Scheduling-Aware Routing for Supercomputers

Jens Domke — TU Dresden
Torsten Hoefler — ETH Zurich

Dipl.-Math. Jens Domke

Research Associate – Technische Universität Dresden
Institute of Computer Engineering – Computer Architecture

Email: jens.domke@tu-dresden.de
Tel.: +49 351 - 463 – 38783

ZIH
Center for Information Services & High Performance Computing

Outline

- **Motivation**
- **Scheduling-Aware Routing**
  - Interface between Batch System and Subnet Manager
  - Routing Optimization with modified DFSSSP
- **Property Preserving Network Updates for IB**
  - Five-phase Update Protocol
  - Current Limitations and Problems
- **Evaluation of Scheduling-Aware Routing**
  - Theoretical Evaluation of Network Metrics
  - Practical Evaluation on a Production System
- **Summary and Conclusions**

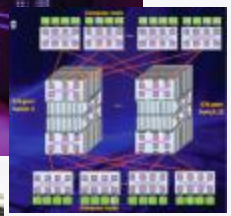
Interconnection Networks for HPC-Systems

Towards ExaScale

- ≥100.000 nodes [Kogge, 2008]
- Fat-trees not sustainable
- Sparse/random topologies (SimFly [Besta, 2014], Dragonfly [Kim, 2008], Jellyfish [Singla, 2012], …)

2013: Tianhe-2 (NUDT)
16,000 Nodes
Fat-Tree

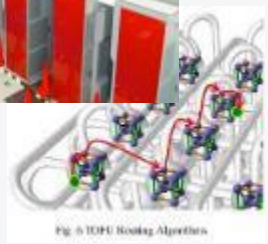

[F7]


[F8]

2011: K (RIKEN)
82,944 Nodes
6D Tofu Network


[F5]

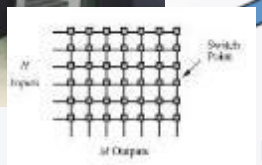
2004: BG/L (LLNL)
16,384 Nodes
3D-Torus Network

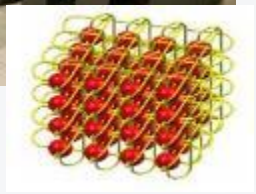

[F6]

1993: NWT (NAL)
140 Nodes
Crossbar Network


[F3]


[F1]


[F2]


[F4]

Routing Metrics:

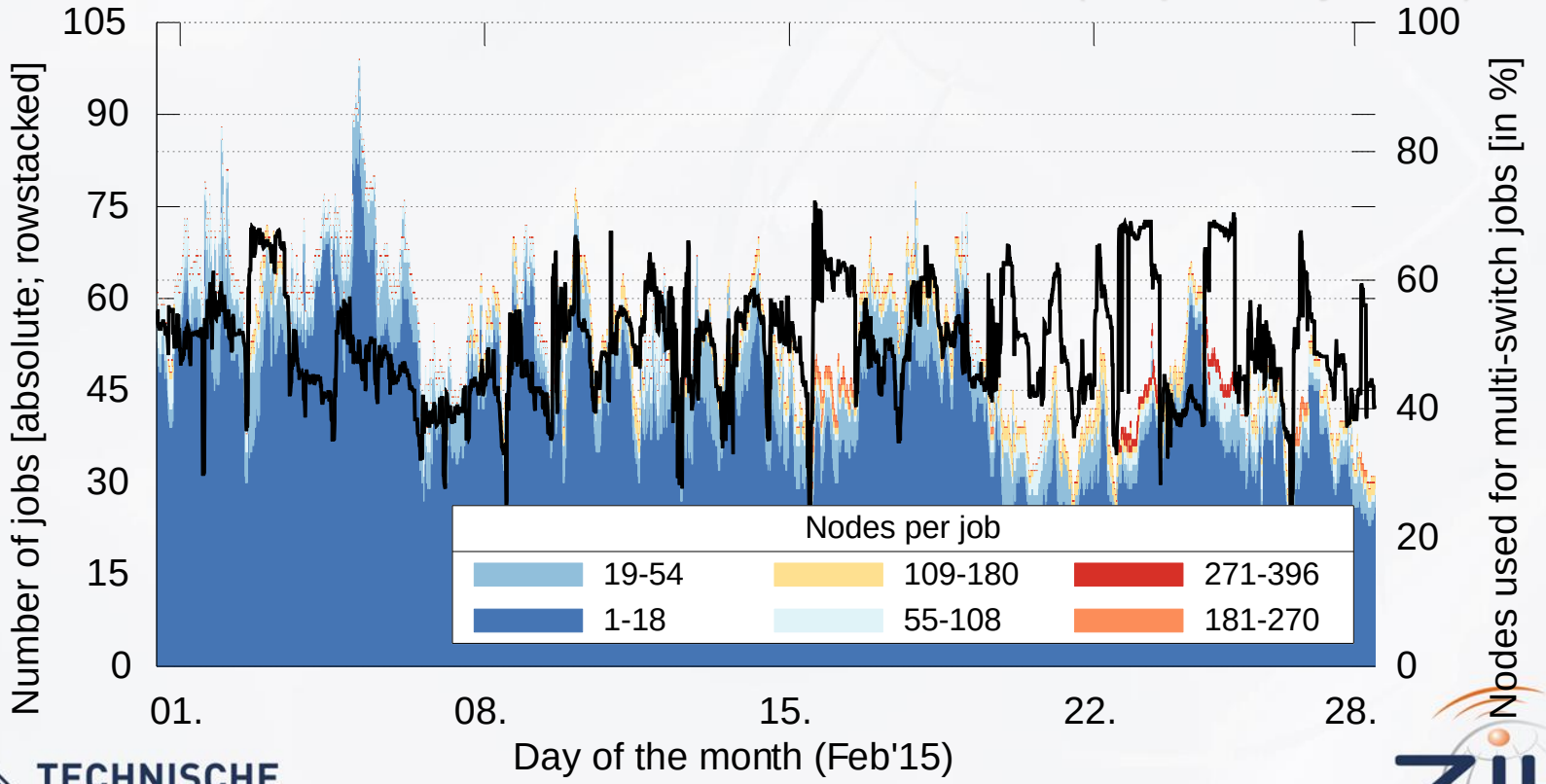
- Low latency
- High throughput
- Low congestion
- Fault-tolerant
- Deadlock-free
- **Utilization**

Massive networks needed to connect all compute nodes of supercomputers (see TOP500 list)

Realistic Workload of Multi-User/Multi-Job HPC Systems

- Avg. 50% of nodes are used for multi-node/multi-switch jobs
- Many small jobs (≤18 nodes) connected to multiple switches
  - ➥ Natural fragmentation of the batch system/supercomputer
  - ➥ Potential to improve network utilization?

Fig. 1: Batch jobs of Tsubame2.5 (sampled every 10 min)



Current state-of-the-art: Flow-Oblivious and Static Routing

- Artificial example
  - Full-bisection fat-tree w/ 180 nodes
  - 3x 60-node jobs (non-contiguous)
- Implication of flow-oblivious DFSSSP
  - Imbalance of intra-job paths
  - Few links underutilized (0 paths)

➥ Known problem: performance degradation through mismatch between comm. pattern and static routing [Hoefler, 2008]

- Alternative approaches, e.g.:
  - Topology mapping [Yu, 2006; Hoefler, 2011]
  - Application-aware routing [Kinsy, 2009]
  - Adaptive routing [Alverson, 2012; Birrittella, 2015]

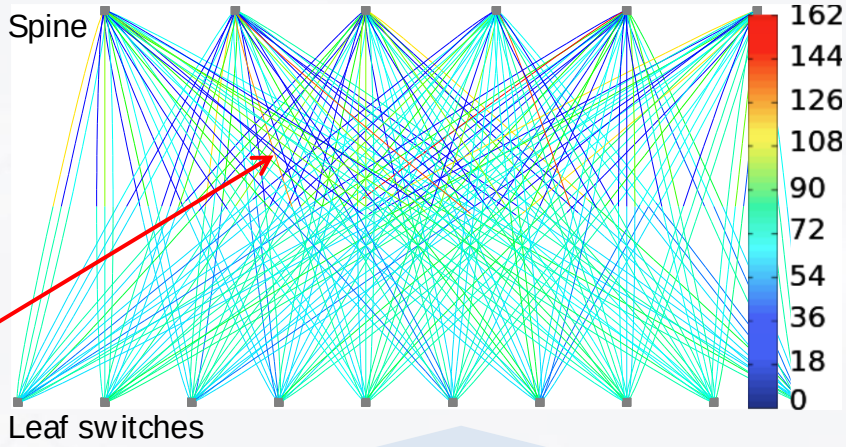
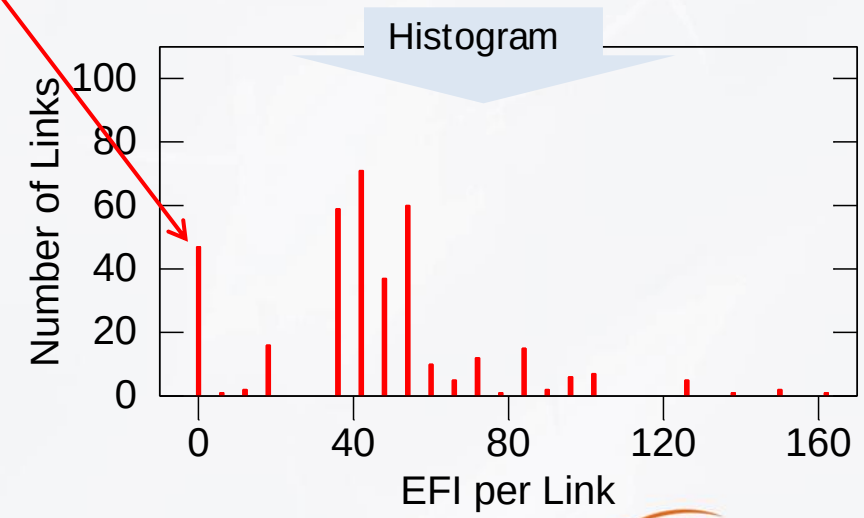


Fig. 2/3: Effective EFI for 3 jobs on 2-level fat-tree



Outline

- Motivation
- **Scheduling-Aware Routing**
  - Interface between Batch System and Subnet Manager
  - Routing Optimization with modified DFSSSP
- Property Preserving Network Updates for IB
  - Five-phase Update Protocol
  - Current Limitations and Problems
- Evaluation of Scheduling-Aware Routing
  - Theoretical Evaluation of Network Metrics
  - Practical Evaluation on a Production System
- Summary and Conclusions

Idea to Improve the Network Utilization and Performance

Initial hypothesis

- Optimizing for **global path balancing suboptimal** for production HPC
- **Inter-job paths not used** (between nodes of different batch jobs)
- InfiniBand/OpenSM allows for **coarse grain routing optimizations**

Requirements for a *feasible* Scheduling-Aware Routing (SAR)

- **Light-weight interface** analyzing jobs which run simultaneously
  - Filtering: collect jobs which require network (at least 2 switches)
  - Inform OpenSM about desired re-routings
- Fast and **optimized** routing calculation **for multi-user environments**
  - Enhancements based on proven techniques (… don't reinvent the wheel)
  - Integrate job locality information into balancing decisions
- **No user interaction** or input needed

Filtering tool: Interface between SLURM and OpenSM

Why not a SLURM plugin?

