INFOCOM 2024 1570935983

A Parallel Algorithm and Scalable Architecture for Routing in Beneš Networks

Rami Zecharia
School of Electrical Engineering
Tel-Aviv University, Israel
ramiz99@yahoo.com

Yuval Shavitt
School of Electrical Engineering
Tel-Aviv University, Israel
shavitt@eng.tau.ac.il

***Abstract*—Beneš/CLOS architectures are common scalable interconnection networks widely used in backbone routers, data centers, on-chip networks, multi-processor systems, and parallel computers. Recent advances in Silicon Photonic technology, especially MZI technology, have made Beneš networks a very attractive scalable architecture for optical circuit switches.**

Numerous routing algorithms for Beneš networks were developed starting with linear algorithms having time complexity of $O(N \log_2 N)$ steps. Parallel routing algorithms were developed to satisfy the stringent timing requirements of high-performance switching networks and have time complexity of $O((\log_2 N)^2)$. However, their implementation requires $O(N^2 \log_2 N)$ wires (termed connectivity complexity), and thus are difficult to scale.

We present a new routing algorithm for Beneš networks combined with a scalable hardware architecture that supports full and partial input permutations. The processing time of the algorithm is limited to $O((\log_2 N)^2)$ steps (iterations) by potentially forfeiting routing of a few input demands; however achieves close to 100% utilization for both full and partial input permutations. The algorithm and architecture allow a reduction of the connectivity complexity to $O(N^2)$, a $\log N$ improvement over previous solutions.

We prove the algorithm correctness, and analyze its performance analytically and with large scale simulations.

***Index Terms*—Beneš Network, Switch Fabric, Optical Switch**

I. Introduction

Crossbar networks, the simplest solution for parallel switches, contain N^2 crosspoints (for $N \times N$ switch) and thus are too costly for large N [1, Ch. 13]. Clos [2] suggested a three-layer network of crossbars that reduces the cost of the switch to only $2\sqrt{2}N^{3/2}$. Indeed, CLOS topologies have become a standard in large scale routers [1].

Beneš networks [3] are a recursive construction of a CLOS network, which result in a multi-stage switching network. It uses 2x2 switching elements (SEs) arranged in a matrix of $N/2$ columns by $2\log_2(N)-1$ rows producing a rearrangeable non-blocking multistage interconnection network [4]–[6]. 'Routing' in a Beneš network refers to the process of producing the state setting of all the SEs in the Beneš network in order to satisfy an input–output routing demand.

The gap between the rapid growth of datacenter traffic and slower growth in Electrical Packet Switches (EPSs) capacity is increasing and is expected to be even worse as EPSs nearly reach their limit in terms of capacity and power consumption, while cloud traffic roughly doubles every year [7], [8].

EPSs require optical-electrical conversion and increased number of SERDES lanes with the increase of switch capacity [9]. High-radix EPSs have scalability issues due to high power consumption, large Silicon size, and cost. Furthermore, as Moore's law approaches its limits, moving beyond 5nm CMOS technology will be extremely challenging.

To resolve the electrical bottleneck of power, size, and cost, Optical Circuit Switches (OCSs) are considered as a replacement for EPSs [10]–[13]. OCSs are transparent to data rates, they can have similar or higher radix than typical EPSs, they are typically used as a cross-bar without any packet processing or buffering hence consume less power than EPSs, and with OCSs there is no need to convert back and forth from electrical signals to light [14]. However, many of the popular technologies used to build OCS suffer from significant drawbacks [15], which hinders their usage for data switches. In particular, micro-electro-mechanical systems (MEMS) have long switching times, while arrayed waveguide grating routers (AWGRs) require expensive multi-color lasers.

Mach–Zehnder interferometer (MZI) is an attractive technology for OCS. Its basic building block is a 2x2 switch that is naturally used to build Beneš networks. Beneš networks are also an attractive architecture for networks-on-chip [16]. In 2014, an experimental recirculating loop through a 2×2 switch, equivalent to a propagation through a 128×128 Beneš network, has been demonstrated [17]. In 2016, a 16×16, 32×32 and 64×64 MZI based Beneš network OCSs were demonstrated [18], [19]. Recently in 2021, capability of building a 256×256 Beneš network based OCS was analyzed and evaluated [20].

The fast switching time (a few nano-seconds) of MZI switches facilitates large high-speed switches, but Beneš networks suffer from routing algorithms that are difficult to scale largely due to the significant amount of wiring required to connect the routing processing elements that are used by today's algorithms. The *connectivity complexity*, which is the number of wires require to connect the switching element control, had only recently emerged as an important obstacle in scaling Beneš networks, since the wires occupy significant space.This work suggests a significant improvement in the connectivity complexity, and thus removes a significant barrier in the scaling of Beneš networks.

The first Beneš routing algorithms were serial [21]–[23] and as a result required at least $O(N \log_2 N)$ operations to

complete. This makes them impractical for large switches.

In the early 1980s, two teams suggested simultaneously parallel routing algorithm [24], [25] with a time complexity of $O((\log_2 N)^2)$. Both algorithms used Completely Interconnected Computers (CIC) topology, and were not capable of handling partial permutations. Since the CIC topology requires all PEs (Processing Elements) to communicate with each other, with at least the PE ID, requiring $\log_2 N$ bits, the connectivity complexity is $O(N^2 \log_2 N)$ for each layer.

In 1995 Lee and Oruc [26] proposed a solution for partial permutations by matching idle input port to idle output port, which requires significant pre-processing and complicates the data structures. In 1996 Lee and Liew published a parallel algorithm [27], [28] running in $O((\log_2 N)^2)$ time complexity using $N/2$ processing elements that can also handle partial permutations. In 2009 Kai et al. [29] developed a design based on Lee and Liew's algorithm that reduces the hardware complexity from $O(N^2)$ to $O(N(\log_2 N)^2)$ and time complexity from $O((\log_2 N)^2)$ to $O(\log N)$ using pipelined architecture, but their solution can not handle partial permutations.

In 2016, Jiang et al. [30], [31] implemented Lee and Liew's algorithm using a centralized memory for small N values, $N \in [8, 32]$, with time complexity of $O((\log N)^2)$ and connectivity complexity of $O(N^2 \log_2 N)$, having a centralized architecture where each PE has a dedicated interface to a centralized control logic and a centralized memory.

Finally, in 2021 Koloko at al. [32] developed yet another parallel implementation to Lee and Liew's algorithm having time complexity of $O((\log_2 N)^2)$ and hardware complexity of $O(N(\log_2 N)^2)$, since each PE communicates with every other PE the connectivity complexity remains $O(N^2 \log_2 N)$.

In this paper, we propose an efficient, functional, fast and, most importantly, scalable parallel algorithm to route a Beneš network. The algorithm works iteratively, where each iteration potentially increases the number of routed input demands to their corresponding outputs hence allowing a trade-off between processing time and number of connected inputs to outputs. The algorithm performs better given partial permutations.

As a trade-off can be made between processing time and the number of serviced input demands of a given input permutation, we can stop processing at a time complexity of $O((\log_2 N)^2)$ and achieve, on average, close to 100% success to route input demand (more specifically $\geq$95% for small size network and $\geq$92% for higher network size). Our solution's connectivity complexity is only $O(N^2)$ wires an improvement of $O(\log_2 N)$ over previous solutions, which makes the algorithm easier to scale to high-radix switches.

