Performance analysis of MCENoC, a Beneš-based predictable Network-on-Chip for EMC2 systems

Steve Kerrison, David May, Kerstin Eder

University of Bristol, United Kingdom

24th Jan 2017

HiPEAC 2017, EMC^2 workshop

The MCENoC is a **switching architecture** designed using the principles of **non-blocking Beneš networks**, combined with **formal verification** to provide a predictable interconnect for **multi-core mixed-criticality embedded systems**.

The MCENoC is a **switching architecture** designed using the principles of **non-blocking Beneš networks**, combined with **formal verification** to provide a predictable interconnect for **multi-core mixed-criticality embedded systems**.

How can we evaluate it?

The MCENoC is a **switching architecture** designed using the principles of **non-blocking Beneš networks**, combined with **formal verification** to provide a predictable interconnect for **multi-core mixed-criticality embedded systems**.

How can we evaluate it?

Acknowledgement

The research leading to these results has received funding from the ARTEMIS Joint Undertaking under grant agreement number 621429 (project EMC2).

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Motivation: NoC

Existing NoCs can deliver high core-count and high performance. Typical structures include meshes, hypercubes, toruses and rings. These pose some challenges:

Motivation: NoC

Existing NoCs can deliver high core-count and high performance. Typical structures include meshes, hypercubes, toruses and rings. These pose some challenges:

- Not all traffic patterns map well onto each structure.
- Congestion can cause reduction in throughput, or in the worst case, deadlock.

Motivation: NoC

Existing NoCs can deliver high core-count and high performance. Typical structures include meshes, hypercubes, toruses and rings. These pose some challenges:

- Not all traffic patterns map well onto each structure.
- Congestion can cause reduction in throughput, or in the worst case, deadlock.
- Varying costs of communicating between nodes.
- Very difficult to guarantee that a low-criticality communication cannot interfere with a high criticality one.

Even with predictable (cache-less, time-deterministic) cores, this is a problem.

Motivation: NoC

Existing NoCs can deliver high core-count and high performance. Typical structures include meshes, hypercubes, toruses and rings. These pose some challenges:

- Not all traffic patterns map well onto each structure.
- Congestion can cause reduction in throughput, or in the worst case, deadlock.
- Varying costs of communicating between nodes.
- Very difficult to guarantee that a low-criticality communication cannot interfere with a high criticality one.

Even with predictable (cache-less, time-deterministic) cores, this is a problem.

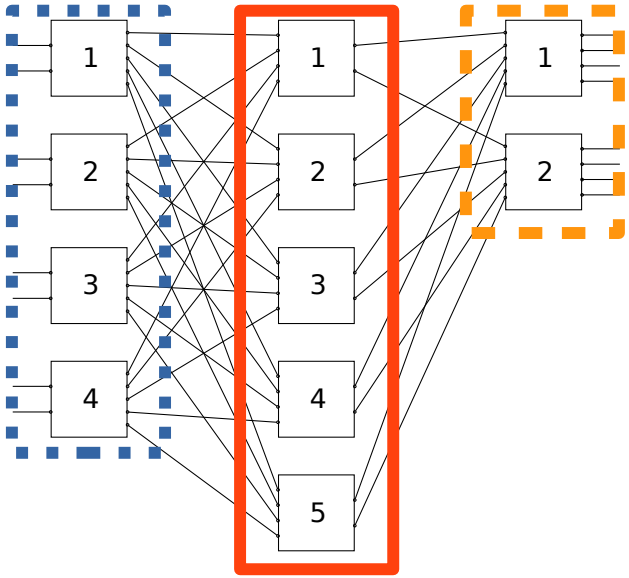
A predictable processor communicating over an unpredictable network is no longer a predictable processor!

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Beneš and Clos style networks

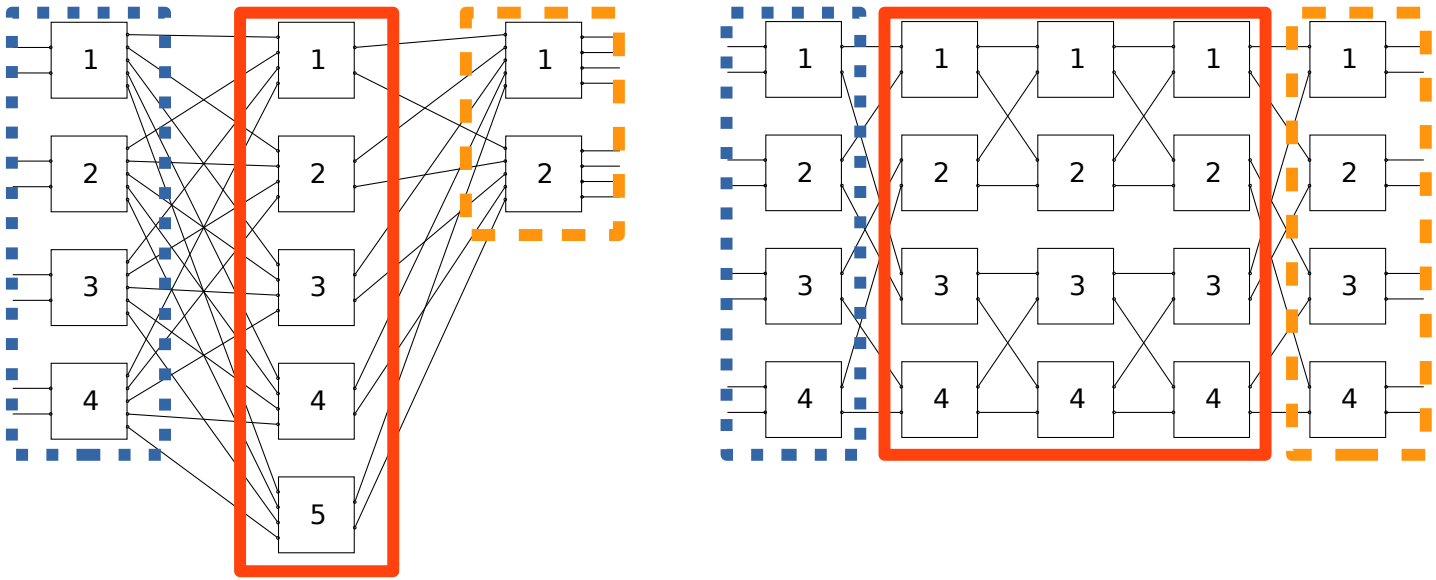
- Clos in 1952 [1], defines a three-stage network for telecoms.
- Beneš network defined by stages of 2-port switching elements [2].
- N–N two-party calls with no blocking.
- Translates well to VLSI [3].



(a) Clos

Beneš and Clos style networks

- Clos in 1952 [1], defines a three-stage network for telecoms.
- Beneš network defined by stages of 2-port switching elements [2].
- N–N two-party calls with no blocking.
- Translates well to VLSI [3].