- Portability to other batch system
- SLURM latency already slow

Filtering tool workflow

- Periodically poll queue state
- Filter out small jobs (attached to only 1 switch)
- Compare job-to-node mapping with previous run
- If changed: prepare input file for OpenSM and send signal to request routing optimization

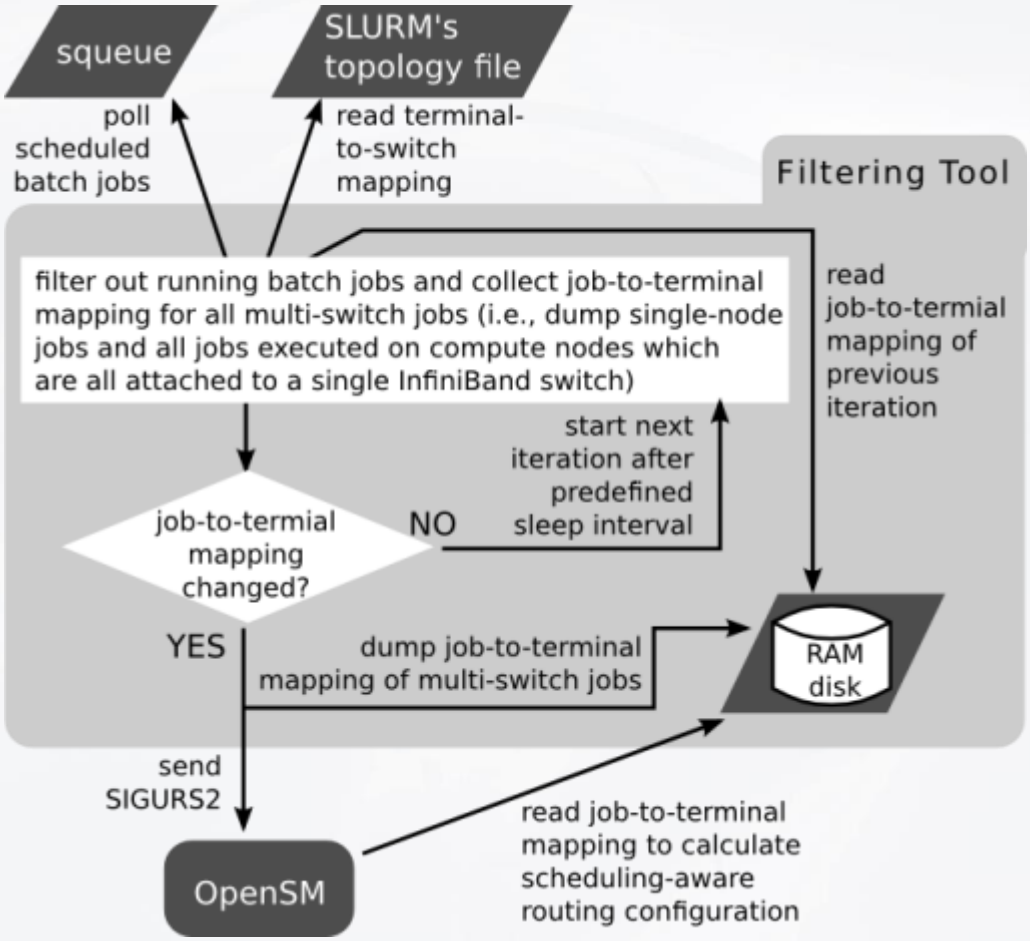


Fig. 4: Flowchart of filtering tool

Routing Optimization with modified (DF-)SSSP

Why deadlock-free single-source shortest-path (DFSSSP) routing [Domke, 2011]?

- Deadlock-free and topology-agnostic ➔ wide support range
- High global throughput even for irregular fat-trees [Domke, 2014]
- Distinguishes three node types: compute, I/O, and other

➥ SAR should inherit these good characteristics

Algorithm 1: Scheduling-aware DFSSSP routing

Input: Network $I = G(N, C)$
Job-to-terminal mapping $B := [(\text{nodeName}, \text{jobID}), \ldots]$
Result: Scheduling-aware and deadlock-free routing configuration
(P_{n_x,n_y} for all $n_x, n_y \in N$)

```
/* Process job-to-terminal mapping                                   */
1  foreach node n ∈ N do
2      n.jobList ← empty list []
3      foreach pair (nodeName,jobID) ∈ B do
4          if n.nodeName = nodeName then n.jobList.append(jobID)
/* Optimize routing for compute nodes                                */
5  N_sorted ← Sort N descending by the job size executed on n ∈ N
6  foreach node n_d ∈ N_sorted do
7      Calculate one path P_{n_x,n_d} for every pair (n_x, n_d), with n_x ∈ N,
       with the modified Dijkstra algorithm (details in [6])
8      foreach node n_x ∈ N do
9          if n_x.jobList ⋂ n_d.jobList ≠ ∅ then
10             Increase edge weight by +1 for each link in path P_{n_x,n_d}
/* Optimize routing for storage nodes                                */
   ...
/* Optimize routing for all other nodes                              */
   ...
/* Create deadlock-free routing configuration                        */
   ...
```

(DFSSSP was a choice, not a requirement ➔ SAR method applicable to other routings, too)

Routing Optimization with modified (DF-)SSSP

Scheduling-aware DFSSSP routing (or SAR) for all $|N| \cdot (|N| - 1)$ routes:

- Read job-to-node mapping file and add job IDs to nodes
- Sort list of nodes by job size (➥ improves balancing for large jobs which need “more network”)
- Search all paths towards a destination (w/ inverse Dijkstra)
- Update edge weights only for intra-job paths
- Calculate balanced routes for remaining nodes and create cycle-free CDG

Algorithm 1: Scheduling-aware DFSSSP routing

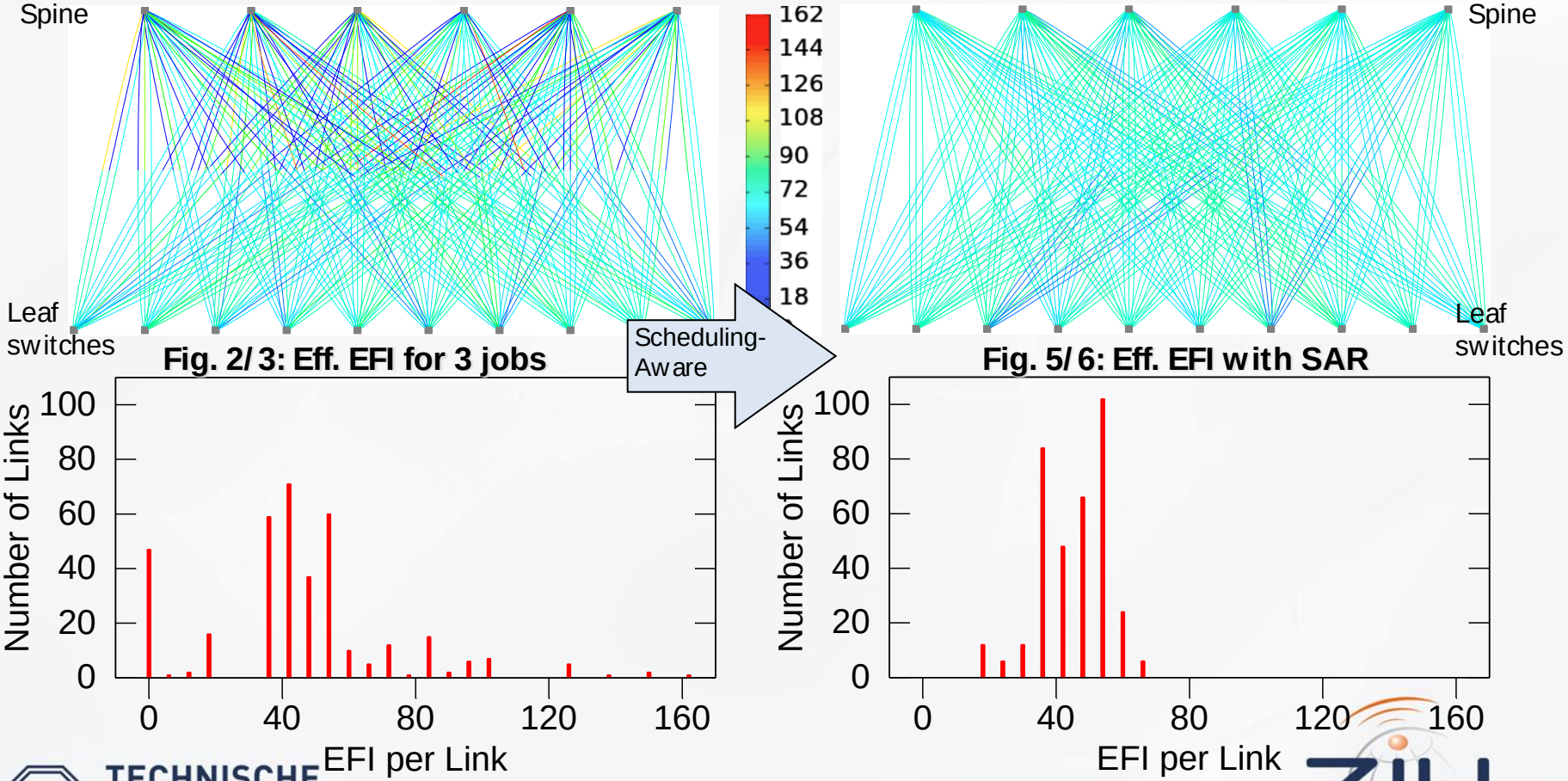
Input: Network $I = G(N, C)$
Job-to-terminal mapping $B := [(\text{nodeName}, \text{jobID}), \ldots]$
Result: Scheduling-aware and deadlock-free routing configuration (P_{n_x,n_y} for all $n_x, n_y \in N$)

```
/* Process job-to-terminal mapping                                   */
1  foreach node n ∈ N do
2      n.jobList ← empty list []
3      foreach pair (nodeName,jobID) ∈ B do
4          if n.nodeName = nodeName then n.jobList.append(jobID)
/* Optimize routing for compute nodes                                */
5  N_sorted ← Sort N descending by the job size executed on n ∈ N
6  foreach node n_d ∈ N_sorted do
7      Calculate one path P_{n_x,n_d} for every pair (n_x, n_d), with n_x ∈ N,
       with the modified Dijkstra algorithm (details in [6])
8      foreach node n_x ∈ N do
9          if n_x.jobList ⋂ n_d.jobList ≠ ∅ then
10             Increase edge weight by +1 for each link in path P_{n_x,n_d}
/* Optimize routing for storage nodes                                */
   ...
/* Optimize routing for all other nodes                              */
   ...
/* Create deadlock-free routing configuration                        */
   ...
```

