UNIVERSITY OF CALIFORNIA
SANTA CRUZ

SOFTWARE-HARDWARE CO-DEFINED NETWORK SWITCH (SHADES) FOR A LABEL SWITCHING PROTOCOL

A dissertation submitted in partial satisfaction of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER ENGINEERING

by

Turhan Karadeniz

September 2015

The Dissertation of Turhan Karadeniz
is approved:

Prof. J.J. Garcia-Luna-Aceves, Chair

Prof. Katia Obraczka

Prof. Brad R. Smith

Tyrus Miller
Vice Provost and Dean of Graduate Studies

Table of Contents

List of Figures

List of Tables

List of Algorithms

Abstract

Software-Hardware Co-Defined Network Switch

(SHADES)

for a Label Switching Protocol

by

Turhan Karadeniz

The network switch is one of the core components of computer communications, functioning as the interconnect among ever increasing number of nodes and a variety of networks. Whether it is used in data centers for connecting large numbers of nodes with high storage and computational capacities, or in the Internet backbone as a core router carrying vast quantity of information among its users, or in medium to large sized local area networks including educational campuses and companies, the switches require ever increasing design needs for scalability, high delivery ratios, low latencies, and quality-of-service (QoS) guarantees.

We believe that switch design should be evaluated as a holistic aspect of network protocol design, since many metrics in protocol design and switch design are closely correlated. Moreover, designing a switch requires the integration of hardware and software, because different functionalities of the switch can be implemented in hardware, software or both depending on different design considerations.

Rapid increase in Software Defined Networks (SDN) related research denotes that the approach of abstracting the data plane from the control plane, thus achieving higher flexibility and architectural simplicity, accounts for a notable research problem. Furthermore, label switching based routing algorithms enable the forwarding layer to be implemented at hardware and thus inflict less per-hop latency. Introducing a hardware component into network protocol design might initially seem to combine two orthogonal design components, however the outcome is an architecture that achieves multiple levels of abstraction, flexibility, simplicity, scalability and high performance at once.

In this thesis, we propose a novel architecture with an emphasis on hardware-software co-design paradigm, resulting in a scalable, flexible and high performance switch. We show that our design can be fully implemented on a Field-Programmable Gate Array (FPGA) based platform. Our system demonstrates an attempt to improve SDN, by taking the hardware component into account in the design flow, resulting in our Software-HArdware co-Defined nEtwork Switch (SHADES) for a label switched routing protocol, delivering load balancing, low latencies, and high delivery ratios.

To my sunshine Kübra,

my extraordinary parents Figen & Hasan,

my amazing cousins Irem & Melis,

my niece Pera,

and my magnificent grandmothers Tulay & Birsen.

Acknowledgments

First of all, I would like to thank my advisor Prof. J.J. Garcia-Luna-Aceves for his support, encouragement and guidance. His wisdom provided me a safe path on which to walk, and his amazing sense of humor allowed me to have the high energy and morale to finish what I started.

I would like to thank Prof. Katia Obraczka and Prof. Brad Smith for being on my defense committee, and for all their valuable feedback.

I have received all sorts of support from the most special people that surrounded me in the period of 5 years, Yusuf Goren, Kenan Sharpe, Serdar Sali, Doruk Sart, Ashok Masilamani, Ali Dabirmoghaddam, Emre Can Kara, Zeynep Mulayim, Dila Beksac, and Renato D'Orfani, who have always been there for me.

My colleagues and friends from CCRG whom I had the pleasure to meet and work with: Spencer, James, Sam, Alison, Yali, Maziar, Duy, Rumi, Ramesh, Ehsan and Nitish. They always came up with the best ideas and offered their help when I needed it the most.

Some mentors and colleagues that I had the pleasure to work with in the past include Nurdan Ugural, Nino Carella, Lou Ungemach Panetta, Cengiz Agalar, Deniz Baysal, Ahmet Demirelli, Ilker Hamzaoglu, Erkay Savas, Serkan Sahin, Georgi Gardadjiev, Onur Can Ulusel, Aydin Aysu, Ece Ercan, Berker Agir, Anil Usumezbas, Stavros Tzilis, Roi Uziel, and Kan Boonyanit. Thank you all!

Last but not least, I want to thank Emily Gregg, Tracie Tucker, Adrienne Bergenfeld, Carol Mullane, Jolinda Singleton and Arielle M.S. Freitas. The amount of patience you had with me is out of this world!

Probably there is a guardian angel as well, of some sorts. Thanks buddy!

Chapter 1

Introduction

In computer networks, the communication between nodes is realized by a broad and diverse body of electronic and optical technology. The network switch is one of the core components of computer communications, functioning as the interconnect among ever increasing number of nodes and in between a variety of networks of various sizes and properties. Whether it is used in data centers for connecting large numbers of nodes with high storage and computational capacities, as Internet backbone core routers carrying vast quantity of information among its users, or in medium to large sized local area networks including educational campuses or companies, the switches require ever increasing design needs for scalability, flexibility, high delivery ratios, low latencies, and load balancing properties.

We believe that switch design should be evaluated as a holistic aspect of network protocol design, since many parameters in protocol and switch design are closely correlated. Moreover, designing a switch requires the integration of hardware and software, because different functionalities of the switch can be implemented in hardware, software or in both depending on different design considerations. The design of a smart switch capable of handling novel routing protocols as well as the TCP/IP stacks, while still delivering high performance, is a problem that cannot be handled in a software or hardware only solution. We perceive the switch as a full embedded system, capable of running software on a microprocessor (µp), and at the same time, able to handle time critical and computationally intensive tasks in hardware.

Interval routing has been shown to minimize routing tables very effectively. In Chapter 2, we propose a feasible multi-root approach for interval routing, by electing the central node and the corner nodes in the network as roots. We show that our BFS pruned/DFS labeled multi-root approach offers $\mathcal{O}(1)$ stretch. Furthermore, we present performance evaluation of our algorithm through simulation of both static and mobile ad-hoc networks. Multi-Root Interval Routing (MINT) achieves high delivery ratios and low latencies comparable to shortest-path based MANET routing protocol OLSR and much better than AODV, while reducing the routing table cost more than an order of magnitude, despite using multiple roots [1].

Software Defined Networking (SDN), as exemplified by OpenFlow, has been proposed as an approach that can simplify the way in which some wired networks operate and are managed. The topology of the network is virtualized at a controller node, which computes routes from sources to destinations using its network-wide view, and can instantiate the "flow state" at each switch of the network. The advantages of SDN are that: (a) signaling is reduced drastically by taking advantage of the global network view available at the controller; (b) switches can be greatly simplified by implementing signaling at the controller; and (c) the control plane is decoupled from the data plane, simplifying forwarding. However, SDN approaches have focused on wired networks operating in server rooms, where the controller is one link away from every switch, the signaling focuses solely on the network layer, and do not address the dynamic placement of content and services. We implement a centralized controller for wireless networks that uses MINT as the underlying routing protocol. MINT enables compact routing tables that facilitate the controller to download the flow state to every node in the network using wireless links. In comparison to MINT, downloading full-size explicit routing tables would be impossible due to wireless bandwidth, in a dynamic scenario.

In Chapter 3, we present our Software-Hardware Co-defined Network Switch (SHADES) for a Label Switching Protocol. Our design is implemented for FPGA platforms and successfully instantiates a switch that enables Layer 2 forwarding by using the communication channels between software routing layer and hardware forwarding logic. SHADES receives control packets including the flow state from the MINT's centralized controller, writes these to the flow tables, which can be then retrieved by the hardware logic, thus enabling Layer 2 forwarding. In this work, we use similar mechanisms to state of the art Cisco switches, which are used in wired networks, where the topology change is not an inherent aspect. Our approach is more dynamic, due to the mobile & dynamic nature of wireless routing, which requires the routing control layer to modify the flow tables dynamically. We present real-time emulation results that are in line with our claim that the point-to-point delays inflicted by both the hardware and software layers between the NIC and software routing layers can be avoided by integrating design principles from SDNs, label switching and HW/SW co-design.

In wired networks, switching latency becomes a major bottleneck and as a result needs to be handled at HW by using high cost specialized queueing and scheduling mechanisms. We implement a mini-router grid (MRG) based switch fabric architecture, which yields comparable performance to state-of-art crossbar switch fabrics. MRG switch fabric are scalable in the sense that the can decouple switch size from the cost growth, at the expense of performance, whereas crossbars always inflict an exponential cost growth. Moreover, the architecture allows the use of output queuing, by bounding the speedup by 3 and resulting in much smaller queuing memory than the current state of the art. Also, MRs naturally provide a highly pipe-staged structure, allowing very high frequency operation. The design consideration for our switches include high delivery ratios, low latencies and load balancing capabilities. In Chapter 4 and Chapter 5 we describe our Mini-Router Grids (MRG) based switch fabric architectures, which deliver these metrics with success [2][3].

Finally, Chapter 6 concludes the thesis.

Chapter 2

Multi-Root Interval Routing (MINT)

2.1 Introduction

Interval routing is a distributed approach in routing protocol design that aims at storing routing tables at each node in a compact manner, by assigning labels to nodes in such a way that the destination addresses that use the same output ports are grouped together with consecutive labeling.

In [4], the authors distinguish between a valid interval labeling scheme and an optimum one, stating that the latter can be achieved when all the paths are shortest paths. Moreover, they describe a number of topologies for which an optimum interval labeling scheme exists. An optimum scheme valid for a uniformly distributed random communication network, however, is far from being achieved. Most of the current schemes use single root mechanisms, which result in low performance due to increased total network distance resulting and high stretch.

In this work, we present the Multi-Root Interval Routing (MINT) algorithm for uniformly distributed static and mobile ad-hoc networks of any size. MINT achieves total network distance comparable to shortest path, and as a result, $\mathcal{O}(1)$ stretch. Our approach improves upon the earlier works on interval routing by electing multiple roots, computing a BFS pruned/DFS labeled spanning tree for each root, and assigning interval labels at each node per root, such that the optimal path for forwarding the packets can be chosen from multiple sets of intervals.

Software Defined Networking (SDN), as exemplified by OpenFlow [5], has been proposed as an approach that can simplify the way in which some wired networks operate and are managed. The topology of the network is virtualized at a controller node, which computes routes from sources to destinations using its network-wide view, and can instantiate the "flow state" at each switch of the network. The advantages of SDN are that: (a) signaling is reduced drastically by taking advantage of the global network view available at the controller; (b) switches can be greatly simplified by implementing signaling at the controller; and (c) the control plane is decoupled from the data plane, simplifying forwarding. However, SDN approaches have focused on wired networks operating in server rooms, where the controller is one link away from every switch, the signaling focuses solely on the network layer, and do not address the dynamic placement of content and services. We implement a centralized controller for wireless networks that uses MINT as the underlying routing protocol. MINT enables compact routing tables that facilitate the controller to download the flow state to every node in the network using wireless links. In comparison to MINT, downloading full-size explicit routing tables would be impossible due to wireless bandwidth, in a dynamic scenario.

The proposed approach of implementing Multi-Root Interval Routing to enable compact routing tables attains comparable stretch to shortest path routing. Additionally, it yields greater scalability and performance compared to single root interval routing solutions. Using a centralized controller in order to virtualize the global network view and to download the flow tables to the nodes reduce control plane overhead and enable scaling to more than 500 nodes.

In Section 2, we outline the related literature. In Section 3, we describe the controller operation and Multi-Root Interval Routing (MINT) algorithm. In Section 4, we present simulation results and compare MINT to other routing protocols such as OLSR and AODV. Finally, Section 5 concludes the chapter.

2.2 Related Work

In [6], the authors discuss enabling compact routing tables by a-priori labeling nodes and links, and compare their approach, namely implicit routing, to maintaining detailed routing information for all destinations at every node, namely explicit routing. In [7] and [4], Van Leeuwen and Tan explore the problem further and describe optimal interval routing schemes for a number of topologies. These approaches use depth-first-search (DFS) based ordering and labeling that implement valid labeling schemes. However, the resulting spanning trees induce high stretch. Similarly, in [8] and [9], Eilam et al. use DFS ordering, and by forcing extensive topology restrictions and assumptions, they achieve a stretch of 3 to 5.

In [10][11], the interval routing is achieved using breadth-first-search (BFS) ordering in order to improve the stretch. In comparison to shortest path routing, this approach improves upon the earlier approaches, however still fails to achieve low stretch.

In [12], [13] and [14], Tse and Lau provide analytical lower and upper bounds for interval routing. Their results show that even with a relatively large number of labels, interval routing still falls short of being optimal for arbitrary graphs.

In [15], the authors provide lower bounds on routing table compactness for interval routing, in an effort to relate the efficiency (measured by stretch factor) to space requirements (measured by compactness or total memory bits). In [16], Bakker et al. introduce multi-label interval routing schemes where links may contain more than one label in an effort to improve routing efficiency.

A number of proposals on using multiple spanning trees, including [17][18], use multiple roots in parallel, but they are not meant for interval routing nor similar label switching based algorithms.

Software Defined Networking (SDN) emerged as a novel computer networking paradigm to abstract the data plane from the control plane, and thus to achieve higher flexibility and architectural simplicity, by using a centralized controller in wired networks. OpenFlow [5][19] is one of the most renowned instantiations of SDN. NetFPGA [20] demonstrates a successful implementation of OpenFlow on FPGA based platforms.

In a number of works, including [21], [22], and [23], the authors propose applying SDN approach to wireless networks, by using a centralized controller to improve control plane overhead and increase scalability.

OLSR [24] is a shortest-path based MANET routing protocol that performs better both in terms of delivery ratio and latency in comparison to AODV [25], when network is composed of a large number of mobile nodes, whereas AODV is more efficient with small number of nodes and resource critical environments. In this work, we compare our interval routing based algorithm to OLSR and AODV in static and mobile scenarios with increasing number of nodes.

OLSR and AODV have been shown to be not loop-free in [26] and [27], respectively. We claim that applying breadth-first-search (BFS) pruning would be a better option for eliminating the looping edges. Moreover, BFS pruning results in wider trees with much smaller total network distance, in comparison to thin and long trees that would arise from DFS pruning. Latency, one of the most important performance metrics of computer networks, is a function of the total network distance. Takuya et al. [28] proposes a similar approach of network pruning.

In Garcia-Luna-Aceves et al. [29] and Sampath et al. [30], the authors propose incremental routing algorithms that use compact routing tables based on prefix labels rather than node identifiers. An important caveat of the algorithm is that prefix labels grow larger at every hop further away from the root node. The fact that the size of the prefix labels cannot be delimited as network size increases renders the approach unscalable. In comparison, interval routing provide fixed-size labels.

2.3 The Architecture & Algorithm

Algorithms that constitute Multi-Root Interval Routing (MINT) run on a centralized controller, connected to every network node on a communication channel orthogonal to the data channel. All nodes send $HELLO$ messages to their 1-hop neighbors for network discovery. Following the discovery stage, nodes report their ID, GPS position and 1-hop neighborhood information to the controller, which in return builds a complete graph of the network, assigns pre-order labels (DFS) to nodes, computes intervals in a way that groups together all consecutively labeled nodes, and finally downloads label and interval information to flow tables at every network node.

In the following subsections, we describe the stages of the algorithm at the controller, followed by the final stage of the algorithm, data plane forwarding.

2.3.1 Centralized Controller

SDN topology comprises one controller node and other regular nodes, as shown in Figure 2.1, where Node C represents the controller. In short, the topology of the network is virtualized at the controller, which computes routes from sources to destinations using its network-wide view, and can instantiate the flow stat at each node/switch of the network. The advantages of SDN are that: (a) signaling is reduced drastically by eliminating multiple protocols running in parallel among peer nodes; (b) switches can be greatly simplified by having to implement only the signaling with the controller; and (c) the configuration and management of the network are simplified greatly by taking advantage of the global network view available at the controller.

The system architecture has several major activities, which we outline here.

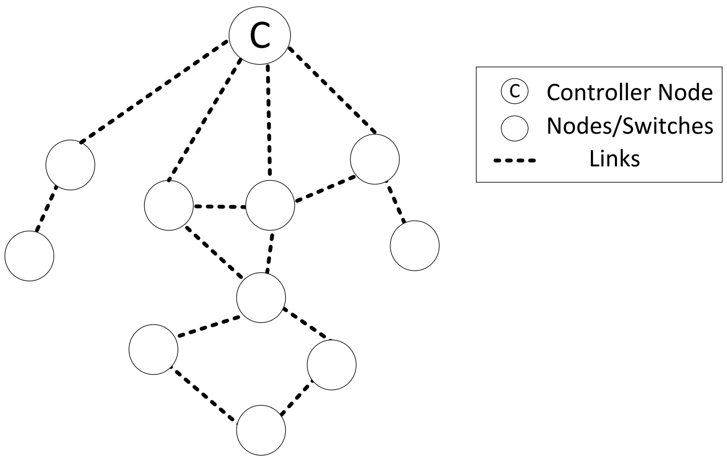


Figure 2.1: Software Defined Network Topology

1. All nodes are assumed to be connected to the centralized controller, using a communication channel orthogonal to data plane.

2. Network nodes monitor their one hop neighborhood information and periodically report it to the controller, as well as their global ID (GID) and global position (GPS).

3. The controller generates a graph based on the information it receives from the nodes, and then uses the algorithm outlined in the following subsections in order to compute the node labels and intervals. The controller selects five roots (four corners + the most central node) for building the spanning trees, using the GPS locations.

4. The controller proactively computes intervals and downloads the flow tables to the nodes.

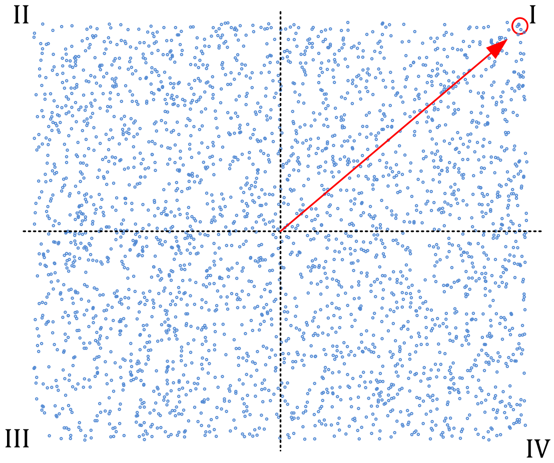


Figure 2.2: Uniformly distributed random network, divided in quadrants.

5. The nodes issue a Route Request (RREQ) to the controller for the destination labels. The controller will issue a Route Reply (RREP) in return, with destination label and root index.

6. The controller offloads most routing related signaling overhead from the network and uses its centralized computing power to compute the intervals. The controller also serves as the name/label resolution service.

2.3.2 Pruning of the BFS Trees

Upon building a complete undirected graph based on the neighborhood information received from the nodes, our algorithm proceeds to electing five roots. Please note that one assumption made here is that every node has reported their GPS coordinates, such that the controller is able to compare the positions of the nodes to the origin on a virtual Cartesian plane. In Figure 2.2, we show how the

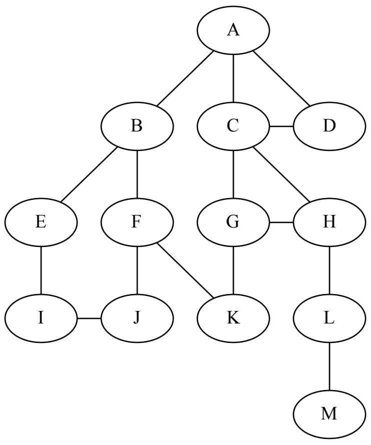


Figure 2.3: Random uniform graph G

controller divides the area into four quadrants, and elects the node with the greatest distance to the origin as a root in each quadrant. The origin is computed as the GPS position of the most central node in the graph, following the centrality algorithm in [31]; as a result it is not a spatial origin, but a logical one. The most central node is also elected as the fifth root.

Using BFS pruning starting from the root nodes, our algorithm builds five spanning trees. By using BFS pruning we achieve wide spanning trees yielding the smallest cumulative network distance.

The rationale behind electing the corner nodes and the most central node as the roots of the spanning trees is as follows:

Let's consider a uniformly distributed random network G of size N in a rectangular area, to be BFS pruned starting from a corner node r as the root, resulting in the BFS tree T. Every node in G is also a node in T. The function $d_N(T,u,v)$ describes the network distance between nodes u and v in the tree T. We define

$$\mu(T,u) = \sum_{n \in T} d_N(T,u,n)\,, \tag{2.1}$$

where $u \in T$ and the function $\mu(T,u)$ is the sum of all network distances from node u to every other node in T.

The reason to define $\mu(u)$ lies in the fact that it is a part of the stretch function,

$$STRETCH(u) = \left(\frac{\mu(T,u)}{\mu(G,u)}\right)\,, \tag{2.2}$$

Because we are trying to show the effect of electing a node as the root of a BFS tree T in a given network G, the only variable part in Def.(2.2) is $\mu(T,u)$.

Our initial claim to support our rationale is as follows: We claim that if $d_N(T,u,r) \leq d_N(T,v,r)$, then $\mu(T,u) \leq \mu(T,v)$ would also hold true. This claim implies that the nodes closer to the root retain more edges during pruning, and that the further a node is from the root node r, the more stretch it would inflict upon the tree and the network.

In Figure 2.4, let node c be the common ancestor of node u and v. Moreover,

$$d_N(T, u, r) = d_N(T, u, c) + d_N(T, c, r)\,, \tag{2.3}$$

$$d_N(T, v, r) = d_N(T, v, c) + d_N(T, c, r)\,, \tag{2.4}$$

Out initial claim stated that

$$d_N(T, u, r) \leq d_N(T, v, r)\,, \tag{2.5}$$

From Defs.(2.3,2.4,2.5),

$$d_N(T, u, c) \leq d_N(T, v, c)\,, \tag{2.6}$$

Also, it follows from Def.(2.6) that the difference Δ as in Def.(2.7) is non-negative.

$$\Delta = d_N(T, v, c) - d_N(T, u, c)\,, \tag{2.7}$$

Computing $\mu(T, m), m \in T$ statistically requires us to know the distribution of nodes over the tree. For example, if the size of the node cluster V'' in Figure 2.4 is much greater than cluster U and/or D, our initial claim would not hold true; $d_N(T, v, v''), v'' \in V''$ would not be effected by Δ, while every $d_N(T, e, v), e \in T - V''$ would incur Δ, resulting in nodes in U and/or D to have greater $\mu()$'s.

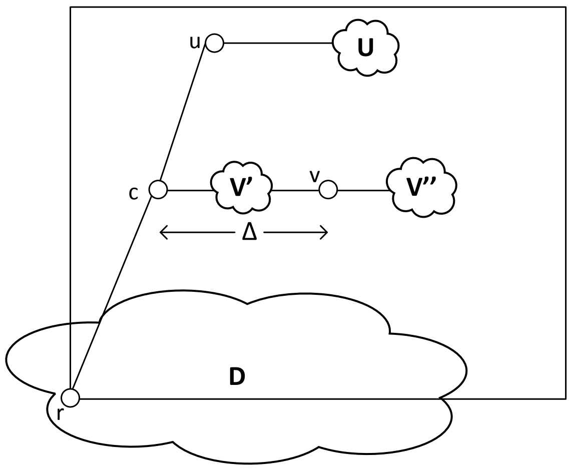


Figure 2.4: Tree T, pruned from root node r, with nodes u, v, c and node clusters U, V', V'' and D

In [32] and [33], the authors prove analytically and empirically that breadth-first-search pruning results in a degree distribution of $\hat{p_k} \approx 1/k^{\tau}$, where k is the node degree and τ stands for the degree exponent of the network. The node distribution probability has a decay of $\mathcal{O}(k^{\tau})$, which implies that V'' cannot be greater than U and/or D in size. As a result, $\mu(T, e)$ is an increasing function, as node e is moved further away from node r.