The main idea that allows us to break away from the wiring scalability challenge is a design of a 'collision' bus. Using this bus, we present an efficient algorithm to route almost 100% of a demand in just $O((\log_2 N)^2)$ steps.

The rest of the paper is organized as follows: Section II provides the theoretical background as the foundations for the algorithm. Section III defines the interfaces between PEs. Section IV presents the routing algorithm. Section VI presents simulation results. and section VII concludes the paper.

II. Preliminaries

In this section we present the theoretical background of the routing algorithm, the utilization of the first column of Beneš network for random setting of first column SEs' states and finally, the routing principles.

The proofs for all lemmas are given in the Appendix.

A. Definitions, Propositions and Lemmas

This section presents properties of the Beneš network that are the basis of this paper's routing algorithm. All letter variables signify integer values for SEs, PEs and ports.

A minimal re-arrangeable non-blocking NxN Beneš network, Fig 1, is recursively built as follows:

1) A first column of $N/2$ 2×2 switching elements (SEs)
2) A last column of $N/2$ 2×2 SEs
3) Middle stage comprised of two $N/2$×$N/2$ sub-networks, upper and lower, that are built recursively
4) Recursion ends at a single column of 2×2 SEs

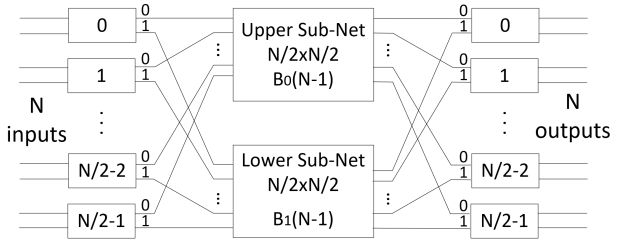


Fig. 1: Recursive construction of NxN Beneš network

Every SE in the first column has one output connected to the upper sub-network in the middle stage and one output to the lower sub-network. The inputs to the SEs of the last column are symmetrically connected to the middle stage.

The entire network can be viewed as a matrix of SEs having $2\log_2 N - 1$ columns and $N/2$ row where there are N connections between each column. SEs in each column are enumerate from 0 (top) to $N/2-1$ (bottom), and the ports of SE_k are denoted by $2k$ and $2k+1$.

Definition 1. *A SE can be in either of two states:*
0) A 'bar' state (= or 0) where $O_0 \leftarrow I_0$ and $O_1 \leftarrow I_1$
1) A 'cross' state (X or 1) where $O_0 \leftarrow I_1$ and $O_1 \leftarrow I_0$

Definition 2. *If SE_k is in bar state, port $2k$ is connected to the upper sub-network and port $2k+1$ to the lower sub-network and the opposite for cross state. The definition is symmetrically applied to the input of last column SEs.*

Given a N×N network, let I be the set of inputs and O be the set outputs, where $I = O = \{0, 1, 2, \ldots, N-1\}$. A function $\pi : I \rightarrow O$ is called a *permutation* if it is bijective (namely, it is one-to-one and onto). We widen the definition of a permutation by denoting $\pi(i) = x$ for an idle input i.

Definition 3. *A function $\pi : I \rightarrow O \cup \{x\}$ is called a* permutation *if $\pi(i) = \pi(j) \in O \implies i = j$.*

The inputs to SE_k are $I_{k0} = \pi(2k)$ and $I_{k1} = \pi(2k+1)$. Using Definition 3, if SE_k state is bar, then $O_{k0} = \pi(2k)$ and $O_{k1} = \pi(2k+1)$ and the opposite if SE_k state is cross.

An example of an input permutation for a 8×8 Beneš network is given in Eq. (1). The upper row in the matrix is the ordered input ports from 0 to N-1 and the lower row is the outputs requested by the corresponding inputs. In this example, input 0 requests route to output 2, input 1 to 3, input 2 is idle, and so on, namely, $\pi(0) = 2, \pi(1) = 4, \pi(2) = x$.

$$\pi = \begin{pmatrix} 0\ 1\ 2\ 3\ 4\ 5\ 6\ 7 \\ 2\ 4\ x\ 5\ 3\ 0\ 1\ 7 \end{pmatrix} \quad (1)$$

In the following text we will use the double division symbol (//) to mark an integer division, namely taking the integer portion of the result: $i//j \triangleq \lfloor i/j \rfloor$. $\pi(i)$ is called an input demand, or simply demand, for input port i. An even/odd input demand is a request of an even/odd output port.

Definition 4. *A collision between two input ports, i and j, of different SEs occurs when they both have demands to the same SE in the last column, that is $\pi(i)//2 = \pi(j)//2$, and both are routed through the same sub-network. The SE in the last column has only one input from each sub-network therefore it can not accept two demands from the same sub-network.*

For example, when applying the permutation shown in Eq. (1) to a 8×8 Beneš network and setting all SEs in the first column to bar state, following collisions are created:

- Input ports 0 and 4 collide through upper sub-network
- Input ports 1 and 3 collide through lower sub-network

A different state setting of the SEs in the first column, such as $\{SE_0, SE_1, SE_2, SE_3\} = \{1,0,0,0\}$, resolves both collisions.

Lemma 1. *A collision occurs between exactly two demands*

Proposition 1. *A single SE in the first column can be involved in at most two collisions, one in the upper sub-network and the other in the lower sub-network.*

Proposition 2. *A collision can only occur between odd input demand and even input demand from different first column SEs, collision can not occur between two even input demands or two odd input demands.*

Lemma 2. *Given 2 colliding demands (SE_m and SE_n in first column, where $m \neq n$, i is an input port to SE_m and j is an input port to SE_n and $\pi(i)//2 = \pi(j)//2$ and both demands are routed through the same sub-network), then flipping the state of either one SE (but not both) resolves this collision but may create a new collision in the same sub-network.*

Lemma 3. *Flipping a state of a first column SE that has two collisions, one in each sub-network, resolves both collisions, but may create a new collision in either of the sub-networks. This means the total number of collisions can not be increased.*

Corollary 1. *Flipping the state of one colliding first column SE can not increase the total number of collisions.*

Following Lemmas 2 and 3, the process of flipping the state of one colliding SE, in each collision, cannot increase the total number of collisions, and thus produces a monotonically non-increasing number of collisions after each flip.

Proposition 3. *Given an input port i with a demand to output port j, that is $j = \pi(i)$, the state of $SE_{\pi(i)//2}$ in the last column is set as follows:*

- *Cross if $\pi(i)$ is odd and routed to upper sub-network*
- *Cross if $\pi(i)$ is even and routed to lower sub-network*
- *Bar for all other cases*

Lemma 4. *If there are no collisions between first column SEs, collisions between last column SEs are not possible.*

B. First column SEs initial state and collision probability

In this section we evaluate the probability for collisions given a random initial setting of states of SEs in the first column and calculate the utilization of Beneš network. Then we present initial setting heuristic that improves the utilization, where the term 'Utilization' is defined as follows:

Definition 5. *Utilization is equal to the number of established routes from inputs to outputs, based on the input demands that the algorithm achieved, divided by the number of valid input demands. A utilization of 100% means that routes were established for all input demands.*

1) Probability of collisions:

Observation 1. *In a full input permutation (no input port is idle) there are exactly N/2 input ports that are connected to the upper (and to lower) sub-network for any given SEs' states.*

Lemma 5. *Based on observation 1, in a full input permutation, a collision in one sub-network always implies a collision in the other sub-network as well.*

The number of collisions, if any, for a full input permutation demand is always even (Lemma 5). Therefore, it is sufficient to only count the number of collisions in the upper sub-network due to this symmetry. When the input permutation is not full, the total number of collisions can be odd.