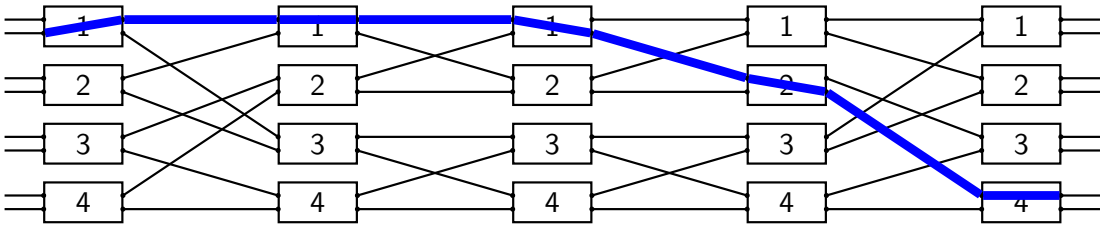


(a) Clos (b) Beneš

Figure: Clos / Beneš conceptual comparison

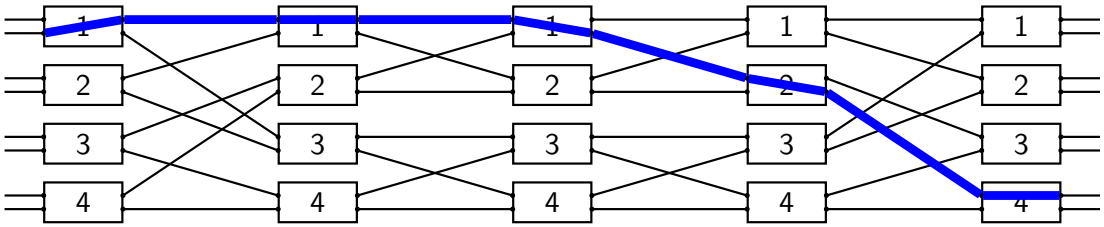
Network folding

- Consider left-side to be input, right to be output.
- Imagine the network folded in half, connecting node input/outputs.

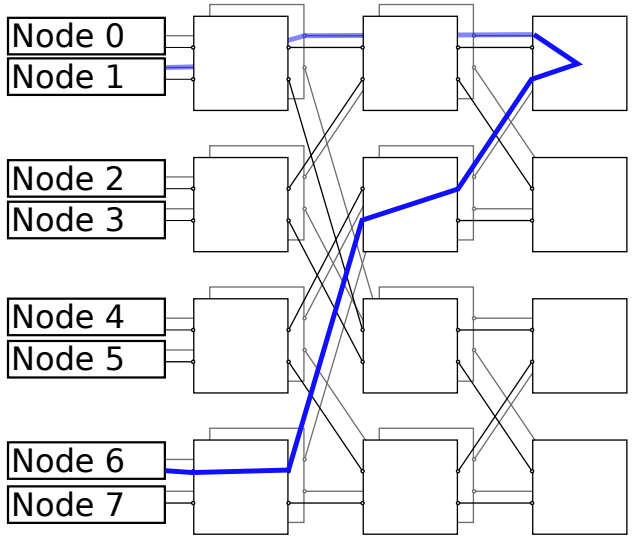


Network folding

- Consider left-side to be input, right to be output.
- Imagine the network folded in half, connecting node input/outputs.

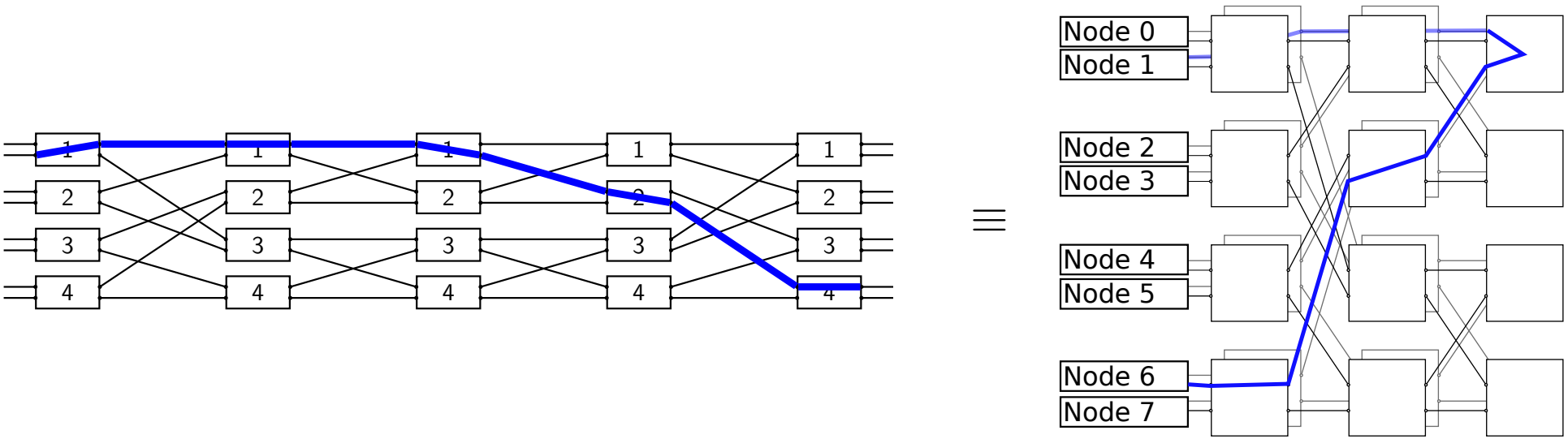


≡



Network folding

- Consider left-side to be input, right to be output.
- Imagine the network folded in half, connecting node input/outputs.



Of particular interest to us:

- **Statically predictable** latencies and contention scenarios.
- Use this style of network to replace mesh, ring, or hierarchical structures.
- The nature of the network gives us **guarantees at a software level** that have the potential to simplify mixed-criticality system certification.

Fully routed example

Eight concurrent communications:

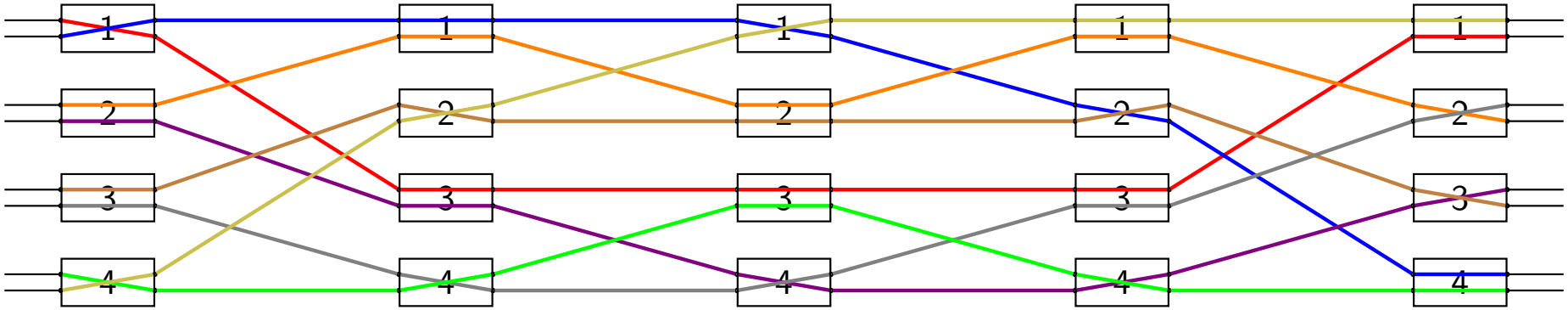


Figure: An eight-node network using two-port switching elements

Other topologies can be emulated with TDM, e.g. four stages for N, S, E, W of a 2 D mesh.