(Furthermore: OpenSM extended to receive SIGUSR2 ➔ triggers re-routing)

Scheduling-Aware Routing applied to previous Example

- Hotspot (max. EFI) reduction from ≥160 to ≈60
  - ➥ theoretically lower worst-case congestion [Heydemann, 1989]
- Overall path balance improved and better utilization (no unused ports)



Outline

- **Motivation**
- **Scheduling-Aware Routing**
  - Interface between Batch System and Subnet Manager
  - Routing Optimization with modified DFSSSP
- **Property Preserving Network Updates for IB**
  - Five-phase Update Protocol
  - Current Limitations and Problems
- **Evaluation of Scheduling-Aware Routing**
  - Theoretical Evaluation of Network Metrics
  - Practical Evaluation on a Production System
- **Summary and Conclusions**

One Implications of Optional Routing Changes

What happens if we change the LFTs while packets are in-flight?

- Assume (simplified):
  - 3-level fat-tree with static, flow-oblivious routing
  - 2 flows (blue & green) to different destinations
  - Blue flow has 5 packets with sequence number 1…5 currently in-flight
  - More packets are waiting (6, …)

➥ congested link between `L0` and `L1` switches

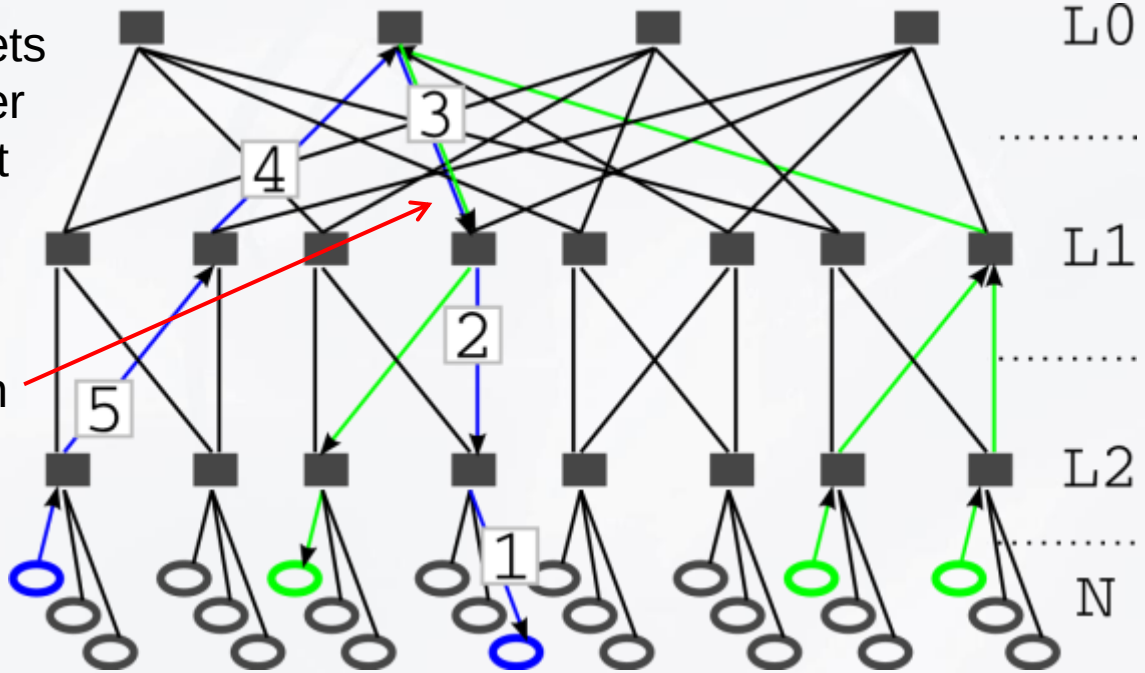


Fig. 7: Out-of-order packet delivery through congestion and re-routing

One Implications of Optional Routing Changes

- Modifying the LFTs (e.g., via SAR) changes blue flow onto red path:
  - Packets 4 and 5 slow via old, congested link
  - Packets 6, 7, … routed via fast and empty links

➥ Packet 6 arrives before packet 4

Consequence for InfiniBand?

- HCA detects out-of-order delivery through packet sequence numbers
- IB doesn't support OOO [IBTA, 2015]

➥ Message dropped
➥ Sender retries delivery
➥ RETRY EXCEEDED ERROR
➥ MPI app. crashes!!!

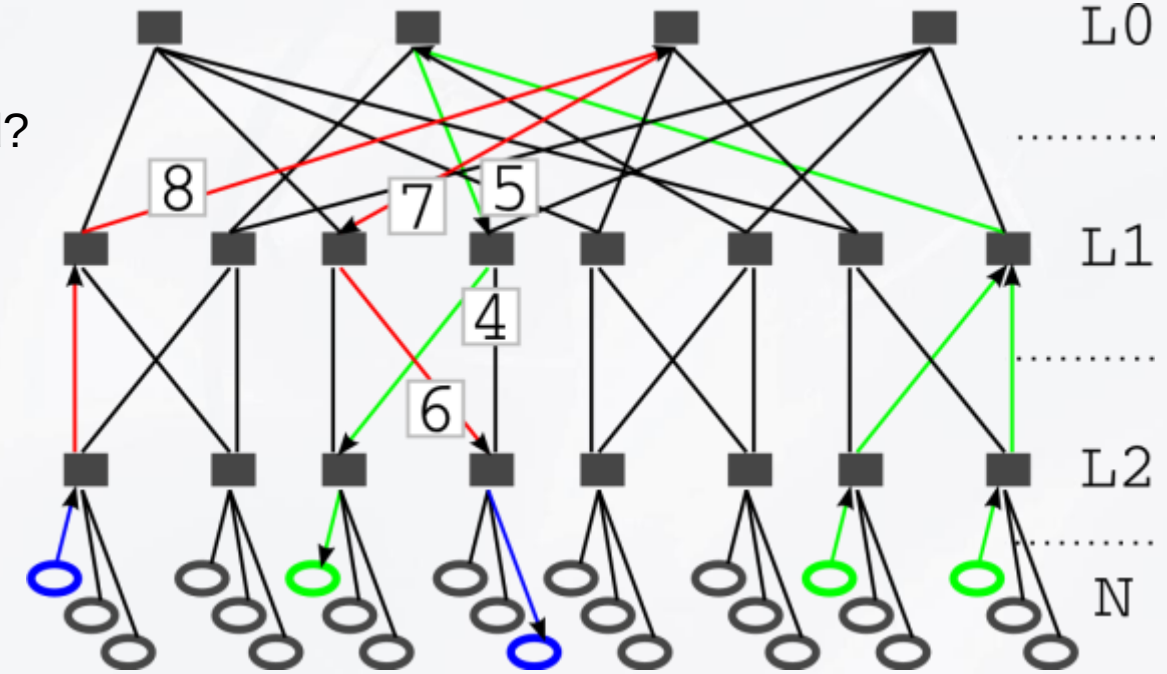


Fig. 7: Out-of-order packet delivery through congestion and re-routing

Property Preserving Network Updates

- Atomic LFT updates impossible in IB (new LFT distributed via 64 B chunks)
  ➥ potential for out-of-order, security vulnerability, packet loss, deadlocks, …
- Existing approaches for SDN/Ethernet not applicable, e.g.
  - Two-phase update [Reitblatt, 2012]
    - Install passive routing configurations
    - Swap passive→active if tagged packet is identified
  - Ordering Updates [McClurg, 2015]
    - Choose a correct order of switch updates
- Requirements for lossless InfiniBand

Property Preserving Network Update

The transition between two routing configurations (i.e., 2 valid LFT sets) is called a property preserving network update if the following holds:

1) **each** configuration itself is **deadlock-free**,
2) the transition is a **per-flow consistent update** (only one routing applies),
3) **simultaneous processing** of flows by both routings is **deadlock-free**.