In order to capture this behavior, we simulate BFS pruning on a uniformly distributed random network, starting from a corner node and the most central node, as shown in Figure 2.5 and Figure 2.6, respectively. Empirical data is very much in line with our formal proof, showing the radially increasing behavior of the function $\mu()$.

When the resulting BFS pruned spanning trees are merged, by adding all

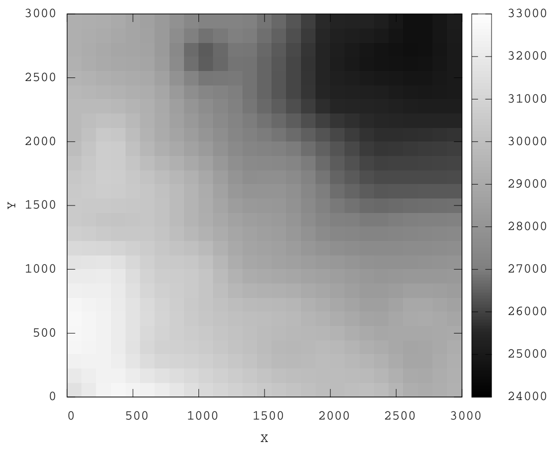


Figure 2.5: Corner node as root. Heatmap shows empirical $\mu()$

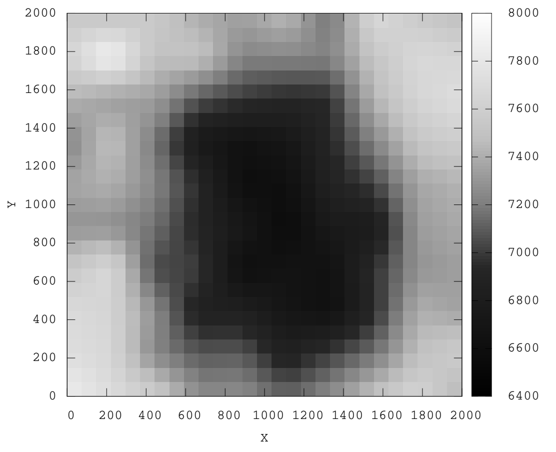


Figure 2.6: Central node as root. Heatmap shows empirical $\mu()$

of the edges from the five trees into a new graph and annotating which BFS tree they belong to, this juxtaposed graph would cover a greater percentage of the edges from the original graph, approaching the total network distance of shortest path routing (i.e. *stretch* ≈ 1), thus inflicting less latency at the forwarding layer.

2.3.3 DFS Ordering & Labeling

After the loop-free BFS trees are pruned by removing the looping edges, the next step is to run DFS for labeling. DFS labels are required by the nature of the interval routing, since intervals are assigned to node ports according to their vertical positions with respect to the root node.

Figures 2.7-2.10 exemplify the procedure of BFS pruning and DFS ordering on Figure 2.3. Corner nodes C1 to C4 are elected as D, B, I and M; BFS spanning trees are ordered and pruned (removed looping edges are denoted by dashed lines); and finally the nodes are assigned the DFS labels (shown in parenthesis). From here and onwards, we will refer to these BFS pruned / DFS labeled trees as BFS/DFS trees.

In order to formulate the problem, we introduce the following notation: For each corner node C = $\{C_1, C_2, C_3, C_4\}$, we have a BFS/DFS tree with labels L = $\{L_{C1}, L_{C2}, L_{C3}, L_{C4}\}$ and for each port per node we have a set of intervals I = $\{I_{C1}, I_{C2}, I_{C3}, I_{C4}\}$. These values are computed proactively at the controller when there is a new update from the nodes. This computationally intensive task, in return, is a trade-off for less signaling overhead within the ad-hoc network and

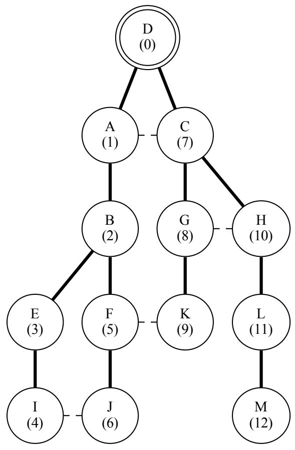


Figure 2.7: Node D is elected as corner 1.

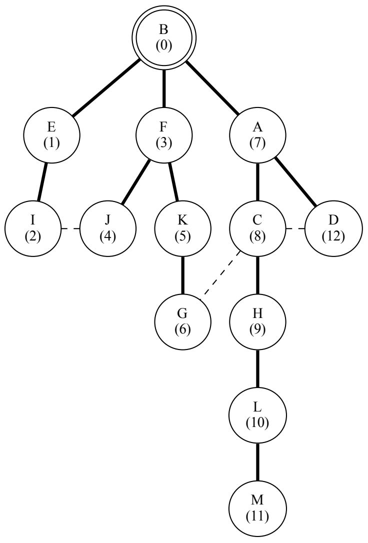
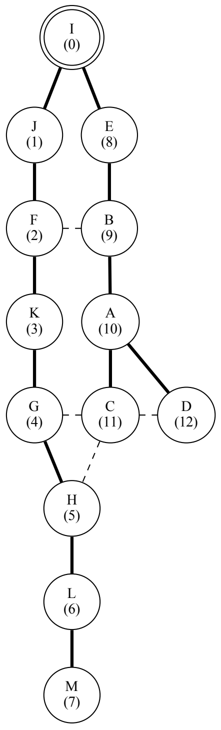


Figure 2.8: Node B is elected as corner 2.



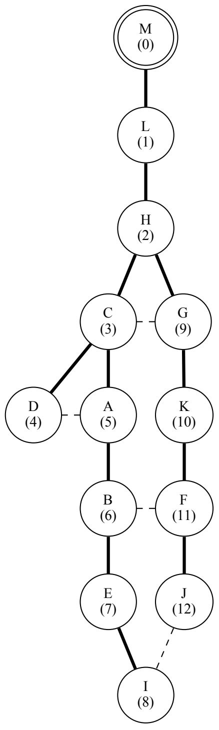


Figure 2.9: Node I is elected as corner 3. Figure 2.10: Node M is elected as corner 4.

much smaller routing tables.

2.3.4 Computing the Intervals

At this stage in the algorithm, five BFS/DFS trees have been generated, which means each node has received five DFS labels, one per each root. Computing the intervals requires the traversal of the trees, such that we can assign intervals at each port of every node, per each root, as described in Algorithm 1. There can be multiple intervals assigned at a port per root. Moreover, we allow forward and reverse directions for intervals. All descendants will have forward intervals; whereas the nodes connecting a node to the root might have reverse intervals.

Figure 2.11 demonstrates the intervals assigned to some of the ports of C1 tree.

2.3.5 Interval Table Minimization

The controller node will download a minimal set of intervals to the nodes, instead of the full five sets of intervals. The size of the interval tables can be reduced since some intervals are either exactly the same even in distinct trees, or they are a subset of one another. In order to determine the redundant intervals, we compare the set of intervals at each port of each node for each BFS/DFS tree, and if the labels point to the same set of nodes, one of the intervals is removed; if one is a subset of another, the smaller set is removed.

After the minimized interval information is downloaded to the nodes by

Algorithm 1 Algorithm for computing the intervals

```
1: Let $r$ be the index variable for BFS/DFS trees.
2: Let $n$ be the index for nodes.
3: Let $p$ be the index for ports.
4: Let $interval[r][n][p]$ be the interval lookup table, a list of lists.
5: Let $labels_fwd$ be a temporary list of labels.
6: Let $labels_rev$ be a temporary list of labels.
7: Let $currentNode$ be a temporary node pointer.
8: for r = BFS/DFS tree iterator; r++ do
9:   for n = Nodes iterator; n++ do
10:    for p = Ports iterator; p++ do
11:      if p connects the node n to the lower levels then
12:        Add all descendants through port p to $labels_fwd$. Sort. Append to $interval[r][n][p]$.
13:      else
14:                                          # comment: p points to n's parent node.
15:        currentNode = n
16:        while currentNode is not root do
17:          Iterate through lower-level ports of currentNode.parent, except the port connecting to currentNode.
18:          Add all descendants to $labels_fwd$. Sort. Append to $interval[r][n][p]$.
19:          Append currentNode to $labels_rev$. Reverse Sort.
20:          currentNode = currentNode.parent()
21:        end while
22:        Group consecutive labels in $labels_rev$. Append each group to $interval[r][n][p]$.
23:      end if
24:    end for
25:  end for
26: end for
```

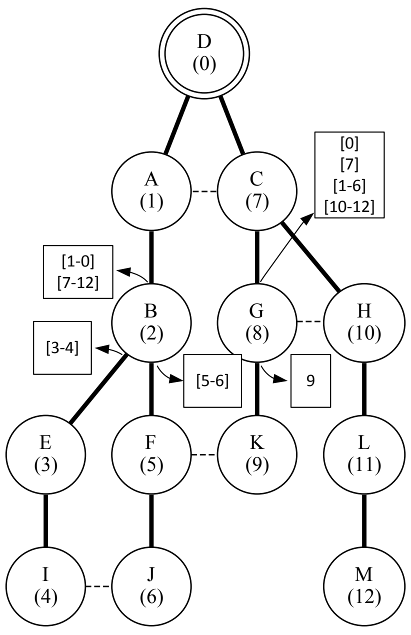


Figure 2.11: Intervals assigned to the ports of nodes B & G in C1 tree.

the controller, the packets are simply forwarded by following the multi-root interval routing algorithm as explained in the next subsection.

2.3.6 Data Plane Forwarding

Having decoupled the control plane from the data plane, our forwarding algorithm simply compares the destination label in the packet header to intervals at every port of the current node. Next-hop is computed incrementally at every node. The packet will either be in the source node, or a forwarding node in the source-destination path.

When destination label is compared with the intervals, the next-hop algorithm chooses between the five trees in order to forward the packet, by comparing the position of the destination label in the interval. In order to avoid loops, the previously selected tree's distance to destination is skewed to be one less than the actual value. Moreover, if there is a tie, we modulo the destination label to the number of trees and pick the closest tree whose index greater than or equal to the modulo value, as a tie-breaker.

2.4 Simulation Results

We run our simulations using NS-3 discrete event network simulator [34] and Boost Graph Library [35], on uniformly distributed mobile networks of various sizes. We use IEEE 802.11ac as the underlying PHY/MAC layers. Our test scenarios consist of networks with 50, 100, 250, 500, 1000, 2000 nodes, and were run over 10 distinct seeds. Finally, the simulation area has been expanded as the number of nodes are increased, in order to keep the average network degree constant.

2.4.1 Efficiency of BFS Pruning/DFS Labeling

In this subsection, we present the impact of BFS pruning on the network stretch, defined as the ratio of the total network distance of the subject algorithm to the total network distance of shortest path routing. Latency is proportional to the network stretch, which makes it an important metric in algorithm design.

In Figure 2.12-2.13, we present the comparison of the following schemes, in terms of stretch:

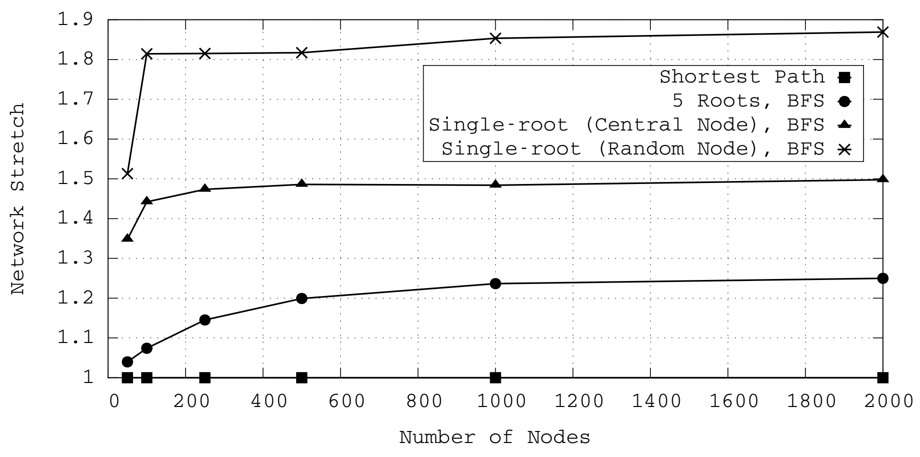


Figure 2.12: Comparison of various interval labeling schemes.

1. Shortest path: All edges are intact, and the distance between any two nodes is equal to the shortest path.

2. Single-root (Central Node), DFS: The tree resulting from the DFS pruning starting from the central node of the network.

3. Single-root (Random Node), BFS: The tree resulting from the BFS pruning starting from a random node of the network.

4. Single-root (Central Node), BFS: The tree resulting from the BFS pruning starting from the central node of the network.

5. Multi-root, BFS: Juxtaposition of BFS pruned trees resulting from choosing the four corner nodes and the central node as roots

The greater the stretch is, the greater the total network distance and latency are for the network. BFS pruned multi-root scheme has the smallest stretch

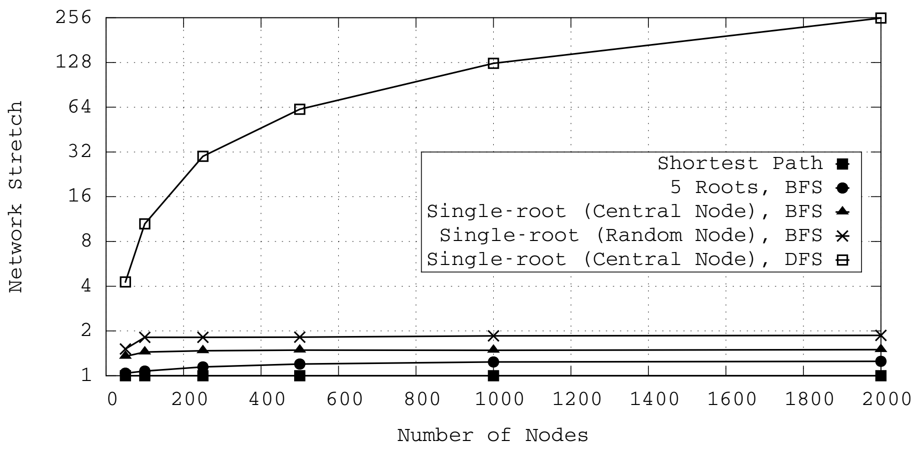


Figure 2.13: Comparison of BFS and DFS pruning.

compared to the other schemes, and thus it would achieve the lowest latency. The fact that the stretch saturates as the number of nodes is incremented is also a strong indication that our algorithm is practical even for large size networks. Furthermore, our results are in line with our findings in Figure 2.5 and Figure 2.6.

2.4.2 Interval Table Compactness

We present the number of interval entries in the flow tables and how much they can be further minimized in Table 2.1 and Figure 2.14. The reported values are the sum of all entries in all the network nodes. Percentage improvement after minimization is between 28 to 44, with a standard deviation of 6.34%. The values demonstrate that the minimization algorithm effectively reduces the number of interval entries in the flow tables.

In Figure 2.15 we compare the MINT to explicit routing algorithms (i.e.

Table 2.1: Total number of interval entries.

Network Size	Intervals	Minimized Intervals	Percentage Min.%
50	4033	2262	43.91
100	9625	6242	35.14
250	36156	26117	27.76
500	80271	55206	31.22
1000	235506	152204	35.37
2000	521177	297994	42.82

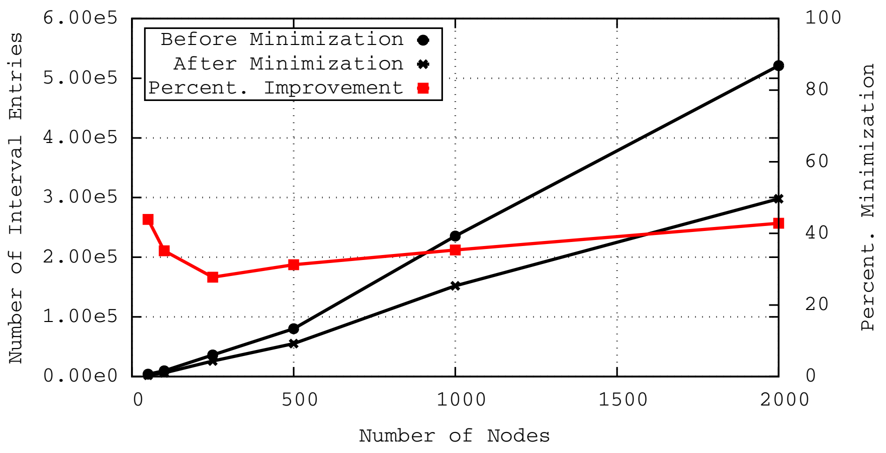


Figure 2.14: Total number of interval entries before and after minimization.

in each node, there is an entry for every other node), in terms of number of routing table entries. In order to emphasize the overhead of our approach using multiple roots, we also include a single-root (central node) scenario. Both single root and five roots scenarios induce flow tables with only a fraction of the size explicit routing algorithms yield.

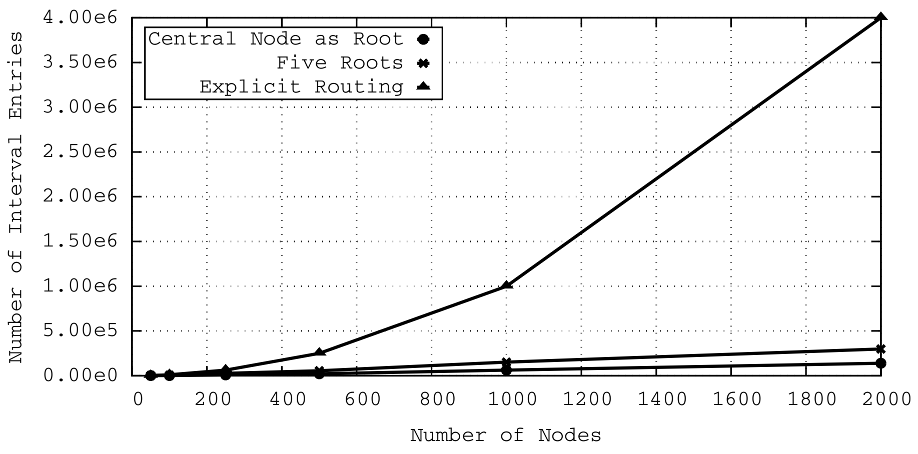


Figure 2.15: Comparison of MINT vs. explicit routing algorithms.

2.4.3 Interval Distribution per Root

Another important design metric for our interval routing algorithm is how balanced the distribution of intervals are per root. The distribution is a strong indicator of how well the load is distributed on various paths available within the network.

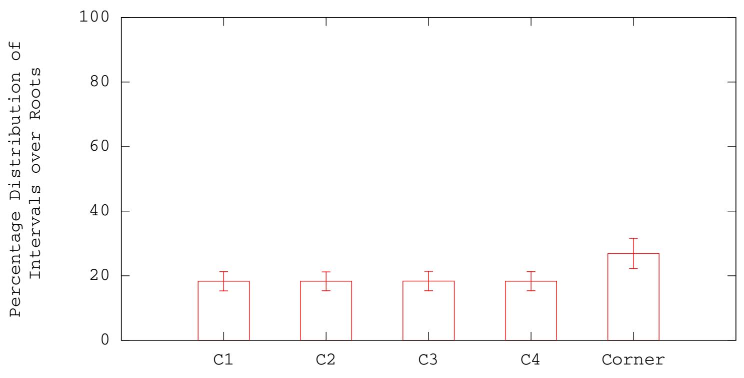


Figure 2.16: Percentage Distribution of Intervals over Roots.

In Figure 2.16, we present the distribution of minimized intervals over roots. Our simulations show that the interval distribution is independent of the network size and has very small standard deviation.

2.4.4 Performance: Delivery Ratio & Latency

In this section, we present the performance evaluation in terms of delivery ratio and latency, by comparing MINT to routing protocols such as OLSR and AODV. MINT achieves high delivery ratios and low latencies comparable to OLSR, while performing much better than AODV, as presented in Figure 2.17 and 2.18

MINT incurs slightly higher end-to-end latency for data delivery compared to OLSR as the number flows increase, which is mostly due to the fact that packets may take routes that are slightly longer than the shortest paths attained with OLSR. However, as the mobility increases, MINT achieves slightly better latency.

2.5 Conclusion & Future Work

Interval routing is an approach in routing protocol design that aims at storing routing tables at each node in a compact manner. A number of topologies that have an optimum interval labeling scheme are described in various publications, however an optimum scheme for uniformly distributed random networks is far from being achieved.