Partitioning example

The network can be partitioned into isolated subnetworks, which can only route within themselves.

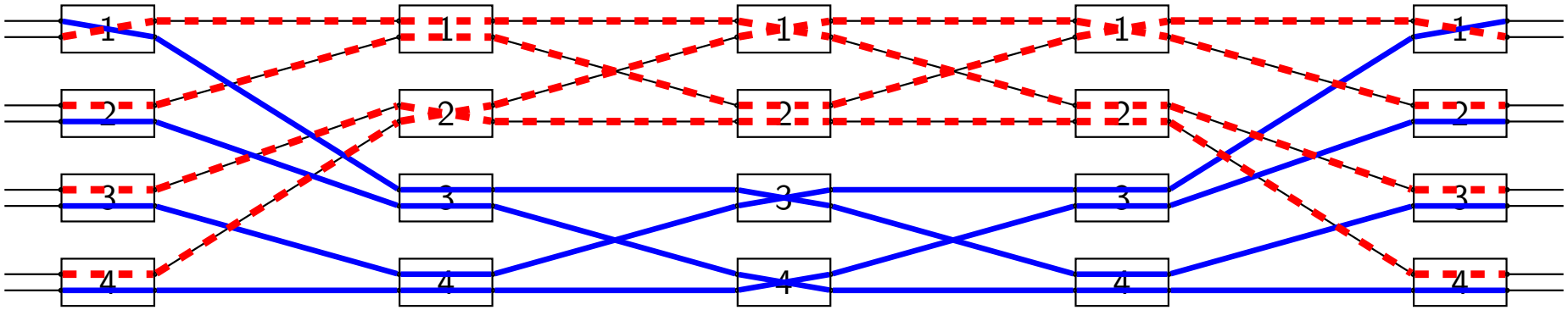


Figure: 4-way sub-networks created from an 8-way system.

Partitioning example

The network can be partitioned into isolated subnetworks, which can only route within themselves.

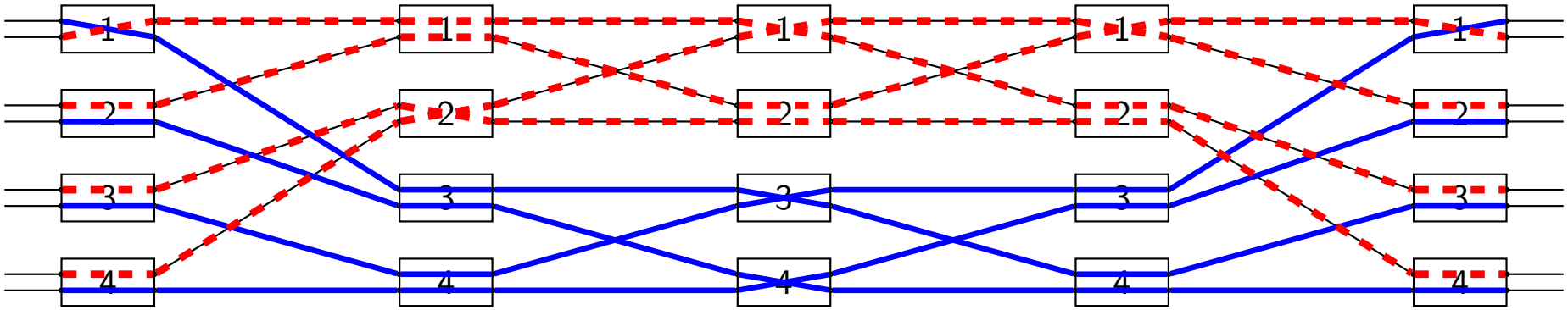


Figure: 4-way sub-networks created from an 8-way system.

- Two sub-networks of nodes: $\{1, 2, 4, 6\}$ and $\{0, 3, 5, 7\}$.
- Each sub-network depicted by line style.
- Connections whithin node groups determined by middle three stages.
- This has benefits for strictly isolated mixed-criticality scenarios.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

MCENoC: Our implementation of a predictable, FV'd NoC

Key features:

- Specification at three levels: switching element, network & system.

MCENoC: Our implementation of a predictable, FV'd NoC

Key features:

- Specification at three levels: switching element, network & system.
- Property definitions for each specification level.

MCENoC: Our implementation of a predictable, FV'd NoC

Key features:

- Specification at three levels: switching element, network & system.
- Property definitions for each specification level.

MCENoC: Our implementation of a predictable, FV'd NoC

Key features:

- Specification at three levels: switching element, network & system.
- Property definitions for each specification level.
- Beneš structure of switching elements, with configurable element size (2, 4, 8, ... ports).

MCENoC: Our implementation of a predictable, FV'd NoC

Key features:

- Specification at three levels: switching element, network & system.
- Property definitions for each specification level.
- Beneš structure of switching elements, with configurable element size (2, 4, 8, ... ports).
- Formal verification and switch and network properties.

MCENoC: Our implementation of a predictable, FV'd NoC

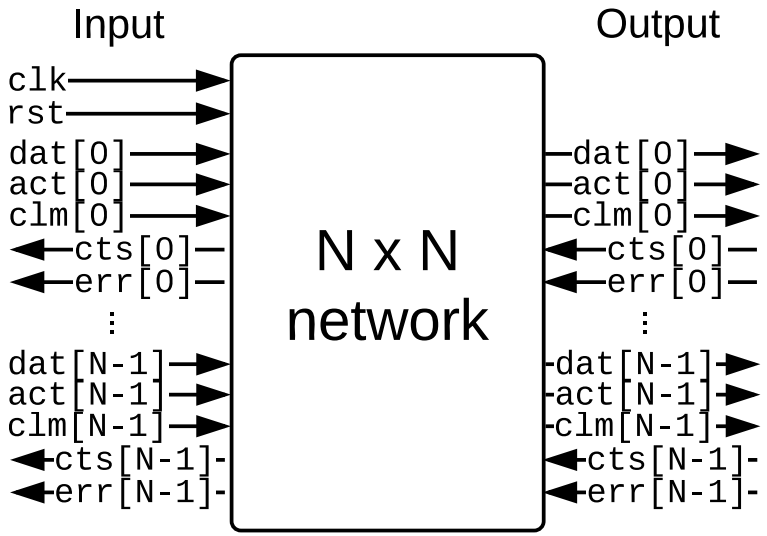
Key features:

- Specification at three levels: switching element, network & system.
- Property definitions for each specification level.
- Beneš structure of switching elements, with configurable element size (2, 4, 8, ... ports).
- Formal verification and switch and network properties.
- Integration with processor: 32 RISC V cores on FPGA.