Five-Phase Property Preserving Update Protocol

SAR build on top of DFSSSP
➥ deadlock-free ➔ (1) ✓

Per-flow consistent update

- Each IB HCA gets 2 LIDs assigned
- SAR routes baseLIDs and uses $0 \leq VL < n - 1$
- Up*/Down* used for highLIDs and uses $VL := n - 1$
- MPI applications subscribe for event forwarding (un-/repath trap)
- Unpath trap (repath similar):
  - Drain send queues of all ranks
  - Trigger path migration (APM)
  - Change LFTs for baseLIDs / SAR

➥ no packets betw. baseLIDs ➔ (2,3) ✓

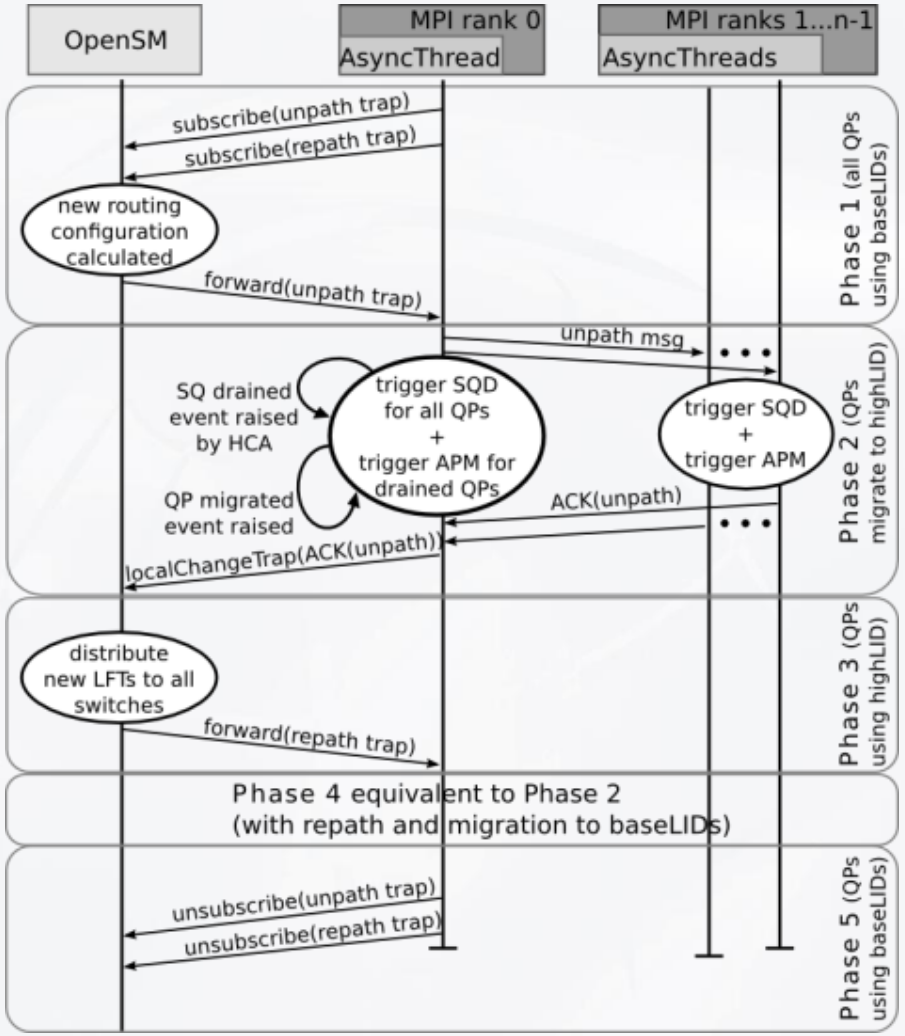


Fig. 8: Sequence diagram of our five-phase update protocol for IB

Current Limitations and Problems

- Potential packet loss between OpenSM and subscribers
  - OpenSM and AsyncThread of rank 0 use (u)MAD packets to subscribe and forward traps ➔ QP0 / QP1 use unreliable transport service
  - MADs usually send multiple times if not acknowledged ☺
- No simultaneous calls to MPI API allowed for Open MPI + openib
  - Workaround: pthread mutex locks to serializing MPI calls between main application and AsyncThread of all ranks ☺
- QP draining impossible with two tested firmware for our IB devices ☹

➥ Implementation challenging but theoretically possible! ☺

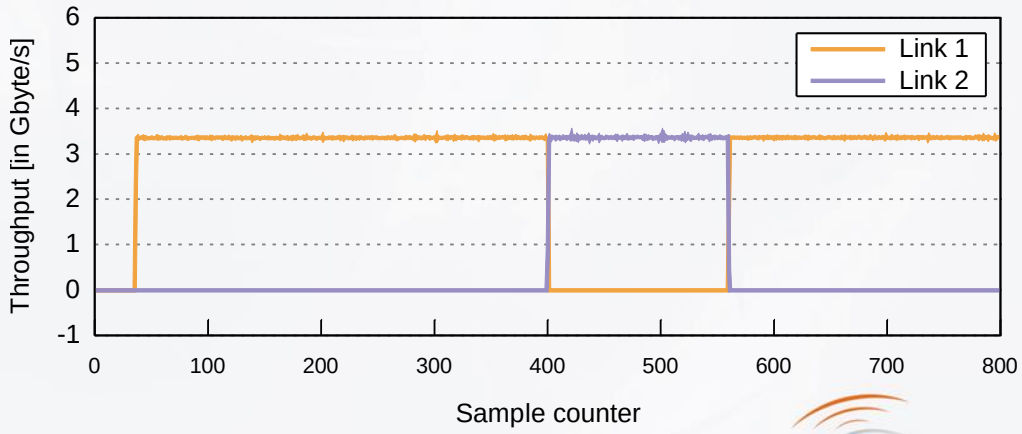


Fig. 9: Network update protocol (w/o QP draining) on testbed

Outline

- **Motivation**
- **Scheduling-Aware Routing**
  - Interface between Batch System and Subnet Manager
  - Routing Optimization with modified DFSSSP
- **Property Preserving Network Updates for IB**
  - Five-phase Update Protocol
  - Current Limitations and Problems
- **Evaluation of Scheduling-Aware Routing**
  - Theoretical Evaluation of Network Metrics
  - Practical Evaluation on a Production System
- **Summary and Conclusions**

Petascale HPC Systems and Workloads

- Modified simulation framework to analyze routing/jobs combinations [Domke, 2014]
- Comparison of four routings:
  - Topology-agnostic: (DF-)SSSP [Hoefler, 2009; Domke, 2011], SAR
  - Topology-aware: fat-tree [Zahavi, 2010], Up*/Down* [Schroeder, 1991]
- Based on two job-depended metrics (eff. EFI and unused ports/links)
- “Replay” exact job history of Feb.’15

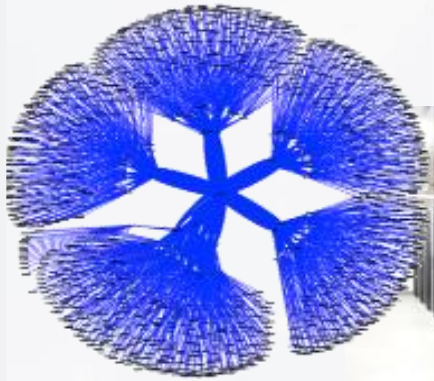
Taurus @TU Dresden

- 2014 compute nodes (1.4 Pflop/s)
- Multiple 2-level full-bisec. FDR/QDR fat-tree islands connected by director

[F9]

Tsubame2.5 @Titech

- 1408 compute nodes (5.7 Pflop/s)
- Two full-bisection fat-tree QDR rails



[F10]

Job-depended Metrics: Effective Edge Forwarding Index

- Common network metrics (e.g., bisection BW, latency, …) not applicable
  - Usually ignore routing algorithm
  - Node locality of batch jobs required to compare SAR to others

- Routes between nodes of different jobs not used (except I/O): EFI ➔ eff. EFI

Effective Edge Forwarding Index

The effective edge forwarding index γ^e of a switch port or outgoing link $c \in C^*$ is the **sum of intra-job routes** being **forwarded via this port**, i.e.,

$$\gamma^e(c) := \sum_j \left| \left\{ P_{n_x,n_y} \mid n_x, n_y \in N_j \text{ and } c \in P_{n_x,n_y} \right\} \right|$$

for all batch jobs $j \in \mathcal{J}$ running on the system.

$\mathcal{J}$	- set of batch jobs
N_j	- set of nodes belonging to job j
C^*	- inter-switch links
P_{n_x,n_y}	- path from n_x to n_y

➥ Prediction of worst-case congestion

Job-depended Metrics: Dark Fiber Percentage

- After filtering unused routes: how many ports/links are actually in use?