In this work we proposed a feasible multi-root approach for interval routing, by electing the central node and the corner nodes as roots. Starting from each root we used BFS pruning and DFS labeling in order to assign labels to the network

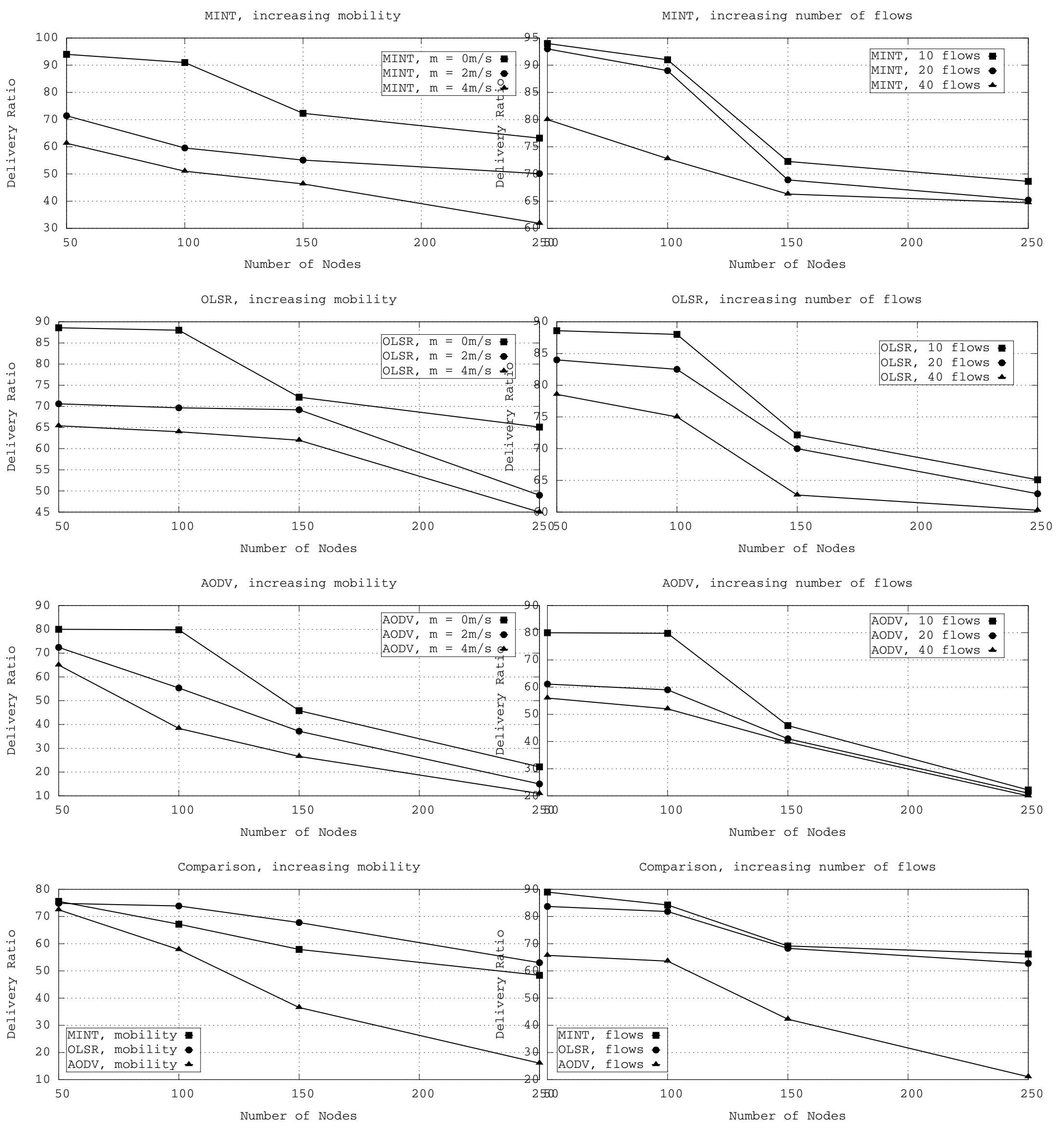


Figure 2.17: Performance Evaluation, Delivery Ratio

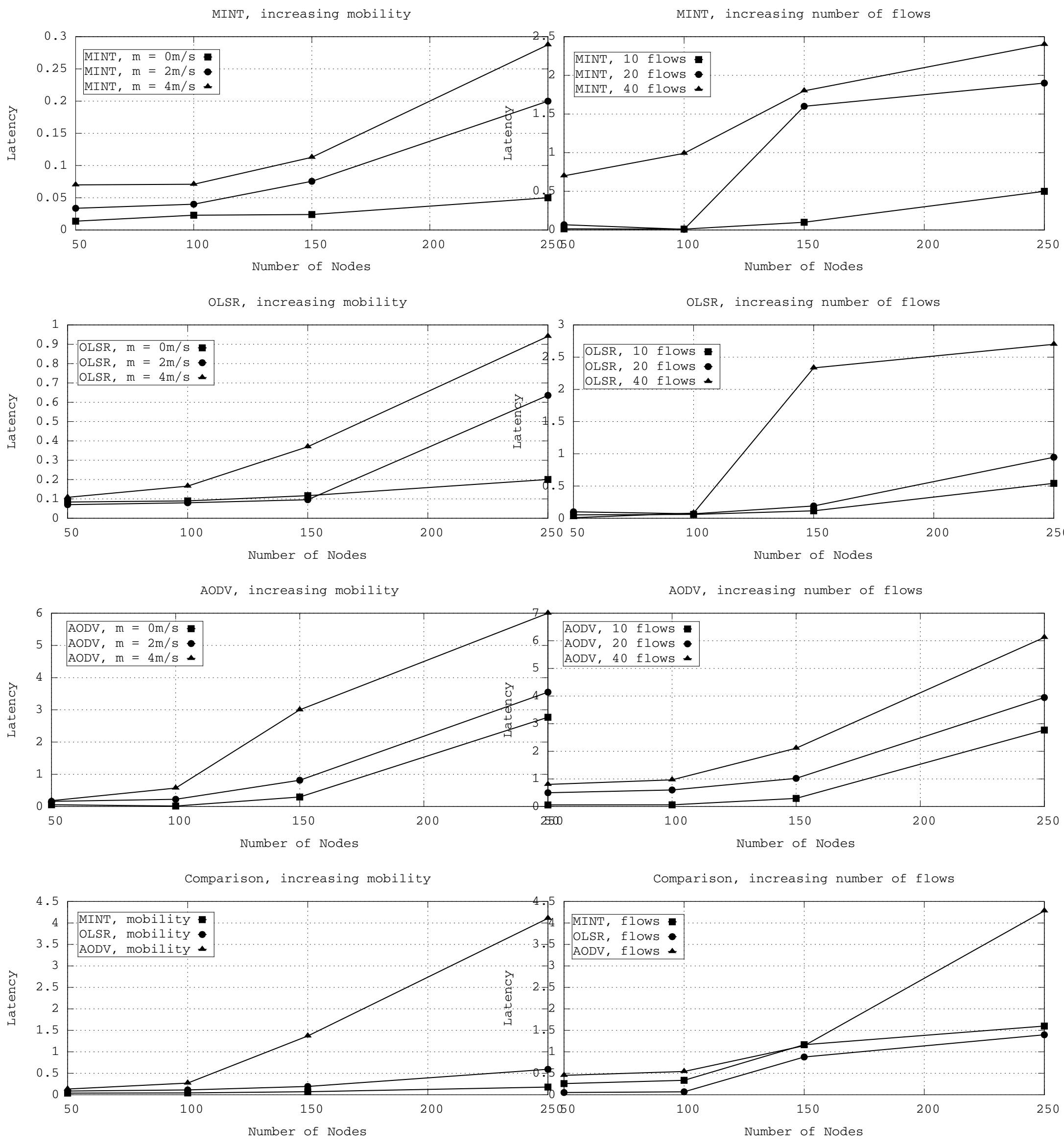


Figure 2.18: Performance Evaluation, Latency (seconds)

nodes, as well as intervals to each port of every node. We showed that our BFS pruned/DFS labeled multi-root approach offers $\mathcal{O}(1)$ stretch, by improving upon the single root interval routing approaches in earlier works.

We implemented a centralized controller for wireless networks that uses MINT as the underlying routing protocol. MINT enabled routing tables an order of magnitude smaller than explicit routing, that facilitate the controller to download the flow state to every node in the network using wireless links. In comparison to MINT, downloading explicit routing tables would be impossible due to the wireless bandwidth in a dynamic scenario. Using a centralized controller in order to virtualize the global network view and download the flow tables to the nodes reduced control plane significantly and enabled scaling to more than 500 nodes.

Furthermore, we presented performance evaluation of our algorithm on NS-3 simulator. Multi-Root Interval Routing (MINT) achieves high delivery ratios and low latencies comparable to shortest-path based MANET routing protocol OLSR and much better than AODV, while reducing the routing table size more than an order of magnitude.

For future work, we plan to incorporate an analytical model of routing tables compactness induced by our approach, in order to demonstrate the efficiency, in terms of the performance to compactness ratio, an important aspect of the interval routing design as described in [36].

Chapter 3

Software-Hardware Co-Defined Network Switch (SHADES) for a Label Switching Protocol

3.1 Introduction & Related Work

In this chapter, we present our Software-Hardware Co-defined Network Switch (SHADES) for a Label Switching Protocol. By taking the hardware component into account in the design flow, our system demonstrates a successful attempt to implement an Hardware-Software co-design embedded system instantiation for network switches.

Software Defined Networking (SDN) emerged as a novel computer networking paradigm to abstract the data plane from the control plane, and thus to achieve higher flexibility and architectural simplicity. This is implemented by computing the forwarding tables at the controller node, which are then disseminated to the other nodes. The data can be forwarded from a node to the next hops using these flow tables. In Section 2.3.1, we have explained in detail how a centralized controller can virtualize the network in order to compute the routes, thus remove the control plane overhead from the data plane links.

By using interval routing we have achieved compact routing tables, that can actually be disseminated to the flow tables in a wireless scenario. Other approaches using explicit routing tables dictate an order of magnitude more information to be downloaded over wireless links, which is not feasible given the low bandwidth of wireless communication. The benefits of using a multi-root interval routing approach has been described in detail in Chapter 2. Figure 2.15 clearly demonstrates that compact routing tables require much less information exchange, which makes this approach ideally suited to mobile scenarios. Using interval labels would enable forwarding to be handled at lower layers (Layer 2), instead of having to transmit the packet to the upper layers of hardware/software and examine the next hop information. IP (or any other routing protocol) data packets can be wrapped into label switched protocol packets, which then are forwarded to the next hops dictated by the label switched protocol.

Our proposal is to merge the SDN philosophy with label switching, which would enable the architecture to be implemented in a hardware-software co-design fashion. Introducing a hardware component into network protocol design might initially seem to combine two orthogonal design components, however the outcome is an architecture that achieves multiple levels of abstraction, flexibility, simplicity, scalability and high performance at once.

Kempf et al. [37] and Das et al. [38] describe extensions on OpenFlow that incorporate MPLS, and propose modification on NetFPGA [20] in order to implement a router for MPLS networks. Please note that this approach serves a different purpose than what we are trying to achieve by using label switching in order to introduce forwarding at lower layers and enable hardware acceleration.

State of the art switches use similar hardware-software co-design techniques for the implementation. In [39][40], Cisco Systems engineers describe the design effort that goes into Cisco switches, which rely on using flow tables written at the software routing layer; these flow tables then can be read by hardware logic components, and used for forwarding packets at Layer 2. Cisco switches also rely on

FPGAs for easier hardware reconfigurability of some components; time-to-market is also another important reason for the use of these easily reconfigurable ICs. Moreover they implement Layer 2 processing of packet headers in order to retrieve the destination address and search for the next-hop information in the flow tables.

Another important aspect of using MINT as the underlying routing layer presents itself when the destination address is searched in the flow tables. Comparing the destination address to all entries in an explicit routing algorithm would be costly, either in terms of time (i.e. CAM) or hardware cost (i.e. TCAM). By using smaller routing state, MINT improves the cost and power for forwarding.

Please note that the software control of the flow tables in Cisco switches is different from what we want to achieve in our work, despite the fact that we are using similar mechanisms to implement it on hardware. Cisco switches are used in wired networks, where the topology change is not an inherent aspect. Our approach is more dynamic, due to the mobile & dynamic nature of wireless routing, which requires the routing control layer to modify the flow tables dynamically. Undoubtedly, achieving Layer 2 processing and forwarding is the common element between our work and state of the art switches.

Hardware-software co-design enables communication channels between hardware components and software that does not exist in regular computing systems. Our System-on-Chip (SOC) is implemented on an Altera Field-Programmable Gate Array (FPGA), using the soft core microprocessor NIOS II with MMU, such that our system supports multiprocessing capabilities at software networking stack. The operating system (OS) is a variant of uClinux [41] with MMU support and it has been modified and recompiled by our team in order to achieve a fully functional switch with four network interfaces (NIC).

We show that the point-to-point delay caused by both hardware and software components from the NIC to the software routing layers can be avoided by forwarding on Layer 2. Moreover, we validate that SHADES successfully supplements the controller and routing layer algorithm described in Chapter 2 by using the control packets received from the controller to populate the flow tables for data plane forwarding.

In Section 2, we describe SHADES architecture and its building blocks. In Section 3, we present our results and describe how SHADES perform better in comparison to components that constitute it. Finally, Section 4 concludes the chapter.

3.2 Implementation

3.2.1 Label Switched Routing

In Chapter 2 we show that multi-root approach has a stretch of $\mathcal{O}(1)$ and proves to attain high delivery ratios and low latencies comparable to shortest path routing, both in static and dynamic networks. By computing the compact routing tables (i.e. intervals) at the controller node and disseminating them at every node, we achieve abstraction of the forwarding plane from control. The ability to realize

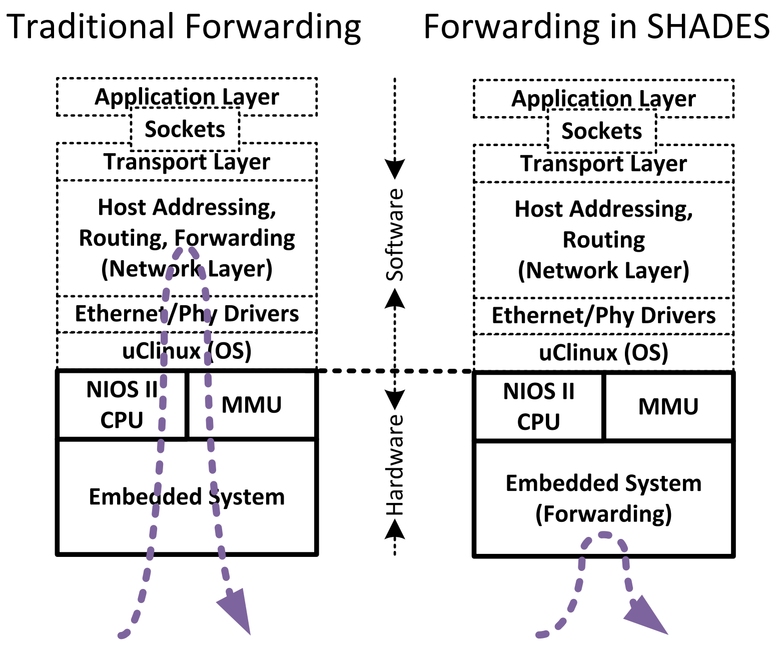


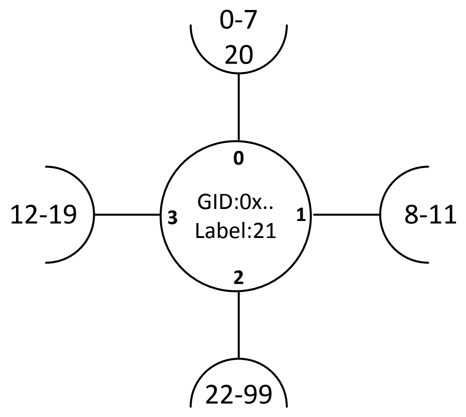
Figure 3.1: Comparison of Traditional Forwarding and SHADES

forwarding at the hardware layer has multiple benefits, such as low latency through the system, more computational resources for other tasks including control plane handling, and less congestion through the SOC bus (Figure 3.1).

Our architecture enables the forwarding function to be handled at the hardware layer by using a dual port/dual clock domain flow table memory, which contains the mapping of the intervals/labels to the next hop ports of the switch. In this way, we allow the software routing layer to modify the flow tables, while enabling the hardware forwarding layer to retrieve the next hop information without transmitting the frames/packets to higher layers.

Figure 3.2 demonstrates the interval label/port information within a node

$Node_A$. When a new data packet arrives at the switch, SHADES engine determines if the frame (1) is intended for the current node/switch, and thus the packet is forwarded to upper software layers, (2) if it needs to be forwarded to another node/switch, or (3) if it is simply dropped. As for forwarding, the next-hop node will be determined by comparing the destination label to the intervals in the flow tables. Finally, the frame will be delivered through the corresponding port.



Next-Hop Intervals	Switch Port Index
0-7	0
20	0
8-11	1
22-99	2
12-19	3

Figure 3.2: $Node_A$ Flow Table for HW Forwarding

When a packet is received by the destination node, SHADES engine forwards it to the software layers, where the original Layer 3 packet is stripped from the label switching encapsulator.

3.2.2 SHADES Top-Level System Architecture

In this subsection, we present the top-level system design for SHADES. Figure 3.3 is the system-on-chip (SOC) block diagram that depicts the integration of various components of our switch.

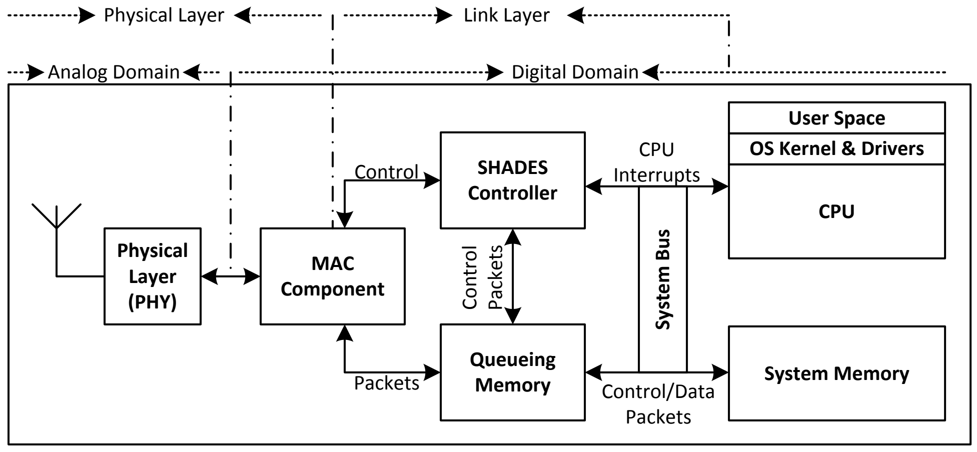


Figure 3.3: Hardware-Software Co-defined Network Switch (SHADES), Abstract

Our understanding of a switch design requires a full system integration, capable of running software in a microprocessor (µp), while still being able to handle time critical and computationally intensive tasks in hardware. If a switch could distinguish control plane and data plane packets at the hardware, it would gain the capability of taking the decision to deliver them to higher layers or forward them to the next hop nodes/switches. This would enable hardware layer forwarding, thus greater performance. The imagining of such a switch necessitates the incorporation of a complete system-on-chip (SoC) approach, which consists of a µp, main memory, various accelerators, a mac controller, and routing related algorithms at both the software and hardware layers.

We implemented a four port switch using a Terasic DE4 board [42] with an

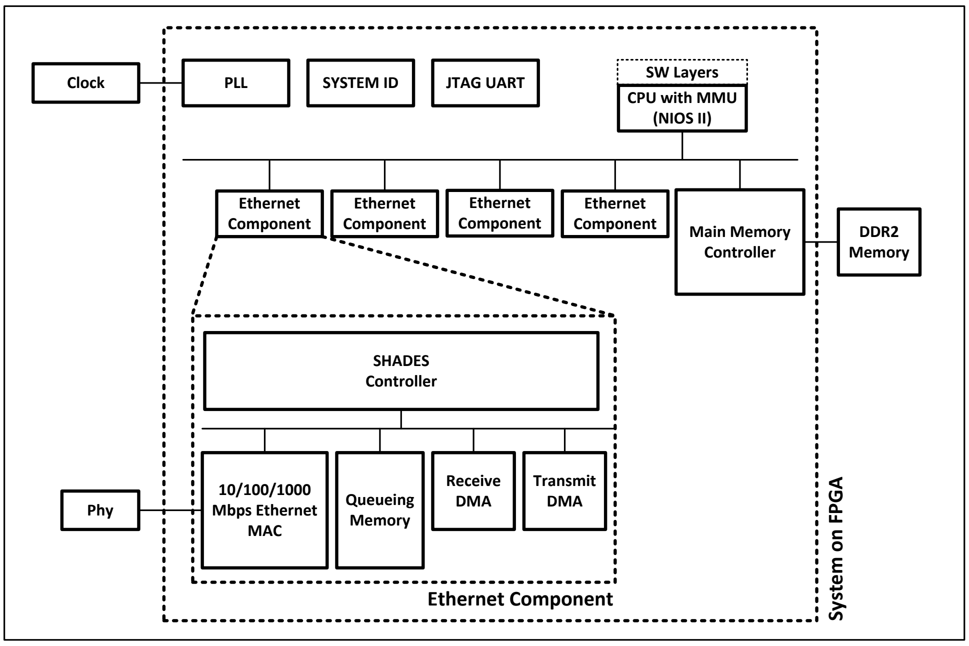


Figure 3.4: Hardware-Software Co-defined Network Switch (SHADES), Implementation for Altera FPGA

Altera Stratix IV GX FPGA (EP4SGX230KF40) [43] and we validated the component connectivity and functionality of all hardware components and software layers. We used the soft core microprocessor NIOS II with MMU to ensure that our system supports multiprocessing capabilities at software networking stack. Figure 3.4 demonstrates the same components from the previous figure, with the emphasis on design for FPGA environment.

We have used several reference designs from Altera, Terasic and other third parties, including [44], [45], [46], [47], [48], [49], [50], [51], and [52]. The operating system (OS) is a variant of uClinux [41] with MMU support and it has been modified and recompiled by our team in order to achieve a fully functional switch with 4 network interfaces (NIC). We have used tools such as SOPC Builder for system-level design, Quartus II for synthesis of the hardware, Modelsim for hardware simulation, Wireshark for capturing network packets, and GCC for miscellaneous compilation tasks, among a number of other tools.

3.3 Performance Results

We have carried out a number of performance evaluation tests with our hardware-software co-designed components, in order to demonstrate that our approach yields considerable improvements over current solutions. In Subsection 3.3.1 we implement a reverse loopback test in order to measure the delay our system would help avoiding when a frame is switched at the hardware layer.

3.3.1 Point-to-Point Round-Trip Delay Time

Forwarding the data packets at the hardware layer makes it possible to avoid the latency that would otherwise be inflicted by various hardware and software layers. The signals that are received by the PHY are deserialized into a frame at the MAC component. MAC Component will write the frame into the temporary Queueing Memory. SHADES Controller determines if the frame should be forwarded to the next hop or if it should be sent to the software layers.

If SHADES Controller did not exist, the frame would need to be transmitted to the DDR2 Memory by the Receive DMA component, through the BUS and the main memory controller. MAC would issue an interrupt to the microprocessor (NIOS II), and the interrupt would trigger the OS/driver routines, and that would initiate the reading of the packet from the main memory. Software Routing Layer would examine the header, make a routing decision, pass the command to the driver/OS layer that would in return issue an interrupt to the MAC component. DMA Transmit component would read the packet from the main memory (via the BUS) into the temporary Queueing Memory, from where the MAC component would serialize the frame and transmit it to the PHY.

In order to measure the latency through the components/layers explained in the last paragraph, we execute the following scheme: We connect a computer $Node_C$ to SHADES from one of the four ports, establishing a point-to-point connection. (i) Measuring the Round-Trip Delay Time (RTT) initiated from $Node_C$ to SHADES and back, including all layers, (ii) Measuring the RTT initiated from $Node_C$ to SHADES and back, bouncing the packet back at SHADES Controller using a reverse loopback mechanism, and finally (iii) computing the delta, we would obtain the latency through the aforementioned components/layers.

In many prominent publications including Carter and Crovella [53], Brik et al [54], and Obraczka and Silva [55], the authors rely on ICMP packets (PING tool) for RTT measurements. Especially in [55], the authors discuss how PING tool consumes few resources and is fast, and as a result it is a good candidate for reporting RTT.

We implement the reverse loopback mechanism based on the reference

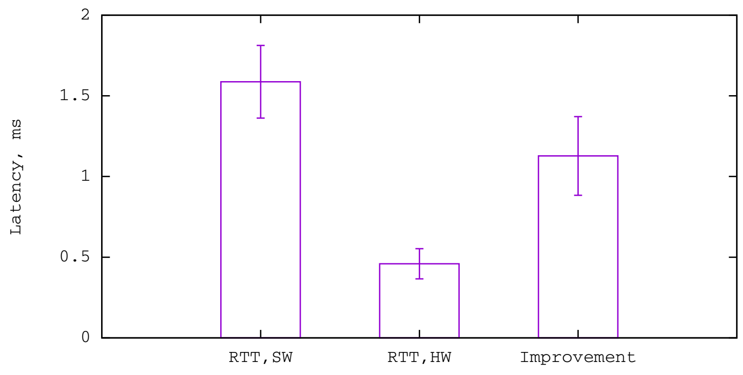


Figure 3.5: Comparing RTT for SW and HW Forwarding

design in [45]. In both cases, we use Wireshark to monitor the packets and measure RTT. In the first test, the RTT values are exactly the same as reported by PING. In the second test, we observe that Wireshark catches the packets that were “bounced” by the reverse loopback circuitry, however PING will not accept them as reply packets, and as a result we only report the Wireshark measurements for both cases. In Figure 3.5 we present the RTT measurements as well as the improvement delta in milliseconds. As one can easily observe, most of the delay is caused by components above the MAC layer, and by combining label switching and hardware forwarding capabilities, this delay can be avoided. Please note that the reported value is per point-to-point connection, thus the approach would improve each hop through which a packet is forwarded. Also note that FPGA systems are slower in terms of clock frequency, and as a result this improvement would be a smaller value in an ASIC system, but would still remain a major percentage of the overall delay.

