An Improved Control Algorithm for a Class of Photonics Timeslot Interchanger

Luai E. Hasnawi and Richard A. Thompson
Graduate Telecommunications and Networking Program,
School of Information Sciences, University of Pittsburgh,
Emails: {leh31,rthompso}@pitt.edu

***Abstract*—All-Optical Networks (AONs) are an active research area for use in Optical Transport Networks (OTNs) and data centers. Adding Optical Time Division Multiplexing (OTDM) to AONs can more efficiently utilize the enormous amount of bandwidth in the fibers. Photonic Timeslot Interchangers (PTSIs) are used in OTDM networks to interchange/switch timeslots. This paper proposes a control algorithm for a previously proposed PTSI that uses multiple feed-forward fiber delay-lines. This algorithm further reduces the required number of delay elements needed for non-blocking operation from the reduction previously reported under a less optimal algorithm. Any reduction in the number of PTSI components should result in increased output signal power, reduced footprint, and reduced manufacturing cost.**

Keywords–Photonic Switching; Optical Time Switching; Photonic Timeslot Interchanger; Circuit Switched; Control Algorithm.

I. Introduction

Cloud computing services are solutions to minimize the initial cost of building new businesses. The enormous amount of data that is processed, stored and shared in the cloud is beyond the capacity of common copper lines. Lately, optics researchers have been proposing a high speed switching system to connect server racks by fiber optics [1][2].

The inflation of file sizes, as well as the transmission rates required by recent applications, has attracted researchers attention to increasing network utilization. Cisco forecasts that IP traffic will reach 1.6 zettabytes per year by 2018 [3]. A common practice to help increase bandwidth is to add more fibers to the network; however, that is an expensive solution. Part of the research investment is focusing on switching the data between data centers and clients in the optical domain using circuit-switched networks [2].

By going back in time to the origin of optical switching network research, we would notice that the majority of research focuses on space switching. Switching in the space domain refers to the operation of switching the signal from one physical fiber to another. Meanwhile, time and wavelength domains have received a reasonable amount of attention. Switching in wavelength domain is a common practice, nowadays. However, switching Optical Time-Division Multiplexed (OTDM) signals in the optical domain has not been commercialized.

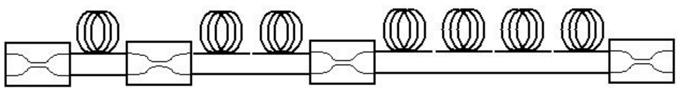
Figure 1. A delay element for 4 timeslots per frame

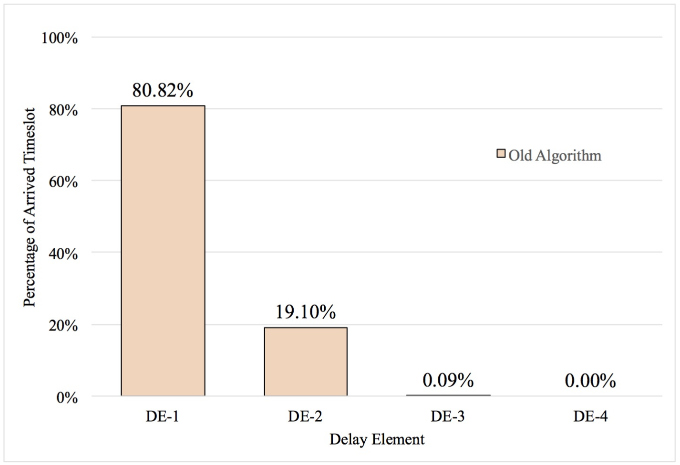

Figure 2. Results from the old control algorithm for 4 timeslots per frame [4]

In OTDM, channels are divided based on time. For the purpose of this paper, we assume that time channels (presented by timeslots) have a fixed duration of time. In addition, this paper presents a circuit switched network with call setup before starting data transmission. For better network utilization, an OTDM network does not require timeslot continuity, in which there has to be at least one timeslot vacant in every link to establish the connection.

If timeslot continuity is required, the exact timeslot index has to be vacant in every hop to establish the connection.If timeslot continuity is not required, a Photonic Timeslot Interchanger (PTSI) should be used at every node. Timeslot interchanging is the operation of switching timeslots with each other. Studies have proven that PTSIs improve system utilization and reduce network probability blocking [5][6].However, none of these studies have used the exact PTSI that was proposed by Thompson [7]. Building a PTSI consists of a number of Delay Elements (DE) connected together. The structure of a DE is presented in Figure 1.