More details in MCSoC'16 paper [4].

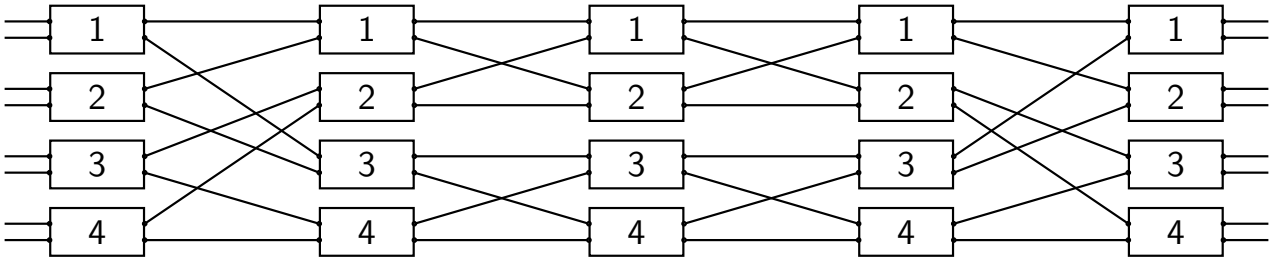
Top level view

- Claim, activity strobe and data signals: `clm`, `act`, `dat`.
- Flow control & error signals: `cts`, `err`.
- *N* ports, equidistant.
- In-band route configuration. Upon claiming a port, first bits configure the switches.



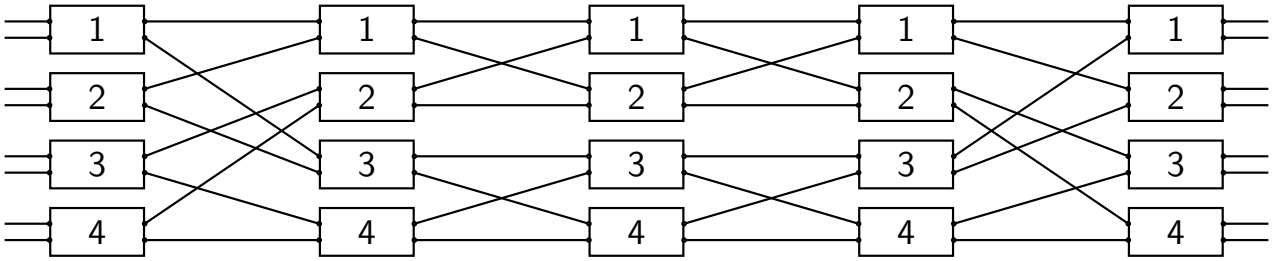
Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.



Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

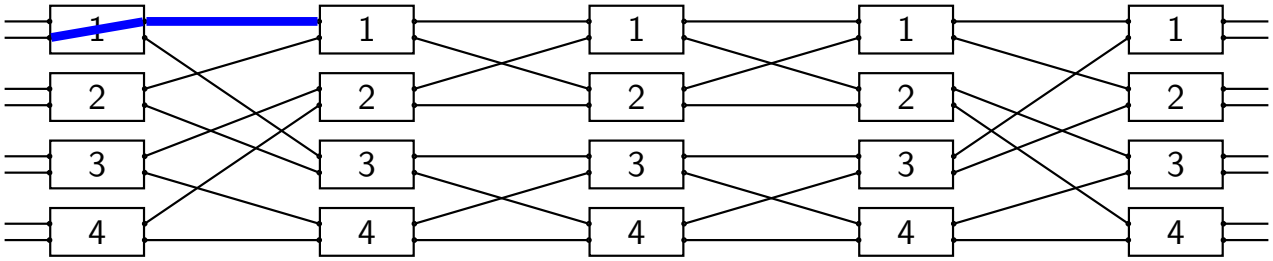


Connecting node 1 with node 6

- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

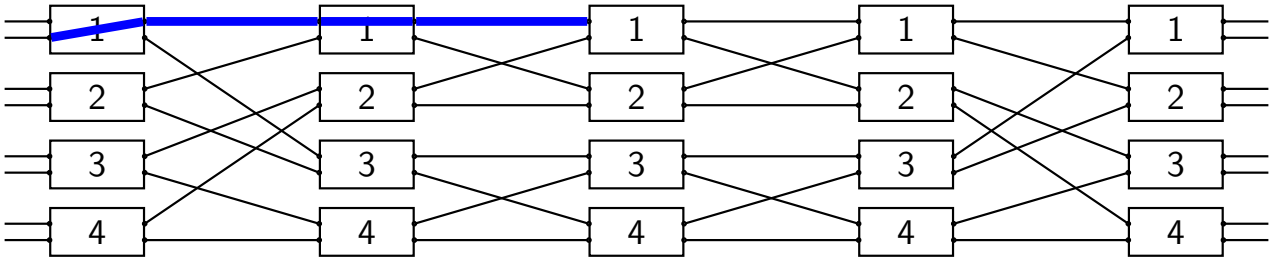


Connecting node 1 with node 6

- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

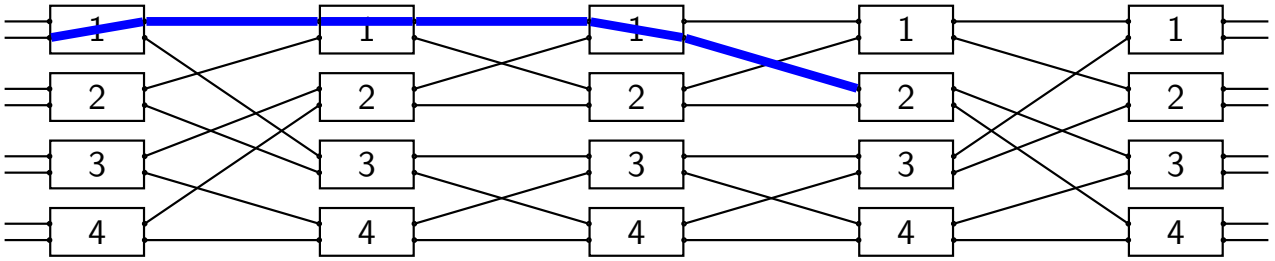


Connecting node 1 with node 6

- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

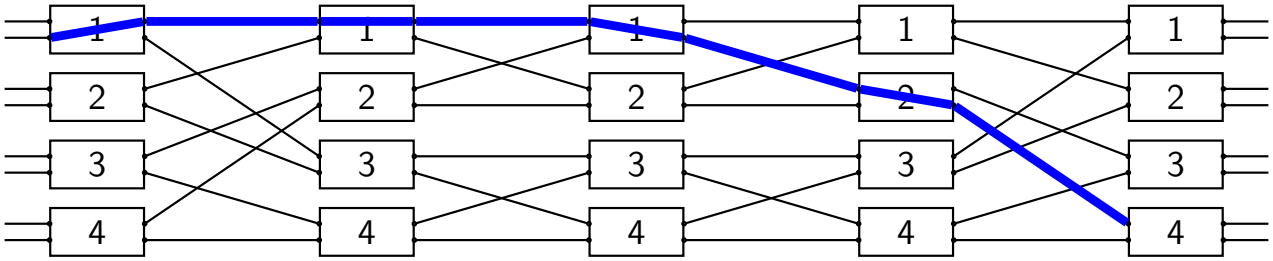


Connecting node 1 with node 6

- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

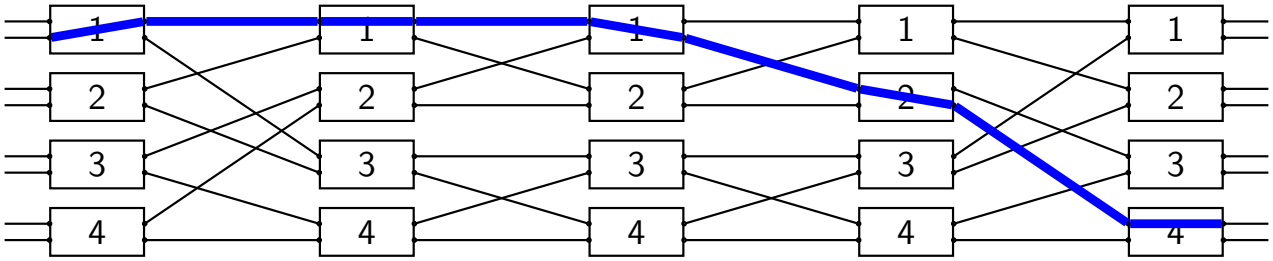


Connecting node 1 with node 6

- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching elements of size 2^b, allowing us to explore the best element size for scaling purposes. Routing is equivalent for an $N \times N$ network, regardless of b; the number of header bits is always the same.

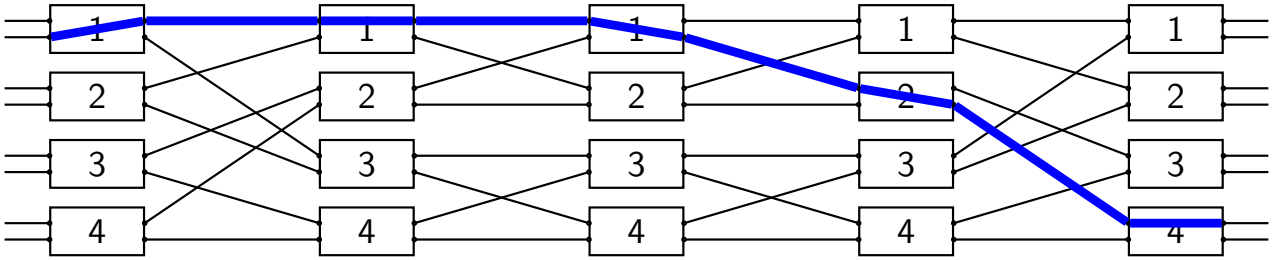


Connecting node 1 with node 6

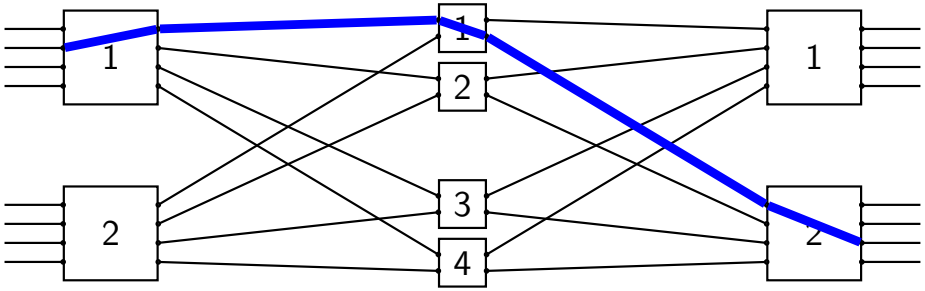
- When interface is claimed, five header bits clocked in.
- Sequence: `0, 0, 1, 1, 0`.
- Each bit performs in-band configuration of the switch.

Our implementation

Switching element sizes can be changed, affecting latency, but not affecting header bits.



(h) Two-port switching elements.



(i) Four-port and two-port elements.

Figure: Example routes in two equivalent eight-node implementations.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

What elements of performance do we care about?

- We claim latency is fixed... is it really?
- How much throughput can be achieved?
- How does the system scale in terms of throughput and logic utilisation?
- What burdens are shifted into other layers of the system stack?
- How to compare to other devices?

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Formally proving properties

We use a combination of SystemVerilog Assertion language, and a formal verification tool.

- SVA properties formalise the specification. This helps us to verify that the specification is **correct and unambiguous**.
- Assertions can be raised in simulation, or explored using a formal verification tool that demonstrates **no counter-example exists**.
- This also exposes bugs that are due to uncaught **specification deficiencies**.

This is not a silver bullet.

Formally proving properties

We use a combination of SystemVerilog Assertion language, and a formal verification tool.

- SVA properties formalise the specification. This helps us to verify that the specification is **correct and unambiguous**.
- Assertions can be raised in simulation, or explored using a formal verification tool that demonstrates **no counter-example exists**.
- This also exposes bugs that are due to uncaught **specification deficiencies**.

This is not a silver bullet.

- Large properties that span many clock cycles can be very slow to prove due to **huge state space**.

Proof performance

- Eleven property definitions.
- Multiple instantiations of some properties.
- Increases with larger switches (32-port switch examines 1216 property assertions).