3.4 Conclusion

SHADES exploits the hardware-software co-design paradigm in order to enable Layer 2 forwarding using MINT routing and a centralized controller to disseminate flow tables to the nodes. We are using similar mechanisms to state of the art network switches in order to implement Layer 2 processing and forwarding capabilities in our switch.

Our approach enables dynamic modification of flow tables that is inherent to the mobile wireless routing. By using interval routing we have achieved compact routing tables, that can actually be disseminated to the flow tables in a wireless scenario. Other approaches using explicit routing tables dictate an order of magnitude more information to be downloaded over wireless links, which is not feasible given the low bandwidth of wireless communication.

We have showed that our design can be implemented and prototyped on a Field-Programmable Gate Array (FPGA) based platform. We present real-time emulation results that are in line with our claim that the point-to-point delays inflicted by both the hardware and software layers between the NIC and software routing layers can be avoided by integrating design principles from SDNs, label switching and HW/SW co-design.

Chapter 4

Hardware Design and Implementation of a Network-on-Chip Based Load Balancing Switch Fabric

4.1 Introduction

In today's world, billions of users all over the world are connected through networks of different sizes, purposes and scopes. A network host might be one of the nodes in a local area network (LAN), the Internet backbone, the infrastructure nodes of a wireless network, or mobile devices in an ad-hoc network.

The communication between the nodes in a network is realized by a broad and diverse body of electronic and optical technology. The network switches in wired networks carry out the task of connecting two or more nodes (or networks), and perform the important functions of (i) determining the next destination of a packet that has been received by the router (control), and (ii) forwarding the data packets to the destination (forwarding). The switch fabric is a key building block of hardware based high-performance network switches, and they implement the second function. When a packet is injected into the router, it is stored in a buffering memory; the output port corresponding to the packet's destination is computed; and finally the packet is forwarded through the output port to its next hop in the network by following the rules in a scheduling algorithm. A switch fabric consists of buffering memories for temporary storage, scheduling unit(s) for forwarding, and other computational components that facilitate these tasks.

A high-performance switch for wired networks is required to support high bandwidths, in order not to become the bottleneck in the communication themselves. The switch fabric, being the key component in a router, constitutes an important part of the router design effort, and therefore it remains to be an open research problem. The switch fabric design consists of architectural design and scheduling algorithm design. The architectural design incorporates the interconnect topology and the buffering memory organization. The scheduling algorithm carries out the task of deciding which packet is to be forwarded, in case a number of packets compete for the same output port, resulting in contention; one of these packets will be forwarded, whereas others will need to be stored in the buffering memory, until later rounds.

In this chapter, we propose two Mini-Router Grid based switch fabric architectures to replace the current state of the art crossbar architectures. Crossbars require both input queuing and crosspoint queuing, and as a result they have very high costs, despite the performance. Moreover, both the input and crosspoint queues are shown to require large size memories. Our proposal replaces the crossbar by following the Network-on-Chip (NOC) approach, in which the switch fabric resembles to a grid network that consists of tiny 3 input/output port Mini-Routers (MR). The connection links between MRs are very small, and the control signals that implement back-pressure are pipelined automatically due to the nature of the point-to-point communication between MRs. In this way, the critical path is very short, yielding high frequencies and throughput.

Our architectures, Unidirectional Network-on-Chip (UDN) and Multidirectional Network-on-Chip (MDN) require very small buffer sizes compared to crossbars, achieves 100% throughput for admissible traffic, and are at least as scalable as other architectures in terms of hardware cost. The switch fabric grid is composed of N rows, one for each input/output port, and M columns of MRs. The number of columns can be decreased at the expense of performance, and as a result the switch size is decoupled from the cost growth, which constitutes the main advantage of UDN and MDN over crossbars.

We (1) carry out feasible hardware implementations of the MRG switch fabrics for FPGAs; (2) execute performance tests, both via RTL simulations and actual execution on FPGA, under uniform traffic flows; and (3) present results in terms of throughput, average latency, and average bitrate.

The rest of the chapter is organized as follows: in Section 2, we present the background information and the related literature. This section includes the information on various architectures that historically have been the milestones in the network switch design. Also, the Network-on-Chip (NOC) related concepts are described. Moreover, we describe the rationale behind how our architecture compares to these milestone architectures. In Section 3, we present our switch fabric architecture and its corresponding hardware implementation; the organization of components such as the buffering memory, Network Interfaces (NI) and Mini-Routers (MR); the routing algorithm (among MRs); and finally the scheduling algorithm. In Section 4, we present the RTL synthesis, RTL simulation and the actual execution results on FPGA. Section 5 concludes the chapter.

4.2 Related Work

4.2.1 The Switch Fabric

The switch fabric is one of the most important building blocks of a network router. Moreover, it requires the implementation of a scheduling unit, which regulates and grants permission for the pairing of input-output ports and buffers in between.

The FIFO scheduling, perhaps the simplest scheduling scheme for input buffering suffers head-of-line (HOL) blocking, where a packet at the head of the queue cannot be delivered, and therefore blocks the others behind it, resulting in throughput decrease (58.6%), increased delays, and congestion. A number of algorithms/architectures were proposed in order to remediate this shortcoming, including PIM, RRM, iSLIP [56][57][58], based on virtual output queues (VOQ), claiming a theoretical 100% throughput.

Another proposal, the load-balancing switch [59], claims greater scalability. VOQ architectures do not scale optimally as the number of ports is increased, and therefore become impractical. The load-balancing switch architecture does not have a scheduler, at the cost of duplicating the packets within the switch fabric.

The main design challenges for implementing switch fabrics include bandwidth, latency, scheduling algorithms, interfacing, and routing algorithms as in NOC based solutions. Several switch fabric architectures have been proposed, including the crossbar [60], shared-bus [61] and shared-memory [62] switches, which deal with these design challenges in various ways. The crossbar switch is the dominant architecture in today's high-performance switches, due to a number of reasons: crossbar switch is more scalable than the shared-bus and shared-memory; this is due to the limitations in bus transfer bandwidth and memory access bandwidth, respectively. Crossbar switch provides point-to-point connections and non-blocking properties, as well as supporting multiple simultaneous transactions, increasing the bandwidth and speed of the router.

Various queuing schemes have been proposed for implementing a crossbar switch. Switch fabrics has evolved from the Output Queued (OQ) to Combined Input and Output Queued (CIOQ) and then to the Combined Input and Crosspoint Queued (CICQ) architecture. This is exemplified by the three generations of the IBM Prizma switch [63]. Also there are a number of other works that combine various queueing schemes, including [64], [65], and [66].

Output queues require a speedup of N, denoting the number of input/output ports, which does not scale with the switch size. CIOQ use complex centralized scheduling units that are not feasible implementation-wise. A number of crossbar switch implementations implement queueing memories only at the crosspoints [67], which proves to require more queuing memory than Combined Input and Crosspoint Queued (CICQ) architectures [68]. CICQ implements smaller crosspoint queues in conjunction with large input queues (usually replaced by large VOQs), which is shown to require smaller memory space with respect to queues at the crosspoint only [69]. CIOQ or CICQ, crossbar switch architecture also require N 1xN demultiplexers and N Nx1 multiplexers for crosspoints, which are very high-cost are logics and make the design unscalable. Moreover, multiplexers and demultiplexers are controlled by a centralized logic component, which yields N^2 complexity for the decision logic. In [70], the authors demonstrate that 20-25 packet-sized queuing memory at each crosspoint is required to achieve 100% delivery ratio, under admissible traffic.

MRG based switch implements distributed control. Buffering and scheduling is handled within MRs. The only control is the backpressure mechanism that signalize the predecessor if there is an available buffer in the next MR; please note that this signal is the critical path of our design, and it is very short. Our architecture does not require any buffers outside the grid, unlike CIOQ or CICQ, and the buffer size is much smaller. In our simulations, every MR has 9 packet-sized queuing memory in total, and in Chapter 5 we analytically reaffirm our simulation results.

A number of papers analyze the power consumption in switch fabrics by analytical methods [71] and by simulation [72][73]. In [72], the breakdown of the power consumption is as follows: 33% for clocking, 22% for buffering memories, 17% on the links, and 15% for the switching activity. Clocking and link power is frequency and technology dependent, and as a result cannot be improved much by architectural design. However, by decreasing the buffering memory size, the system's power consumption can be improved. Our buffering mechanism would definitely be a step in the right direction.

4.2.2 Network-on-Chip

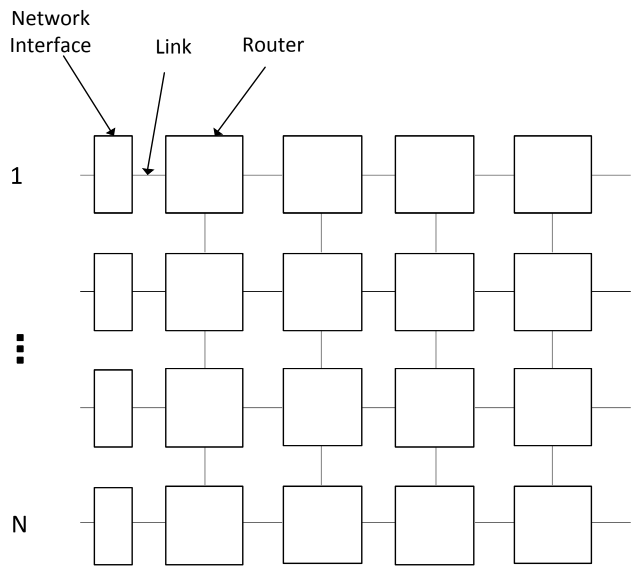


Figure 4.1: NOC Switch Fabric

Network-on-chip, a relatively new concept that emerged as a system-on-chip (SOC) communication methodology, borrows many ideas from the computer networks, the domain in which the research on routers and packet switching has matured. However, they need to be adapted, since there is no direct translation of these methodologies.

Figure 4.1 represents a NOC router, in the form of a regular N-by-N mini-router (MR) grid. Computational cores in a SOC are connected to each other via this communication fabric, composed of network interfaces (NI), and MRs.

The NIs act as an abstraction layer between the computational cores and MRs. The data to be communicated in between these cores are packetized in NIs and transmitted to the next-hop router, in equally sized flits.

An MR might have multiple packets in the buffers competing for the same output port, resulting in contention. A scheduling algorithm computes which packet has to be forwarded prior to the other packets. The arbiter, the hardware embodiment of the scheduling algorithm, makes a link in between the chosen buffer and output port, such that the packet is forwarded. In this chapter, the scheduling and arbitration terms are used interchangeably. Round Robin offers fair scheduling, assigning each resource equal usage in circular order, which results in a starvation-free system.

The communication through the NOC is pipelined automatically due to the nature of the point-to-point communication in between MRs. In this way, the critical path is restricted to the control signals in the switch NOC fabric, improving the scalability and throughput.

Some other important concepts in NOC are topology, routing, flow control, buffer management, quality of service and network interfaces and they have been studied in acclaimed proposals such as Aetherial [74], Nostrum [75], Xpipes [76], Intel 80 Core NOC [77], and Mango [78]. They all make different design decisions, to achieve their design goals.

Mesh-based NOC architectures and NOC routing algorithm have been discussed in various other publications, including [79, 80, 81, 82].

4.2.3 Network-on-Chip Based Switch Fabric

Recently, functional-level designs of two novel Network-on-Chip (NOC) based switch fabric architectures were proposed: Unidirectional NOC (UDN) and Multidirectional NOC (MDN) [83][84], as a replacement of the buffered crossbar switch fabric architecture (Figure 4.2), targeting greater scalability and flexibility, as well as greater performance per hardware cost, compared to buffered crossbar switch fabric.

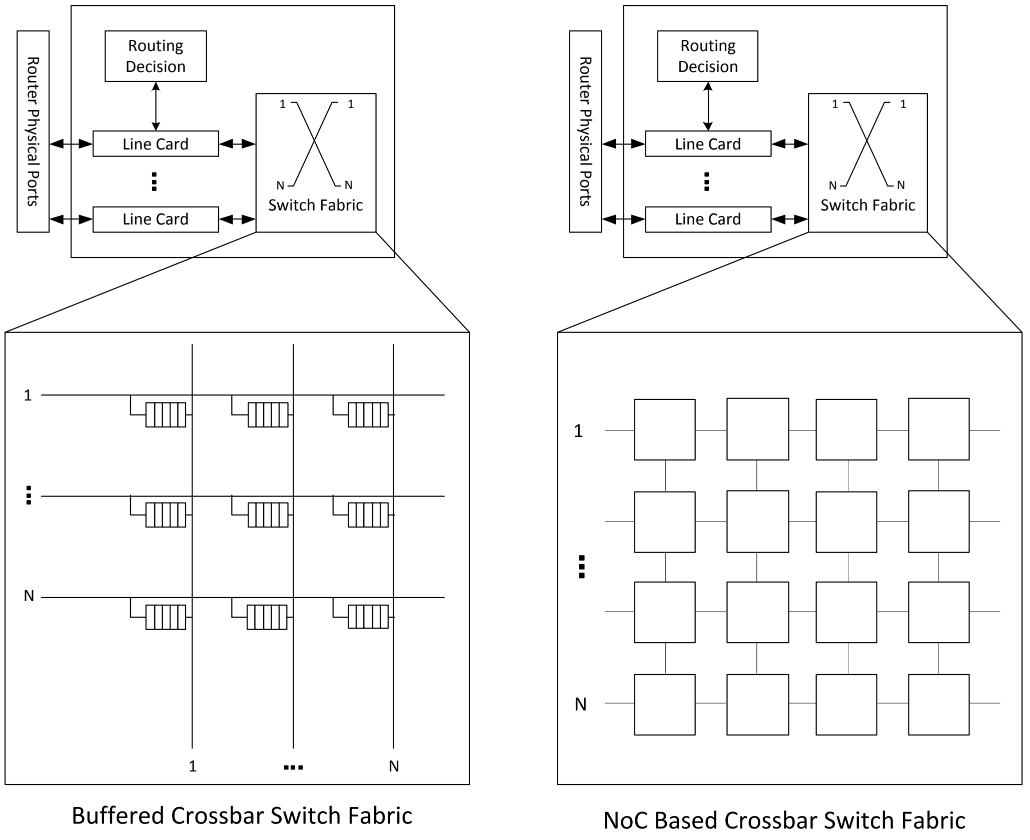


Figure 4.2: Buffered Crossbar and NOC Based Crossbar Switch Architectures

The crossbar-based switch fabric architectures offer very high performance and are widely used in high-performance routers. However, their cost grows quadratically with the input/output port count, since they require internal crosspoints (and buffers) for every input/output port pair. The UDN proposal decouples the number of ports from the cost growth, and therefore is able to achieve subquadratic growth. The MDN, in return, is quadratic, however it achieves greater performance/cost ratio, for smaller number of ports. Both of the proposals introduce load-balancing without duplicating the packets, which in return improves the throughput and latency.

NOC, a paradigm of on-chip communications, with its basic concepts borrowed from computer networks, is proposed to be applied back to its original domain, to remedy some shortcomings in the switch fabric design. In regard to this matter, the basic building blocks of NOC, including the buffers, flow control, arbitration and routing decision units need to be used in the correct combination of schemes/specifications, to be able to provide a competitive solution.

4.3 Hardware Design

4.3.1 UDN and MDN Architectures & Algorithms

The block diagrams for UDN and MDN switch architectures are presented in Figure 4.3 and Figure 4.4. The main difference among the two switch fabrics is how their input/output pins are placed in the layout. In UDN, the input pins are placed on the west side of the layout, whereas the output pins are on the east side. On the other hand, in the MDN switch, the pins are placed all around the peripheral, where input and output pins are next to each other. UDN MRs have either 2 or 3 I/O ports, whereas MDN MRs have 4 I/O ports.

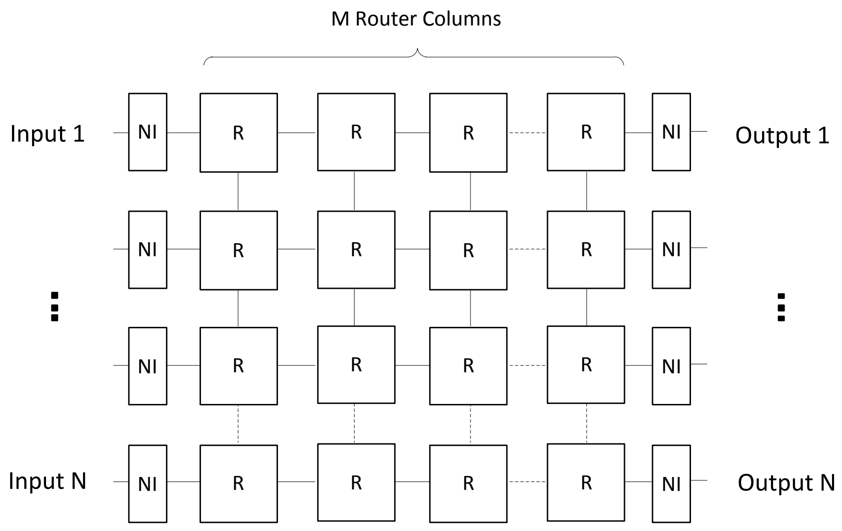


Figure 4.3: UDN Architecture

The UDN and MDN switch architectures have the same NOC specifications. The switching mode is store and forward. We choose to apply buffered flow control, implemented by FIFO input buffers. Buffered flow control requires a (buffer management/backpressure) mechanism, which we choose to be the valid/accept scheme, instead of the credit-based scheme in the original proposal [83], in order to decrease inter-MR communication overhead. The buffer size is four packets. The scheduling is based on iSLIP [58]. XY Modulo Algorithm allows deterministic and adaptive routing within the MR mesh, where routing path decision is made in an incremental fashion, at each MR. We choose to implement our switch fabric for

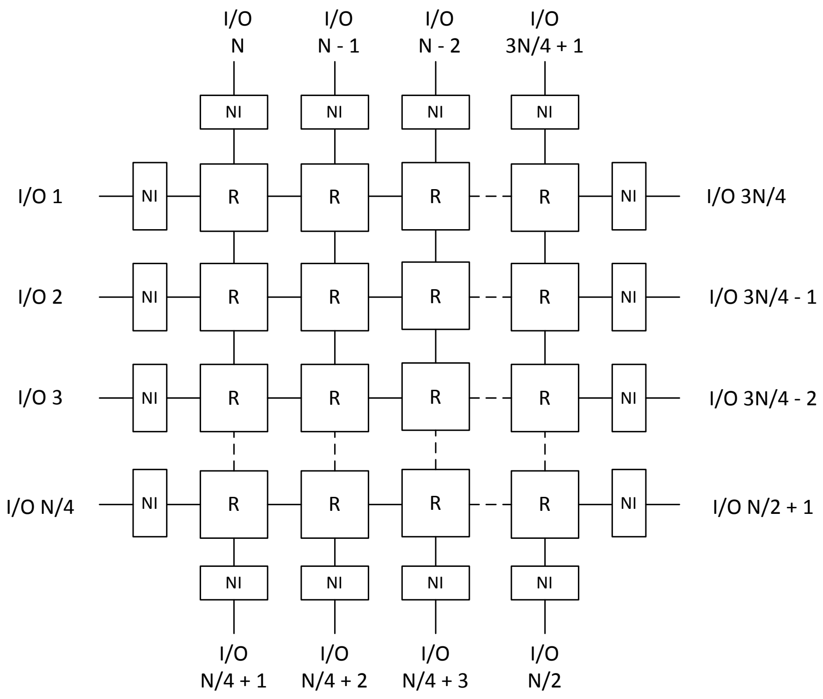


Figure 4.4: MDN Architecture

fixed-length packets, for simplicity.

The size of the UDN switch, as shown in Figure 4.3, is defined by the 2-tuple (N, M), where N denotes the number of input-output ports, and M denotes the number of MR columns. Some restrictions apply to (N, M) values: $N \in \mathbb{N}$, and $N \geq 2$; $M \leq N$ and $M = 2^m$, where $m \in \mathbb{N}_0$. The restrictions on M are caused by the routing algorithm involving Modulo M operations (See Section 4.3.6 and [83]). Because Modulo M operation requires division in case $M \neq 2^m$, but it is a simple bit-selection operation in case $M = 2^m$, we can avoid the extra cycles caused by the division operation by applying this restriction. With some minor modification in the routing algorithm, $M = N - 1 \Rightarrow N = 2^n$, where $n \in \mathbb{N}_1$ is also a possible (N,

M) combination. The number of routers in the UDN switch is equal to N $\times$ M.

The size of the MDN switch, as shown in Figure 4.4, is a function of N, which denotes the number of input/output ports. Because the input/output ports are placed around the peripheral of the switch, some restrictions apply to N: N = 4 $\times$ n, where n $\in \mathbb{N}_1$. The input/output ports are placed counter-clockwise, starting from the West side of the layout. The ports in between 1 and (N/4) are placed on the West; (N/4 + 1) and (N/2) on the South; (N/2 + 1) and (3N/4) on the East; and (3N/4 + 1) and N on the North side of the layout. The number of routers is equal to $(N/4)^2$.

In the UDN and MDN switch design, the same network interface, flow control unit, and buffering memory modules are used. The UDN and MDN MRs are different, due to the different number of ports.

The UDN is a deadlock-free architecture due to its unidirectional nature, whereas the possibility of deadlocks in MDN are avoided with the use of virtual channels [85].

4.3.2 Input Buffers (FIFOs)

The input buffer is implemented as a circular queue. The read/write register positions are marked by the Read Pointer (RP) and Write Pointer (WP). During a write operation, the data is written into this register, and the WP is incremented. In the same way, after a successful read operation, the RP is incremented to point to the following data. Status register (SR) is incremented after each write operation and decremented after each read operation. If SR = 0, then the buffer is empty (Empty Status Signal is set); therefore there cannot be made any read operations. If SR = Buffer Size, then the buffer is full (Full Status Signal is set); therefore no more write operations are allowed until SR is decremented. The Empty Status Signal informs the packet forwarding unit (PFU) in the current module (NI or Routers) about the availability of a valid packet for a read operation. The Full Status Signal generates Accept Signal, which informs the previous module's (NI or MR) PFU about the availability of buffer space.

4.3.3 Network Interface (NI)

The Network Interface (NI) is the module that acts as an abstraction layer in between the network protocol and internal UDN/MDN switch protocols. There are two types of NI: Input NI (INI) (Figure 4.5) and Output NI (ONI). When a packet is injected into the switch, INI encapsulates (packetizes) them into U(M)DN packets, and transmits them to the next router, in equally sized flits. ONI, in return, receives the U(M)DN flits, strips (depacketizes) the original packet, and ejects it from the switch.