Dark Fiber Percentage

The dark fiber percentage is the **percentage of links** in the system, which are **not used for intra-job routes**, and can therefore be derived from γ^e in the following way:

$$\theta := \frac{|\{ c \in C^* \mid \gamma^e(c) = 0 \}|}{|C^*|}$$

C^* - inter-switch links
γ^e - effective edge forwarding index

➥ Utilization of network hardware

Relative Improvements for Tsubame2.5 (base: fat-tree)

Maximum γ^e for all jobs

Avg. max. γ^e across jobs

Dark fiber percentage

Used ports/links (avg. across jobs)

(higher is better)

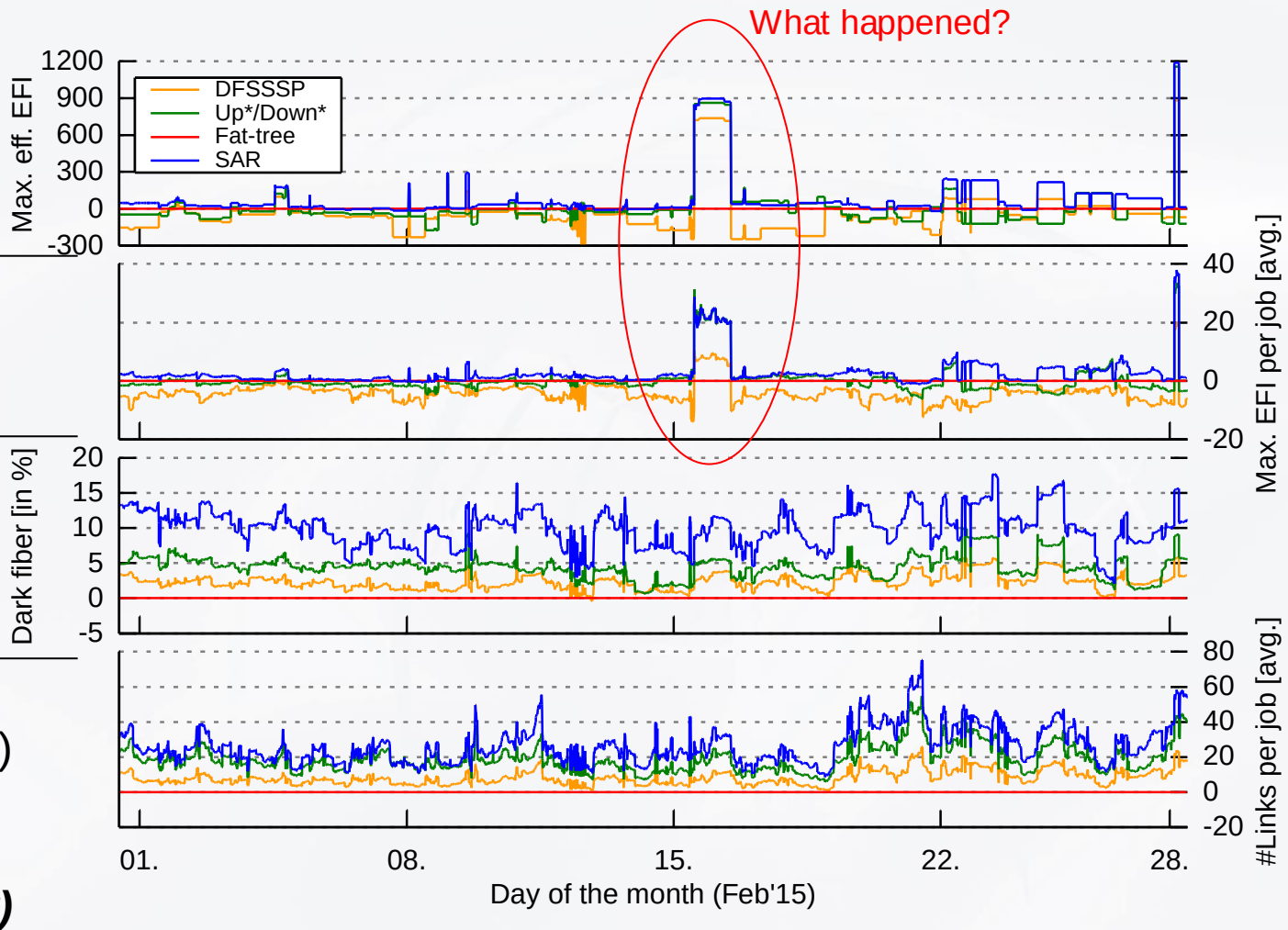
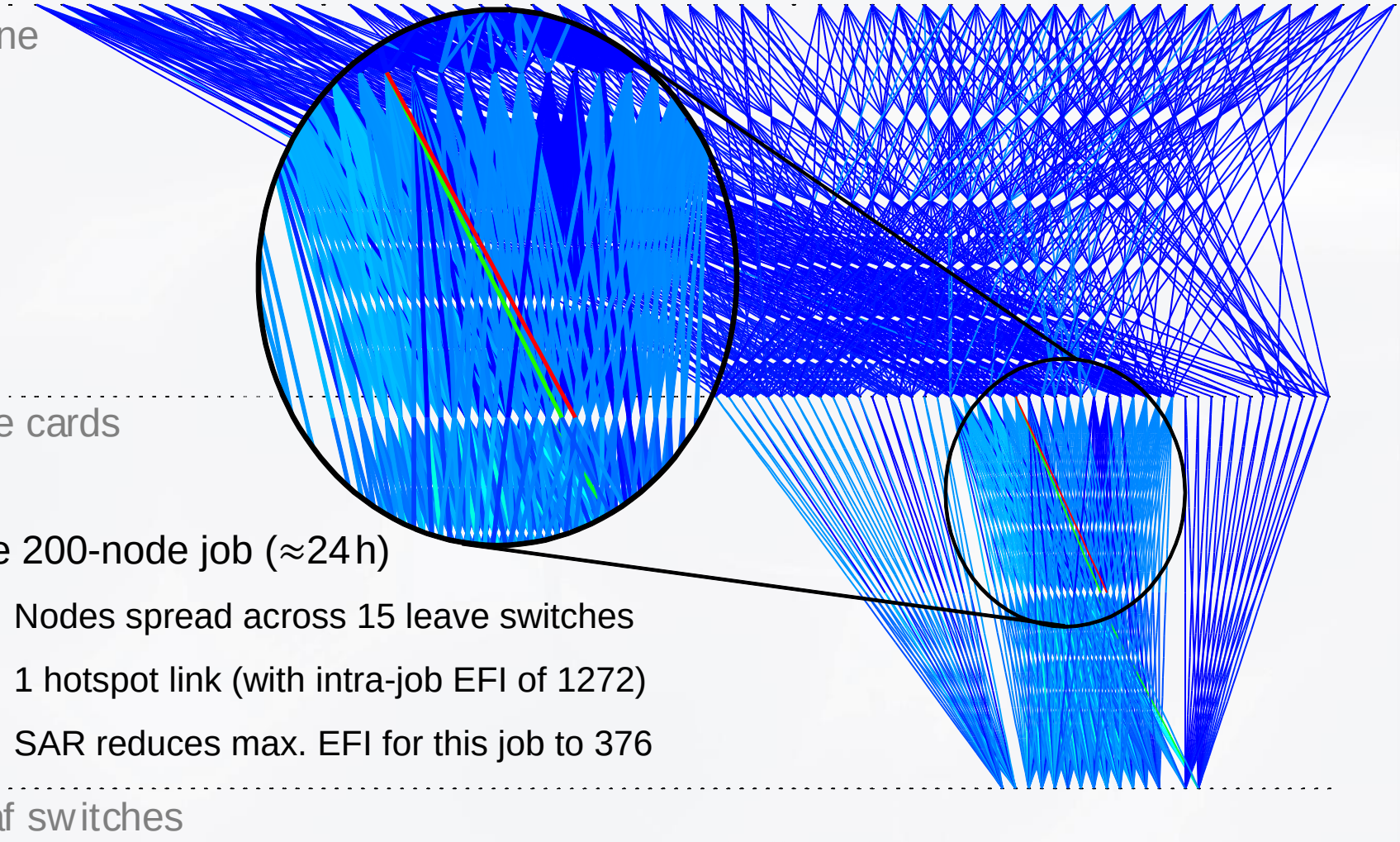


Fig. 11: Replay of job history for Tsubame2.5 (four routings); Values relative to fat-tree routing

Outlier for Fat-Tree Routing on Tsubame2.5 on 02/16/2015



- One 200-node job (≈24 h)
  - Nodes spread across 15 leave switches
  - 1 hotspot link (with intra-job EFI of 1272)
  - SAR reduces max. EFI for this job to 376

Fig. 12: Heatmap of eff. EFI for one job on first rail of Tsubame2.5 supercomputer

Collected Metrics for Taurus and Tsubame2.5

- Maximum and average improvements by SAR for full month (Feb.'15), e.g.:
  - Taurus
    - Maximum γ^e reduced by 279.0 (50.8%) compared to DFSSSP
    - Avg. θ improved between 4% and 9% (dep. on routing)
  - Tsubame2.5
    - Max. θ improved by up to 17.7%
    - On avg. 27% more ports/links available per job (compared to fat-tree)

➥ Overall: remarkable benefits through SAR

TABLE 1. IMPROVEMENTS BY OUR SCHEDULING-AWARE ROUTING COMPARED TO DFSSSP, FAT-TREE, AND UP*/DOWN* ROUTING

	Taurus HPC system						Tsubame2.5 HPC system					
Metric	DFSSSP		fat-tree		Up*/Down*		DFSSSP		fat-tree		Up*/Down*	
	max. / in %	avg. / in %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %	max. / %	avg. / %
$\max_{c_q \in C^*} \gamma^e(c_q)$	279.0 / 50.8	57.3 / 23.3	18.0 / 21.2	-1.1 / -0.6	18.0 / 21.2	-1.1 / -0.6	321.0 / 61.2	119.4 / 38.5	1186.0 / 71.2	80.5 / 29.7	354.0 / 48.7	69.9 / 26.8
$\text{avg}_{j \in J} \gamma_{max}(j)$	16.0 / 39.0	4.3 / 23.4	1.6 / 11.4	-0.3 / -2.1	1.6 / 11.4	-0.3 / -2.1	18.9 / 46.0	6.7 / 30.1	38.0 / 49.8	2.7 / 14.8	10.6 / 29.9	2.4 / 13.2
θ [in %]	9.38	6.03	7.64	3.62	7.64	3.62	12.06	7.63	17.74	9.99	9.24	5.51
$\text{avg}_{j \in J}$ #links(j)	16.5 / 15.0	7.7 / 11.1	14.1 / 13.8	6.6 / 9.4	14.1 / 13.8	6.6 / 9.4	49.2 / 26.7	18.3 / 17.4	75.1 / 43.9	26.5 / 27.3	22.3 / 14.9	7.4 / 6.4