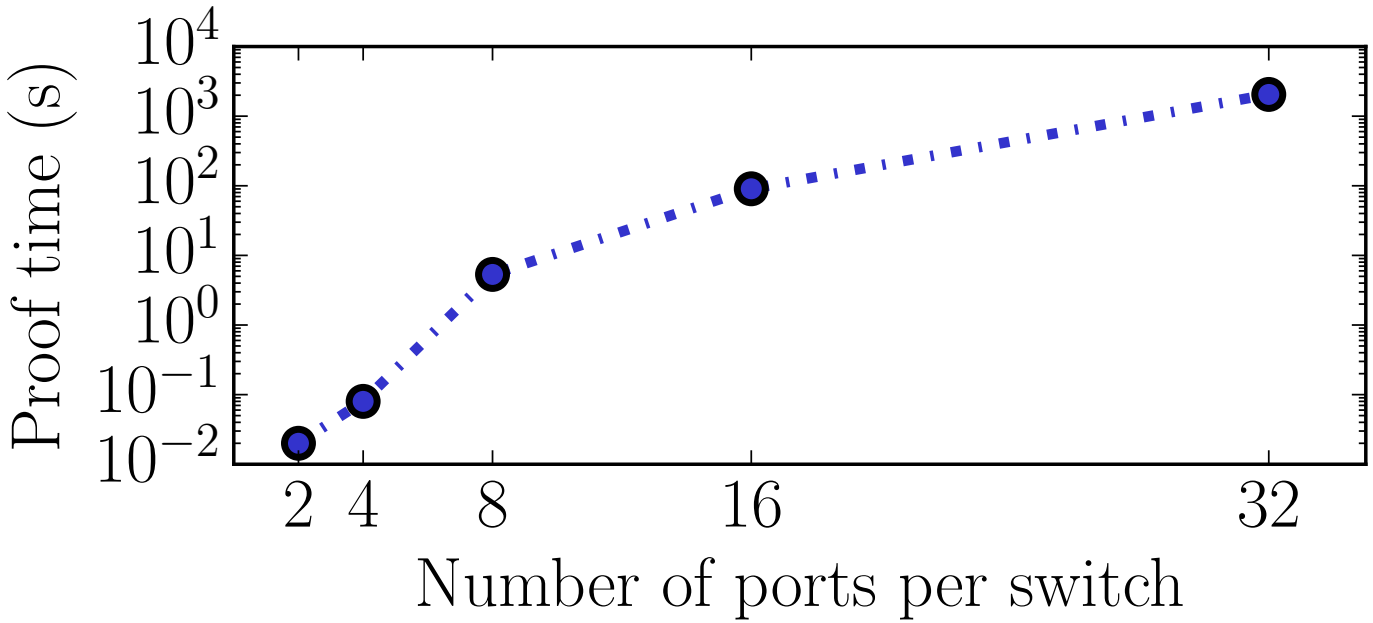


Figure: Proof time for a single switching element (core).

Proof performance

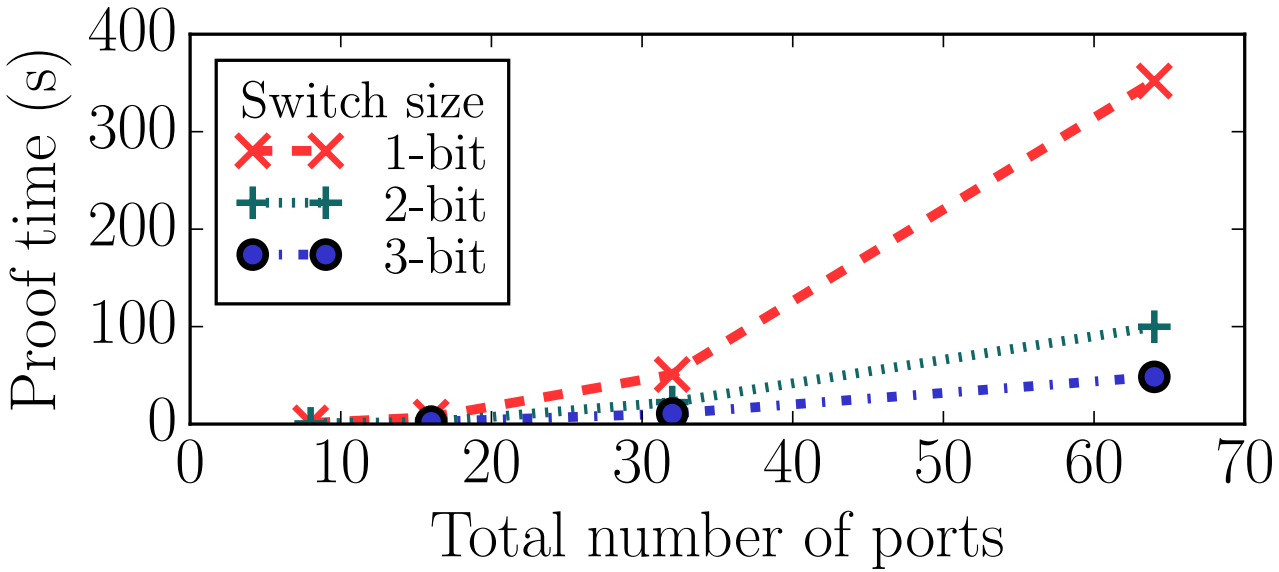


Figure: Proof time for networks of varying size & switching element size.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Number of stages

For N nodes, there will be S stages.

Standard Beneš

$$S = 2\log_2(N) - 1 \tag{1}$$

Larger switch sizes

$$S = 2\log_B(N) - 1, \quad \text{if} \quad \exists x \in \mathbb{N} \mid B^x = N. \tag{2}$$

Number of stages

For N nodes, there will be S stages.

Standard Beneš

$$S = 2\log_2(N) - 1 \tag{1}$$

Larger switch sizes

$$S = 2\log_B(N) - 1, \quad \text{if} \quad \exists x \in \mathbb{N} \mid B^x = N. \tag{2}$$

Latency scales logarithmically with number of nodes. Buffers at edge of network, software stack will add extra latency.

Practical performance

Synthesis

- Eight-node system with four-port switching elements synthesizes to a Kintex-7 at 364 MHz.
- Single RISC-V core "PicoRV32" achieves similar[a].
- Multiple PicoRV32 cores further constrain clock.
- **Network is not the bottleneck**.

[a] https://github.com/cliffordwolf/picorv32

Throughput

- Eight-node: 2.9 Gbit/s bisection bandwidth.
- 32-node: 11.6 Gbit/s bisection bandwidth.
- Per-node, per-bit-width, 32-node MCENoC achieves 725 Mbit/s vs. 64-node Epiphany 199 Mbit/s.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Scheduling

- Communications must be scheduled into TDM phases to avoid contention.
- If a node must communicate with M other nodes, $>= M$ phases will be needed.
- Determining routes used in a single phase is a Matrix permutation problem [5].

Scheduling

- Communications must be scheduled into TDM phases to avoid contention.
- If a node must communicate with M other nodes, $>= M$ phases will be needed.
- Determining routes used in a single phase is a Matrix permutation problem [5].
- If bandwidth requirements varied, communications may be subdivided into further phases.
- Use of TDM with routes motivates a rapid teardown/re-build time.

Scheduling

- Communications must be scheduled into TDM phases to avoid contention.
- If a node must communicate with M other nodes, $>= M$ phases will be needed.
- Determining routes used in a single phase is a Matrix permutation problem [5].
- If bandwidth requirements varied, communications may be subdivided into further phases.
- Use of TDM with routes motivates a rapid teardown/re-build time.