Corollary 2. *The maximum number of collisions for NxN Beneš network is $N/4$ for each sub-network, that is a total of $N/2$ collision for the entire network.*

We now calculate the probability of a collision between SEs in first column to one sub-network given full input permutation where the total number of collisions is exactly twice.

We first calculate the total number of permutations for $N/2$ ports in one sub-network. The first input port selects 1 out of N output ports to be routed to, second input port remains with 1 out of $N-1$ output ports, k^{th} output port remain with 1 out of $N-(k-1)$ ports, and so on. The number of ways to connect $N/2$ input ports to one sub-network is:

$$N \cdot (N-1) \ldots \left(N - \tfrac{N}{2} + 1\right) = \prod_{i=0}^{\frac{N}{2}-1} (N-i) = \tfrac{N!}{(\frac{N}{2})!} \quad (2)$$

To count the number of input permutations having exactly k collisions we count the number of k pairs of input ports that can be selected, shown in Eq. (3). First pair has $N/2$ options, second pair has $N/2-2$ options and so on. As the ordering between the selected pairs is irrelevant, we divide by $k!$, Thus:

$$\frac{1}{k!}\cdot\binom{\frac{N}{2}}{2}\cdot\binom{\frac{N}{2}-2}{2}\ldots\binom{\frac{N}{2}-2(k-1)}{2}=\frac{1}{k!}\cdot\frac{(\frac{N}{2})!}{2^k\cdot(\frac{N}{2}-2k)!} \quad (3)$$

Next, we count the number of options of selecting output ports. Each pair of colliding input ports and each single non-colliding input port select an output port leaving two less ports to select from. A colliding pair of input ports are routed to the two output ports and a single non-colliding input port must not allow the other output port of the same SE to be selected by others. In total there are $N/2-k$ selections where the first has N ports to choose from, the second has $N/2$ ports to choose from and so on. This count is depicted in Eq. (4).

$$N\cdot(N-2)\ldots(N-2(\tfrac{n}{2}-1-k))=\frac{2^{\frac{N}{2}-k}\cdot(\frac{N}{2})!}{k!} \quad (4)$$

By multiplying Eq. (3) by Eq. (4), dividing by Eq. (2) and re-arranging the equation, we get $Pr(N,k)$, the probability of k-collisions in a single sub-network of a N×N Beneš network when setting SEs in the first column to random states (bar or cross) for a full input permutation demand, Eq. (5), and the distribution of k-collisions is shown in Fig 2b. The probabilities for the entire N×N Beneš network are doubled.

$$Pr(N,k)=\frac{[(\frac{N}{2})!]^3}{N!}\cdot\frac{2^{\frac{N}{2}-2k}}{(k!)^2\cdot(\frac{N}{2}-2k)!} \quad (5)$$

Each sub-network has $N/2$ ports hence at most $N/4$ collisions. The maximum probability of collisions occurs for k=$N/8$, which, due to symmetry, it relies at half of the maximum number of collisions. From Eq. (5), the approximate probabilities of no collisions (k=0), of maximum collisions (k=$N/4$) and the maximum probability (k=$N/8$) after applying Stirling's approximation are shown in Eq. (6)

$$Pr(N,0)\approx\sqrt{\frac{\pi N}{2^{N-1}}}\ ,\quad Pr(N,N/4)\approx\sqrt{\frac{1}{2^{N-1}}}$$

$$Pr(N,N/8)\approx\frac{4}{\sqrt{\pi N}} \quad (6)$$

Another way to calculate $Pr(N,0)$ is by first counting the number of input permutations having no collisions. The first input port chooses 1 out of N output ports, the second chooses 1 out of N-2 ports (to avoid collision), the third chooses 1 out of N-4 ports and so on for $N/2$ input ports, shown in Eq. (7).

$$N\cdot(N-2)\cdot(N-4)\ldots2=2^{\frac{N}{2}}\cdot(\frac{N}{2})! \quad (7)$$

Next we divide expression (7) by (2) and, with Stirling's approximation, we get the same result as shown in Eq. (6).

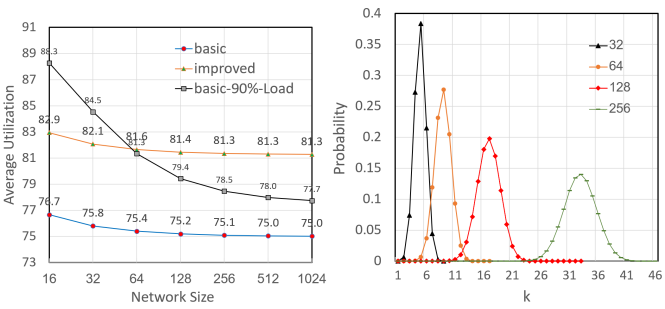
2) First column utilization:

A collision is resolved by forfeiting an input demand from one of the colliding input ports, resulting in reduced utilization.

The average utilization of first column SEs having arbitrary states given 100% input load is derived by multiplying Eq. (5) by the remaining input demands after forfeiting colliding demands and summarize for all collisions, shown in Eq. (8).

$$U_1(N)=\frac{1}{N}\sum_{i=0}^{\frac{N}{4}}Prob(N,i)\cdot(N-2i) \quad (8)$$

The line marked as **basic** in Fig 2a shows the calculated first column utilization, based on Eq. (8), as a function of Beneš network size which converges to 75%.



(a) Average utilization (b) k-collisions probability

Fig. 2: (a) average first column utilization for basic and improve methods (b) K-collisions Probability distribution

The average utilization for the entire NxN Beneš network, when SEs in all layers are set to a random state, is always above the multiplication of U_1 for all sub-networks, Eq. (9).

$$U(N)>\prod_{i=2}^{\log_2 N}U_1(2^i) \quad (9)$$

This is because when demands are forfeited, the number of demands to the sub-networks are reduced resulting in lower probability for collisions, hence higher utilization in the sub-networks. For example, the line marked as **basic-90%-Load**, in Fig 2a, shows the simulated first column average utilization for 90% input load using random state settings, which is higher than the average utilization for full permutation.

3) Improved initial state of the first column:

No collisions occur if all demands to even output ports are directed to the upper sub-network and all demands to odd output ports are directed to the lower sub-network (or the opposite). Based on that, we introduced a heuristic:

- For each SE$_k$, if input I_0 is odd and input I_1 is even, set SE state to cross, otherwise set SE state to bar

Simulations show that this heuristic increases the average utilization of the first column by more than 6%, as shown in Fig 2a in the line marked **improved**.

C. Routing Principles

Parallel Beneš network routing algorithm, as presented in [27], calculates states of SEs in the first and last columns and sets demands to the sub-networks. This process is repeated recursively in each sub-network until all layers are routed.

The routing algorithm in this work uses the same methodology, but with the ability to tradeoff between completion time of calculating the states of the first columns and utilization (which can be below 100%) which allows for easy scalability.

The crux of the routing algorithm presented in this work is based on Corollary 1 and the fact that it is enough for a PE to know that it has a colliding demand without knowing which is the other colliding PE.

To resolve the states of the first column SEs, the algorithm operates in iterations. In each iteration, the PEs communicate with each other to detect and resolve collisions. Once the iteration process is completed, the states of the last column SEs and demands to the sub-networks are calculated. At the end of the recursion, the innermost sub-networks indicate to the outmost layer about forfeited input demands due to collisions.