The rest of this paper is organized as follows: Section II describe the motivation behind this work. The proposed control algorithm is described in Section III. The results are presented in Section IV. Finally, the conclusion and future works are presented in Sections V and VI, respectively.

II. Motivation

The result from the first control algorithm [4] that controls such a PTSI shows that some of the DEs have been utilized less than 1% of the time, as shown in Figure 2.

Hence, we assume that if we make the algorithm more sophisticated, we might obtain better results and eliminate additional DEs. There are a number of benefits to reducing the number of DEs in the PTSI including:

- Reducing the number of hardware components in the fabric increases the overall availability [8].
- Adding more hardware to the fabric increases the cost, footprint and control algorithm complexity. A simple control algorithm reduces the fabric development and manufacturing time [8].
- As the light beam passes through optical components, it suffers from losses, mainly insertion loss. Hence, reducing the number of components that the signal passes through improves the signal strength.

III. Proposed Model

Building a PTSI has been discussed in great detail in Thompson's initial study [7] and in Hasnawi and Thompson [4]. There has not been any change to the PTSI components. In this paper, our focus is to improve the control algorithm to eliminate as many additional DEs as possible. PTSI consists of DEs similar to the one in Figure 1. DEs are connected to form a binary tree on both sides, as in Figure 3. In this paper, we will generalize the switching components (splitters, switches, and combiners) using the term *Switching Module* (SWM). We assumed that the links between SWMs are perfect connectors with no loss or delay.

A. Switching Assignment Matrix

A Switching Assignment (SWA) is defined as permuting input timeslots with output timeslots. Each input timeslot, S_i^{in} , may be switched into *TS* possible output timeslots S_j^{out}. There are $TS!$ possible SWAs. All possible SWAs form a *[TS!][TS]* matrix. Each row is indexed from *0* to *TS!-1* . Every SWA is presented in cyclic notation. For example: SWA (a)(bc) is read as Timeslot (a) at frame *F-1* is assigned to be switched to Timeslot (a) in the next frame, *F*, while Timeslot (b) and (c) at frame *F-1* are assigned to be switched to Timeslots (c) and (b) at frame *F*, respectively. The SWA index is assigned to the simulation during the initialization of each run. Since there are *M = 2* frames processed for every run, this study has two cases:

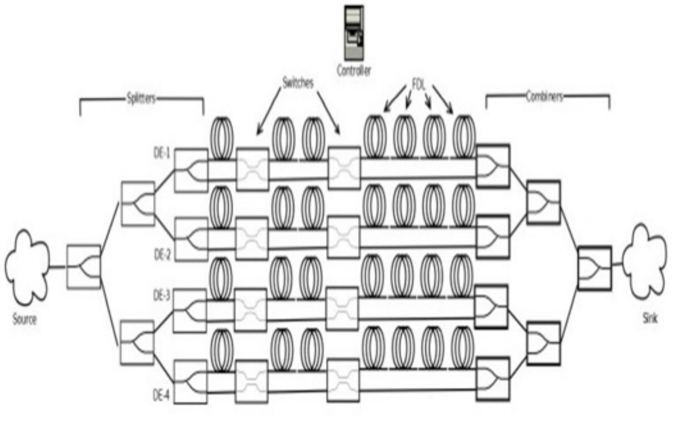

Figure 3. A complete PTSI for 4 timeslots per frame

- Case 1: Static Switching Assignment: both frames have the same switching assignment. Thus, there are (*TS!*) total number of possible permutations.
- Case 2: Dynamic Switching Assignments. Frames are independent from each other. Hence, Timeslots in the frame *F-1* should have independent SWA from *F*. In this case, there are *(TS!)M* total number of Switching Assignments.

In this study, we are going to apply every possible SWA for TS = 2, 4 and 8 for static cases. In addition, for the dynamic case, we are trying every SWA for TS = 2 and 4. We will omit TS = 8 for the dynamic switching assignment because the total number of runs required for every possible permutation would be greater than 1.6 billion.

TABLE I. Sample of the switching assignment for TS=4.

SW Index	Input Timeslot			
	S_0	S_1	S_2	S_3
0	0	1	2	3
1	0	1	3	2
2	0	2	1	3
.	.	.	.	.
.	.	.	.	.
.	.	.	.	.
23	3	2	1	0

B. Delay Matrix

Considering a synchronized system, every SWM must be pre-set during guard time, prior to the timeslot. Any misbehavior may affect the overall system and result in system failure. Some such scenarios include:

- If a timeslot arrives at SWM that is pre-set to an undesired switching state, the timeslot will exit at an undesired port.
- If two timeslots arrive at a switch simultaneously, each requiring a different switching state, at least one will be switched to an undesired output.
- If two timeslots arrive at a combiner simultaneously, at least one timeslot will be blocked.

