

Evaluation of Speedup and Expansion in Terabit Switch Fabrics

Sarah Ruepp, Andreas Rytlig, Michael Berger, Henrik Wessing, Anna V. Manolova, Hao Yu, Anders Rasmussen

DTU Fotonik, Technical University of Denmark, 2800 Kgs. Lyngby, Denmark

E-mail: {srru,s052800,msbe,hewe,anva,haoyu,anras}@fotonik.dtu.dk

Abstract

This paper evaluates speedup and expansion in a multi-stage Terabit switch fabric. Single-stage switch fabrics, e.g. crossbar switches, are difficult to scale up to a Terabit system. Hence, a multi-stage switch fabric, where traffic can be distributed on different chips, offers a promising perspective. We evaluate buffer usage, number of out-of-sequence cells and fabric delay. Simulation results obtained in OPNET Modeler show that speedup outperforms expansion and that both approaches significantly gain from applying an output line speedup. By reducing the buffer usage in the middle stage, fewer cells arrive out-of-sequence and hence the need for re-sequencing is decreased as well.

Introduction

The Internet is increasingly populated by bandwidth-demanding applications. Following the evolution of 10 and 40 Gigabit Ethernet (GE), 100 Gigabit Ethernet is currently emerging as a promising candidate to fulfill the request for increased line speed.

Switching 100 GE signals requires that the selected switch architecture is scalable enough to accommodate high capacity transmission [1], as the 100 GE port speed easily accumulates to the Terabit range within the switch fabric. It is difficult, if not impossible, to obtain Terabit speeds in single stage switch fabrics. A promising alternative is therefore to use a multi-stage switch fabric where the traffic can be distributed on different chips. In particular, the Clos' architecture relies on three stages of switching modules, where each module connects to all the modules in the adjacent stages via a unique path [2], as illustrated in Figure 1. The modules are named Input Module (IM), Central Module (CM) and Output Module (OM), respectively [3].

In a Space-Space-Space (S3) architecture, which does not contain any buffers, switching is carried out only in the space domain. Since no buffers are available, an advanced scheduling algorithm is required to avoid blocking, which increases the complexity and implementation cost considerably due to hardware constraints at increasing speeds. The Memory-Space-Memory (MSM) architecture is bufferless in the middle stage (CM), which avoids the out-of-sequence problem. Out-of-sequence problems are caused by cells belonging to the same packet being distributed over multiple CMs, hence being delayed for different amounts of time in the individual CM queues. The Concurrent Round-Robin Dispatching (CRRD) scheme [3], developed to schedule cells to the middle stage efficiently, allows for 100% throughput under uniform traffic, but its performance drops significantly under unbalanced traffic. Scheduling for the MSM requires a request-grant-accept (RGA) handshaking scheme, whose implementation is non-trivial and costly. If buffers are used in all stages, the architecture is referred to as Memory-Memory-Memory. In this case the buffers at the input and output stages operate like those in the MSM architecture, and the buffers in the middle stage are organized as

output queues. This avoids contention in the central modules, which simplifies the scheduling scheme considerably, but may cause out-of-sequence problems. It has also been suggested to only place buffers in the central stage, leading to a Space-Memory-Space [4] architecture. But for this approach to be practical, schedulers are needed like those studied in [5]. Again this causes scalability problems when speeds increase, and they can be costly to implement in hardware.

Each internal module (i.e. IM, CM, OM) may be constructed as an individual crossbar switch. However, the sheer amount of data that must be switched in a Terabit system causes a number of challenges [6], especially related to delay, buffer and queue management.

To increase the switch throughput and reduce the delay, a speedup can be introduced where the internal link speed in the switch is higher than the interface speed [6]. A speedup can compensate for flow control in case of limited internal buffering resources or ensure faster transmission of packets over the switch fabric leading to a performance closer to an output buffered switch, which provides optimal throughput and is thus usually used as a reference for performance analysis. For multistage switches, speedup can also compensate for a non-optimal traffic distribution scheme (load-balancing).

Speedup in multistage switches, e.g. Clos' networks, can be obtained by link speedup or by expansion (explained in detail in the next section).

The behaviour of a Clos'-based switch fabric under uniform and bursty traffic without speedup or expansion has been studied in [8]. In this paper, we analyze different speedup and expansion methods, and evaluate their performance in terms of buffer usage, cell reordering and delay.