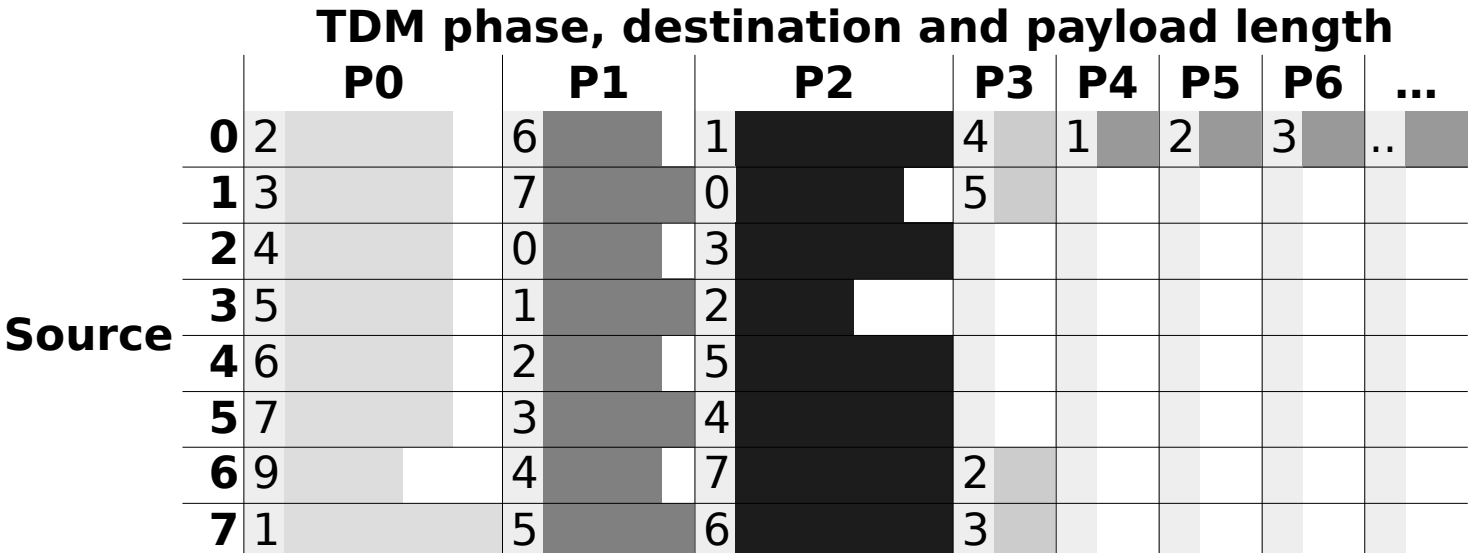


Figure: Examples of communication phases, revealing overheads and slack.

Permutation performance

Evaluation for a schedule where $N = M$. Payload efficiency indicates ratio fo data to header.

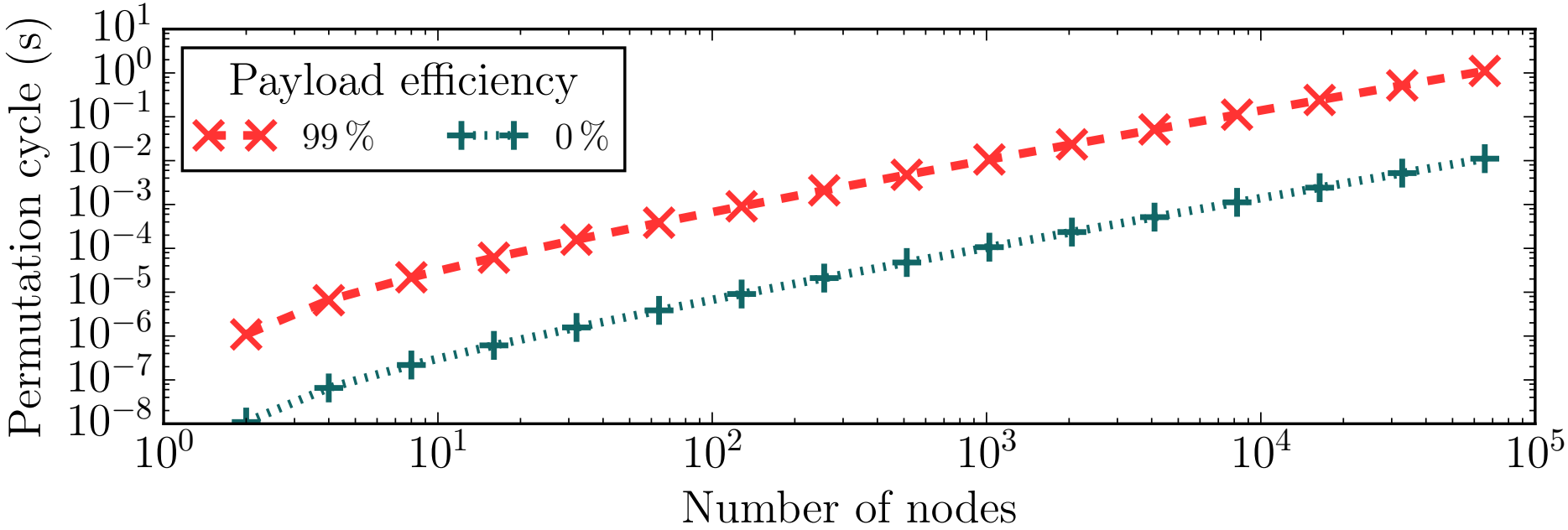


Figure: Time taken to perform N permutations with 364 Mbit/s/port, considering a desired payload efficiency and 2-port switches.

Permutation performance

Evaluation for a schedule where $N = M$. Payload efficiency indicates ratio fo data to header.

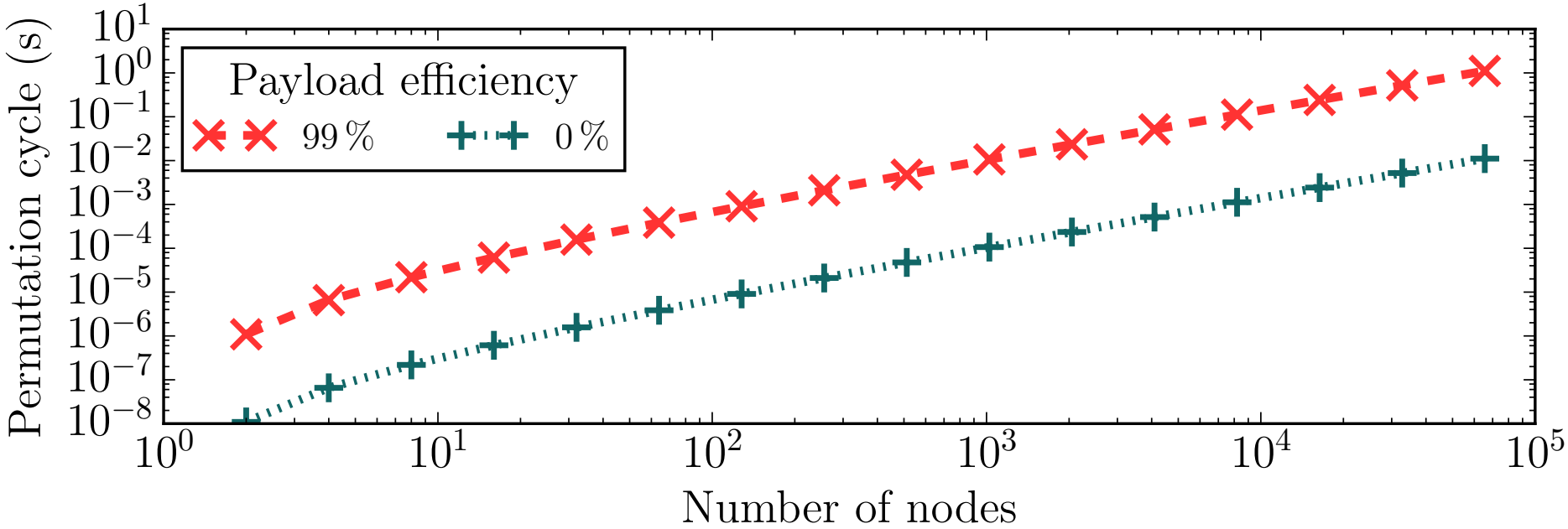


Figure: Time taken to perform N permutations with 364 Mbit/s/port, considering a desired payload efficiency and 2-port switches.