Every SWM must be reserved (set to busy) and pre-set depending on the delay required for the timeslot. Computing the delay required to switch timeslots S_i^{in} to S_j^{out} is given by $D = TS{+}j{-}i$; where *j* is the timeslot output index and *i* is the timeslot input index. For example, assuming TS = 4 and SW index = 2 from Table I, we can extract that the cyclic notation for this assignment is $(S_0^{in})(S_1^{in}\ S_2^{in})(S_3^{in})$. Using the delay equation, S_0^{in} and S_3^{in} each must be delayed by a duration equivalent to 4 timeslots, while S_1^{in} must be delayed by 5 timeslots, and S_2^{in} must be delayed by only 3. The required switching operation, as well as the required delay per timeslot, is illustrated in Figure 4. This first stage of the PTSI is the splitter stage, as in Figure 3, which forms a perfect binary tree with a total number of leaves $l = TS$ (equal to the number of DEs), and the depth of the tree given by $d = log_2 l + 1$. Thus, each timeslot passes through d splitters before it enters a DE. The combiner stage is identical to the splitter stage but on the other side of the fabric. Splitters do not perform interchange operations, thus, no delay at this stage, as seen in Table II.

However, switches and combiners do require delays before they change their status. Going back to the example in the

TABLE II. DELAY MATRIX FOR TS=4

	Splitters Stage			Switches Stage		Combiners Stage		
	0	*1*	*2*	*3*	*4*	*5*	*6*	*7*
Delay 0	0	0	0	0	0	0	0	0
Delay 1	0	0	0	1	1	1	1	1
Delay 2	0	0	0	0	2	2	2	2
Delay 3	0	0	0	1	3	3	3	3
Delay 4	0	0	0	0	0	4	4	4
Delay 5	0	0	0	1	1	5	5	5
Delay 6	0	0	0	0	2	6	6	6
Delay 7	0	0	0	1	3	7	7	7

previous paragraph, switching S_0^{in} to S_0^{out} requires a delay with a duration of 4 timeslots. Thus, every splitter and switch stage in Figure 3 must change its stage during the guard time prior to the timeslot. The combiner stage can be set to busy and change its switching state during the guard time, prior to the timeslot transmission, until the timeslot arrives. However, doing so will prevent other timeslots from using these combiners from time t=0 to t=4. Therefore, to better utilize the fabric, the combiner stage will change its state after a delay of 4 timeslots.

The delay required for each SWM to change its state while efficiently utilizing the fabric is presented by a $[D_{max}][totalNumberOfSWMinThePath]$ matrix, as shown in Table II. D_{max} is defined as the maximum delay required to interchange the first timeslot, S_0^{in} , with the last timeslot, S_{TS}^{in} , and is given by $D_{max} = 2TS - 1$.

The start holding time (*startHoldingTime*) is a parameter used in the simulation. It is a *timeStamp* at which the SWM should start holding its current state before releasing to the idle state and being set to free. Using Table II, $startHoldingTime = [currentSimulationTime] + [TS_{dur} * Delay]$. Once a timeslot exits the SWM, then SWM is set to free. For M=2 and TS=4, the initial module [7] utilizes each DE one fourth of the time.

C. Select Path Algorithm

Select Path algorithm is what distinguishes this work from others. This algorithm is responsible for switching timeslots to the optimum DE in the PTSI. The first proposed PTSI model did not implement a path selection (or routing) algorithm.