Fabric Speedup and Expansion

A decrease in buffer usage and delays can be achieved in two ways: Either by speeding up the internal connections (i.e., running them at a higher speed than the line rate); or by adding extra central modules (CMs) in the Clos' design. Both approaches are explained in the following sections, and are illustrated in Figure 1.

Using speedup, the cells are forwarded faster from the IM through the CM to the output modules (OMs).

We call this approach *internal speedup*. The aim of internal speedup is to lower the buffer usage in the CMs of the Clos' network. In addition to speeding up the lines between the CMs and the OMs, we investigate the effect of *full speedup*, where also the connection between the OM to the linecard is also sped up. *Expansion* describes the approach where more CMs than IMs/OMs are present in the switch fabric. The idea behind this approach is that the traffic can be distributed over a larger number of CMs, hence avoiding delays caused by CM-buffering. For a Clos' network the expansion factor is defined as the number of outputs divided by the number of inputs at the first stage. The approach is illustrated in Figure 1, where an extra module is added to the central stage.

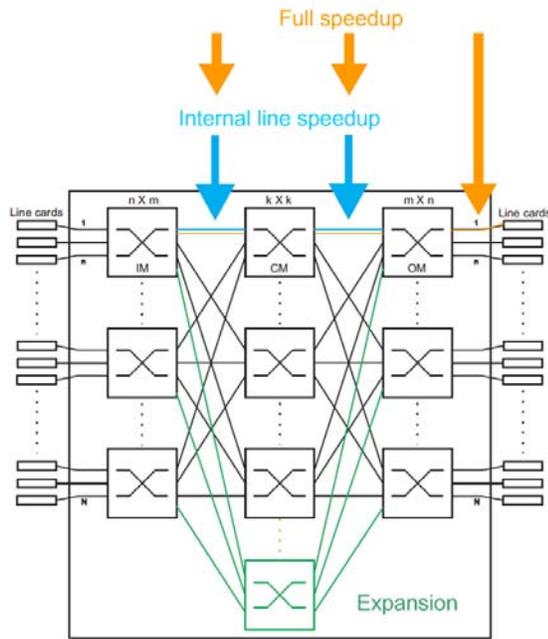


Figure 1: Clos' fabric speedup and expansion

OPNET Model of Switch

In this work, we focus on a switch with 16 ports configured as a Clos' (4,4,4) system, as illustrated in Figure 2. Any other configurations are however possible as well. Buffers are present in the CMs and the OMs leading to a Space-Memory-Memory configuration [10].

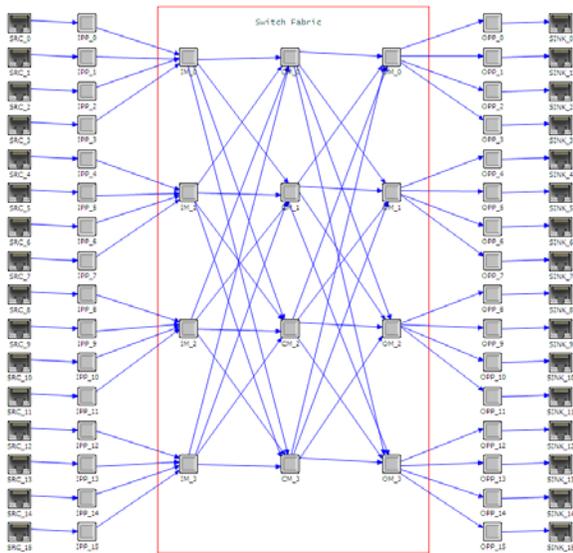


Figure 2: Switch configuration used for simulation study

Process models

To understand how the model works we need to take a look at how the switch processes are organized internally.

Source (SRC)

The source process is generating and sending packets towards the switch fabric. It generates a number of child processes that represent individual sources. The traffic rate of the entire source

is shared between all of its child processes. This means that a source with a specified rate of 100 Gbit/s and 10 individual sub-sources, every sub-source generates traffic at 10 Gbit/s. It should be noted that the possibility of using multiple sub-sources is designed to employ different traffic pattern, such as bursty (Pareto) and uniform (Bernoulli) traffic.