Runtime Measurement for MPI_Alltoall on Taurus

- Modified OSU MPI_Alltoall benchmark (const. message size of 1 MiB)
- 28 nodes (1 ppn) allocated via SLURM: system fragmentation ➔ 10 switches
- Seamless routing switch (fat-tree routing ➔ DFSSSP ➔ SAR)
  - ☹ Runtime increase of 7.1% for DFSSSP
  - ☺ SAR decreases runtime by 17.6% (DFSSSP) or 11.7% (fat-tree)
  - ☺ Congestion overhead reduced by 50% for SAR vs. fat-tree

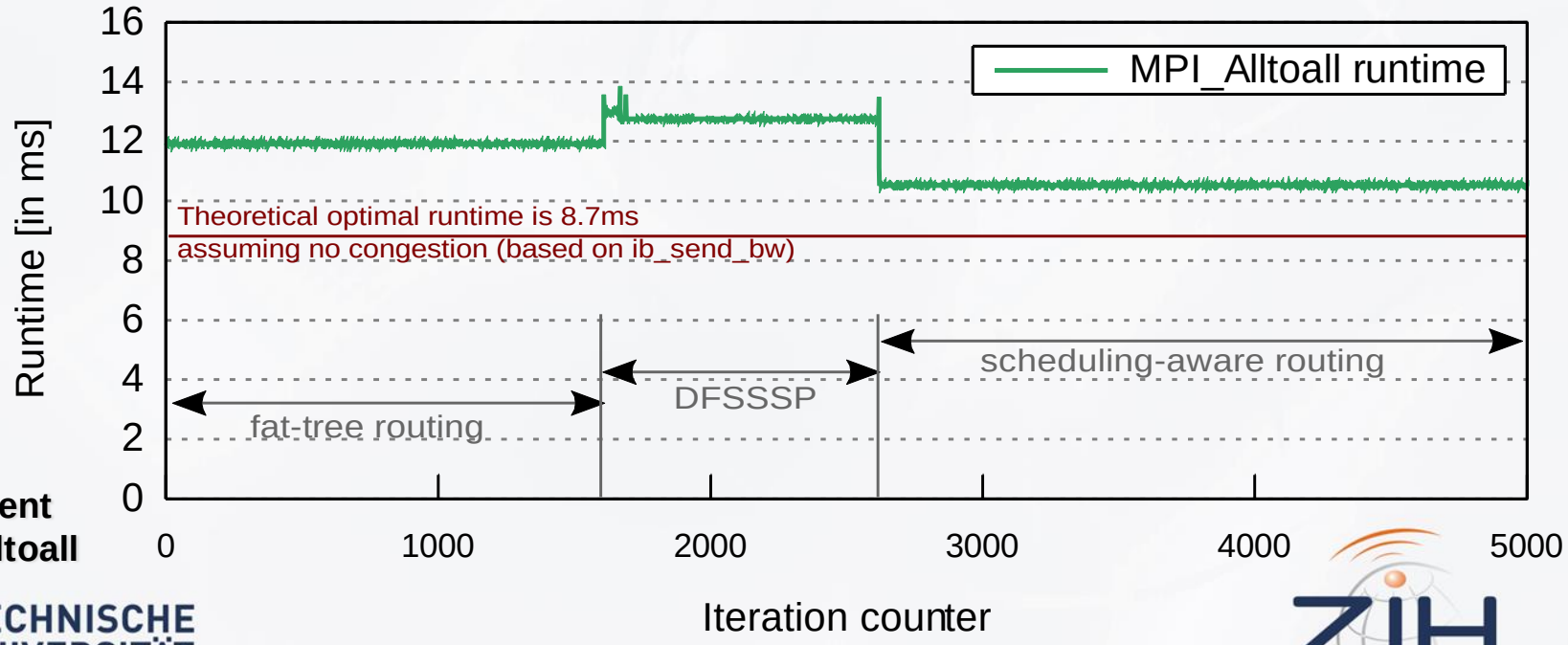


Fig. 13: Runtime measurement for MPI_Alltoall

Statistics for 1 year of SAR on Taurus HPC System

- Runtime of the filtering tool (scheduled to run every 5 min on Taurus)
  - Depends almost entirely on *squeue* latency
  - Recorded min./avg.: 0.02 s and 16 s
  - Worst case within a year:
    - ≤ 2 min for 99.1% of the runs
    - 3 runs with ≥ 10 min
- Routing overhead induced by SAR (compared to DF-/SSSP)
  - Negligible; same runtime complexity of $\mathcal{O}(|N|^2 \cdot log|N|)$
  - Total runtime ≤1 s for Taurus with 2014 compute nodes

Statistics for 1 year of SAR on Taurus HPC System

- New routing configurations calculated per day
  - Between 0 and 57 re-routings by SAR (avg. of 14) ➔ approx. every 2 h
  - 4 days without re-routings: 3x on weekend; 1x Monday
- Time needed to reconfigure all 210 switches of Taurus
  - Avg. of 4.6 μs to send LFT block and receive ACK
  - Usually $\approx$0.8 s to reconfigure full fabric (incl. OpenSM-internal overhead)
- Application crashes due out-of-order packages in these 0.8 s?
  - Probably mitigated through IB's end-to-end error detection and retry
  - No crashes reported by users

Outline

- **Motivation**
- **Scheduling-Aware Routing**
  - Interface between Batch System and Subnet Manager
  - Routing Optimization with modified DFSSSP
- **Property Preserving Network Updates for IB**
  - Five-phase Update Protocol
  - Current Limitations and Problems
- **Evaluation of Scheduling-Aware Routing**
  - Theoretical Evaluation of Network Metrics
  - Practical Evaluation on a Production System
- **Summary and Conclusions**

Summary and Conclusions

State-of-the-art static routings are suboptimal for production systems!

- Optimizing for global path balancing ➔ only effective if whole system used by single parallel application
- We created low-overhead filtering tool to interface SLURM and OpenSM (avg. runtime of 16 s; but depends on SLURM latency)
- We enhanced topology-agnostic DFSSSP to consider job-to-node mapping ➔ SAR inherits features: deadlock-freedom, separate I/O balancing,…
- Our scheduling-aware routing (SAR) outperforms other flow-oblivious routings
  - Up to 70% reduced path overlap for production workloads
  - More inter-switch links available per batch job ➔ higher network utilization

Summary and Conclusions

Reconfiguring switch LFTs can cause out-of-order packages in IB!

- We designed a reliable update protocol to prevent out-of-order
- Implementation in practice “failed” (vendor firmware not 100% IB-compliant)

SAR is default on petascale production HPC systems!

- Stable operation for more than one year
- No user interaction/input needed
- No application crashes despite missing update protocol
- Avg. of 4% less dark fiber compared to fat-tree routing (suggested by vendor)

Acknowledgments and Downloadable Content

Prof. Nagel and his team provided the batch job history of the Taurus HPC system installed at **TU Dresden** and allowed us to modify Taurus' routing algorithm over a longer period of time.

Prof. Matsuoka and his team gave us access to their batch job history of the Tsubame2.5 supercomputer located at the **Tokyo Institute of Technology**.

SAR for InfiniBand (OpenSM implementation):

- https://gitlab.com/domke/osm-routing-dev/tree/sar-3.3.20
- http://jdomke.info/#research

References (A-H)

[Alverson, 2014] B. Alverson, E. Froese, L. Kaplan, and D. Roweth, “Whitepaper: Cray XC Series Network,” Cray Inc., Tech. Rep. WP-Aries01-1112. [Online: http://www.cray.com/sites/default/files/resources/CrayXCNetwork.pdf]

[Besta, 2014] M. Besta and T. Hoefler, "Slim Fly: A Cost Effective Low-Diameter Network Topology," New Orleans, LA, USA, 2014.

[Birrittella, 2015] M. S. Birrittella, M. Debbage, R. Huggahalli, J. Kunz, T. Lovett, T. Rimmer, K. D. Underwood, and R. C. Zak, “Intel Omni-path Architecture: Enabling Scalable, High Performance Fabrics,” in 2015 IEEE 23rd Annual Symposium on High-Performance Interconnects (HOTI). Santa Clara, CA: IEEE, Aug. 2015, pp. 1–9.

[Domke, 2011] J. Domke, T. Hoefler, and W. E. Nagel, “Deadlock-Free Oblivious Routing for Arbitrary Topologies,” in Proceedings of the 25th IEEE International Parallel & Distributed Processing Symposium (IPDPS). Washington, DC, USA: IEEE Computer Society, May 2011, pp. 613– 624.

[Domke, 2014] J. Domke, T.Hoefler, and S.Matsuoka, “Fail-in-Place Network Design: Interaction between Topology, Routing Algorithm and Failures,” in Proceedings of the IEEE/ACM International Conference for High Performance Computing, Networking, Storage and Analysis (SC14), ser. SC ’14. New Orleans, LA, USA: IEEE Press, Nov. 2014, pp. 597–608.

[Heydemann, 1989] M. C. Heydemann, J. Meyer, and D. Sotteau, “On Forwarding Indices of Networks,” Discrete Appl. Math., vol. 23, no. 2, pp. 103–123, May 1989.

[Hoefler, 2008] T. Hoefler, T. Schneider, and A. Lumsdaine, “Multistage Switches are not Crossbars: Effects of Static Routing in High-Performance Networks,” in Proceedings of the 2008 IEEE International Conference on Cluster Computing. IEEE Computer Society, Oct. 2008.

[Hoefler, 2009] T. Hoefler, T. Schneider, and A. Lumsdaine, “Optimized Routing for Large-Scale InfiniBand Networks,” in 17th Annual IEEE Symposium on High Performance Interconnects (HOTI 2009), Aug. 2009.

[Hoefler, 2011] T. Hoefler and M. Snir, “Generic Topology Mapping Strategies for Large-scale Parallel Architectures,” in Proceedings of the 2011 ACM International Conference on Supercomputing (ICS’11). Tucson, AZ: ACM, Jun. 2011, pp. 75–85.

References (I-Z)

[IBTA, 2015] InfiniBand Trade Association, “InfiniBandTM Architecture Specification Volume 1 Release 1.3 (General Specifications),” Mar. 2015.

