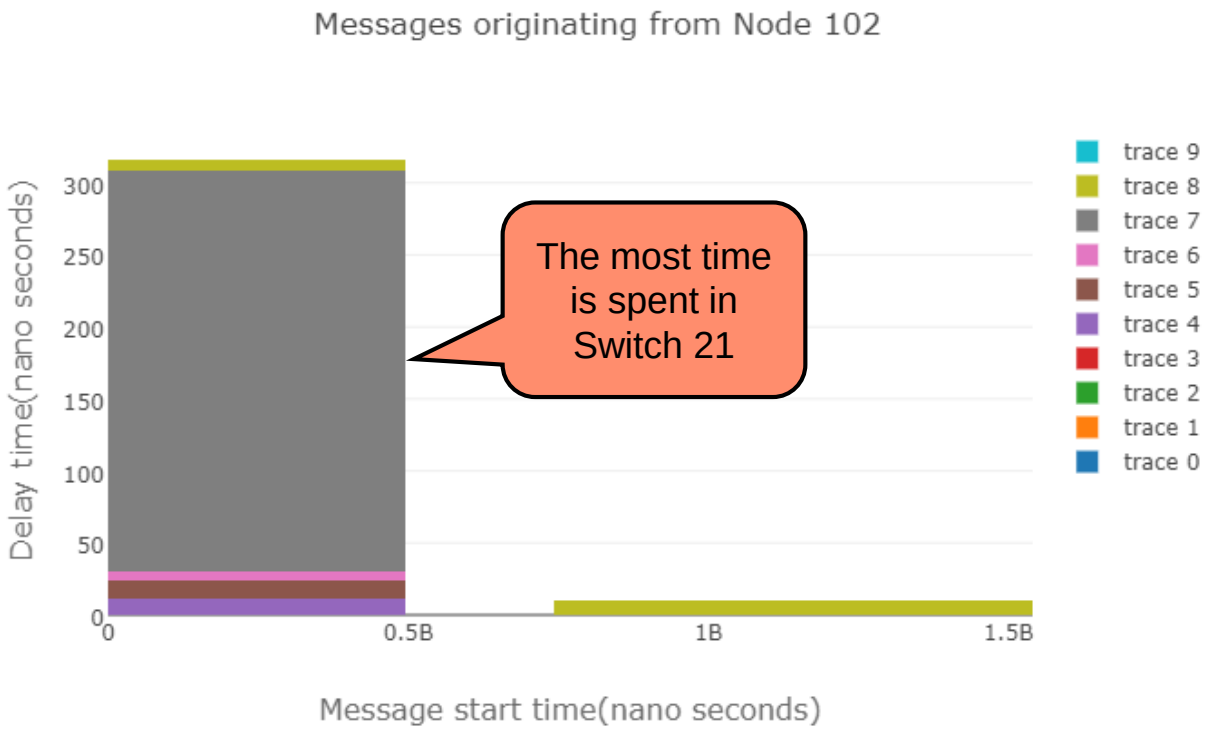
- Evaluation parameters:
  - Is our prototype effective in identifying performance bottlenecks in the network due to application source code?
  - Is the overhead of network low enough to be practical?

Heatmap of Ncast program



Per-node Bar Graph of Ncast





Hewlett Packard Enterprise

Efficiency of Network Monitoring

- Average simulation time overhead of 0.16% for all three applications with sampling rate of 1 in every 100 packets
- Average increase in wall clock time was 4.8% over base execution without our extensions
- Average size of log files generated was 61 MB

Conclusion

- Lightweight monitoring of network internals is essential to provide deeper insights into performance problems
- Solution: protocol extensions, hardware extensions and software solutions to provide deeper insights into performance problems
- Designed and prototyped an extendable performance analysis framework with broad applicability

Future Work

- In-depth assessment of this capability on more realistic and mixed workloads
- Working with vendors to include the capability in the hardware

Thank you!

Our tool is available at https://github.com/HewlettPackard/genz_tools_network_monitoring