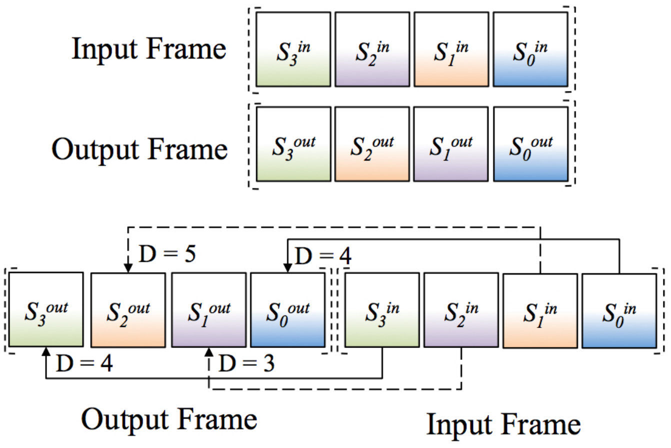

Figure 4. A graphical representation of PTSI operation for SWA index 2 with the required delay per timeslot.

```
selectPath (delay, startHoldingTime)

Input: delay, startHoldingTime
Output: path, SWCIndex
path ← 0
SWCIndex ← TS * (delay-1)
pathFound ← false
do
    for all switches j in path p where {j: number of switches in path p }
        Con. 1: in path p, at least one queue corresponding to switch j is not empty.
        Con. 2: in path p, at least one switch j is set to busy at startHoldingTime.
        Con. 3: in path p, at least one queue corresponding to switch j is in conflict.
    If (condition 1 == false)
        pathFound = true
        return path, SWCIndex
    else if (condition 1 == true && condition 2 == false)
        pathFound = true
        return path, SWCIndex
    else if (condition 1 == true && condition 2 == true && condition 3 == false )
        pathFound = true
        return path, SWCIndex
    else
        path++
        SWCIndex ++
while (!pathFound)
* Conflict occurs when 2 TS require different SWC at the same startHoldingTime
```

Figure 5. Select path algorithm

It assumed that blocking was avoided by switching S_i^{in} to DE_i. Hence, every row should be utilized $M/(M * TS)$ of the time. This result led us to build a new path selection algorithm that would reduce the number of DEs in the PTSI, and increase the utilization to the maximum while maintaining zero probability of blocking.

The first select path algorithm was published by Hasnawi [4]. The algorithm did not allow 2 timeslots to travel through a switch simultaneously even if both required the same switching state.

The improved algorithm differs from the previously published algorithm by allowing 2 timeslots to pass through a switch simultaneously, if, and only if, both required the same switching state. The algorithm is presented in Figure 5.

D. Switching Control

Switching Control (SWC) is a control signal that travels from the controller to the SWM. The purpose of this signal is to change the switching state of SWM. Switching control contains three fields:

- Switching State (Bool): changes the state of an SWM to *BAR* (0) or *CROSS* (1).
- Busy (Bool): if the SWM is reserved, then it is set to busy (1); otherwise, it is set to free (0).
- *startHoldingTime* (double): is the time at which the SWM should be switched to the new state and set to busy.
- *releaseTime* (double): is the time at which the SWM releases its current state and is set to free.

We assume that every SWM is a Lithium Niobate ($LiNbO_3$) directional coupler, which only requires a simple SWC, presented as a voltage. Hence, only the switching state is passed from the controller to SWM and the rest of the fields are used inside the controller.

Once a path is selected, the controller calls another algorithm to reserve the path for the incoming timeslot. The algorithm is presented in Figure 6.

E. Insert Switching Control

Every SWM has a SWC queue at the controller. Switching Controls are inserted in order, based on the $startHoldingTime$. If two or more SWC for a given SWM have the same switching state at $startHoldingTime$, then only one SWC will be stored in the queue and the rest will be discarded.

F. Send Switching Control

At every guard time prior to any timeslot, each head of the queue is examined. If the current simulation time, $simTime$ equals $startHoldingTime$, then the SWC is popped and sent to its corresponding SWM; otherwise, the queue will be ignored and examined at the next guard time.

G. Simulation Tool and Assumption

We used an Omnet++ [9] simulation tool to build every component in our PTSI. For each simulation run, we assumed the following:

- There were M=2 frames per run.
- Each frame had TS = 2, 4 or 8 timeslots.
- The size of each timeslot was fixed and equal to 10^6 bits.
- The data rate equaled $R = 10^9$ bps
- A Timeslot Duration (TS_{dur}) equals the timeslot size divided by the data rate.
- There was a guard time between consecutive timeslots. This guard time equaled the SWM switching speed.
- Timeslots SWAs' follow the permutation table, discussed in subsection A.
- There is frame integrity, in which timeslots are switched within the boundary of the frame.
- Each run starts with an empty fabric; then, timeslots are generated at the deterministic rate of TS_{dur}.
- Every channel is a point-to-point connection.
- Each simulation run is independent. For each run, we verified that each timeslot generated by the source was received at the destination.

```
reservePath (path, delay)

Input: path, delay

    for all switches j in path p where {j: number of switches in path p }
            setBusy ();
            SwitchingControl SWC_j;
            SWC_j → setHoldingTime (simTime() + getDelay[j][delay] *
                      getTimeslotDuration())
            SWC_j → setReleaseTime (simTime() + getDelay[j][delay] *
                      getTimeslotDuration() + getGuardTime())
            SWC_j → setSwitchingState (path,j)
                  if ( duplicate* exist )
                       delete SWC_j
                  else
                       insertToOrderedQueue (SWC_j)
```