4.3.4 UDN 3 I/O MR

The block diagram for a 3 I/O Port Router is given in Figure 4.6. There are 3 input ports, West, North and South; and there are 3 output ports, East, North and South. The router input buffer modules are placed at each input port, whereas

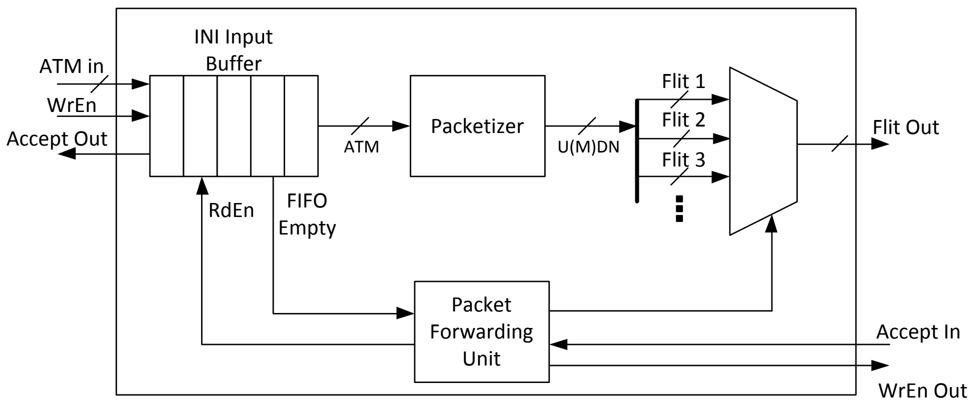


Figure 4.5: Input Network Interface Block Diagram

the PFUs are placed at each output port. At each round, the arbiter virtually connects input ports to output ports, according to the scheduling algorithm.

4.3.5 MDN 4 I/O MR

The MDN architecture is based on UDN; therefore, most of the UDN modules are also being used in the MDN switch, including NIs, PFUs and Input Buffer. The arbiter is also very similar in terms of basic principles, however it is more complex due to handling the connectivity between four input buffers and four PFUs. Moreover, MDN requires virtual channels to avoid deadlocks: WEST, NORTH and SOUTH input ports of the WEST-most MR column; EAST, NORTH and SOUTH input ports of the EAST-most MR column; and NORTH and SOUTH ports of the other MDN MRs have a pair of buffers, demuxed at their input and muxed at their output (Figure 4.7). In the functional-design proposal [83], the virtual channels are not muxed; however this results in the arbiter handling seven pairings, rather than

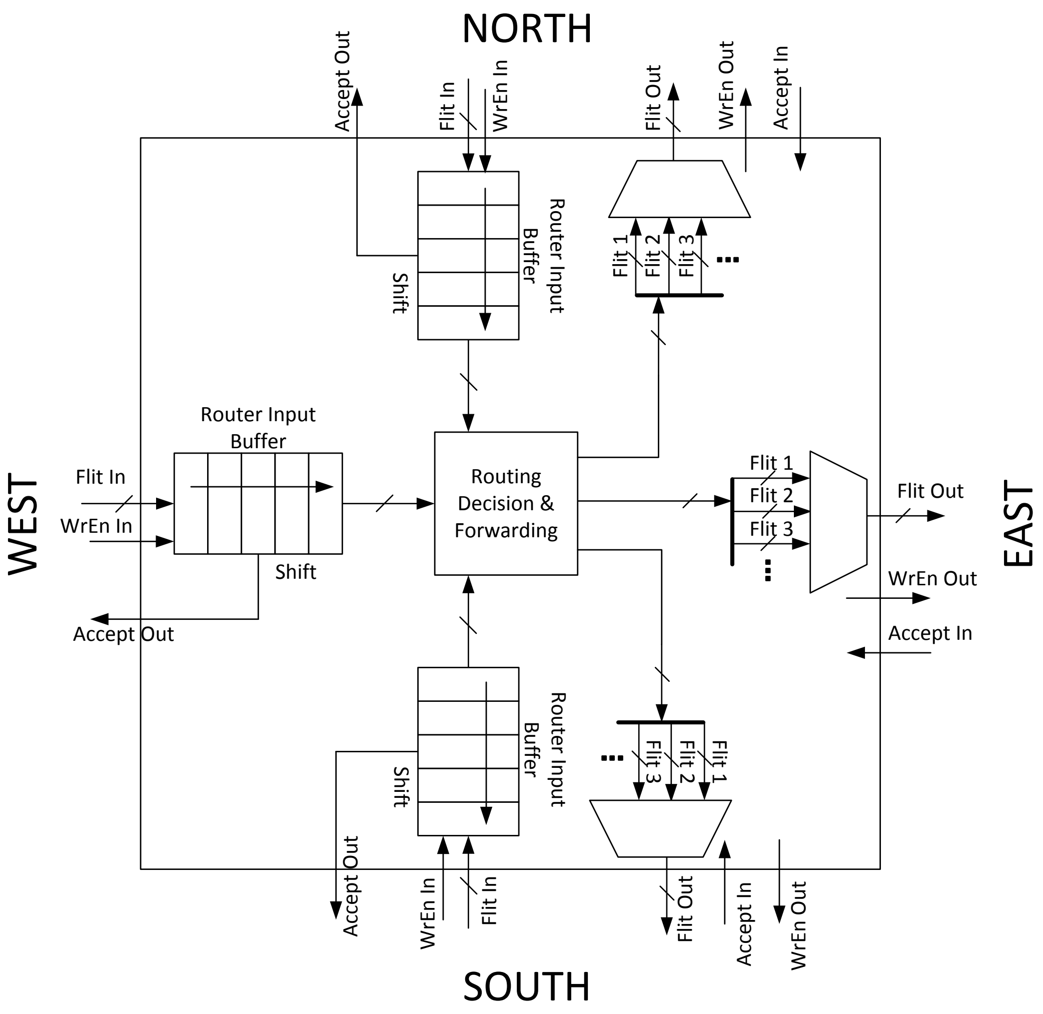


Figure 4.6: 3 I/O Port UDN Router, Top-Level Block Diagram

four, and thus not feasible in terms of an actual hardware implementation.

4.3.6 Modulo XY Routing Algorithm

Both UDN and MDN routing algorithms are based on modulo operation. The algorithms make a balanced distribution of the traffic over the columns or rows, thus earning its name XY Modulo; this means that the modulo operation is applied

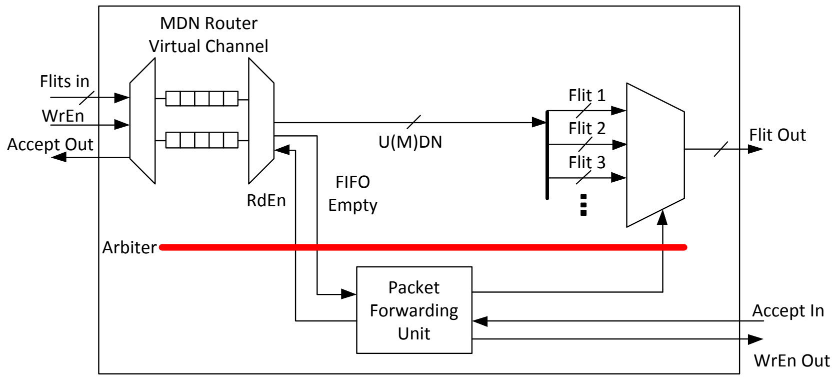


Figure 4.7: Virtual Channel for an Input Ports of MDN Router

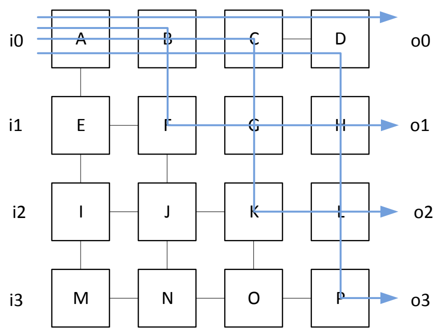


Figure 4.8: XY Modulo Routing

if the communication is on the X axis (from MR's West Input Port to MR's East Output Port as in UDN and MDN, or from MR's East Input Port to MR's West Output Port or vice versa as in MDN) or on the Y axis (from MR's North Input Port to MR's South Output Port, or vice versa, as in MDN). In Figure 4.8, we exemplify the XY Modulo routing on the X axis, where packets are injected into the switch from i0 input port, with destinations to the o0 - o3 output ports. The packets are routed on different router columns per input/output port pairs, distributing the load on the switch.

4.3.7 Scheduling Algorithm

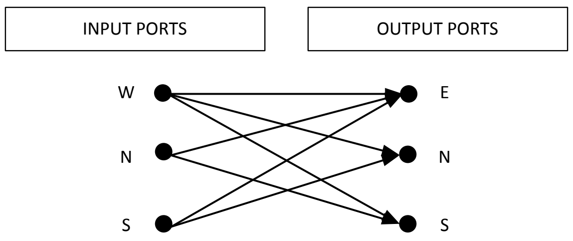


Figure 4.9: Bipartite Graph Matching Problem in UDN

The scheduling unit resolves the contention among the input buffers of a router, competing for the same output port, as well as controlling the arbiter to virtually connect the chosen input buffer to the output port.

The scheduling of the UDN switch is a bipartite graph matching problem. This can be formulated as G = (I, O, E) where I denotes input ports (W, N, S), O denotes output ports (E, N, S) and E denotes the edges. The graph is a directed graph. The unmatched graph is presented in Figure 4.9. Similarly, MDN scheduling is a H = (I', O', F); I' = (W, E, N, S), O' = (W, E, N, S) and F denotes the edges. The scheduling is based on the iSLIP algorithm [58] running a single round.

4.4 Simulations, Execution & Results

4.4.1 RTL Synthesis

The RTL synthesis is carried out on Xilinx ISE 11.1, using Xilinx Synthesis Technology (XST) tool, with the settings ‘Optimization Goal: Speed’, ‘Optimization Effort: Normal’ and ‘Keep Hierarchy: No’. The specific FPGA device is Virtex 5 - XC5VTX240T, FF1759, -2.

The XST tool reports the area results in terms of ‘Number of Slice LUTs’ and ‘Number of Slice Registers’, unlike the results for older platforms like Virtex 4, which reported a single value for slice usage; therefore, all the performance & cost analysis will be reported in 2-tuple. We do not present the results in metrics such as logic cell count, gate count or ASIC area size, since the design is made for the reconfigurable platforms, and the conversions from FPGA metrics are not meaningful, even in terms of providing estimates.

The modules that constitute the UDN/MDN switches have different tasks and therefore different weights on the combinational and sequential circuits. This can be observed by comparing the weights of the number of LUTs and Registers for any module. The synthesis results of the individual modules are given in Table 4.1.

Synthesis results show that the frequency of the UDN switch fabric is independent of M. In Table 4.2 we present frequency results for various (N, 1) UDN and (N) MDN switches. (N, N-1) UDN has higher operational frequencies than the corresponding (N) MDN switch, at the expense of greater cost per port number.

Table 4.1: Synthesis Results for individual UDN/MDN Modules

Module Name	**Size (# of Slice LUTs)**		**Max Frequency**
	as LUTs	**as Regs**	**(MHz)**
NI Buffer	1278	853	734.7
Input NI	359	857	597.229
Output NI	450	857	510.843
XY MODULO	2	-	-
Router Buffer	531	1016	508.414
Virtual Channel	532	1016	508.414
Router (2 I/O Port)	1679	2045	269.485
Router (3 I/O Port)	3073	3073	257.107
Router (4 I/O Port)	6197	4112	255.115

UDN arbiter is responsible for three I/O ports, whereas MDN's is responsible for four ports, resulting in longer critical path. The (8, 7), (16, 15), (32, 31) UDN switches require more than 100% of the resources on a Virtex-5 (XC5VTX240T, FF1759, -2), and therefore cannot be placed on the FPGA. On the other hand, only (32) MDN switch cannot be placed on the FPGA.

Table 4.2: Comparison of Synthesis Results, I

Switch Sizes		**# of Slice LUTs**		**# of Slice REGs**	
UDN	**MDN**	**UDN**	**MDN**	**UDN**	**MDN**
(4, 3)	**4**	31759	8933	37590	10969
(8, 7)	**8**	179978	30387	171508	30153
(16, 15)	**16**	762421	108996	734114	93203
(32, 31)	**32**	3137269	410697	3039532	318016

Switch Sizes		**Frequencies**	
UDN	**MDN**	**UDN**	**MDN**
(4, 3)	**4**	290.252	250.591
(8, 7)	**8**	283.168	240.667
(16, 15)	**16**	252.139	212.227
(32, 31)	**32**	220.146	173.214

The UDN vs. MDN switches comparison is tabulated in Table 4.3. The cost of the UDN switch is a function of the product of N and M, with M varying from any small number to N-1. The UDN cost growth is only quadratic in the worst case of M = N-1 (yielding the greatest throughput); however keeping M small, it's possible to keep the growth subquadratic, unlike the crossbar switch fabric, while achieving acceptable throughputs. The MDN yields quadratic growth of $(N/4)^2$. Even though the 4 I/O Port MDN MR consumes twice the amount of resources as the 3 I/O Port UDN MR, and 3.5 times as the 2 I/O Port UDN MR, the comparison of the overall UDN and MDN switch fabrics show that MDN is more cost efficient, for small N values. On the other hand, the operational frequencies of the MDN switches of various sizes are below the operational frequencies of the UDN switches, which would affect the performance.

Table 4.3: Comparison of Synthesis Results, II

	Arbiter Size	**MR Size (LUTs/REGs)**	**MRs**	**Cost Increase**
UDN	3	3073/3073	N×M	Subquadratic
MDN	4	6197/4112	$(N / 4)^2$	Quadratic

4.4.2 On-Chip Packet Switching Latency

We have validated the UDN switch (N, M) = (4, 3) on FPGA. Each input port of the switch fabric is connected to a linear feedback shift register (LFSR) based pseudo-random packet generator [86], as shown in Figure 4.10 and Figure 4.11.

The packet generators and switch fabric are initially idle. Once the CPU

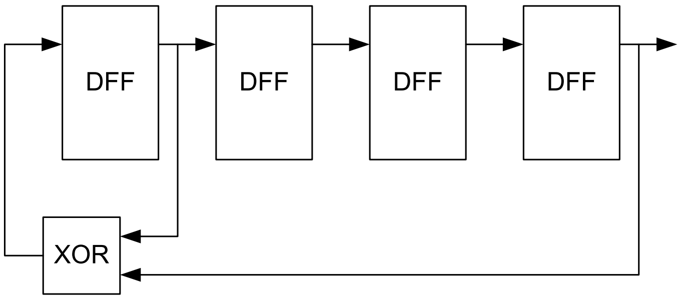


Figure 4.10: Linear feedback shift registers (LFSR) for pseudo-random packet generation.

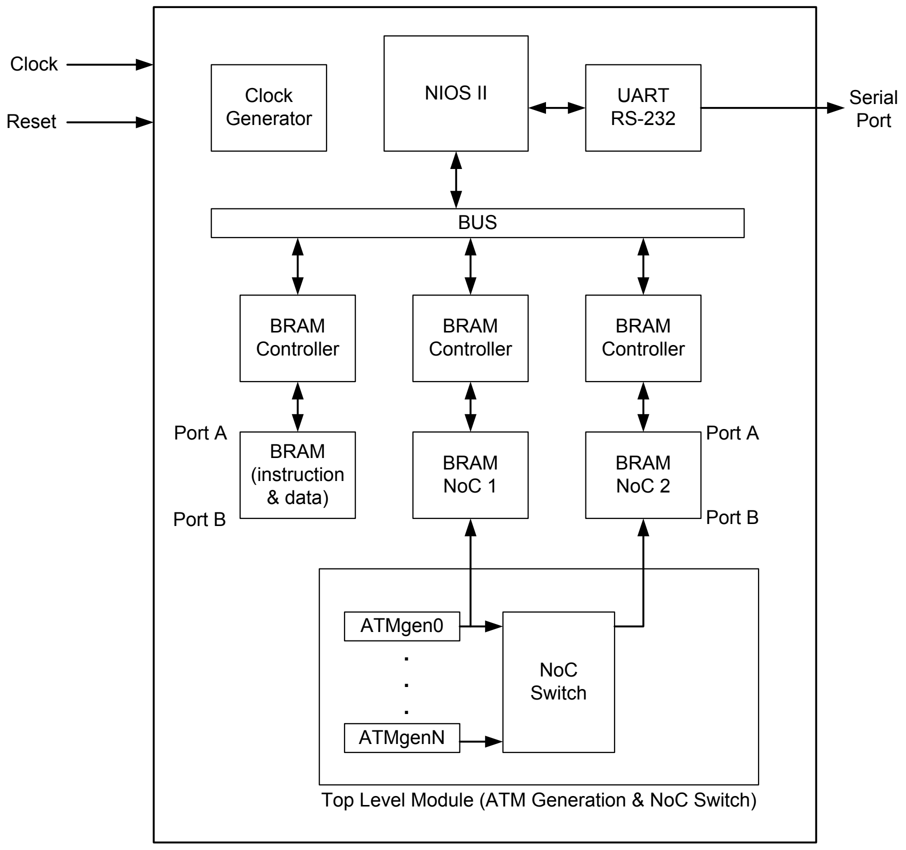


Figure 4.11: Block diagram for the run-time on-chip testing environment.

is initiated and our validation software starts to run, a trigger register is set to 1, which starts the packet generation. The headers of the incoming packets are written on a dual port RAM, which can be accessed by the CPU, after the packet generation comes to an end. When the packets are transmitted through the destination output port, their headers are also written to a dual port RAM. In this way, the validation software can compare the inbound and outbound packets, and verify if the system works correctly. With this approach, we were able to validate that all the packets have been switched correctly.

Using this system, we also carried out some performance analysis. Once the system runs for a certain period of time (i.e. cold start), the trigger register is reset, stopping the packet generation. Measuring the time period (in terms of cycles) in between the set and reset of the trigger register, it has been observed that the performance on the FPGA matches the simulation results. The error percentage is 5% to 9% for both latency and delivery ratio in all of the tests we have run, which confirms the validity of our approach. The error percentage can be attributed mainly to the fact that pseudo-random generation based on LFSR suffers from generating exactly one more 1 than 0s (or viceversa), and as a result traffic flows are not perfectly uniform. Additionally, software timers are not very accurate, inflicting extra latency themselves, which might be a minor factor in the error [87].

4.4.3 Simulation

The simulations were carried out on RTL descriptions of hardware components, with functional models of traffic generators/sinks. For the UDN, M=N-1 yields the greatest throughput for all N values; therefore, in this section M is always chosen to be N-1. The throughput for (N, N-1) UDN switches, under uniform traffic, is 99.54% and 97.34%, where N=4 and N=32 respectively. The throughput for (N) MDN switches, under uniform traffic, is 98.23% and 96.12%, where N=4 and N=32 respectively. The average latency for (N, N-1) UDN switches, under uniform traffic, is 10.8 and 94.7 cycles/'Number of flits a packet is divided to', where N=4 and N=32 respectively. The average latency for (N) MDN UDN switches, under uniform traffic, is 14.2 and 126.7 cycles/'Number of flits a packet is divided to', where N=4 and N=32 respectively.

For the bitrate computations, we use both the frequency results of the RTL synthesis, as well as the RTL simulation results. The UDN offers 6.56 and 42.12 Gbytes/sec aggregate bandwidth for (4, 3) and (32, 31) UDN switches; which imply 1.64 and 1.32 Gbytes/sec bandwidth per port, respectively. On the other hand, the MDN offers 5.22 and 21.36 Gbytes/sec aggregate bandwidth for (4) and (32) switch sizes; which imply 1.31 and 0.67 Gbytes/sec bandwidth per port, respectively. High performance Ethernet cables offer 1.25 Gbytes/sec bandwidth; therefore, the UDN switch proves to be a competitive architecture to comply with the market products, whereas MDN's performance does not match this bandwidth, as the number of input/output ports is increased.

4.5 Conclusion

In this chapter, we proposed the hardware design and implementation of the two NOC based switch fabric architectures (UDN and MDN) for FPGA. We further improved the routing and scheduling algorithms, for the feasibility of their hardware design. The synthesis and RTL simulations are carried out over a range of switch sizes. The simulation results are also validated on FPGA, with packets generated by LFSR based pseudo-random traffic generators. The results show that UDN outperforms MDN in terms of throughput, whereas MDN offers greater performance-cost ratio. UDN's high performance makes it suitable for performance critical cases, whereas MDN is a better solution for cases that require cost efficiency. Both architectures offer scalability, flexibility and high performance, supporting our claims.

The results show that our architecture 1) performs as good as other buffering schemes/scheduling algorithms that theoretically achieve 100% throughput, 2) is at least as scalable as other architectures in terms of hardware cost, 3) is perfectly implementable, 4) implements load-balancing by nature and 5) introduces NOC concepts, which have originally been borrowed from computer networks, back into computer networks.

The UDN and MDN NOC switches would benefit from fault-tolerance, which can be implemented by exploiting FPGAs' property of dynamic reconfigurability. In case of a malfunctioning MR, other UDN and MDN MRs could be reconfigured to route the packets through an alternative path. This is left as future work, to improve the systems' performance, reliability and service capability further.

Chapter 5

A New Approach to Switch Fabrics based on Mini-Router Grids and Output Queueing

5.1 Introduction

Whether it is used in routers for computer communications or in the system-on-chip domain as a replacement of buses, switch fabric design is an important part of the effort carried out for packet-switched information transmission paradigm. The switch fabric is the hardware component that acts as the intermediate connection point of ingress ports to egress ports in the aforementioned and other similar information systems.

From the computer networks perspective, the routers require ever increasing need for scalable switch fabrics with high delivery ratios and low latencies. From the system-on-chip (SoC) perspective, using packet switched communication remedies the difficulty of interconnecting components with heterogeneous interfaces, thus abstracting data from the signaling interface, while offering high degree of parallelism and pipelining, which results in greater throughput.

A switch fabric consists of queueing memories and memory controllers for temporary storage of the packets, scheduling unit(s) to avoid contention, and other computational components that facilitate these tasks. Switch fabrics need to support high delivery ratios, in order not to become the bottleneck in the communication themselves, and therefore their design remains to be an open research problem.

In Chapter 4, we proposed a feasible hardware design and implementation of two Mini-Router Grid (MRG) based switch fabric architectures (UDN and MDN) for FPGA platforms. Our results point to a high performance switch fabric, that decouples switch size from the cost growth, at the expense of performance. We also demonstrated that UDN and MDN switch fabrics are comparable to crossbar switch fabrics in terms of performance.

In this chapter we improve upon our work in Chapter 4, by adding wraparound links between North and South Mini-Router (MR) rows, changing from Virtual-Output Queueing to Output Queueing, which results in a highly homogenized fabric with improved load balancing and uniformity. Furthermore we present an analytical model for the switch fabric that closely matches the simulation results, supporting our hypothesis.

The rest of the chapter is organized as follows: in Section 2, we outline the related literature. In Section 3, we describe a novel switch fabric architecture. In Section 4 we describe its analytical model and in Section 5, correlate the model to the simulation results. In Sections 6 and 7, we discuss a possible placement and comparison to other switches, respectively. Finally, Section 8 concludes the chapter.

5.2 Related Work

The switch fabric is one of the most important building blocks of communication related hardware, and various architectures have been proposed for both packet-switched computer networks and system-on-chip communication. Most of these architectures require a combination of queueing units, scheduling units and various other hardware components.

The main design challenges for a switch fabric include delivery ratio, bandwidth, latency, scheduling algorithms, interfacing, and routing algorithms as required in grid-based solutions. Several switch fabric architectures have been proposed, including the crossbar, shared-bus and shared-memory switches, which deal with these design challenges in various ways. The crossbar switch is the dominant architecture in today's high-performance switches, due to a number of reasons: crossbar switch is more scalable than the shared-bus and shared-memory; this is due to the limitations in bus transfer bandwidth and memory access bandwidth, respectively. Crossbar switch [60] provides point-to-point connections and non-blocking properties, as well as supporting multiple simultaneous transactions, increasing the bandwidth and speed of the router. However, their cost grows quadratically with the number of ports, since they require internal crosspoints and queues for every input/output port pair.