The Finite State Machine (FSM) can be seen on Figure 3. In the *init* state it creates the sub-sources, discovers the reachable destinations (sinks) and set up relevant data structures. It will then enter the *idle* state at the designated start time and here the child processes will be awakened, meaning traffic generation will start. The source can change the way traffic is distributed during the simulation, usually from uniform to unbalanced. It uses this temporary distribution for a given time interval after which it will change back. The *dist_change* state is used for this purpose, as the process will go from the *init* state to this state and stay here for as long as the temporary distribution lasts. Finally, if a stop time was given it will go to the *stop* state and destroy all of its child processes, thereby ending the traffic generation.

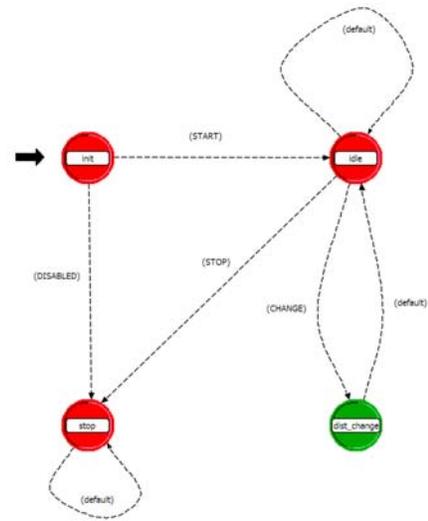


Figure 3: Parent source process

Source Child Process

As described before, the source child process is used to generate traffic and is controlled by the parent source process. The FSM for the child process is illustrated on Figure 4. In the *init* state all the necessary parameters are read and the needed distributions set up. If the chosen arrival method is Bernoulli, it will not use the off and on states, but will instead transition to the *idle* state and from there to the *pkgen* state. In this state packets are generated and sent depending on the level of utilization and other traffic parameters. As long as the child process is alive, the state machine alternates between the *idle* and *pkgen* states. However, if the ON-OFF Pareto method has been selected, the on and off states are used to start and stop traffic generation. When the process is on, traffic is generated just like in the Bernoulli case, and when it is off nothing is sent.

At the start of the simulation the process will transition from the *init* state, through the *idle* state, to the *off* state. Here the next on period is calculated and a self interrupt of type "on" is scheduled to this time. This results in an event that causes the child process to enter the on state at the scheduled time. Here it calculates the time until the next off period and schedules a self interrupt of

type "off" to this time, and in this way the child source alternates between being on and off. To calculate the on and off times two Pareto distributions are used.

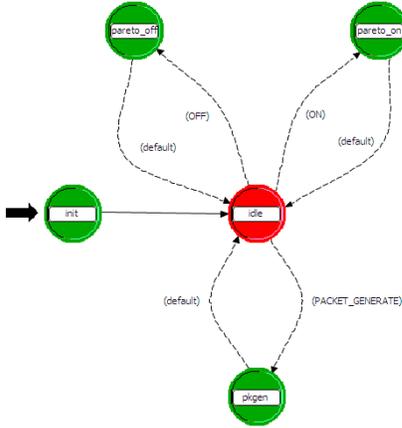


Figure 4: Child Source Process

Input Port Process (IPP)

The sources are connected to input port processes which are equivalent to traffic managers in a real switch. When packets arrive at the IPP they are split into cells of fixed using the segmentation and reassembly package provided by OPNET (SAR package). Using the SAR package allows for the cells to be treated as packets internally in OPNET, meaning they trigger the same packet stream interrupts. After a packet has been segmented, the resulting cells are buffered in Virtual Output Queues (VOQs). The FSM for this module is seen in Figure 5.

The *init* state is used for the usual functions, like reading parameters from the simulation console and to initialize variables. From the *init* state the process advances to the *idle* state. Here it awaits either a stream interrupt signalling a new packet has arrived from the source, or a self interrupt causing the process to transmit a cell. The *strm_int* state is triggered by the stream interrupts and deals with the received packets, after which the process returns to the *idle* state. The *self_int* state is used for transmitting cells at regular intervals (every timeslot) and it is triggered by a self interrupt. Finally, the *end* state is used when the simulation ends to collect statistics.