Through the rest of the paper, we refer to PE_k as the processing element of SE_k in first and in last columns.

III. Interfaces between PEs

PEs communicate between them in order to detect and resolve collisions. In this chapter we review this communication. Collision bus in Section III-A, An Iteration process in Section III-B and Forfeit bus in III-C.

A. Collision bus

A centralized bus, named "Collision bus", is used for the communication between PEs. Each bit in the bus represents a required SE in the last column, therefore the size of the bus is $N/2$ and the collision bus is an input to all PE.

Each PE has $N/2$ bits output bus. PE_k can set bus bits $\pi(2k)//2$ or $\pi(2k+1)//2$ indicating the index of the required SE in the last column, based on its valid input demands, $\pi(2k)$, $\pi(2k+1)$. An idle input demand can not set any bit.

Collision bus bit [i] is the result of an OR gate between bit [i] of output busses of all $N/2$ PEs. Wired-OR technology can eliminate the use of actual OR gates. The implementation of the collision bus based on wired-OR is shown in Fig 3.

In total, each PE has N interface bits, $N/2$ output bits and $N/2$ input bits, and the total wires for $N/2$ PEs is $N^2/2$, that is $O(N^2)$, a reduction of $log_2 N$ compared to prior works.

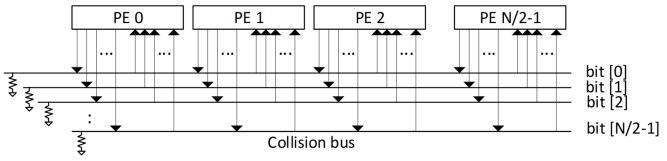


Fig. 3: Collision bus implemented with wired-OR

Proposition 4. *Based on Lemma 1 and proposition 2, exactly 2 ports, i and j, having demands π(i) and π(j) from 2 different PEs, PE_m and PE_n, can collide. Therefore, if bit [π(i)//2] on the collision bus is set by PE_m, only PE_n relates to that bit and, π(i) and π(j) are not both odd or both even.*

Proposition 5. *Based on Proposition 4, when all PEs with demands to even output ports routed through the upper sub-network set their corresponding bits on the collision bus, each bit on the bus can be set by one and only one such PE.*

A set bit on the bus can match a demand of one and only one PE that has demand to an odd output port that is routed through the upper sub-network

When PE_k sets bit [d] on its output bus, it means that PE_k 'publishes' demand to SE_d in the last column. Based on proposition 5, when all PEs with even (or odd) output demands to upper (or lower) sub-network publish their demands, a single bit on the collision bus can be set by only one PE. It is not possible for two such PEs to set the same bit.

This property can only be achieved if PEs are divided to sets of publishing and listening PEs for the same sub-network, set 1 publish and set 2 listen or the opposite and set 3 publish and set 4 listen or the opposite. Sets are as follows:

1) PEs with even demand routed via upper sub-network
2) PEs with odd demand routed via upper sub-network
3) PEs with even demand routed via lower sub-network
4) PEs with odd demand routed via lower sub-network

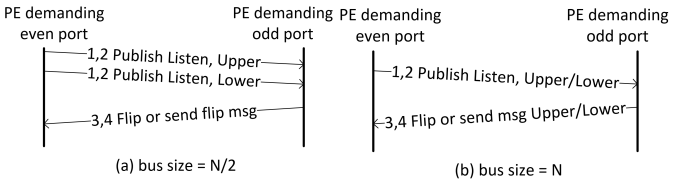
B. An Iteration

Following are iteration steps to detect/resolve collisions:

1) **Publish demands**: PEs of one set publish their required SE in last column on the collision bus (that is demand//2).
2) **Listen and detect collision**: Each PE in an opposite set listens to a bit in the collision bus, corresponding to its required SE in the last column. If the bit is set, a collision is detected by the listener PE.
3) **Flip state**: Listening PEs having two collisions flips their state to resolve the collision, if having one collision, they flip their state with probability 1/2.
4) **Flip message**: PEs that decided not to flip their state send a similar message to 'publish', on the collision bus. The publishing PEs listen to the bus and flip their state if their demand is equal to the set bit on the bus.

A single iteration, shown in Fig 4a, contains steps 1,2 for the upper sub-network (sets 1 and 2), then repeat steps 1,2 for the lower sub-network (sets 3 and 4), then steps 3,4.

An iteration can be reduced to 2 messaging phases by doubling the collision bus width to N bits, shown in Fig 4b, such that two publish phases are performed in one message where each uses half of the bus. Selecting $N/2$ or N bits bus depends on implementation constraints.



(a) $N/2$ bits Collision bus (b) N bits Collision bus

Fig. 4: Collision bus steps for (a) $N/2$ bits, (b) N bits

C. Forfeit bus

Input demands may be forfeited in any sub-network due to unresolved collisions following the iterations process. The forfeited input demands must be propagated to the outmost layer. For that purpose, input demands are propagated in full to all sub-layers, during the recursive iterative process, even if they are forfeited in some middle layer. At the end of the recursion, the sub-networks in the innermost layer contains all valid, invalid or forfeited input demands, however without the knowledge of their corresponding input port.

To pass the information of invalid (forfeited) demands from innermost to outmost layer, a dedicates bus, named "Forfeit bus", is used from innermost layer PEs to outmost layer PEs. The structure of this bus is the same as the collision bus, it contains N bits where bit [i] corresponds to input demand [i].

Each PE in the innermost layer has N outputs to the Forfeit bus corresponding to N input demands. Each PE can set up at most two bits on this bus corresponding to its invalid/forfeited input demands. All N bits from all innermost PEs are ORed together, similar to the collision bus, to create the forfeit bus.

Each PE in the outmost layer listens to the forfeit bus and checks if its valid input demands are set on the bus, if so, the PE invalidate these input demands.

IV. Algorithm

In this section we present our novel Beneš network routing algorithm, contributing: (1) a new iterative approach to resolve routing conflicts; (2) reduces connectivity between PEs; and (3) fast average completion time of $O((\log_2 N)^2)$.

To the best of our knowledge this is the first Beneš routing algorithm that trades-off processing time and utilization.

The routing algorithm contains the following six steps:

1) Distribute demand and Set Initial state
2) Modify states of SEs in first column by performing iterative collision detection and resolution process
3) Remove demands of remaining colliding inputs
4) Set switching states of SEs in last column
5) Set demand to upper and lower sub-networks
6) Inform outmost layer about forfeiting input demands

Steps 1-5 are performed on all layers recursively followed by step 6. Steps 2, 3 and 6 are unique to our proposed algorithm, where the other steps are used by (almost) all parallel algorithms. The recursive nature of the algorithm allows the use of the same circuit for every sub-layer with a half the number of inputs–outputs per sub-network.

To implement the algorithm each PE maintains the following variables, all are reset prior to enabling the algorithm:

- $CollUp$: Collision detected in upper sub-network
- $CollDown$: Collision detected in lower sub-network
- $FirstState$: State of SE in first column
- $LastState$: State of SE in last column
- $Flip$: SE state was flipped in previous iteration

We denote the collision bus by CB, the output bus of each PE by OB, and the forfeit bus by FB.