Recently, there have been a number of proposals on Mini-Router Grid (MRG) based switch fabric architectures for system-on-chip communication as a replacement of bus (a paradigm also referred to as network-on-chip), as well as for routers used in computer communications as a replacement of crossbars, including [83, 2, 79, 81].

Goossens et al [83] and Karadeniz et al [2], as described in the previous chapter, propose the UDN switch fabric, its functional model and the hardware design respectively, with the premise that they provide i) high throughput due to their pipelined nature, ii) low latencies, iii) ability to decouple switch size from cost

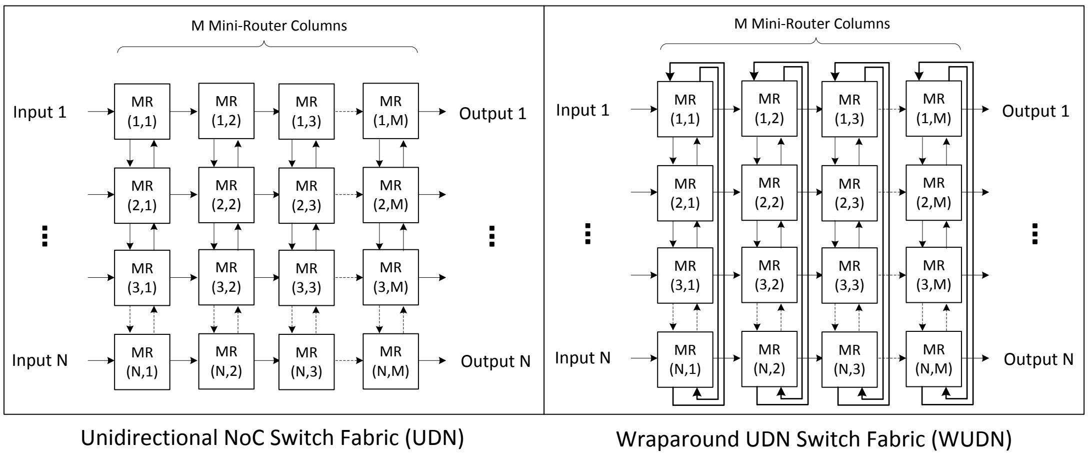


Figure 5.1: UDN / WUDN Architectures

growth, and iv) load balancing properties. UDN is in the form of an $N \times M$ MRG, where the input ports are placed on the West of the grid, and output ports are placed on the East, as shown in Figure 5.1. These proposals achieve sub-quadratic cost growth by decoupling the number of ports from the switch cost, at the expense of performance. Moreover, they introduce load-balancing without duplicating the packets, which in return improves the delivery ratio and latency. Goossens et al [83] describe the aforementioned architecture, presents some limited analytical modeling and functional level simulations. Karadeniz et al [2], as described in the previous chapter, propose a feasible hardware implementation and cost analysis; however, both lack an in-depth analytical model correlating the design parameters to each other, as well as to the performance metrics.

A number of architectures based on virtual output queues (VOQ) were proposed in order to remediate head-of-line (HOL) blocking of FIFO scheduling, including PIM, RRM, iSLIP [56][57][58], claiming a theoretical delivery ratio of 100%. However, VOQs do not scale optimally as the number of ports is increased, and therefore they are impractical. Another proposal, the load-balancing switch [59], claims greater scalability, by eliminating the need for a scheduler at the cost of duplicating the packets.

Output queueing (OQ) provides high performance, due to the fact the ingress packets are forwarded to the output queues without any delays, and as a result they achieve a theoretical delivery ratio of 100%. However, OQ suffers from the speedup problem, which means if there are N contending packets, they need to be written to the queue in the same time cycle requiring a frequency speedup of N, as described in [88]. Prabhakar and McKeown [89] propose a combined input and output queued (CIOQ) architecture, which bounds speedup by 4. We replace VOQs we have used in our switch fabric design (as described in Chapter 4) by OQs. Our MRG architecture bounds the speedup by 3 due to the use of 3 I/O MRs, which is the lowest speedup bound for any switch size when using only output queues, to our knowledge. Other approaches of bounding speedup include using input-output buffering, that can bound the speedup by blocking some of the HOL packets at the input queues, however this approach induces HOL blocking and increased delays; as a result, it suffers from reduced delivery ratios, as described in [90][91].

By adding wraparound links between North and South Mini-Router (MR) rows, we enable using the same exact 3 I/O port MR component, increasing the implementability and homogeneity in our design. Furthermore, wraparound links between North and South MR rows improve the average and maximum latency by 50%, in addition to improved load balancing and uniformity.

In this work, we present the first in-depth analytical model of switch fabric architectures based on OQ MRG and validate our model with register-transfer level (RTL) simulations in the synthesizable subset of SystemC by showing that the analytical and simulation results have close correlation over a range of design parameters.

5.3 MRG Based Switch Fabric Architecture

5.3.1 Architecture Design Parameters

In this work, we propose an output-queued, wraparound version of UDN (WUDN), where the North ports of the North-most row are connected to South ports of the South-most row, reducing the maximum vertical packet transmission from N hops to $N/2$ hops, and increase the load balancing and traffic uniformity further. The comparison of the UDN and WUDN architectures are presented in Figure 5.1.

The size of the WUDN switch is defined by the 2-tuple (N, M), where N denotes the number of input-output (I/O) ports. M denotes the number of MR columns. The number of MRs in the WUDN switch is equal to $N \times M$, and they all have 3 I/O ports. Hardware design constraints and the modulo based routing algorithm (Subsection 5.3.4) inflict some restrictions to the (N, M) values:

- $N \in \mathbb{N}$, and $N \geq 2$, trivial restriction that ensures the number of ports is greater than or equal to 2.
- $M \leq N$ and $M = 2^{\mathrm{m}}$, where $m \in \mathbb{N}_0$: The restrictions on M are inflicted by the routing algorithm involving Modulo M operations (See Section 5.3.4). Because Modulo M operation requires division in case $M \neq 2^m$, but it is a simple bit-selection operation in case $M = 2^m$, we can avoid the extra cycles caused by the division operation by applying this restriction.
- $M \mid N$: This ensures that the load is distributed uniformly on M columns, in order to avoid congestion due to structural non-uniformity.

The WUDN switch architecture has the wormhole switching mode, buffered flow control implemented by output queueing scheme, Modulo XY Algorithm allows deterministic-uniform routing within the MRG, and incremental routing path decision. For simplicity, we use fixed-length cells.

The data delivery through the MRG is pipelined due to the point-to-point nature of MRs. This restricts the critical path to the control logic in a single MR and improves the scalability and throughput. WUDN is a deadlock-free since it is unidirectional.

By introducing output queuing (OQ) scheme to MRGs, we show that the speedup problem can be bounded by 3 and therefore be feasibly implemented in

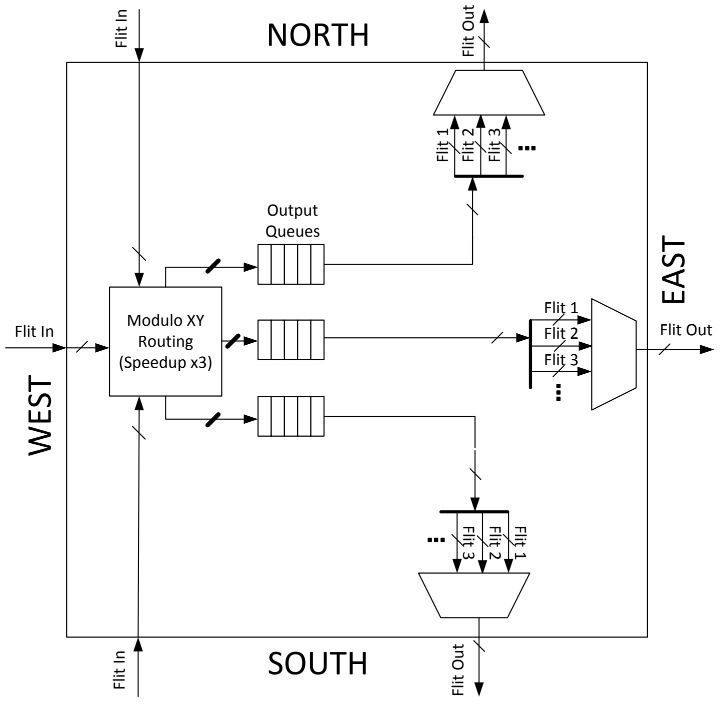


Figure 5.2: 3 I/O Port WUDN MR, Top-Level Block Diagram

hardware. Our architecture offers a theoretical throughput of 100%, smaller queue sizes, and QoS guarantees, while not suffering from unscalability due to speedup.

The cost of WUDN is a function of N and M, and it is directly proportional to the number of MRs. As N is increased, more MR columns would be required to support the traffic; however the architecture permits M to be decoupled from N, trading performance for lower cost. For constant performance, M is a function of N.

5.3.2 WUDN 3 I/O Port Mini-Router

The block diagram for a 3 I/O Port WUDN MR is given in Figure 5.2. There are 3 input ports, West, North and South; and there are 3 output ports, East, North and South. The output queues and memory controllers are placed at each output port, whereas the next-hop logic and demultiplexers are placed at the input ports. Packets are transmitted in equally size flits in between MRs.

5.3.3 Memory Organization for Output Queues

MRs require a maximum speedup of 3, which is the number of MR I/O ports. In order to overcome the speedup problem that affects only the write operations, we use a dual clock-domain memory controller and dual-port queues, as shown in Figure 5.4.

First, the next-hop port for the ingress packets are computed, which generates the signal to demultiplex the packet into the appropriate output queue controller; then, within the memory controller for write operation, contending packets are multiplexed and serially written to the queueing memory that is on the higher frequency clock domain. We use a Round Robin (RR) based scheduling algorithm to multiplex and serialize the packets into the output queues. The priorities of the three input ports are shifted in a circular fashion, and the present contending packets are serialized according to these priorities. The output queue control is implemented in a circular fashion.

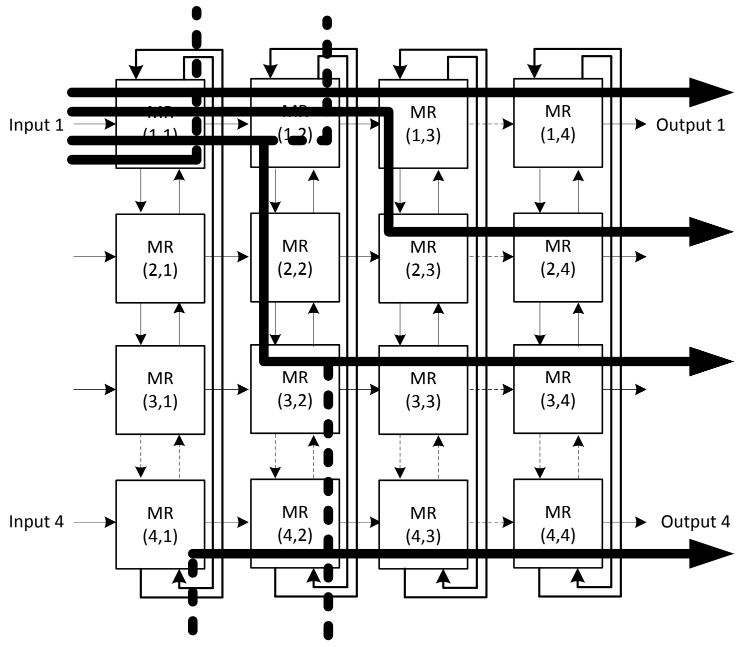


Figure 5.3: Modulo XY Routing for WUDN

5.3.4 Modulo XY Routing in the Mini-Router Grid

We present Modulo XY Routing algorithm for WUDN (Algorithm 2), a special case of Manhattan routing, which makes a balanced distribution of the vertical traffic over the MR columns by using the modulo operation. The algorithm is applied to each packet at each MR, thus implementing an incremental routing scheme. Figure 5.3 exemplifies 4 packet transmissions from I_1 to O_{1-4}. The packets are routed on different MR columns per I/O port pairs, distributing the load on the switch. A turning point MR (MR_{TP}) denotes an MR in which a horizontal flow is directed into a vertical one ($MR_{TP,H\to V}$), or vice versa ($MR_{TP,V\to H}$). $TP_{H\to V}$ occurs when the algorithm chooses the column for vertical transmission. Vertical

Algorithm 2 Modulo XY Algorithm for WUDN

```
1: Let inputPort denote the input port of the MR.
2: Let nexthop denote the output port of the MR & return value of the algorithm.
3: Let $IDX$ denote the row index of the current MR.
4: Let $IDY$ denote the column index of the current MR.
5: Let source denote the input port of the switch fabric the packet has entered the
   switch.
6: Let destination denote the output port of the switch fabric the packet is destined
   to.
7: Let $N$ denote the number of switch fabric input ports.
8: Let $M$ denote the number of MR columns.
9: Let $RR$ denote one bit round robin alternator variable.
10: if inputPort is NORTH then
11:   if $IDX$ is destination then
12:     nexthop ← $EAST$              # Turning point MR
13:   else
14:     nexthop ← $SOUTH$             # Keep moving vertically
15:   end if
```

Algorithm 2 Modulo XY Algorithm for WUDN (continued)

```
16: else if $inputPort$ is SOUTH then
17:     if $IDX$ is $destination$ then
18:         $nexthop \leftarrow EAST$                  # Turning point MR
19:     else
20:         $nexthop \leftarrow NORTH$                 # Keep moving vertically
21:     end if
22: else if $inputPort$ is WEST then
23:     if $IDX$ is $destination$ then
24:         $nexthop \leftarrow EAST$                  # Keep moving horizontally
25:     else if $(destination + source)\%M == IDY$ then
26:         if $abs(IDX - destination) > (N/2)$ then
27:             if $(IDX - destination) < 0$ then
28:                 $nexthop \leftarrow NORTH$         # Turning point MR
29:             else
30:                 $nexthop \leftarrow SOUTH$         # Turning point MR
31:             end if
```

Algorithm 2 Modulo XY Algorithm for WUDN (continued)

```
32:       else if $abs(IDX - destination) == (N/2)$ then
33:          if $RR == 0$ then
34:             $nexthop \leftarrow NORTH$          # Turning point MR
35:          else
36:             $nexthop \leftarrow SOUTH$          # Turning point MR
37:          end if
38:          $RR \leftarrow !RR$                     # Alternate $RR$
39:       else if $abs(IDX - destination) < (N/2)$ then
40:          if $(IDX - destination) < 0$ then
41:             $nexthop \leftarrow SOUTH$          # Turning point MR
42:          else
43:             $nexthop \leftarrow NORTH$          # Turning point MR
44:          end if
45:       end if
46:    else
47:       $nexthop \leftarrow EAST$                 # Keep moving horizontally
48:    end if
49: end if
```

transmission is either South-bound or North-bound according to the vertical hop count to the destination row; if the distances are equal, one is intermittently chosen. $TP_{V \to H}$ occurs when the packet arrives to the destination row. If the packet's source and destination are the same (e.g. I_1 to O_1), a single horizontal tranmission will be made through a single MR row, and no TPs will be elected by the algorithm.

In the Algorithm 2, there can be observed some computational operations, including modulo (Line 25), absolute value (Line 26, Line 32, Line 39), sign function (Line 27, Line 40), subtraction (Line 26, Line 32, Line 39), and comparison (Line 26, Line 32, Line 39). With the restrictions we have defined in Subsection 5.3.1, the modulo operation is reduced to a simple bit shift, and the remaining operations are interrelated such that they can be computed in fewer number of operations, using 2's complement algebra.

5.4 Analytical Model

In this section, we present a mathematical analysis of WUDN switch fabric that provides useful insights into evaluation of the performance and scalability of the proposed architecture.

In this section, we use notation R_{xy} to denote the MR at coordinate (x, y) on the grid. I_s and O_t are used when referring to input port s and output port t, respectively. Further, $s \rightsquigarrow t$ is to denote the path between I_s and O_t found by the modulo algorithm and $|s - t|$ denotes the lattice distance between I_s and O_t.

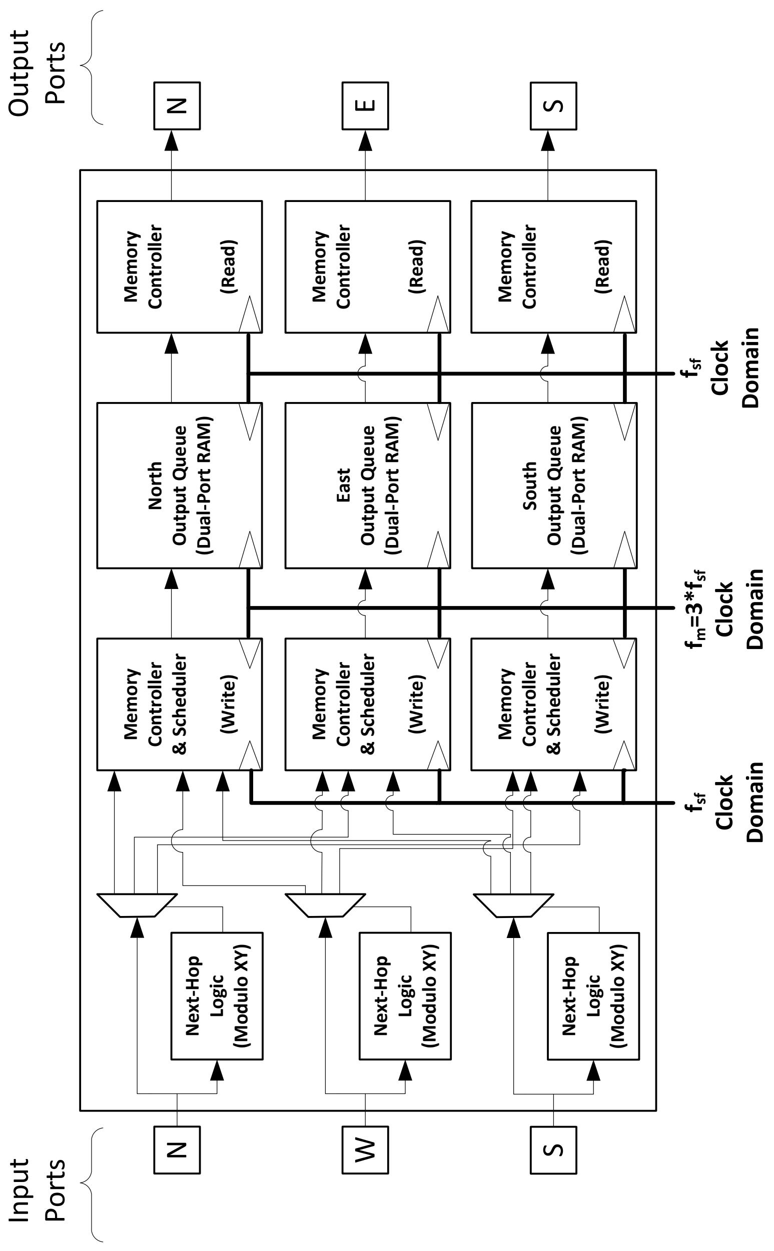

Output Ports
N
E
S
Memory Controller (Read)
Memory Controller (Read)
Memory Controller (Read)
North Output Queue (Dual-Port RAM)
East Output Queue (Dual-Port RAM)
South Output Queue (Dual-Port RAM)
Memory Controller & Scheduler (Write)
Memory Controller & Scheduler (Write)
Memory Controller & Scheduler (Write)
f_{sf} Clock Domain
f_m=3*f_{sf} Clock Domain
f_{sf} Clock Domain
Next-Hop Logic (Modulo XY)
Next-Hop Logic (Modulo XY)
Next-Hop Logic (Modulo XY)
Input Ports
N
W
S

Figure 5.4: Memory Organization for Output Queues

5.4.1 Analysis of the Modulo Routing Algorithm

We begin with characterization of the routing algorithm. Modulo routes exhibit some useful properties that make the analysis of the system easier. In the sequel, we review some of these properties.

Property 1. *For any given source-destination pair, modulo routing finds the shortest path between the two end.*

Proof: The modulo routing is a greedy forwarding scheme in a taxicab geometry in which at each hop, the lattice distance between the packet and destination is decremented by one. This is, indeed, the maximum the grid structure can offer and thus, guarantees that the routing is performed over the shortest path. Such a path, however, is not necessarily unique. ■

An immediate deduction from Property 1 is that the length of the modulo route between an arbitrary choice of I_s and O_t is indeed $|s-t|$, that is the lattice distance between source and destination. On a $N \times M$ grid and with a uniform and independent selection of source and destination, every path is comprised of an exact number of M horizontal and an average of $N/4$ vertical transmissions. The latter can be inferred noting the fact that $|s-t|_v$ takes on values between 0 and $N/2$ with equal probability. Therefore, the average path length is $M + N/4$ hops. We shall use this fact later on when proving Property 3.

Property 2. *Modulo routes are loop-free.*

Proof: The previous property also implies that modulo routes cannot contain cycles. ■

Property 3. *Under a uniform and independent selection of source-destination pairs, modulo routing maintains a uniform distribution of traffic across all MRs throughout the grid.*

Proof: We quantify the fraction of transmissions contributed by each MR in the grid. In a general case and on a $N \times M$ grid, the modulo algorithm can generate a total of N^2 deterministic paths connecting every source to every destination port. Assuming that source-destination pairs are chosen uniformly and independently, then the choice of every path is also uniform; that is, all paths are equally utilized.

Suppose that every source deterministically generates a packet for every destination. From Property 1, we recall that a modulo path comprises an average of $M + N/4$ hops. Thus, the total number of transmissions, T, to be made over the entire grid is given by

$$T = N^2\Big(M + \frac{N}{4}\Big). \tag{5.1}$$

Consider the x^{th} row, denoted by R_{x*}, on the grid. We enumerate the total number of horizontal transmissions over R_{x*}. Note that such transmissions are either from the traffic originated from I_x or destined at O_x. We consider each case separately.

1. Traffic originated from I_x (but not destined at O_x): There are N packets coming out of I_x, each of which takes between 0 to $M - 1$ horizontal hops on

R_{x*} with equal probability before getting to the turning point. That makes a total of $N(M-1)/2$ horizontal transmissions on R_{x*}.

2. Traffic destined at O_x: There are N such packets each taking between 1 to M horizontal hops on R_{x*} with equal probability. This adds up to a total of $N(M+1)/2$ horizontal transmissions on R_{x*}.

The total number of horizontal transmissions across R_{x*} is thus given by

$$T_h(x) = NM\,. \tag{5.2}$$

In order to enumerate the vertical transmissions across row R_{x*}, we perform a transformation on the grid. Let us assume that the grid is horizontally contracted such that all MRs in every row are consolidated into a single MR. This transforms the original $N \times M$ grid into a $N \times 1$ layout. This transformation is safe, in the sense that it does not affect the number of vertical transmissions. In fact, all vertical transmissions that were supposed to be carried out across R_{x*} will now take place on $R_x \equiv R_{x1}$. Therefore, the number of vertical transmissions across row R_{x*} is now equal to the number of individual paths intersecting R_x.