*** Duplicate happens when a *SWC* has previously inserted with the same *switchingState* for the same *startHoldingTime* discussed in subsection G.**

Figure 6. Reserve path algorithm

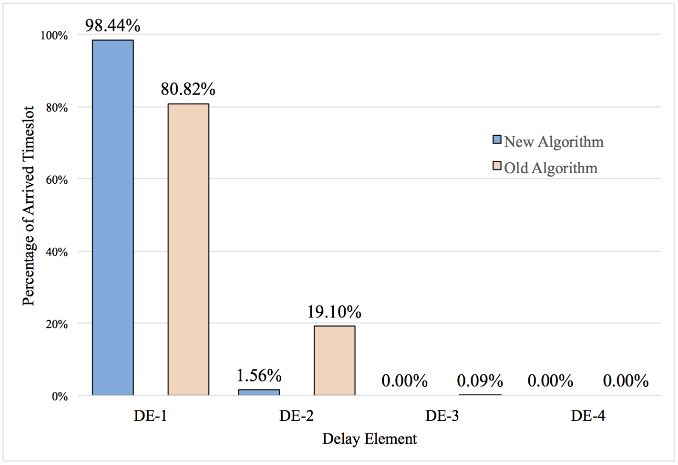

Figure 7. Comparing the old and new Select Path algorithm for TS=4. Each bar represents the utilization of the DE

IV. Results

The new algorithm provides a significant improvement. We were able to further reduce the number of DEs for every case. The results show that for 2, 4, and 8, the number of DEs required to provide non-blocking PTSI is 1, 2, and 2, respectively. Note that, in this study, we only simulated the static case for TS=8 and we were able to interchange every timeslot with only two DEs.

The number of DEs required to interchange TS = 4 timeslots per frame using the improved algorithm is illustrated in Figure 7. Similarly, the number of DEs required to interchange TS = 8 timeslots per frame for the static case using the improved algorithm is illustrated in Figure 8.

From a software perspective, reducing the size of a PTSI results in easier and faster control. Searching for a path from three available paths is faster than searching from eight.

Lastly, eliminating half the DEs from a PTSI for TS = 4, not only reduced the footprint size, but also improved the received signal power at the destination node. Each timeslot passed through 6 SWMs instead of 8, and passed through 9 timeslots instead of 11 for TS = 8. The average insertion loss for SWM was between 4 and 5dB for wavelength =1550 [10][11][12]. Hence, reducing the PTSI size improved the received power signal by an average of $10dB$ for TS = 4 and TS = 8.

V. Conclusion

In this work, we were able to further reduce the number of DEs in a PTSI. The algorithm improvement focused on additional reductions on the number of DEs while maintaining the same code complexity.The impact of this additional improvement is a scalable system. The number of channels (presented in timeslots) does not increase proportionally with the size of the PTSI. We were able to reduce an additional 25% of the number of DEs for TS = 4, compared with the previous work [4].In addition, for TS = 8, we reduced the number of DEs by an additional 12.5%, compared with the previous work. Both works show that the dynamic case requires one additional DE than the static case to provide non-blocking interchanging. If this statement is generally true, then for TS = 8, the

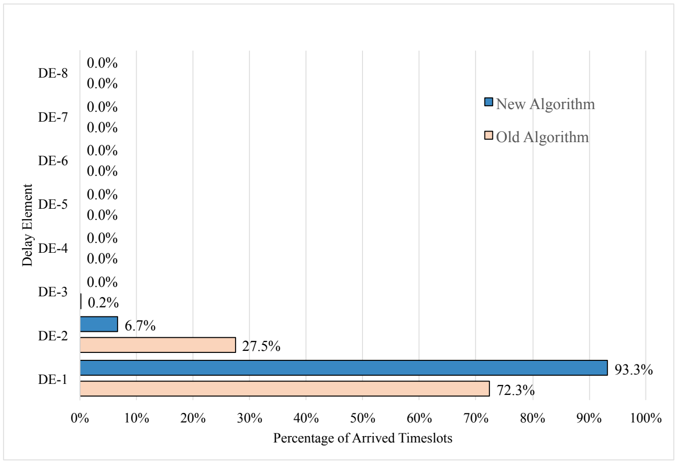

Figure 8. Comparing the old and new Select Path algorithm for TS=8 (static case). Each bar represents the utilization of the DE

required number of DEs to provide non-blocking switching is 3. These six scenarios (TS = 2, 4, and 8 for both static and dynamic cases), result in reducing the number of DEs required for non-blocking timeslot interchanging to $log_2(TS)$; however, it needs to be proven for $TS = 2^N$. The improvement on the number of DEs from the initial work [7], on the first (old) control algorithm [4], to the second (improved) control algorithm is presented in Figure 9. Additional benefits from this improvement include reducing the footprint, circuit temperature, power loss and monetary cost.