A. Summary of Heuristics in the algorithm

Following are the heuristics used in the algorithm:

1) *Improved initial state (described in section II-B3)*
2) *Iteration sequence*: publish up→publish down→flip: Get collisions from both networks then decide to flip.
3) *Flip decision*: PEs with two collisions flip their state, PEs with one collision flip their state with probability 1/2
4) *Prev. flip state*: PEs that flip their state in previous iteration do not flip their state in current iteration unless they detect two collisions

B. Step 1: Set demand and initial state

In this step, demands are distributed to each PE and each PE sets its initial state as describe in section II-B3.

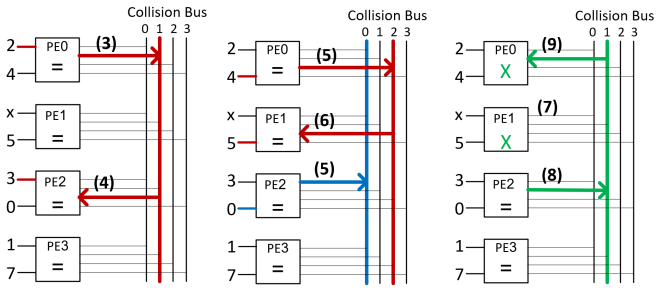
C. Step 2: Iterative collision detection and resolution process

An iteration has nine phases, shown in Alg. 1:

Example, given the input demand in Eq. (1) and initial states of PEs in the first column is bar, that is: $\{PE_0, PE_1, PE_2, PE_3\}$ = $\{0, 0, 0, 0\}$, phase 2 operations, shown in Fig 5, are:

1) PE_0: Out_{00}=2, Out_{01}=4 PE_1: Out_{10}=x, Out_{11}=5, ...
2) Set1={0}, Set2={2,3}, Set3={0,2}, Set4={1,3}
3) PE_0 sets $CB[1] \leftarrow 1$ since its output port is 2 (2//2)
4) PE_2 listen to $CB[1]$ (input 4 requires output 3) and sets $CollUp \leftarrow 1$ because $CB[1] = 1$
5) PE_0 sets $CB[2] \leftarrow 1$ since its output port is 4 (4//2)
   PE_3 sets $CB[0] \leftarrow 1$ since its output port is 0 (0//2)
6) PE_1 listen to $CB[2]$ (input 5 requires output 5) and sets $CollDown \leftarrow 1$ because $CB[2] = 1$
7) PE_2 has $CollUp$=1, it decides not to flip its own state
   PE_1 has $CollDown$=1 and it decides to flip its own state
8) PE_2 sets $CB[1] \leftarrow 1$
9) PE_0 listen to $CB[0]$ which is set, it flips its own state

As a result of this iteration, the states of PEs in the first column are changed to $\{PE_0,PE_1,PE_2,PE_3\}$ = $\{1,1,0,0\}$. This setting resolves the collision between input ports 0 and 4 but input ports 1 and 3 are now colliding in the upper sub-network.



(a) Steps 3,4 (b) Steps 5,6 (c) Steps 7,8,9

Fig. 5: Example for step 2, (a) upper sub-network (b) lower sub-network, (c) flip and flip message

D. Step 3: Forfeit colliding demands

This step is perform another iteration, as step 2, where a PE forfeits its colliding input instead of flipping its state. At the end of this step half of the colliding input ports forfeit their demands, hence reduced utilization. The randomness of selecting input demands to be forfeited guarantees fairness between demands over multiple input permutations.

1) **Phases 1-6**: Same as Step 2
2) **Phase 7**: PEs in sets 2 and 4 forfeit their colliding demand with probability 1/2.
3) **Phase 8**: Same as Step 2, Publish by PEs of sets 2 and 4 that decide not to forfeit their input demand
4) **Phase 9**: Same as Step 2, Listen by PEs of sets 1, 3 if accepting flip message, forfeit corresponding demand

Algorithm 1 Step 2: Iterations

Phase 2.1: Set PEs outputs based on their state
Set outputs PE_{k,O_0}, PE_{k,O_1} based on $PE_{k.In\_State}$
Phase 2.2: Divide all PEs into 4 sets
Set 1: PEs with valid even demand to upper sub-network
Set 2: PEs with valid odd demand to upper sub-network
Set 3: PEs with valid even demand to lower sub-network
Set 4: PEs with valid odd demand to lower sub-network
Phase 2.3: Publish even demands to upper sub-network
for $PE_k \mid \{k \in$ **Set1**$\}$ **do** $PE_k : OB[PE_{k,O_0}//2] \leftarrow 1$
Phase 2.4: Detect collision in upper sub-network
for each $PE_k \mid \{k \in$ **Set2**$\}$ **do**
    if $CB[PE_{k,O_0}//2] = 1$ **then** $PE_{k.CollUp} \leftarrow 1$
Phase 2.5: Publish even demands to lower sub-network
for $PE_k \mid \{k \in$ **Set3**$\}$
    do $PE_k : OB[PE_{k,O_1}//2] \leftarrow 1$
Phase 2.6: Detect collision in lower sub-network
for each $PE_k \mid \{k \in$ **Set4**$\}$ **do**
    if $CB[PE_{k,O_1}//2] = 1$ **then** $PE_{k.CollDown} \leftarrow 1$
Phase 2.7: Collision resolution by listening PEs
for each $PE_k \mid \{k \in$ **Set2** $\cup$ **Set4**$\}$ **do**
    if $PE_{k.CollUp} = 1$ **and** $PE_{k.CollDown} = 1$ **then**
        Flip $PE_{k.FirstState}$, $PE_{k.Flip} \leftarrow 1$
    else if $PE_{k.Flip} = 0$ **and**
    $(PE_{k.CollUp} = 1$ **or** $PE_{k.CollDown} = 1)$ **then**
        $X \leftarrow [0\ or\ 1]$ randomly with even probability
        if $X = 1$ **then**
            Flip $PE_{k.FirstState}$, $PE_{k.Flip} \leftarrow 1$
            $PE_{k.CollUp} \leftarrow 0$, $PE_{k.CollDown} \leftarrow 0$
        end if
    end if
end for
Phase 2.8: Publish flip for upper/lower sub-network
for each $PE_k \mid \{k \in$ **Set2**$\}$ **do**
    if $PE_{k.CollUp} = 1$ **then**
        $PE_k : OB[PE_{k,O0}//2] \leftarrow 1$, $PE_{k.CollUp} \leftarrow 0$
for each $PE_k \mid \{k \in$ **Set4**$\}$ **do**
    if $PE_{k,CollDown} = 1$ **then**
        $PE_k : OB[PE_{k,O1}//2] \leftarrow 1$, $PE_{k.CollDown} \leftarrow 0$
Phase 2.9: Respond to flip for upper/lower sub-network
for each $PE_k \mid \{k \in$ **Set1** $\cup$ **Set3**$\}$ **do**
    if $CB[PE_{k,O_0}//2] = 1$ **or** $CB[PE_{k,O_1}//2] = 1$ **then**
        Flip $PE_{k.FirstState}$, $PE_{k,Flip} \leftarrow 1$

E. Step 4: Set states of SEs in last column

Last columns SEs' states are set based on proposition 3 and lemma 4 using the collision bus, shown in Alg.2.