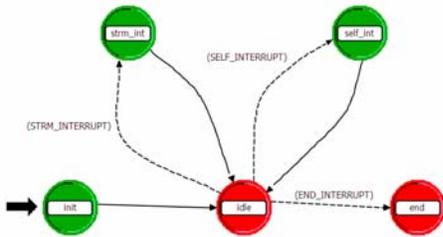


Figure 5: Input Port Process

Input Module (IM)

This module is a key module in the entire model, mainly because of its load balancing functions. It takes incoming cells and forwards them immediately to the CM's. The actual forwarding procedure depends on the kind of connection scheme used (e.g. static, random, round robin). The general layout and

functionality of the FSM in this module is quite similar to that of the IPP, but it does have one more state as can be observed in Figure 6. The *init* state is used for reading the parameters, initializing data structures, discovering the topology and so on. After leaving the *idle* state, the process will enter the *idle* state. To leave the *idle* state the IM must receive one of four interrupts: a stream interrupt indicating that a cell has been received from an IPP, a remote interrupt from either a CM or OM signaling backpressure changes¹, a self interrupt used for timeslot behavior - meaning it should send the received cells towards the CM's, or lastly an end simulation interrupt. In the *strm_int* state a received cell is inspected and it is determined to which CM it will be forwarded to. The *remote_int* state is used for handling incoming backpressure notifications. The *self_int* is used for the timeslot behavior (forwarding cells), and the *end* state is used for statistics collection, as in the IPP.

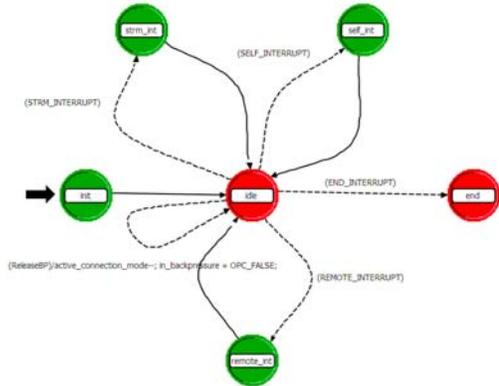


Figure 6: Input Module Process

Central Module (CM)

The CM takes the incoming cells from the IM and stores them in its queues, which can be organized in different ways. Each timeslot, the queues are served and cells are sent to the OM's. The layout of the FSM is the same as for the IM albeit the functionality of each state is a bit different, therefore no illustration of it is given – please see Figure 6 instead.

The *strm_int* state handles received cells, determines their destination OM and stores them in the internal queues. The *self_int* state serves these queues each timeslot, handling output contention for cells destined to the same OM. Every time the process returns to the *idle* state, the size of the buffers are checked and if an overflow is impending while backpressure is enabled, it will transmit remote interrupts to the relevant IM's. The *remote_int_state* is used for incoming backpressure signals that will halt transmission from certain queues.

Output Module (OM)

The OM is the last stage of the internal switch fabric, situated between the CM and the OPP. It forwards cells to the OPP at every timeslot, serving its internal queues. Like the queues in the CM, these can be organized in different ways. As with CM, the FSM looks similar to the one in the IM case seen on Figure 6.

¹ The model is designed to handle backpressure operations. However, we do not use any backpressure in the results presented here – hence backpressure is not explained in detail.

Therefore it also uses the *init* state for discovering the topology, creating data structures and so on. Likewise, the *idle* state contains buffer size checks, and it transitions from this state to either the *strm_int*, *self_int*, *remote_int* or the *end* state, depending on if it receives a cell, needs to send a cell, etc.

Output Port Process (OPP)

When the cells arrive at the OPP, they are reassembled to complete source packets that are forwarded to the destination sink. But before this operation the OPP checks if the cells arrive out of order, using the information conveyed in the cell ICI. If they do, they are temporarily stored in a re-sequencing buffer until the correct order is restored. The internal state-machine looks like the one in the IPP, as illustrated in Figure 5, but working in reverse order. This means the internal states are used in almost the same way, except cells are received and treated in the *strm_int* state and packets are sent in the *self_int* state. Additionally, the *self_int* state is also used to check for how long the cells have resided in the buffers. If they have been there longer than a specified flush time, they are removed as it is assumed some of the cells belonging to the same packets have been lost inside the fabric, thus the packet can never be completed.