We take advantage of the symmetry of vertical connections on the grid to enumerate all such paths. To that end, note that R_x is used on every $s \rightsquigarrow t$ where $|s-x| + |x-t| = |s-t|$. By Property 1, $|s-t| \leq 1 + N/2$. However, where $|s-t| = 1 + N/2$, there are exactly two such paths between s and t, only one of which uses R_x if $s \neq x$. In that case, the grid uses both paths alternatively to maintain a fair balance of traffic over all MRs. In our analysis, we count such cases as *half-paths.* It is easy to observe that there are exactly N half-paths that cross through any given R_x. Offsetting all such paths that are double-counted, we obtain the following equation for the total number of paths intersecting R_{x*}

$$T_v(x) = 2\sum_{i=0}^{\frac{N}{2}-1}\left(\frac{N}{2} - i\right) - \frac{N}{2} = \frac{N^2}{4}\,, \tag{5.3}$$

which is indeed equivalent to the number of vertical transmissions carried out across R_{x*}.

From Equations (5.1), (5.2) and (5.3), we realize that R_{x*} handles $1/N$ of the total transmissions. For uniformly and independently chosen I_s and O_t, choice of the turning point column (*i.e.*, $(I_s + O_t)\%M$) is also uniform (see [92] for the actual theorem and a rigorous proof). This implies that every column also carries equal share of the traffic. Therefore, the total number of horizontal transmissions, $\bar{T}_h$, and vertical transmissions, $\bar{T}_v$, contributed by each MR is

$$\bar{T}_h = N\,, \qquad \bar{T}_v = \frac{N^2}{4M}\,. \tag{5.4}$$

Overall, each MR transmits a total of $N + N^2/4M$ packets, which is in fact $1/MN$ of the total traffic given by Equation (5.1). ■

In the following, we use the foregoing discussion to obtain a detailed characterization of the dynamics of the proposed architecture.

5.4.2 Detailed Characterization of Traffic Distribution for MRs

Inside an individual MR, let us denote with P_E, P_N and P_S the probabilities of sending a packet on the east, north and south ports, respectively. Using

Property 3, we note that these probabilities are identical for all MRs throughout the grid. From Equation (5.4) and the facts that

$$\frac{\bar{T}_h}{\bar{T}_v} = \frac{P_E}{P_N + P_S}, P_N = P_S \tag{5.5}$$

we readily obtain that

$$P_E = \frac{4}{\beta + 4}, \quad P_N = P_S = \frac{1}{2}\Big(\frac{\beta}{\beta + 4}\Big), \tag{5.6}$$

where $\beta := N/M$ is the grid's aspect ratio.

As seen, these probabilities depend on number of inputs N and number of layers M only through β. In fact, grids with similar aspect ratio exhibit similar behavior in terms of traffic distribution. We call such grids *isomorphic* and shall further investigate on their properties later in Section 5.5.

A second important observation from Equation (5.6) is that when $\beta = 8$, we have $P_E = P_N = P_S$. In that case, all three queues are equally utilized and there appears a balanced distribution of traffic across all directions on the grid.

5.4.3 Incoming Traffic Rate and Stability

For a switch with infinite capacity queues, using Equation (5.6), we find the maximum rate of incoming traffic (denoted by λ) under which the switch would be stable. Let us denote with λ^* the total input rate to an arbitrary edge MR, R_{x1}.

In the steady state, the output rate is equal to the input rate. Thus,

$$\begin{aligned}\lambda^* &= \lambda + 2 \times \frac{1}{2}\Big(\frac{\beta}{\beta+4}\Big)\lambda^* \\ &= \lambda\left(1+\frac{\beta}{4}\right). \end{aligned} \tag{5.7}$$

The stability condition requires to maintain a utilization factor $\rho := \lambda^*/\mu$ of less than one. Therefore, the sufficient condition for stability is to maintain

$$\lambda < \frac{4\,\mu}{\beta+4} = \lambda_{max}\,. \tag{5.8}$$

The service rate, μ, is deterministic and is equal to 3, since MR is able to transmit 3 packets every round. Thus, the maximum tolerable input rate is 2.4 when $\beta = 1$ (*i.e.*, perfect square grids). The switch would only be able to handle lower rates as β increases. Of course, with finite capacity queues, irrespective of the queue size, the switch would always remain stable; yet certain fraction of the incoming traffic would be blocked and the system would inevitably be subject to packet losses should the system is loaded with a rate higher than λ_{max}.

5.4.4 State Distribution and Optimal Queue Size

The incoming traffic being a memory-less process, a WUDN switch can effectively be modeled as a Jackson network of $M^X/D^Y/1/N$ queues[1]. Due to the architectural symmetry and traffic uniformity (Property 3), most of the analysis of

[1]$M^X/D^Y/1/N$ is Kendall's notation describing memory-less batch arrival and deterministic batch service distributions, single server, with queue size of $N-1$. Here M and N are part of a standard notation and should not be confused with what we have used for specifying grid dimensions.

such a complex system can be well reduced to the analysis of a single $M/D/1/N$ queue.

Even in finite capacity systems, a delivery ratio of close to 1 is often desired. In order to obtain insights into finding the optimal queue size, we concentrate on the analysis of the more generic case of $M/D/1$ with infinite capacity and characterize the state distribution. The state distribution, $P(i)$, for $M/D/1$ queues can be computed according to Fry's derivation [93] when $n = 1$, as follows:

$$P(i) = (1-\rho)\sum_{j=0}^{i} e^{\rho j}\left(\frac{(-\rho j)^{i-j}}{(i-j)!} - \frac{(-\rho j)^{i-j-1}}{(i-j-1)!}\right). \tag{5.9}$$

Using that, we readily compute the cumulative state distribution, $S(i)$, as follows.

$$S(i) = \sum_{j=0}^{i} P(j) = (1-\rho)\sum_{j=0}^{i} \frac{\big(\rho(j-i)\big)^{j}}{j!} \cdot e^{\rho(i-j)}. \tag{5.10}$$

In Figure 5.7, the dashed lines illustrate $S(i)$, the probability of having at most i packets in each MR. In the actual $M/D/1/N$ architecture, there is a chance of packet loss due to blocking at the queues. By numerical evaluation of $S(i)$ for a given utilization factor, we are able to pinpoint the optimal queue size by looking at the k^{th}-percentile on the cumulative distribution, when a steady-state delivery ratio of $k\%$ is expected.

5.5 The Model & Simulations

5.5.1 RTL & Functional Models

The switch fabric model and simulations are implemented in SystemC [94]. SystemC is an event-driven simulation library for C++ that imitates Hardware Description Languages (HDL) by simulating concurrent events. SystemC provides a synthesizable subset to approach RTL design languages like Verilog [95] or VHDL [96] further.

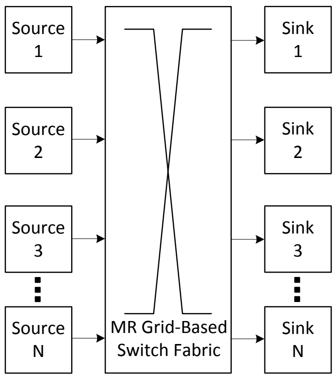


Figure 5.5: Concurrent Source & Sink Modules, connected to the Switch Fabric

We implement most of our switch fabric components, including *queue memory controllers*, *Modulo XY Routing unit*, and other *forwarding related circuitry* in synthesizable SystemC, whereas *clock generation,Source* and *Sink* modules are functional models. The Source module is instantiated N times, once per ingress port, and it generates random traffic according to a Poisson distribution. Similarly, the Sink module is instantiated N times, once per egress port, and it is responsible for performance reporting. In Figure 5.5 we present a schematic view of the simulation components.

5.5.2 Simulation Parameters & Performance Metrics

The input parameters to our simulation experiments are switch size N, number of MR layers M, maximum queue size QS, incoming traffic rate λ, and the simulation runtime duration T.

The performance metrics of interest are as follows:

- Delivery Ratio (DR): Ratio of the packets successfully delivered to the destination ports. We analyze this metric under variable input rate and queue size;

- Local traffic distribution: Fraction of packets being sent over each output port within MRs. For this measure, we only evaluate P_E, probability of transmitting over east port. P_S and P_N provide no further insight and can be computed accordingly;

- Global distribution of traffic across the entire MR grid.

5.5.3 Delivery Ratio under Variable Traffic Rates

Our first experiment examines the impact of increasing traffic rate on the overall delivery ratio. We simulate the performance of a switch with 64 input ports, where the number of layers M are $\{1, 8, 64\}$. For this analysis, we use

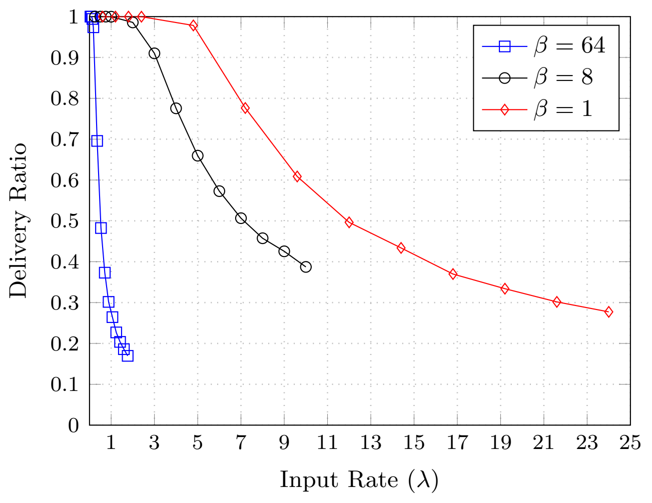


Figure 5.6: Delivery Ratio Vs. Input Rate

Equation (5.10) to anticipate and employ a queue capacity for which a delivery ratio of close to 1 is achieved when the system is highly loaded. Note that the optimal queue size also depends on the grid's aspect ratio as seen in Section 5.4. Therefore, the queue sizes used on each of the three individual grid layouts are different.

Figure 5.6 shows how the delivery ratio decays with an increasing load of traffic over the x-axis. λ_{max} for $\beta = \{1, 8, 64\}$ are $\{2.4, 1, 0.176\}$, respectively. As clearly seen, a delivery ratio of 1 is obtained in all cases when $\lambda < \lambda_{max}$. Recall from Section 5.4 that this corresponds to a utilization factor (ρ) of 1. The utilization factor, in fact, determines the fraction of time that the system is busy (*i.e.*, non-empty MRs). Having $\rho \geq 1$ results in instability (infinite queue size and waiting time) in infinite capacity systems and packet drops in finite capacity queues. This behavior is accurately captured both by model and simulations. Even though reporting results for $\rho \geq 1$ might seem futile, it's actually useful to take into account how the system would react to bursty traffic or unexpected high loads of short durations.

5.5.4 Delivery Ratio as a Function of Queue Size

We study the impact of changing the queue size on the overall delivery ratio. In every experiment, the queue size is fixed and identical for all MRs throughout the grid. Also, the queue sizes for north, east and south ports are the same and are equal to one-third of the total MR capacity. This, in fact, justifies the reason why we have used queue sizes that are multiples of 3.

In order to conduct a fair experiment, we have used a 64×8 grid to maintain an aspect ratio (β) of 8 for which all queues are equally utilized.

Figure 5.7 demonstrates how increasing the system capacity enhances overall delivery ratio. The solid curves show the simulation results, while dotted lines correspond to theoretical values obtained by Equation (5.10). As seen, the behavior of the system is accurately predicted by the model.

5.5.5 Impact of Grid's Layout on Local Traffic Distribution

In this subsection, we verify the property captured by Equation (5.6) that the within-MR distribution of traffic across all ports depends on N and M only through β. More precisely, grids with similar aspect ratios (isomorphic grids) result in similar traffic distributions across their output ports.

For this experiment, we generate multiple grid instances of different sizes that have identical aspect ratios. For every layout, we calculate the fraction of

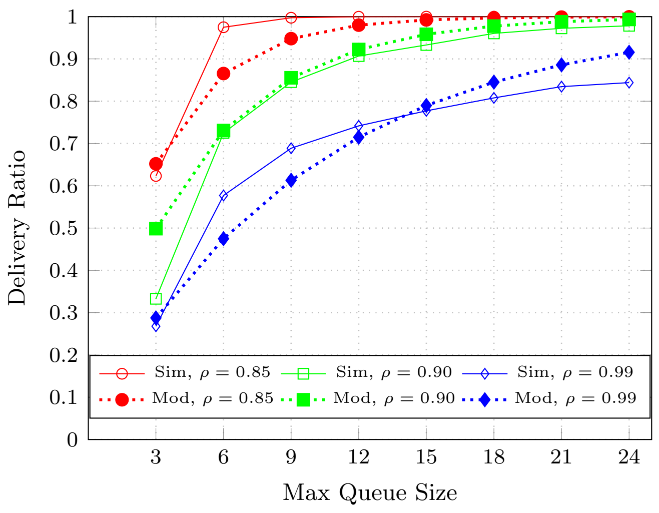


Figure 5.7: Delivery Ratio vs. Max Queue Size

packets transmitted through the east port (*i.e.*, P_E). Figure 5.8 demonstrates that the simulation results (solid line) accurately support our theoretical analyses (dashed line). The simulation results, in fact, are the average values for different isomorphic grids. The standard deviations, shown as error bars, are so small that are barely visible on the graph.

According to Figure 5.8, the more the aspect ratio is, the lighter becomes the eastbound traffic. This behavior is intuitive noting the fact that for a fixed M (say, $M = 1$), a larger β corresponds to having a larger number of sources (N). In that case, higher proportion of traffic should be carried through a fixed number of columns (M), which results in a lower P_E. This also points out to the fact that isomorphic grids provide identical distributions of traffic across output ports.

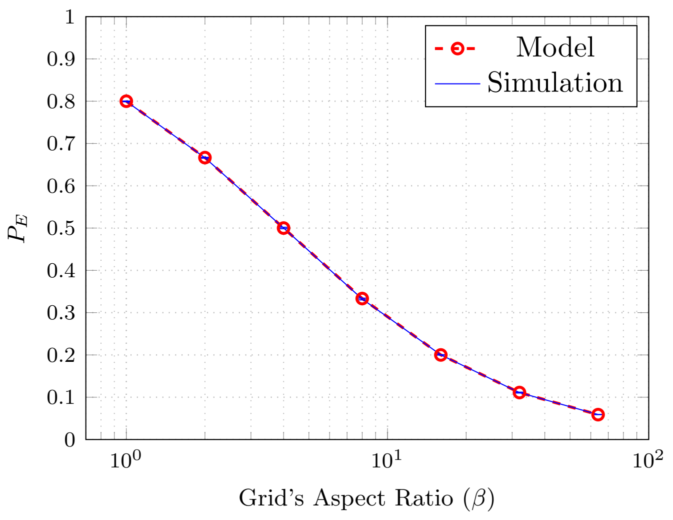


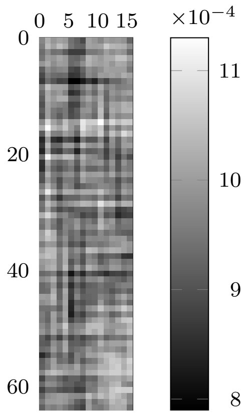
Figure 5.8: Probability of sending on east vs. grid's aspect ratio

5.5.6 Global Distribution of Traffic Across the Grid

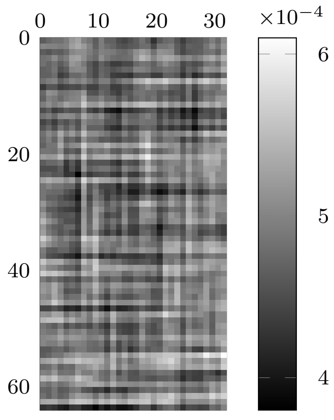
We provide experimental results to demonstrate the uniformity of the traffic across the entire grid. This is, indeed, an attribute of the XY modulo routing that we discussed earlier as Property 3 of the routing algorithm.

We run extensive simulations on three different grid layouts and quantify the portion of traffic handled by each MR. We illustrate the traffic through individual MRs in the grid using heatmaps in Figure 5.9. Lighter colors represent higher load intensity.

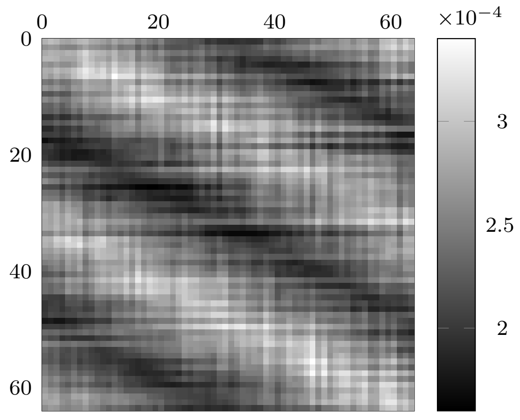
As illustrated in Figure 5.9, the load range is very tight all over the grid, and the varying colors on the heatmap only point to very small variances. Therefore, in all scenarios, the traffic distribution is very balanced and the grid allows fair routing. A slight pattern can be observed in Figure 5.9c, but the variance between the limiting values within the colormap is so low that the pattern is virtually negligible.



(a) A 64 × 16 Grid



(b) A 64 × 32 Grid



(c) A 64 × 64 Grid

Figure 5.9: Heatmaps illustrating uniform distribution of the the traffic across the grid

5.6 Placement Considerations

The block diagram of WUDN Figure 5.1 might incorrectly imply that the wraparound links connecting North ports of the North-most MR row to South ports of the South-most MR row are implemented as long buses, which in return would inflict asymmetric delays between the MRs, requiring the clocking to be slowed down. Moreover, having a long bus proportional to N would render the switch fabric unscalable due to the propagation delay of O(N). On the contrary, WUDN can be placed on the chip in such a way that the wraparound links are not longer than the other vertical or horizontal links, preventing such issues from arising.

The idea behind WUDN placement is "folding" the columns in the mesh such that each column has the "U" shape, as opposed to "I". This is shown in Figure 5.10, where wraparound links are not a function of switch size N anymore. Since the topology and the architecture remain the same, the descriptions and analysis in the previous sections hold.

Please note that, with this approach the area of the switch fabric remains the same; the height is halved and the width is doubled. After the placement, WUDN has longer horizontal links that connect every other column, but the total link length throughout the switch fabric remains the same. In addition, to our advantage, the length of any link between the MRs is now decoupled from the switch size N (as well as, number of layers M), to yield a scalable and low delay architecture.

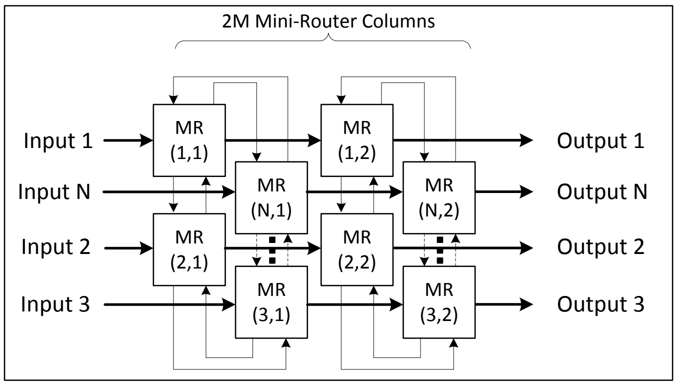


Figure 5.10: Placement of WUDN

5.7 Comparison & Discussion

Crossbar, in terms of many performance aspects, resemble MRG based UDN and WUDN, supporting point-to-point connections and non-blocking properties that allows multiple simultaneous transactions. However, their cost grows quadratically with the number of ports, since they require internal crosspoints and queueing memories for every input/output port pair. MRG based switches improve upon crossbar switches for their ability to lower switch cost growth from $O(N^2)$ at the expense of performance. This is due to their ability to add or remove MR columns (or layers) as necessary.

UDN inflicts increased congestion in the central regions, since there is more traffic towards the center due to its lack of uniformity. UDN's non-uniformity is captured in Figure 5.11, emphasizing the need for distributing the load better. WUDN improves upon the uniformity by adding the wraparound links that balances the traffic over all of the rows, resulting in less congestion, as presented in Figure 5.9.

Moreover, the wraparound links decrease the vertical latency by a factor of 2.

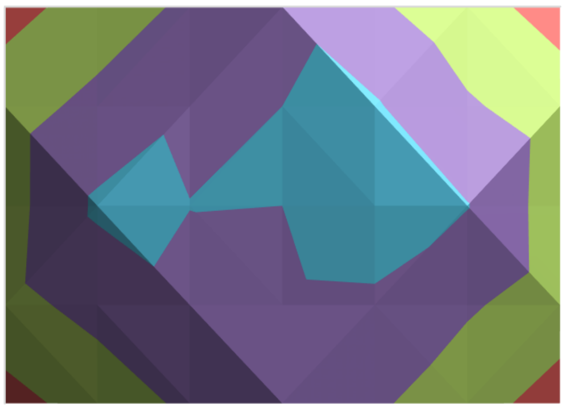


Figure 5.11: UDN, Non-uniform Traffic

It should be mentioned that WUDN is more cost efficient for larger switch sizes. For example, in the case of a switch with 4 ports, it would be more efficient to remedy a speedup of 4 instead of implementing it using 16 3-port mini-routers, which is clearly inefficient. However, as the switch size increases, the speedup cannot be remedied by a single switch, and WUDN becomes a viable option.

Another important point that the reader should note is that we chose to carry out in-depth analysis and simulations assuming uniform all-to-all traffic. [83] reports good performance for unbalanced and bursty traffic patterns, and our proposal with wraparound links and OQ would only improve those results further.

WUDN has queuing memory at every output port for each MR; however, observing Figure 5.7 (where values are provided in terms of the total queueing memory size per MR) it's possible to deduce that the WUDN requires very small queue sizes, and thus that this does not in fact issue a threat on the cost.

5.8 Conclusion

In this chapter, we have proposed a novel switch fabric based on OQ MRGs, which offers promising delivery ratios, small queue sizes, and low latencies. Moreover, we showed that the speedup problem introduced by OQ can be bounded by 3 by using MRGs. We have presented the first in-depth analytical model of switch fabric architectures based on OQ MRs, where we correlate design parameters to several performance metrics, such as the maximum supportable input rate λ_{max}, the optimal queue size, and local and global distributions of traffic. We have supported and validated our model with RTL simulations and showed that the simulation results closely match the analytical model. Finally, we have shown that WUDN does not inflict additional delays over UDN due to the wraparound links, by describing a feasible placement on-chip. Power profiling and comparison to other architectures in terms of power consumption remain as future work. Analytical modeling under unbalanced traffic patterns would also be beneficial to this work.

Chapter 6

Conclusion

In this thesis, we described an embedded switch architecture for label switched networks with an emphasis on hardware-software co-design paradigm, resulting in a scalable, flexible and high performance switch. We have proposed improvements over both wireless and wired network scenarios.

For wireless networks we have proposed a multi-root interval routing algorithm (Chapter 2) that yields routing tables more than an order of magnitude smaller than explicit routing algorithms. Moreover, by using multiple roots, we achieve a stretch of $\mathcal{O}(1)$ and comparable latencies and delivery ratios to shortest path based routing algorithms like OLSR. Small routing tables enable us to use a centralized controller that can compute all the routes and download them to individual nodes, thus eliminating a major part of the control plane overhead. Our embedded switch (Chapter 3), implemented on FPGA, retrieves the flow table information that comes from the controller at the software routing layer; it writes the flow state to memory that can be read by hardware logic, so that data forwarding can be carried out at Layer 2. SHADES significantly improves point-to-point latencies more than 1ms per link.