Algorithm 2 Step 4: Setting states to SEs in last column

Phase 5.1: Set states of SEs in last col. to bar, Set outputs of first col. SEs based on input state and create 4 sets
for $k \leftarrow 0$ to $N/2-1$ **do**
    $PE_{k.LastState} \leftarrow Bar$
    Set outputs PE_{k,O_0}, PE_{k,O_1} based on $PE_{k.In\_State}$
Set1/2: Even/Odd demands to upper sub-network
Set3/4: Even/Odd demands to lower sub-network
Phase 5.2: Publish demands, odd to upper even to lower
for each $PE_k \mid \{k \in$ **Set2**$\}$
    $PE_k : OB[PE_{k,O_0}//2] \leftarrow 1$
for each $PE_k \mid \{k \in$ **Set3**$\}$
    $PE_k : OB[PE_{k,O_1}//2] \leftarrow 1$
Phase 5.3: Set state of SEs in last column to Cross
for $k \leftarrow 0$ to $N/2-1$ **do**
    if $CB[k] = 1$ **then** $PE_{k.LastState} \leftarrow Cross$

F. Step 5: Deliver demands to upper/lower sub-networks

The delivery of demands from the first column to the sub-networks is via a dedicated interface identical to the natural connectivity of Beneš network.

- Upper net $PE_{k,I_0}, PE_{k,I_1} \leftarrow PE_{2k,O_0}, PE_{2k+1,O_0}$
- Lower net $PE_{k,I_0}, PE_{k,I_1} \leftarrow PE_{2k,O_1}, PE_{2k+1,O_1}$

G. Step 6: Inform forfeited demands

Informing the outmost layer about input demands that were forfeited is done using the dedicated forfeit bus, as described in section III-C. This process is shown in Alg.3.

The selection of a new permutation given forfeited demands at current permutation is out of the scope of this algorithm.

Algorithm 3 Step 6: Inform outer layer of forfeited demands

phase 6.1: Innermost layer PEs publish invalid demands
for $k \leftarrow 0$ to $N/2-1$ **do** ▷ of all innermost layers
    if PE_{k,I_0} is invalid **then** $PE_k : FB[PE_{k,I_0}] \leftarrow 1$
    if PE_{k,I_1} is invalid **then** $PE_k : FB[PE_{k,I_1}] \leftarrow 1$
end for
phase 6.2: PEs of outmost layer listen to forfeit bus and invalidate input demands
for $k \leftarrow 0$ to $N/2-1$ **do** ▷ of all outmost layers
    if $FB[PE_{k.I_0}] = 1$ **then** PE_k invalidates its input I_0
    if $FB[PE_{k.I_1}] = 1$ **then** PE_k invalidates its input I_1
end for

H. Termination of recursion

The recursive iterative process stops at 8×8 sub-networks which are resolved in O(1) time in 2 steps as follows:

1) Resolve first and last columns in 8×8 Beneš network: Based on Waksman [33], the state of one first column SE can be fixed to cross or bar and complete routing can still be found. Once fixing a state to an SE, three SEs remain unset, resulting in 8 (2^3) setting options. A simple combinatorial logic evaluates these 8 options in parallel within O(1) time.

2) Resolve entire 4×4 Beneš network:

A pre-configured 24-entry table (4! input permutations) is used. Each entry contains the states of all six SEs of the 4x4 network. Partial permutations are filled prior to accessing the table. Resolving a 4×4 network is done within $O(1)$ time.

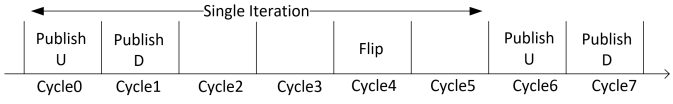
V. Implementation and Timing Budget

A. Collision bus sizing and related pipeline

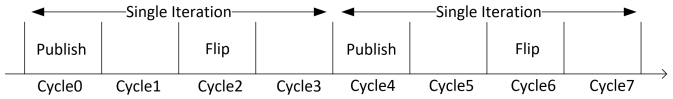
A single clock cycle may not be sufficient to deliver and process messages on the collision bus, especially for high-radix networks. Therefore, we suggest a pipeline design such that one clock cycle is dedicated to propagate a message and sample it at the receiving PEs and another clock cycle to process the received message.

N/2 bits collision bus: 6 clock cycles per iteration. Cycles 0, 1 are used to deliver collision messages for upper and lower sub-networks. The messages are accepted in cycles 1 and 2. Cycle 4 is used to deliver the flip messages, Fig 6a.

N bit collision bus: 4 clock cycles per iteration. Cycle 0 is used to publish demands for both upper and lower networks and cycle 2 is used to send flip messages, Fig 6b.



(a) $N/2$ bits collision bus width



(b) N bits collision bus width

Fig. 6: Single iteration pipeline

B. Relating number of iterations to clock cycles

Given, I, the number of iterations performed in step 2 per layer, processing a single layer requires $I + 1.5$ iterations, when considering all steps (except 1, 5 and 6). Step 2 takes I iterations, step 3 takes 1 iteration and step 4 takes $1/2$ iteration.

Step 1 and 5 are required by all parallel algorithms and are implementation specific so they are not counted. Step 6 is performed once and require 3 clock cycles so it is ignored.

In order to complete the routing of the entire NxN network within $O((\log_2 N)^2)$ iterations, we set I such that the number of iterations for all $\log_2(N/8)$ layers is less than or equal to $(\log_2 N)^2$. as shown in Eq. 10.

$$(\log_2 N)^2 \geq (I + 1.5) \cdot \log_2(N/8)$$
$$\Rightarrow I \leq \frac{(\log_2 N)^2}{\log_2(\frac{N}{8})} - 1.5 \qquad (10)$$

To meet $O((\log_2 N)^2)$ iterations to rout a complete Beneš network for N=32, $I \leq 11$, for N=64 and N=128 $I \leq 10.5$, for N=256 $I \leq 11.3$, for N=512 $I \leq 12$, and for N=1024 $I \leq 12.8$.

The total number of clock cycles is derived by multiplying the total number of iterations to complete routing by 4 or 6, depending on the size of the collision bus.

For example, when using N bits collision bus on 32x32 network, I=8 requires 76 clock cycles to complete routing.

C. Summary of busses and sizing

Routing the entire N×N Beneš network requires only $N/2$ PEs. The outmost layer uses all $N/2$ PEs along with the entire collision bus and the entire forfeit bus. Next layer contains two sub-networks where each requires $N/4$ PEs along with half of the collision bus and half of the forfeit bus and so on.

Summary of interfaces and connectivity per PE is shown in Table I. Reduced or extended collisions bus are considered based on implementation constraints. Other implementation dependent interfaces are not shown.

TABLE I: Summary of required wiring

	Per PE		For all PEs
Reduce Coll. bus	$N/2$ in, $N/2$ out	(N)	$N^2/2$
Extend Coll. bus	N in, N out	$(2N)$	N^2
Forfeit bus	N in, N out	$(2N)$	N^2
Alg. Step 6	1 in, 1 out	(2)	N

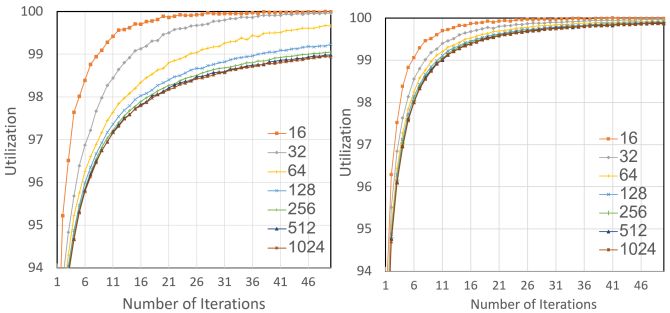
VI. Evaluation

In this section we evaluate the performance of the routing algorithm for Beneš networks as a function of network size (N=32...1024) and number of iterations. The initial state of PEs is as defined in algorithm step 1.