This can improve with wider data-path and larger switching elements.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

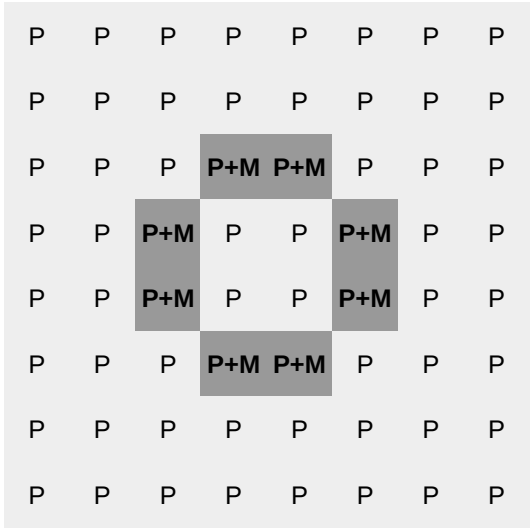
Software benchmarking

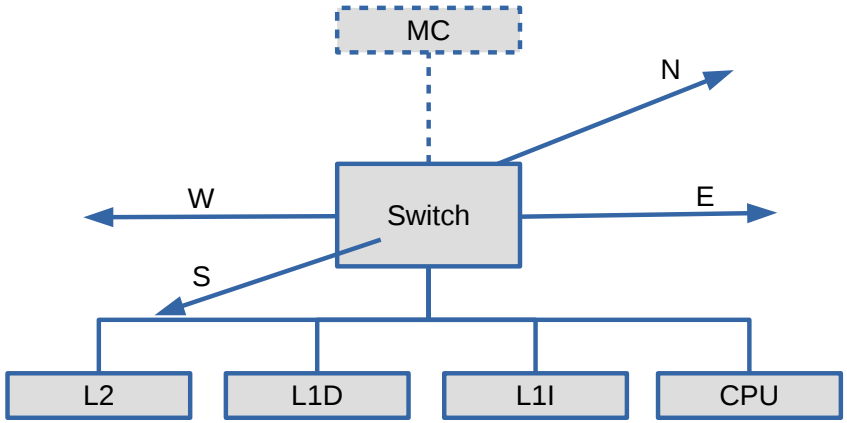
- Challenge: construct use case, or use existing benchmarks.
- Intent: Do both!

First. . .

- Benchmark MCENoC against standard benchmarks.
- Use `Netrace` approach to avoid implementing entire SW & peripheral stack [6].

py-netrace





- Implementation in Python.
- Uses Parsec benchmark traces[a] (Alpha simulation in {Ge}M5).
- 64-core, 2 GHz system, L1 cache, 64-banks L2 cache, 8 mem controllers.
- Mesh: 64 switches. MCs in diamond configuration.
- MCENoC: Doesn't matter!

[a] `http://www.cs.utexas.edu/~netrace/`

Early results

Comparing a single-cycle uncontended network, mesh and MCENoC:

- Mesh network 0.73% slower than uncontended.
- MCENoC between 0.45% and 2.10% slower than uncontended, depending on TDM scheduling pessimism.

Considerations:

Early results

Comparing a single-cycle uncontended network, mesh and MCENoC:

- Mesh network 0.73% slower than uncontended.
- MCENoC between 0.45% and 2.10% slower than uncontended, depending on TDM scheduling pessimism.

Considerations:

- MCE^2 system not likely to contain caches.
- Traffic trace is from a non-deterministic environment: dynamic, not static.
- Not embedded benchmarks.
- Netrace itself introduces error.

In this presentation

- Review: Beneš and Clos networks.
- Implementation: The MCENoC design.
- Performance: A multidimensional problem.
  - Verification.
  - Link latency and throughput.
  - Hardware scaling.
  - Scheduling.
  - Software benchmarking.
- Next steps.

Next steps

- Complete evaluation of scenarios in netrace.
- SW stack for RISC-V.
- Explore EMC^2 scenarios on a fully implemented system.

Next steps

- Complete evaluation of scenarios in netrace.
- SW stack for RISC-V.
- Explore EMC^2 scenarios on a fully implemented system.

Potential future work

- Combining static and dynamic communication scheduling [7].
- Formal verification of the software (kernel).
- Fault injection & handling.

Thank you

Contact information

Steve Kerrison, David May & Kerstin Eder
firstname.lastname@bristol.ac.uk

References

[1] Charles Clos. "A Study of Non-Blocking Switching Networks". In: *Bell System Technical Journal* (1952), pp. 406–424. DOI: 10.1002/j.1538-7305.1953.tb01433.x.

[2] V. E. Beneš. "On Rearrangeable Three-Stage Connecting Networks". In: *Bell System Technical Journal* 41.5 (Sept. 1962), pp. 1481–1492. DOI: 10.1002/j.1538-7305.1962.tb03990.x.

[3] Yikun Jiang and Mei Yang. "On circuit design of on-chip non-blocking interconnection networks". In: *2014 27th IEEE International System-on-Chip Conference (SOCC)*. Vol. 40. 8. IEEE, Sept. 2014, pp. 192–197. DOI: 10.1109/SOCC.2014.6948925.

[4] Steve Kerrison, David May, and Kerstin Eder. "A Benes Based NoC Switching Architecture for Mixed Criticality Embedded Systems". In: *IEEE 10th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC 2016) [to appear]*. Lyon, France, 2016, pp. 1–8.

[5] Abraham Waksman. "A Permutation Network". In: *Journal of the ACM* 15.1 (Jan. 1968), pp. 159–163. DOI: 10.1145/321439.321449.

[6] J. Hestness, B. Grot, and S. W. Weckler. "Netrace: Dependency-Driven, Trace-Based Network-on-Chip Simulation". In: *3rd International Workshop on Network on Chip Architectures (NoCArc)*. Dec. 2010.

[7] Adam Kostrzewa, Selma Saidi, and Rolf Ernst. "Slack-Based Resource Arbitration for Real-Time". In: *Design, Automation Test in Europe Conference Exhibition (DATE), 2016*. 2016, pp. 1012–1017. ISBN: 9783981537062.