[Kinsy, 2009] M. A. Kinsy, M. H. Cho, T. Wen, E. Suh, M. van Dijk, and S. Devadas, “Application-aware deadlock-free oblivious routing,” in Proceedings of the 36th annual international symposium on Computer architecture, ser. ISCA ’09. New York, NY, USA: ACM, 2009, pp. 208–219.

[Kogge, 2008] P. Kogge, K. Bergman, and S. Borkar, “ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems,” University of Notre Dame, Department of Computer Science and Engineering, Notre Dame, Indiana, Tech. Rep. TR-2008-13, Sep. 2008.

[McClurg, 2015] J. McClurg, H. Hojjat, P. Cerny, and N. Foster, “Efficient Synthesis of Network Updates,” in Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, ser. PLDI 2015. New York, NY, USA: ACM, 2015, pp. 196–207.

[Reitblatt, 2012] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and D. Walker, “Abstractions for Network Update,” in Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication, ser. SIGCOMM ’12. New York, NY, USA: ACM, 2012, pp. 323–334.

[Schroeder, 1991] M. D. Schroeder, A. Birell, M. Burrows, H. Murray, R. Needham, T. Rodeheffer, E. Satterthwaite, and C. Thacker, “Autonet: A High- speed, Self-Configuring Local Area Network Using Point-to-Point Links,” IEEE Journal on Selected Areas in Communications, vol. 9, no. 8, Oct. 1991.

[Singla, 2012] A. Singla, C.-Y. Hong, L. Popa, and P. B. Godfrey, "Jellyfish: Networking Data Centers Randomly," in Presented as part of the 9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12), San Jose, CA, 2012, pp. 225-238.

[Yu, 2006] H. Yu, I.-H. Chung, and J. Moreira, “Topology Mapping for Blue Gene/L Supercomputer,” in Proceedings of the 2006 ACM/IEEE Conference on Supercomputing, ser. SC ’06. New York, NY, USA: ACM, 2006.

[Zahavi, 2010] E. Zahavi, G. Johnson, D. J. Kerbyson, and M. Lang, “Optimized InfiniBand fat-tree routing for shift all-to-all communication patterns,” Concurr. Comput. : Pract. Exper., vol. 22, no. 2, pp. 217–231, Feb. 2010.

Figure References (1-10)

[F1] http://museum.ipsj.or.jp/en/computer/super/0020.html

[F2] http://wiki.expertiza.ncsu.edu/index.php/CSC/ECE_506_Spring_2010/ch_12_PP

[F3] https://asc.llnl.gov/computing_resources/bluegenel/

[F4] https://asc.llnl.gov/computing_resources/bluegenel/configuration.html

[F5] http://www.fujitsu.com/global/about/resources/news/press-releases/2011/0620-02.html

[F6] http://www.fujitsu.com/downloads/TC/sc10/interconnect-of-k-computer.pdf

[F7] http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf

[F8] http://www.netlib.org/utk/people/JackDongarra/PAPERS/tianhe-2-dongarra-report.pdf

[F9] https://gauss-allianz.de/en/profile/Technische%20Universit%C3%A4t%20Dresden

[F10] http://pc.watch.impress.co.jp/img/pcw/docs/609/529/html/271.jpg.html

BACKUP

Assumptions and Goals for the Remainder of the Talk

- Requirements and assumptions:
  - Network I consists of
    $I = G(N, C)$
    $with \;\; C \subseteq N \times N$
    - switches, terminals (N) and full-duplex channels/links (C)
    - subset of inter-switch links $C^* \subset C$
  - Routing R should be
    $R(c_i, n_d) = c_{i+1}$
    $with \;\; n_d \in N \wedge c_i \in C$
    - shortest-path and balanced for realistic HPC workloads
    - destination-based (and unicast)
    - deadlock-free (for lossless technologies, e.g., InfiniBand)
    - support arbitrary topologies
    - no user-interaction required
  - Compute resources are limited

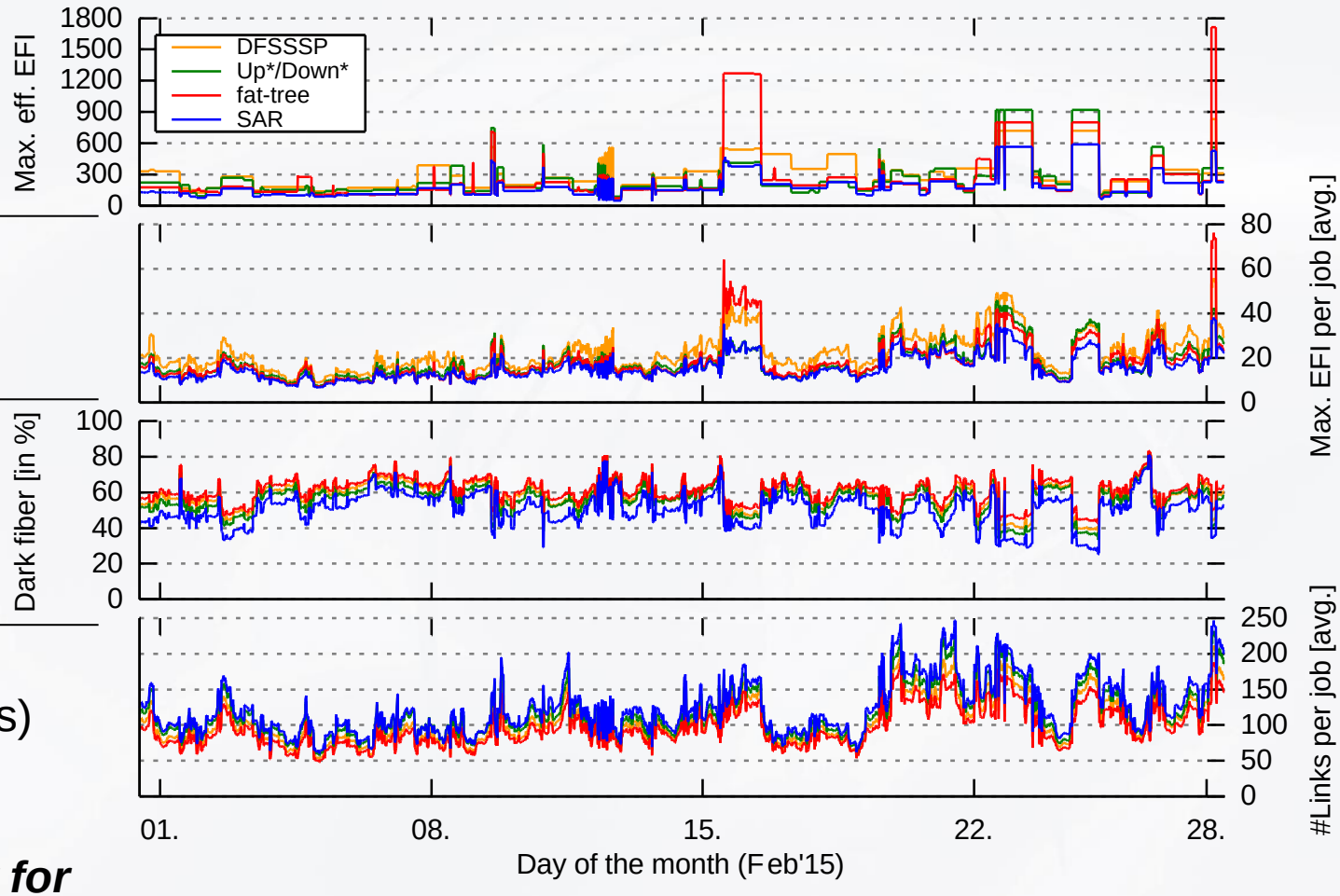
Collected Metrics for Tsubame2.5

Maximum γ^e for all jobs

Avg. max. γ^e across jobs

Dark fiber percentage

Used ports/links (avg. across jobs)



(lower is better for first three plots)

Fig. 10: Replay of job history for Tsubame2.5 (four routings applied per 10 min sampling point)

Collected Metrics for Taurus

Maximum γ^e for all jobs

Avg. max. γ^e across jobs

Dark fiber percentage

Used ports/links (avg. across jobs)

(lower is better for first three plots)