As mentioned in the introduction, Beneš networks are currently mainly suggested as an attractive solution for optical cross-connects. In this context, the full features of EPSs such as queuing engine, congestion control, scheduling, and more, that greatly affect the input permutations, are not pertinent to our proposed algorithm. Therefore, we used random input permutations (full and partial) with no specific workload related traffic patterns. Additionally, the algorithm does not consider forfeited demands of previous permutations when generating a new permutation, due to the same reasons.

A. First column utilization

Average utilization of the first column only was measured for 100% and 90% input loads, shown in Fig 7. Each point is calculated based on 50,000 random permutations. High average utilization, >95%, is achieved with just few iterations.

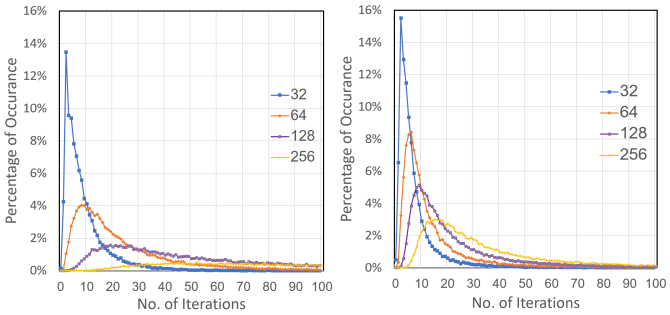


(a) 100% load (b) 90% load

Fig. 7: First column average utilization

The utilization distribution of the first column only is shown in Fig 8 for 100% and 90% input load.

At 100% load, the average number of iterations is 10.0 for N=32, 25.4 for N=64, for 82.6 for N=128 and 131.4 for



(a) 100% load (b) 90% load

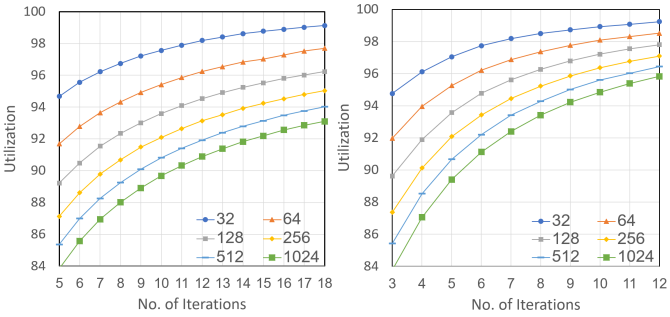
Fig. 8: Histogram of the number of iterations

N=256. At 90% load, 8.6 for N=32, 15.8 for N=64, 27.5 for N=128 and 46.4 for N=256. High number of iterations is required to achieve 100% utilization even for relatively small networks. The long tails are due to rare cyclic behavior of the collision resolution algorithm that may randomly return to a PE state already visited. As 90% input load imposes less constraints on the routing algorithm, therefore, higher utilization is achieved with less iterations. For example, N=256 at 90% requires a third of the number required for 100% load.

As we want the algorithm to perform within $O((\log_2 N)^2)$ steps (iterations), best known time complexity for parallel algorithms [28], we limit the total number of iterations as shown in Eq. 10, achieving less than 100% utilization but with less connectivity between PEs which allows for easier scalability to high-radix OCSs.

B. Fixed number of iterations per layer

In this section we examine the algorithm performance for the entire Beneš network where the number of iterations (I) is fixed per layer where layers 8×8 and 4×4 are resolved in $O(1)$. In total $I \cdot \log_2(N/8)$ iterations are performed in the entire network. The initial state of PEs is as described in algorithm step 1.Results are shown in Fig 9.



(a) 100% load (b) 90% load

Fig. 9: Average utilization of N×N network, fixed mode for (a) 100% input load and for (b) 90% input load

At 100% input load the algorithm achieves above 95% average utilization for 32×32 with I≥6 and for 64×64 with I≥10 and higher network sizes achieve above 92% average utilization all using about $(\log_2 N)^2$ total iterations.

At 90% input load, all network sizes achieve above 95% average utilization using less than $(\log_2 N)^2$ iterations.

Next, we show the probability distribution of the utilization for the entire N×N Beneš network, Fig 10, for 100% input load and 12 iterations per layer with the same setup described above. The spread of the utilization is relatively narrow and gets narrower when switch size grows indicating that close to average utilization is achieved with very high probability.

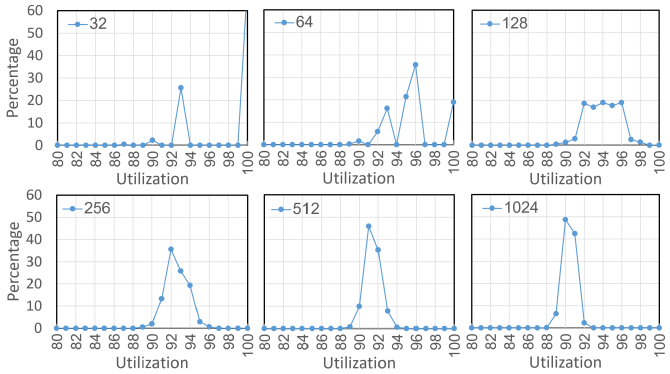


Fig. 10: Distribution of Utilization for 100% input demands for 12 iterations per layer

VII. Conclusions

In this paper we presented a novel approach to Beneš network routing algorithm and architecture by using an iterative algorithm that trades-off between utilization and completion time. The algorithm is scalable to high-radix networks suitable for OCSs, achieved by a reduced connectivity complexity.

The proposed algorithm, with its dedicated hardware architecture, achieves over 94% utilization for full input demands on network 128x128 and below,and over 92% for 256×256 up to 1024×1024. It performs within $(\log_2 N)^2$ iterations and with connectivity complexity of $O(N^2)$, a reduction of $\log_2 N$. The algorithm achieves higher utilization for partial permutations.

The reduction in the connectivity complexity and the comparable time complexity is achieved at the cost of having average utilization below but nearly 100%, using fairly small amount of logic per each PE without the usage of special blocks such as SRAM or DSP.

Acknowledgment

Special thanks to Otchi Zion Shlomo Gal for participating in developing the core algorithm and for his meaningful insights.

References

[1] G. Varghese, *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005.

[2] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, no. 2, pp. 406–424, 1953.

[3] V. E. Beneš *et al.*, *Mathematical theory of connecting networks and telephone traffic*. Academic press, 1965.

[4] F.-C. Shao and A. Y. Oruç, "Efficient nonblocking switching networks for interprocessor communications in multiprocessor systems," *IEEE transactions on parallel and distributed systems*, vol. 6, no. 2, pp. 132–141, 1995.

[5] A. Jajszczyk, "Nonblocking, repackable, and rearrangeable Clos networks: fifty years of the theory evolution," *IEEE Communications Magazine*, vol. 41, no. 10, pp. 28–33, 2003.

[6] S. Sharma, P. Bansal, and K. Kahlon, "On a class of multistage interconnection network in parallel processing," *International Journal of Computer Science and Network Security*, vol. 8, no. 5, pp. 287–291, 2008.

[7] H. Ballani, P. Costa, R. Behrendt, D. Cletheroe, I. Haller, K. Jozwik, F. Karinou, S. Lange, K. Shi, B. Thomsen *et al.*, "Sirius: A flat datacenter network with nanosecond optical switching," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, 2020, pp. 782–797.