Sink

The sink represents the output of the switch. It is a modified standard OPNET model that simply discards the arriving packets, while updating some statistics. The FSM is seen on Figure 7. After the simulation has started, it transitions from the *idle* state to the *discard* state, where it awaits packet arrivals. Basically it just loops the discard state every time a packet arrives.

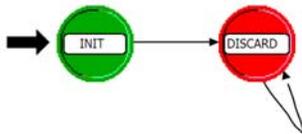


Figure 7: Sink Process

Simulation Scenario and Results

The performance of internal speedup, full speedup and expansion is evaluated in a simulation study using OPNET Modeler [9], and compared to a switch that does not use speedup for benchmarking purposes.

Simulations are carried out on a switch with 16 ports configured as a Clos' (4,4,4) system, as illustrated in Figure 2. Buffers are present in the CMs and the OMs (Space-Memory-Memory configuration [10]). The traffic is of the uniform Bernoulli type and the connection scheme within the switch is Desynchronized Static Round Robin (DSRR) [10]. The incoming packets are of variable size, and are split into fixed sized cells of 512 bits for transmission over the switch fabric. Simulations are carried out under heavy traffic load of 0.99. The line speed is 150 Gbit/s for the 1.5 speedup ratio and 200 Gbit/s for the 2.0 speedup ratio, related to the input speed of 100 Gbit/s.

Using speedup, the cells are forwarded faster from the IM through the CM to the OM, which should potentially lower the buffer usage in the CMs. The resulting reduction in buffer usage in the CMs is illustrated in Figure 8. The original switch without

speedup (i.e., Ratio 1.0) has a mean buffer size of 1643 cells, please note the different axis for this plot. The buffer usage clearly decreases when the line is sped up internally, both for the Ratio 1.5 case and even more if the internal line speed is run at twice the external line speed, i.e. Ratio 2.0. Furthermore, it is interesting to note that the line speedup performs better than adding more CMs using expansion.

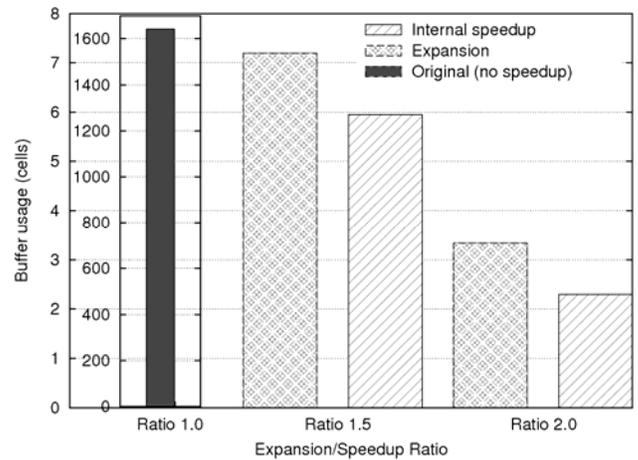


Figure 8: Mean number of cells in the CMs.

As seen in Figure 8, the buffer usage in the CMs is clearly reduced by both internal speedup and expansion. Next we evaluate how these two methods affect the buffer usage in the OMs, which is illustrated in Figure 9. The results clearly show that there is a bottleneck in the OM's. With expansion ratios above 1.0, the OMs are overloaded with traffic and cannot dispatch it fast enough. The original switch without speedup actually has less cells residing in the OMs, but this is because they are situated in the CMs instead.

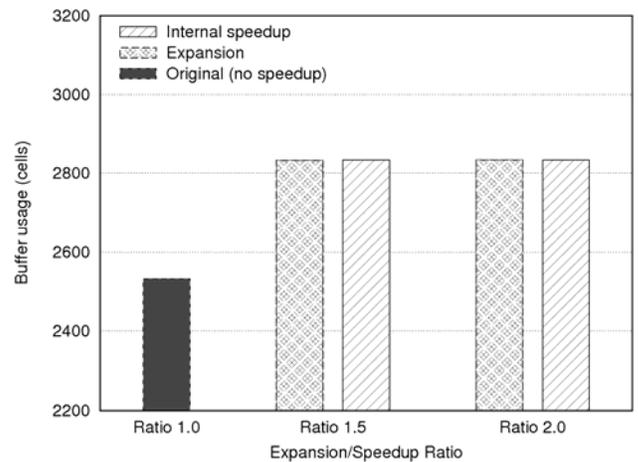


Figure 9: Mean number of cells in the OMs (using internal speedup only)