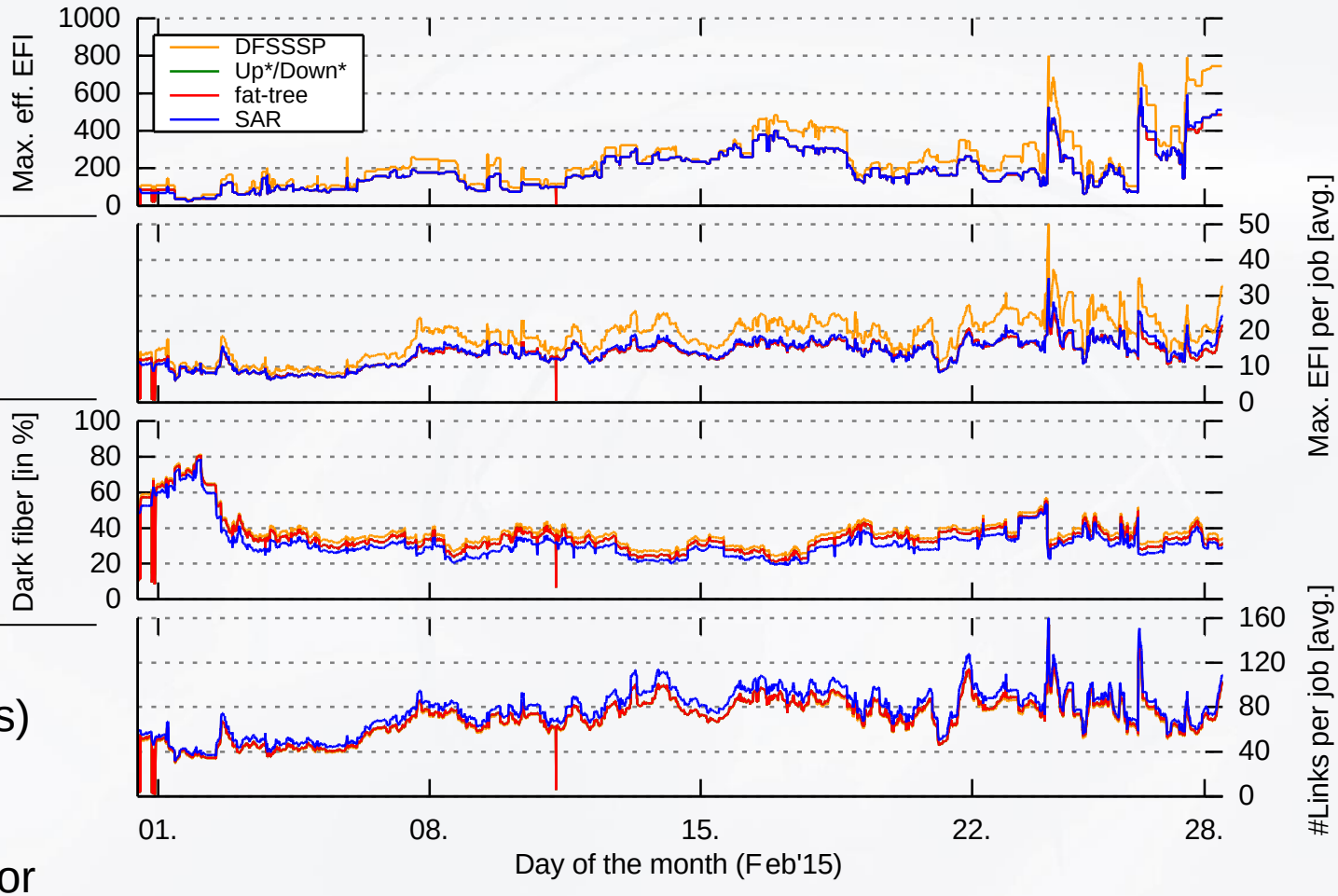


Fig. 14: Replay of job history for Taurus (four routings applied per 10 min sampling point)

Relative Improvements for Taurus (base: fat-tree)

Maximum γ^e for all jobs

Avg. max. γ^e across jobs

Dark fiber percentage

Used ports/links (avg. across jobs)

(lower is better for first three plots)

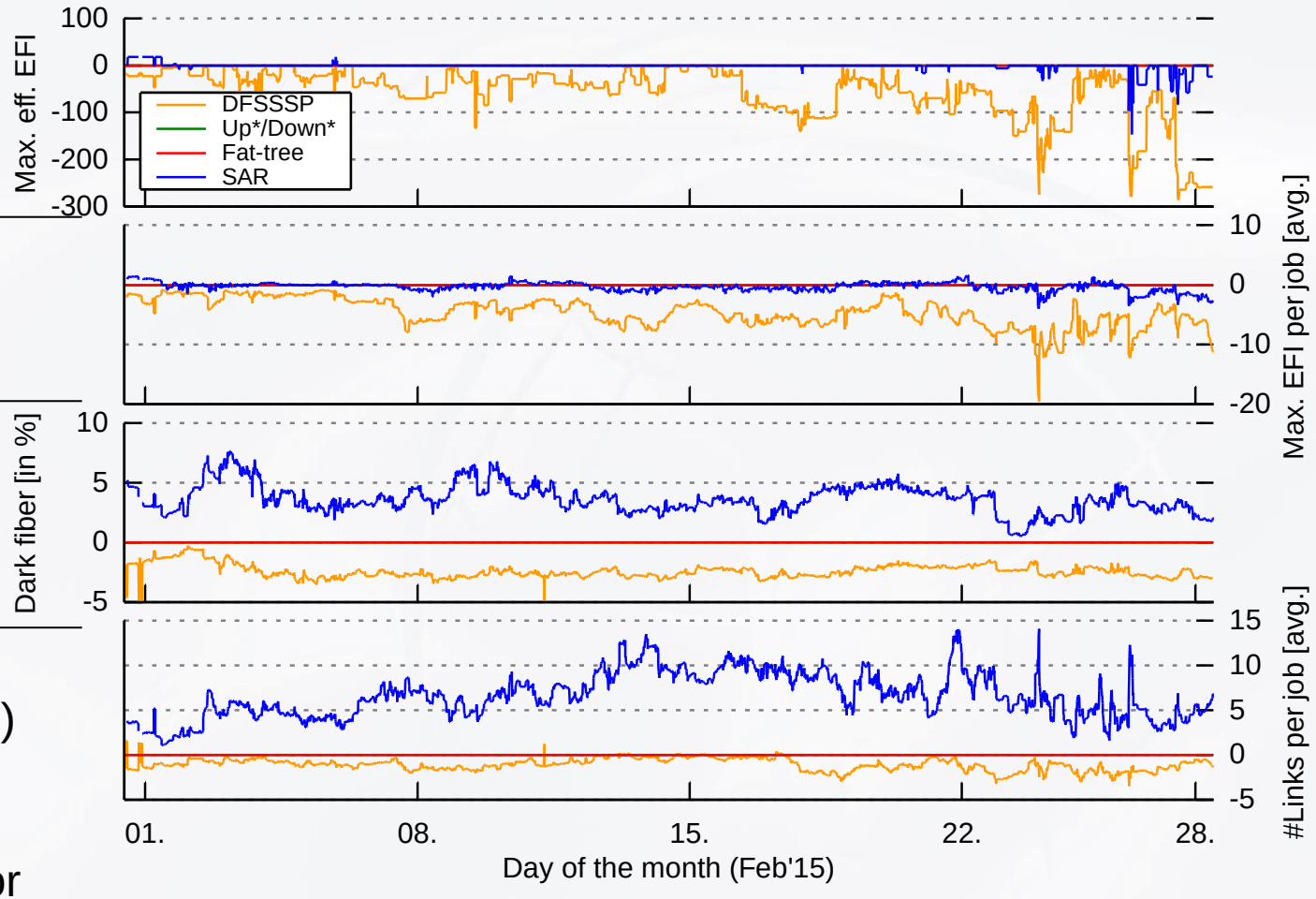


Fig. 15: Replay of job history for Taurus (four routings applied per 10 min sampling point)

Working Network Updates on Testbed (w/o QP draining)

- Small test system w/ 2 IB QDR switches (connected by two links) and 4 nodes
- MPI benchmark: repeatedly MPI_Bcast with 1 MiB send buffer
- Use *perfquery* for inter-switch links every ≈0.07 s to calculate throughput
- Artificial delay (10 s) between unpath and repath traps (samples: 400→560)

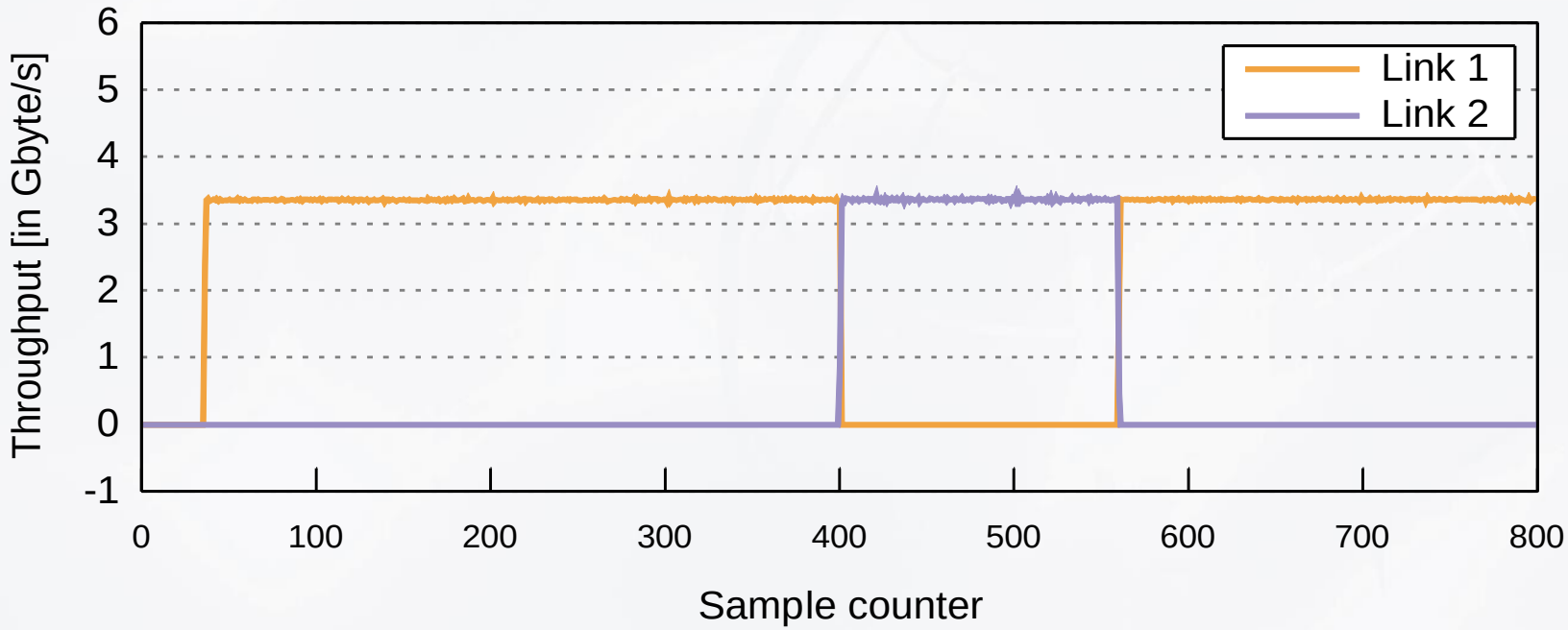


Fig. 16: Visualization of network update protocol (w/o QP draining) and APM betw. 2 links on testbed during high MPI load