[8] A. Singh, J. Ong, A. Agarwal, G. Anderson, A. Armistead, R. Bannon, S. Boving, G. Desai, B. Felderman, P. Germano *et al.*, "Jupiter rising: A decade of Clos topologies and centralized control in google's datacenter network," *ACM SIGCOMM computer communication review*, vol. 45, no. 4, pp. 183–197, 2015.

[9] C. Minkenberg, R. Krishnaswamy, A. Zilkie, and D. Nelson, "Co-packaged datacenter optics: Opportunities and challenges," *IET Optoelectronics*, vol. 15, no. 2, pp. 77–91, 2021.

[10] N. Zilberman, G. Bracha, and G. Schzukin, "Stardust: Divide and conquer in the data center network," in *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, 2019, pp. 141–160.

[11] W. M. Mellette, R. McGuinness, A. Roy, A. Forencich, G. Papen, A. C. Snoeren, and G. Porter, "Rotornet: A scalable, low-complexity, optical datacenter network," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, 2017, pp. 267–280.

[12] W. M. Mellette, R. Das, Y. Guo, R. McGuinness, A. C. Snoeren, and G. Porter, "Expanding across time to deliver bandwidth efficiency and low latency," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 1–18.

[13] R. Urata, H. Liu, K. Yasumura, E. Mao, J. Berger, X. Zhou, C. Lam, R. Bannon, D. Hutchinson, D. Nelson *et al.*, "Mission apollo: Landing optical circuit switching at datacenter scale," *arXiv preprint arXiv:2208.10041*, 2022.

[14] A. S. Raja, S. Lange, M. Karpov, K. Shi, X. Fu, R. Behrendt, D. Cletheroe, A. Lukashchuk, I. Haller, F. Karinou *et al.*, "Ultrafast optical circuit switching for data centers using integrated soliton microcombs," *Nature communications*, vol. 12, no. 1, pp. 1–7, 2021.

[15] Y. Mori and K.-I. Sato, "High-port-count optical circuit switches for intra-datacenter networks [invited tutorial]," *Journal of Optical Communications and Networking*, vol. 13, no. 8, pp. D43–D52, 2021.

[16] L. Zhao, P. Shi, and H. Zhang, "Bi-directional benes with large port-counts and low waveguide crossings for optical network-on-chip," *IEEE Access*, vol. 9, pp. 115 788–115 800, 2021.

[17] Q. Cheng, A. Wonfor, J. Wei, R. Penty, and I. White, "Demonstration of the feasibility of large-port-count optical switching using a hybrid mach–zehnder interferometer–semiconductor optical amplifier switch module in a recirculating loop," *Optics letters*, vol. 39, no. 18, pp. 5244–5247, 2014.

[18] L. Lu, S. Zhao, L. Zhou, D. Li, Z. Li, M. Wang, X. Li, and J. Chen, "16× 16 non-blocking silicon optical switch based on electro-optic mach-zehnder interferometers," *Optics express*, vol. 24, no. 9, pp. 9295–9307, 2016.

[19] L. Qiao, W. TAng, and T. Chu, "Ultra-large-scale silicon optical switches," in *2016 IEEE 13th International Conference on Group IV Photonics (GFP)*. IEEE, 2016, pp. 1–2.

[20] M. Kynigos, J. A. Pascual, J. Navaridas, M. Luján, and J. Goodacre, "On the routing and scalability of mzi-based optical beneš interconnects," *Nano Communication Networks*, vol. 27, p. 100337, 2021.

[21] D. C. Opferman and N. T. Tsao-Wu, "On a class of rearrangeable switching networks part i: Control algorithm," *The Bell System Technical Journal*, vol. 50, no. 5, pp. 1579–1600, 1971.

[22] A. Chakrabarty, M. Collier, and S. Mukhopadhyay, "Matrix-based non-blocking routing algorithm for beneš networks," in *2009 Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*. IEEE, 2009, pp. 551–556.

[23] D. Nikolaidis, P. Groumas, C. Kouloumentas, and H. Avramopoulos, "Novel benes network routing algorithm and hardware implementation," *Technologies*, vol. 10, no. 1, p. 16, 2022.

[24] D. Nassimi and S. Sahni, "Parallel algorithms to set up the benes permutation network," *IEEE Transactions on Computers*, vol. 31, no. 02, pp. 148–154, 1982.

[25] G. F. Lev, N. Pippenger, and L. G. Valiant, "A fast parallel algorithm for routing in permutation networks," *IEEE transactions on Computers*, vol. C-30, no. 2, pp. 93–100, Feb. 1981.

[26] C.-Y. Lee and A. Y. Oruç, "A fast parallel algorithm for routing unicast assignments in benes networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, no. 3, pp. 329–334, 1995.

[27] T. T. Lee and S.-Y. LIew, "Parallel routing algorithms in Benes-Clos networks," in *IEEE INFOCOM'96*, vol. 1, 1996, pp. 279–286.

[28] T. T. Lee and S.-Y. Liew, "Parallel routing algorithms in Benes-Clos networks," *IEEE transactions on communications*, vol. 50, no. 11, pp. 1841–1847, 2002.

[29] Y. Kai, K. Hamada, Y. Miao, and H. Obara, "Design of partially-asynchronous parallel processing elements for setting up Benes networks in $O(\log_2 n)$ time," in *International Conference on Photonics in Switching*. IEEE, 2009, pp. 1–2.

[30] Y. Jiang and M. YAng, "Hardware implementation of parallel algorithm for setting up benes networks," in *The International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2016, p. 10.

[31] Y. Jiang and M. Yang, "Hardware design of parallel switch setting algorithm for benes networks," *International Journal of High Performance Systems Architecture*, vol. 7, no. 1, pp. 26–40, 2017.

[32] L. Koloko, T. Matsumoto, and H. Obara, "Design and implementation of fast and hardware-efficient parallel processing elements to set full and partial permutations in beneš networks," *The Journal of Engineering*, vol. 2021, no. 6, pp. 312–320, 2021.

[33] A. Waksman, "A permutation network," *Journal of the ACM (JACM)*, vol. 15, no. 1, pp. 159–163, 1968.

APPENDIX A: PROOFS OF LEMMAS

Proof to lemma 1:

Proof. Since a SE in the last column is connected to 2 output ports, it is not possible for that SE to be requested by another input demand (input demand = output port). □

Proof to lemma 2:

Proof. Flipping a state of a colliding SE in the first column resolve the collision to the corresponding SE in the last column as the colliding demand is now routed through the other sub-network. However, flipping the state of the colliding SE brings another demand to the sub-network which may collide with another demand already routed in this sub-network.

□

Proof to lemma 3:

Proof. Similar to lemma 2 proof extended to two collisions. □

Proof to lemma 4:

Proof. No collisions from the first column means traffic to SE_k in the last column arrives one from the upper sub-network and one from the lower sub-network. This means even and odd outputs for SE_k in the last column can not arrive from the same sub-network and therefore collisions are not possible. □

Proof to lemma 5:

Proof. A collision between two input demands to SE_k in the last column through one sub-network implies that there is no demand to SE_k from the other sub-network. Therefore, the $N/2$ input demands connected to this other sub-network has only $N/2-1$ available destination ports, using the pigeonhole principle, two requests must share one of the output ports of the other sub-network, hence a collision in the other sub-network as well.

□