The effect of the long CM queues can be seen in the resequencing buffer, illustrated in Figure 10, where the amount of cells arriving out-of-sequence is shown. The result for the original switch is that 50% of the cells arrive out-of-sequence (not plotted here). In comparison, the switch configurations with

internal speedup and expansion benefit from a lower amount of out-of-sequence cells, below 5% in all cases. This confirms that the amount of out-of-sequence cells is directly tied to the buffer usage in the CMs. The internal speedup again outperforms the expansion method.

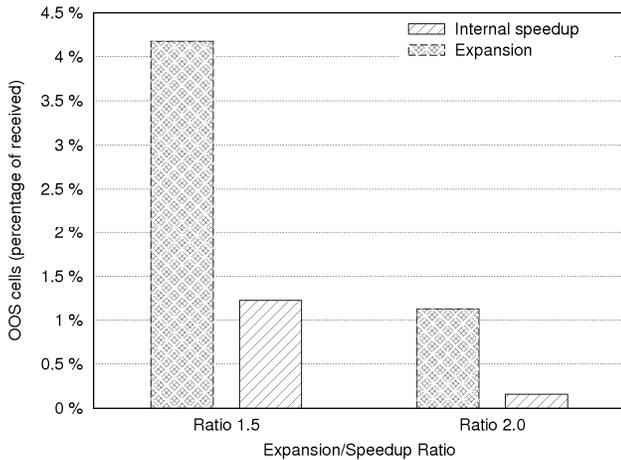


Figure 10: Percentage of Out-of-Sequence (OOS) cells

The end-to-end delay is shown in Figure 11. The original switch has the worst delays, but due to the bottleneck in the OMs neither speedup nor expansion provides significant improvements. While the combined mean buffer usage in the CMs gets lower as the expansion ratio increases, the end-to-end delays does not change significantly due to the bottleneck at the OMs.

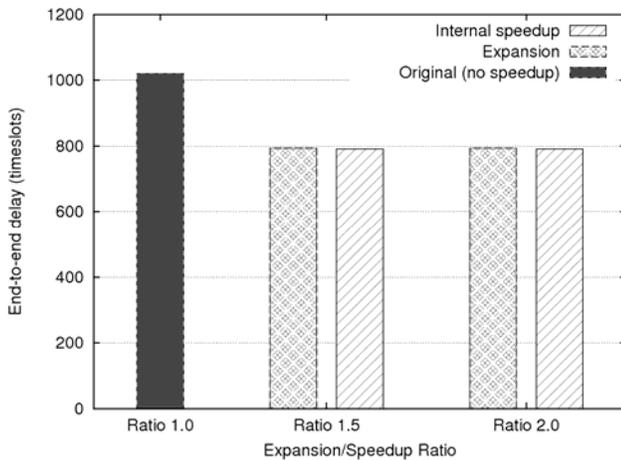


Figure 11: Mean end-to-end delay (using internal speedup only)

A full speedup from the OM to the linecard could potentially alleviate the bottleneck. Results of the OM buffer usage are shown in Figure 12. The buffer usage in the OMs is significantly decreased compared to both the original switch without speedup as well as for internal speedup (shown on Figure 9). Furthermore, the performance of expansion is improved by full speedup since it also benefits from the speedup from the OMs to the linecards. Speedup still slightly outperforms expansion. The CM usage stays the same for both internal and full speedup (not shown here).