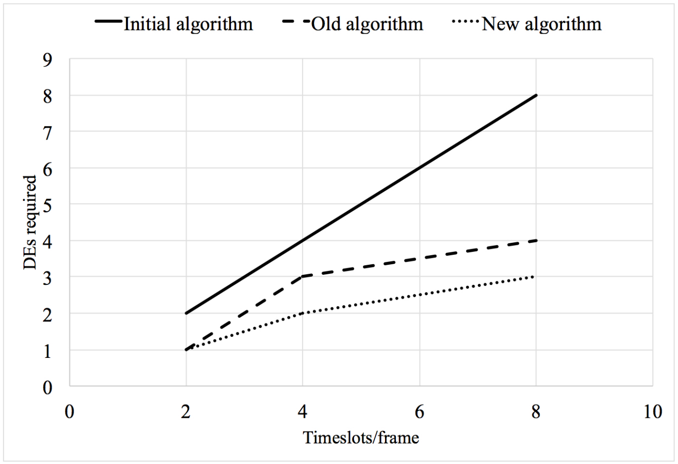

Figure 9. PTSI improvement over time based on the number of DEs

VI. Future Work

This simulation starts with an empty network and then the source starts generating timeslots until the network reaches 100% utilization. The work will be tested on a testbed network, such as NSFNet under different loads. In addition, this work could extend to larger number of timeslots per frame such as 64, 128, . 2^n.

Acknowledgment

The authors would like to thank Taibah University for supporting and partially sponsoring this work. We would like to thank Rakan and Anh for help in debugging and improving the code.

References

[1] C. Develder, M. De Leenheer, B. Dhoedt, M. Pickavet, D. Colle, F. De Turck, and P. Demeester, "Optical networks for grid and cloud computing applications," Proceedings of the IEEE, vol. 100, no. 5, 2012, pp. 1149–1167.

[2] C. Kachris and I. Tomkos, "A survey on optical interconnects for data centers," Communications Surveys & Tutorials, IEEE, vol. 14, no. 4, 2012, pp. 1021–1036.

[3] C. V. N. Index, "The zettabyte era–trends and analysis," Cisco white paper, 2013.

[4] L. E. Hasnawi and R. A. Thompson, "Photonic timeslot interchangers with a reduced number of feed-forward fiber delay lines," Procedia Computer Science, vol. 34, 2014, pp. 47–54.

[5] J. Yates, J. Lacey, and D. Everitt, "Blocking in multiwavelength tdm networks," Telecommunication Systems, vol. 12, no. 1, 1999, pp. 1–19.

[6] Y. C. Huei, P. H. Keng, and N. Krivulin, "Average network blocking probabilities for tdm wdm optical networks with otsis and without wc," in Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2007. MASCOTS'07. 15th International Symposium on. IEEE, 2007, pp. 424–431.

[7] R. A. Thompson, "Optimizing photonic variable-integer-delay circuits," in Photonic Switching. Springer, 1988, pp. 158–166.

[8] M. Zdeblick, "Design variables prevent a single industry standard," Laser Focus World, vol. 37, no. 3, 2001.

[9] "Omnet++ discrete event simulator," http://www.omnetpp.org/, accessed: 2015-03-01.

[10] X. Ma and G.-S. Kuo, "Optical switching technology comparison: optical mems vs. other technologies," Communications Magazine, IEEE, vol. 41, no. 11, 2003, pp. S16–S23.

[11] "Coralign low loss moving fiber optical switches," http://luminos.com/products/switches/downloads/switch_datasheet_detail.pdf, accessed: 2015-03-01.

[12] "Jdsu 2x2 interferometric switch," http://www.jdsu.com/en-us/Optical-Communications/Products/a-z-product-list/Pages/switch-2x2-interferometric-lithium-niobate.aspx#.VP0albPF840, accessed: 2015-03-01.