In wired network switches, the switch fabric is the key components that queues the incoming packets and schedules them for delivery. The design effort includes the interconnect topology, queueing memory organization and scheduling logic design and a successful design should yield high delivery ratios, low latencies and load balancing capabilities. In Chapter 4 and Chapter 5 we described our Mini-Router Grid (MRG) based switch fabric architectures, which successfully delivers these metrics. Our proposal is to replace the current state of the art crossbar architectures that require both input queuing and crosspoint queuing, inducing high costs for the performance. Using both input and crosspoint queues are shown to require large size memories. Our proposal uses Network-on-Chip (NOC) approach, in which the switch fabric resembles to a grid network that consists of tiny 3 input/output port Mini-Routers (MR). The connection links between MRs are very small, and the control signals that implement back-pressure are pipelined automatically due to the nature of the point-to-point communication between MRs. In this way, the critical path is very short, yielding high frequency and throughput. The switch fabric grid is composed of N rows, one for each input/output port, and M columns of MRs. The number of columns can be decreased at the expense of performance, and as a result the switch size is decoupled from the cost growth, which constitutes the main advantage of UDN and MDN over crossbars.

The research we have carried out has been published in the following conference proceedings:

[1] T. Karadeniz, L. Mhamdi, K. Goossens, and J. J. Garcia-Luna-Aceves, "Hardware design and implementation of a Network-on-Chip based load balancing switch fabric," in 2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig), 2012, pp. 1 7.

[2] T. Karadeniz, A. N. Masilamani, and J. J. Garcia-Luna-Aceves, "Neighbor Discovery Using Galois Fields and its Hardware Implementation," presented at the

Milcom 2013, San Diego, CA, 2013.

[3] T. Karadeniz, A. Dabirmoghaddam, Y. Goren, and J. J. Garcia-Luna-Aceves, “A New Approach to Switch Fabrics based on Mini-Router Grids and Output Queueing,” presented at the International Conference on Computing, Networking and Communications, Communications QoS Workshop (ICNC, CNC 2015), Anaheim, California, USA., 2015.

In addition, the following papers are on submission or yet to be submitted to various international conferences:

[4] M. Mosco, A. N. Masilamani, T. Karadeniz, and J. J. Garcia-Luna-Aceves, “Towards Software-Defined MANETs”

[5] T. Karadeniz, M. M. Barijough, and J. J. Garcia-Luna-Aceves, “Multi-Root Interval Routing (MINT)”

Table 6.1 summarizes our contributions, describing the mapping of the logical components in the system to the physical components. Moreover, it shows whether each component was implemented in software, hardware or as a co-design. Also, the rationale behind implementing each component is presented in the last column.

Table 6.1: Contributions, Summary

Logical Component	**Physical Component**	**HW/SW**	**Reason**
Interval Routing	MINT Routing Layer	SW	Scalability
SDN (Flow Based Routing)	Flow Tables	CO	Abstraction/Flexibility
Label Switching	MINT/Flow Tables	CO	Abstraction/Flexibility
Top-Level System	SHADES on FPGA	CO	Integration/Performance
Switch Fabric/Forwarding	Switch Fabric, Q Memory	HW	Scalability/Performance

My doctoral journey has been a long winding road [97], and it required distilling the right steps from all the wrong ones I took. I believe I have achieved the research tasks described in my research proposal, presented on 10 September, 2013 as a part of the advancement to candidacy.

Santa Cruz, CA, June 2015 □

Bibliography

[1] Turhan Karadeniz, Maziar Mirzazad Barijough, and J. J. Garcia-Luna-Aceves. Multi-Root Interval Routing (MINT). 2015. To be submitted.

[2] Turhan Karadeniz, Lotfi Mhamdi, Kees Goossens, and J.J. Garcia-Luna-Aceves. Hardware design and implementation of a Network-on-Chip based load balancing switch fabric. In *2012 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1 –7, December 2012.

[3] Turhan Karadeniz, Ali Dabirmoghaddam, Yusuf Goren, and J.J. Garcia-Luna-Aceves. A New Approach to Switch Fabrics based on Mini-Router Grids and Output Queueing. Anaheim, California, USA., February 2015.

[4] J. Van Leeuwen and R. B. Tan. Interval Routing. *The Computer Journal*, 30(4):298 – 307, January 1987.

[5] The OpenFlow Switch Consortium. http://openflowswitch.org/.

[6] N. Santoro and R. Khatib. Labelling and Implicit Routing in Networks. *The Computer Journal*, 28(1):5 –8, January 1985.

[7] J. van Leeuwen and R. B. Tan. Computer Networks with Compact Routing Tables. In *The Book of L*, pages 259–273. Springer Berlin Heidelberg, January 1986.

[8] T. Eilam, S. Moran, and S. Zaks. A simple DFS-Based algorithm for linear interval routing. In Marios Mavronicolas and Philippas Tsigas, editors, *Distributed Algorithms*, number 1320 in Lecture Notes in Computer Science, pages 37–51. Springer Berlin Heidelberg, January 1997.

[9] Tamar Eilam, Cyril Gavoille, and David Peleg. Compact Routing Schemes with Low Stretch Factor (Extended Abstract). In *Proceedings of the Seventeenth Annual ACM Symposium on Principles of Distributed Computing*, PODC '98, pages 11–20, New York, NY, USA, 1998. ACM.

[10] Nicolas Nisse, Ivan Rapaport, and Karol Suchan. Distributed Computing of Efficient Routing Schemes in Generalized Chordal Graphs. In Shay Kutten and Janez Žerovnik, editors, *Structural Information and Communication Complexity*, number 5869 in Lecture Notes in Computer Science, pages 252–265. Springer Berlin Heidelberg, January 2010.

[11] A Zabian and M.A Bonuccelli. On the latency of BFS based interval cooperative Web caching. In *2004 International Conference on Information and Communication Technologies: From Theory to Applications, 2004. Proceedings*, pages 637–638, April 2004.

[12] Savio S. H. Tse and Francis C. M. Lau. Lower bounds for multi-label interval routing. pages 123–134, 1995.

[13] Savio S. H. Tse and Francis C. M. Lau. Two Lower Bounds for Multi-Label Interval Routing. pages 123–134, 1997.

[14] Savio S. H. Tse and Francis C. M. Lau. New bounds for multi-label interval routing. *Theoretical Computer Science*, 310(1–3):61–77, January 2004.

[15] R. Královič, P. Ružička, and D. Štefankovič. The complexity of shortest path and dilation bounded interval routing. In Christian Lengauer, Martin Griebl, and Sergei Gorlatch, editors, *Euro-Par'97 Parallel Processing*, number 1300 in Lecture Notes in Computer Science, pages 258–265. Springer Berlin Heidelberg, January 1997.

[16] Erwin M. Bakker, Jan Van Leeuwen, and Richard B. Tan. Linear Interval Routing. *The Computer Journal*, 30:298–307, 1991.

[17] Marco Di Benedetto, Ramana Mellacheruvu, Norman W. Finn, and Umesh Mahajan. Multiple instance spanning tree protocol, April 2012. U.S. Classification 370/256, 370/402, 370/395.53, 370/254; International Classification H04L12/46, H04L12/56, H04L12/28; Cooperative Classification H04L12/4695, H04L45/48, H04L12/4641, H04L45/02, H04L12/4691, H04L12/462; European Classification H04L12/46V, H04L12/46B7, H04L12/46V3B3, H04L12/46V3B2, H04L45/48, H04L45/02.

[18] G. Ibanez, A. Garcia, and A. Azcorra. Alternative multiple spanning tree protocol (AMSTP) for optical Ethernet backbones. In *29th Annual IEEE International Conference on Local Computer Networks, 2004*, pages 744–751, November 2004.

[19] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[20] Jad Naous, David Erickson, G. Adam Covington, Guido Appenzeller, and Nick McKeown. Implementing an OpenFlow switch on the NetFPGA platform. In *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, ANCS '08, pages 1–9, New York, NY, USA, 2008. ACM.

[21] Mao Yang, Yong Li, Depeng Jin, Li Su, Shaowu Ma, and Lieguang Zeng. OpenRAN: A Software-defined Ran Architecture via Virtualization. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*, SIGCOMM '13, pages 549–550, New York, NY, USA, 2013. ACM.

[22] S. Costanzo, L. Galluccio, G. Morabito, and S. Palazzo. Software Defined Wireless Networks: Unbridling SDNs. In *2012 European Workshop on Software Defined Networking (EWSDN)*, pages 1–6, October 2012.

[23] H. Ali-Ahmad, C. Cicconetti, A. De La Oliva, V. Mancuso, M. Reddy Sama, P. Seite, and S. Shanmugalingam. An SDN-Based Network Architecture for Extremely Dense Wireless Networks. In *Future Networks and Services (SDN4FNS), 2013 IEEE SDN for*, pages 1–7, November 2013.

[24] Thomas Clausen, Philippe Jacquet, Cédric Adjih, Anis Laouiti, Pascale Minet, Paul Muhlethaler, Amir Qayyum, and Laurent Viennot. Optimized Link State Routing Protocol (OLSR). 2003.

[25] Charles Perkins, Elizabeth Belding-Royer, Samir Das, and others. RFC 3561-ad hoc on-demand distance vector (AODV) routing. *Internet RFCs*, pages 1–38, 2003.

[26] C.R. Aponte and S. Bohacek. OLSR and approximate distance routing: Loops, black holes, and path stretch. In *2012 Fourth International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–10, January 2012.

[27] Rob van Glabbeek, Peter Höfner, Wee Lum Tan, and Marius Portmann. Sequence Numbers Do Not Guarantee Loop Freedom: AODV Can Yield Routing Loops. In *Proceedings of the 16th ACM International Conference on Modeling, Analysis & Simulation of Wireless and Mobile Systems*, MSWiM '13, pages 91–100, New York, NY, USA, 2013. ACM.

[28] Takuya Akiba, Yoichi Iwata, and Yuichi Yoshida. Fast Exact Shortest-path

Distance Queries on Large Networks by Pruned Landmark Labeling. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, SIGMOD '13, pages 349–360, New York, NY, USA, 2013. ACM.

[29] J.J. Garcia-Luna-Aceves and D. Sampath. Scalable integrated routing using prefix labels and distributed hash tables for MANETs. In *IEEE 6th International Conference on Mobile Adhoc and Sensor Systems, 2009. MASS '09*, pages 188–198, October 2009.

[30] Dhananjay Sampath, Suchit Agarwal, and J.J. Garcia-Luna-Aceves. "Ethernet on AIR': Scalable Routing in very Large Ethernet-Based Networks. In *2010 IEEE 30th International Conference on Distributed Computing Systems*, pages 1–9, Genoa, Italy, June 2010.

[31] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social Networks*, 32(3):245–251, July 2010.

[32] Maciej Kurant, Athina Markopoulou, and Patrick Thiran. Towards Unbiased BFS Sampling. *arXiv:1102.4599 [cs, stat]*, February 2011. arXiv: 1102.4599.

[33] Shankar Bhamidi, Jesse Goodman, Remco van der Hofstad, and Júlia Komjáthy. Degree distribution of shortest path trees and bias of network sampling algorithms. *arXiv:1310.5672 [math]*, October 2013. arXiv: 1310.5672.

[34] Thomas R. Henderson, Mathieu Lacage, George F. Riley, C. Dowell, and J. B.

Kopena. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, 2008.

[35] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *Boost Graph Library: User Guide and Reference Manual, The.* Pearson Education, 2001.

[36] David Peleg and Eli Upfal. A Trade-off Between Space and Efficiency for Routing Tables. *J. ACM*, 36(3):510–530, July 1989.

[37] James Kempf, Scott Whyte, Jonathan Ellithorpe, Peyman Kazemian, Mart Haitjema, Neda Beheshti, Stephen Stuart, and Howard Green. OpenFlow MPLS and the open source label switched router. In *Proceedings of the 23rd International Teletraffic Congress*, ITC '11, pages 8–14, San Francisco, California, 2011. ITCP.

[38] S. Das, A.R. Sharafat, G. Parulkar, and N. McKeown. MPLS with a simple OPEN control plane. In *Optical Fiber Communication Conference and Exposition (OFC/NFOEC), 2011 and the National Fiber Optic Engineers Conference*, pages 1–3, 2011.

[39] Cisco. Cisco Nexus 7000 Switch Architecture (2013 Orlando).

[40] Cisco. Exploring the engineering behind the making of a switch (2012 San Diego).

[41] David McCullough. uCLinux for Linux programmers. *Linux J.*, 2004(123):7–, July 2004.

[42] Terasic Technologies. http://www.terasic.com, 2015.

[43] Altera Corporation. https://documentation.altera.com/, 2015.

[44] Altera. Triple Speed Ethernet Data Path Reference Design. https://www.altera.com/en_US/pdfs/literature/an/an483.pdf, 2009.

[45] Altera Wiki. Triple Speed Ethernet with MII/GMII Interface Hardware Test By Using System Console Reference Design. http://www.alterawiki.com/wiki/Triple_Speed_Ethernet_with_MII/GMII_Interface_Hardware_Test_By_Using_System_Console_Reference_Design, 2010.

[46] Altera Wiki. UClinuxDist. http://www.alterawiki.com/wiki/UClinuxDist, 2013.

[47] Altera Wiki. Devicetree. http://www.alterawiki.com/wiki/Devicetree, 2013.

[48] Altera. Accelerating Nios II Networking Applications. https://www.altera.com/en_US/pdfs/literature/an/an440.pdf, 2013.

[49] Altera. Using Triple-Speed Ethernet on DE4 Boards. ftp://ftp.altera.com/up/pub/Altera_Material/14.0/Tutorials/DE4/using_triple_speed_ethernet.pdf, 2014.

[50] Mooi Mooi Tan. Nios II Linux User Manual for Stratix

IV. http://www.rocketboards.org/foswiki/Documentation/NiosIILinuxUserManualForStratixIV, 2015.

[51] Altera Forums. TSE (Triple Speed Ethernet) MAC on uClinux with MMU. http://www.alteraforum.com/forum/showthread.php?t=31098, 2011.

[52] Terasic. Terasic DE4 Reference Design. http://www.terasic.com.tw/cgi-bin/page/archive.pl?Language=English&CategoryNo=138&No=501&PartNo=4, 2014.

[53] R.L. Carter and M.E. Crovella. Server selection using dynamic path characterization in wide-area networks. In , *Proceedings IEEE INFOCOM '97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution*, volume 3, pages 1014–1021 vol.3, April 1997.

[54] Vladimir Brik, Arunesh Mishra, and Suman Banerjee. Eliminating Handoff Latencies in 802.11 WLANs Using Multiple Radios: Applications, Experience, and Evaluation. In *Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement*, IMC '05, pages 27–27, Berkeley, CA, USA, 2005. USENIX Association.

[55] K. Obraczka and F. Silva. Network latency metrics for server proximity. In *IEEE Global Telecommunications Conference, 2000. GLOBECOM '00*, volume 1, pages 421–427 vol.1, 2000.

[56] Thomas E. Anderson, Susan S. Owicki, James B. Saxe, and Charles P. Thacker. High-speed switch scheduling for local-area networks. *ACM Transactions on Computer Systems*, 11:319–352, November 1993.

[57] Nick McKeown and Thomas E Anderson. A Quantitative Comparison of Iterative Scheduling Algorithms for Input-Queued Switches. *COMPUTER NETWORKS AND ISDN SYSTEMS*, 30:2309–2326, 1998.

[58] N. McKeown. The iSLIP scheduling algorithm for input-queued switches. *IEEE/ACM Transactions on Networking*, 7(2):188–201, April 1999.

[59] I. Keslassy, Cheng-Shang Chang, N. McKeown, and Duan-Shin Lee. Optimal load-balancing. In *Proceedings IEEE INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1712– 1722 vol. 3. IEEE, March 2005.

[60] Nick McKeown, Martin Izzard, Adisak Mekkittikul, William Ellersick, and Mark Horowitz. *The Tiny Tera: A Packet Switch Core.* 1996.

[61] D. Torres, J. Gonzalez, M. Guzman, and L. Nunez. A new bus assignment in a designed shared bus switch fabric. In *Circuits and Systems, 1999. ISCAS '99. Proceedings of the 1999 IEEE International Symposium on*, volume 1, pages 423–426 vol.1, 1999.

[62] H. Kuwahara, N. Endo, M. Ogino, T. Kozaki, Y. Sakurai, and S. Gohara. A shared buffer memory switch for an ATM exchange. In *Communications, 1989.*

ICC '89, BOSTONICC/89. Conference record. 'World Prosperity Through Communications', IEEE International Conference on, pages 118–122 vol.1, 1989.

[63] A.P.J. Engbersen. Prizma switch technology. *IBM Journal of Research and Development*, 47(2.3):195–209, March 2003.

[64] L. Mhamdi. A Partially Buffered Crossbar packet switching architecture and its scheduling. In *Computers and Communications, 2008. ISCC 2008. IEEE Symposium on*, pages 942–948, 2008.

[65] Xiao Zhang and L.N. Bhuyan. An efficient scheduling algorithm for combined input-crosspoint-queued (CICQ) switches. In *IEEE Global Telecommunications Conference, 2004. GLOBECOM '04*, volume 2, pages 1168–1173 Vol.2, November 2004.

[66] T. Javidi, R. Magill, and T. Hrabik. A high-throughput scheduling algorithm for a buffered crossbar switch fabric. In *IEEE International Conference on Communications, 2001. ICC 2001*, volume 5, pages 1586–1591 vol.5, 2001.

[67] S. Nojima, E. Tsutsui, H. Fukuda, and M. Hashimoto. Integrated Services Packet Network Using Bus Matrix Switch. *IEEE Journal on Selected Areas in Communications*, 5(8):1284–1292, October 1987.

[68] M. Katevenis, G. Passas, D. Simos, I. Papaefstathiou, and N. Chrysos. Variable

packet size buffered crossbar (CICQ) switches. In *Communications, 2004 IEEE International Conference on*, volume 2, pages 1090–1096 Vol.2, 2004.

[69] Masayoshi Nabeshima and Thisletter Prt. *Performance Evaluation of a Combined Input- and Crosspoint-Queued Switch.* 2000.

[70] Neda Beheshti, Yashar Ganjali, Ramesh Rajaduray, Daniel Blumenthal, and Nick McKeown. Buffer Sizing in All-Optical Packet Switches. In *Optical Fiber Communication Conference and Exposition and The National Fiber Optic Engineers Conference (2006), paper OThF8*, page OThF8. Optical Society of America, March 2006.

[71] Jingcao Hu and Radu Marculescu. Energy-aware Mapping for Tile-based NoC Architectures Under Performance Constraints. In *Proceedings of the 2003 Asia and South Pacific Design Automation Conference*, ASP-DAC '03, pages 233–239, New York, NY, USA, 2003. ACM.

[72] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-stage Design Space Exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, DATE '09, pages 423–428, 3001 Leuven, Belgium, Belgium, 2009. European Design and Automation Association.

[73] Jian Wang and T.H. Szymanski. Power analysis of Input-Queued and

Crosspoint-Queued crossbar switches. In *Canadian Conference on Electrical and Computer Engineering, 2009. CCECE '09*, pages 273–278, May 2009.

[74] K. Goossens, J. Dielissen, and A. Radulescu. AEthereal network on chip: concepts, architectures, and implementations. *Design & Test of Computers, IEEE*, 22(5):414–421, 2005.

[75] M. Millberg, E. Nilsson, R. Thid, and A. Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the nostrum network on chip. In *Design, Automation and Test in Europe Conference and Exhibition, 2004. Proceedings*, volume 2, pages 890–895 Vol.2, 2004.

[76] M. Dall'Osso, G. Biccari, L. Giovannini, D. Bertozzi, and L. Benini. Xpipes: a latency insensitive parameterized network-on-chip architecture for multiprocessor SoCs. In *Computer Design, 2003. Proceedings. 21st International Conference on*, pages 536–539, 2003.

[77] T.G. Mattson, R. Van der Wijngaart, and M. Frumkin. Programming the Intel 80-core network-on-a-chip Terascale Processor. In *High Performance Computing, Networking, Storage and Analysis, 2008. SC 2008. International Conference for*, pages 1–11, 2008.

[78] T. Bjerregaard and J. Sparso. A router architecture for connection-oriented service guarantees in the MANGO clockless network-on-chip. In *Design, Automation and Test in Europe, 2005. Proceedings*, pages 1226–1231 Vol. 2, 2005.

[79] D. Wiklund, A. Ehliar, and D. Liu. Design of an Internet core router using the SoCBUS network on chip. In *Signals, Circuits and Systems, 2005. ISSCS 2005. International Symposium on*, volume 2, pages 513 – 516 Vol. 2, July 2005.

[80] A. Ehliar and Dake Liu. An FPGA Based Open Source Network-on-Chip Architecture. In *Field Programmable Logic and Applications, 2007. FPL 2007. International Conference on*, pages 800 –803, August 2007.

[81] Geng Luo-Feng, Du Gao-ming, Zhang Duo-Li, Gao Ming-Lun, Hou Ning, and Song Yu-Kun. Design and performance evaluation of a 2d-mesh Network on Chip prototype using FPGA. In *Circuits and Systems, 2008. APCCAS 2008. IEEE Asia Pacific Conference on*, pages 1264 –1267, December 2008.

[82] S.A. Asghari, H. Pedram, and M. Khademi. A flexible design of network on chip router based on handshaking communication mechanism. In *Computer Conference, 2009. CSICC 2009. 14th International CSI*, pages 225 –230, October 2009.

[83] K. Goossens, L. Mhamdi, and I.V. Senin. Internet-Router Buffered Crossbars Based on Networks on Chip. In *Digital System Design, Architectures, Methods and Tools, 2009. DSD '09. 12th Euromicro Conference on*, pages 365 –374, August 2009.

[84] Lotfi Mhamdi, Kees Goossens, and Iria Varela Senin. Buffered Crossbar Fab-

rics Based on Networks on Chip. In *Communication Networks and Services Research Conference (CNSR), 2010 Eighth Annual*, pages 74 –79, May 2010.

[85] W.J. Dally and C.L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *Computers, IEEE Transactions on*, C-36(5):547–553, 1987.

[86] D. Gollmann and W.G. Chambers. Clock-controlled shift registers: a review. *IEEE Journal on Selected Areas in Communications*, 7(4):525–533, May 1989.

[87] Altera. Profiling Nios II Systems, High-Resolution Timer. `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/an/an391.pdf`, 2011.

[88] Hosein F. Badran and H. T. Mouftah. ATM switch architectures with input-output-buffering: effect of input traffic correlation, contention resolution policies, buffer allocation strategies and delay in backpressure signal. *Computer Networks and ISDN Systems*, 26(9):1187–1213, May 1994.

[89] Balaji Prabhakar and Nick McKeown. On the speedup required for combined input- and output-queued switching. *Automatica*, 35(12):1909–1920, December 1999.

[90] Achille Pattavina and Giacomo Bruzzi. Analysis of Input and Output Queueing for Nonblocking ATM Switches. *IEEE/ACM Trans. Netw.*, 1(3):314–328, June 1993.

[91] D.C.W. Pao and S.C. Leung. Sharing buffer in an input-output buffered ATM switch without scaling up memory bandwidth requirement. In *Computer Communications and Networks, 1995. Proceedings., Fourth International Conference on*, pages 339–343, 1995.

[92] Paola Scozzafava. Uniform distribution and sum modulo m of independent random variables. *Statistics & Probability Letters*, 18(4):313–314, November 1993.

[93] Thornton C. Fry. *Probability and its engineering uses,*. D. Van Nostrand Company, inc, 1928.

[94] Open SystemC Initiative. IEEE standard SystemC language reference manual. *IEEE Computer Society*, pages 1666–2005, 2006.

[95] S. Sutherland. *Verilog 2001: A guide to the new Verilog standard*. Kluwer Academic Publishers, Boston, Massachusetts, 2001.

[96] VHDL Language reference manual. *IEEE Std*, pages 1076–1987, 1988.

[97] The Beatles. The Long and Winding Road, 1970.