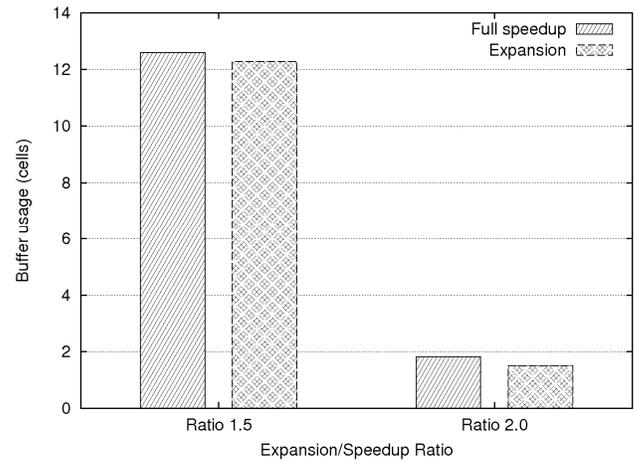


Figure 12: Mean number of cells in the OMs (using full speedup)

The significant drop in mean buffer usage in the OMs affects the end-to-end delay for the packets, which is illustrated in Figure 13. The packet delays for full speedup shown here are much lower than in the internal speedup case (seen on Figure 11).

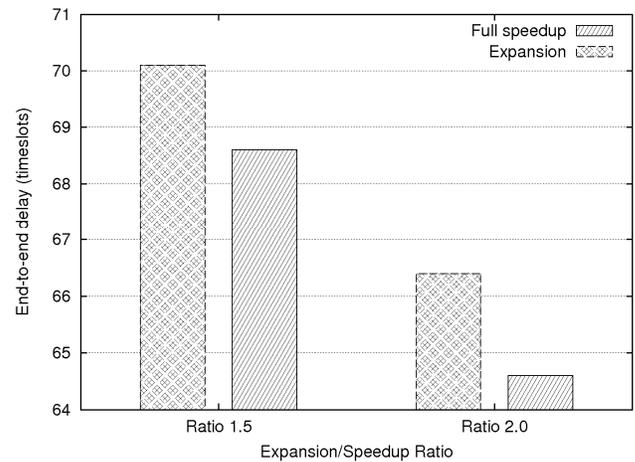


Figure 13: Mean end-to-end delay (using full speedup)

Conclusion

In this paper, we analyze speedup and expansion for a Terabit switch fabric. Simulation results show that an increase in speedup or expansion ratio is most beneficial when the output line of the switch is also run at a higher speed. This lowers the buffer usage and thereby the delays in the switch. A reduction in CM buffer usage also has the benefit that fewer cells arrive out-of-sequence. Finally we see that speedup outperforms expansion.

Acknowledgment

This work has been partially supported by the Danish Advanced Technology Foundation (Højteknologifonden) through the research project "The Road to 100 Gigabit Ethernet".

References

- [1] C. Hermsmeyer et al., "Towards 100G packet processing: Challenges and technologies," Bell Labs Technical Journal, vol. 14, no. 2, 2009.
- [2] C. Clos, "A study of non-blocking switching networks," Bell Systems Technical Journal, pp. 406–424, 1953.
- [3] E. Oki, Z. Jing, R. Rojas-Cessa, and H. Chao, "Concurrent Round-Robin-Based Dispatching Schemes for Clos-Network Switches," in IEEE/ACM Transactions on Networking, 2002.
- [4] F. Wang and M. Hamdi, "Analysis on the Central-stage Buffered Clos-network for packet switching," in IEEE ICC, 2005.
- [5] J. Kleban and S. Piotrowski, "Performance Evaluation of Selected Packet Dispatching Schemes for the CBC Switches," in Communications Letters, IEEE, 2009.
- [6] "Facing the Challenges Of Developing 100 Gbps Platforms," Road to 100G Alliance, 2008, <http://www.ethernetalliance.org>.
- [7] C. Shang-Tse, A. Goel, N. McKeown, B. Prabhakar. "Matching output queueing with a combined input/output-queued switch", IEEE Journal on Selected Areas in Communications, Jun 1999, Volume: 17 Issue:6 On page(s): 1030 - 1039
- [8] S. Ruepp, A. Rytlig, A. Manolova, M. Berger, H. Wessing, H. Yu, and L. Dittmann, "Performance evaluation of 100 Gigabit Ethernet switches under bursty traffic," in In proc. of 15th International Conference on Optical Network Design and Modeling (ONDM), Bologna, Italy, 2011.
- [9] OPNET Technologies, Inc., <http://www.opnet.com>.
- [10] X. Li, Z. Zhou, and M. Hamdi, "Space-Memory-Memory Architecture for CLOS-network Packet Switches," in IEEE International Conference on Communications (ICC), 2005, pp. 1031–1035.