Cut-through network switches: architecture, design and implementation

Noa Zilberman Łukasz Dudziak Matthew Jadczak
Thomas Parks Alessandro Rietmann Vadim Safronov
Daniel Zuo

Abstract

Cut-through switches are increasingly used within data-centres and high-performance networked systems. Despite their popularity, little is known of the architecture of cut-through switches used in today's networks. In this paper we introduce three different cut-through switch architectures, designed over the NetFPGA platform. Beyond exploring architectural and design considerations, we compare and contrast the different architectures, providing insights into real-world switch design. Last, we provide an evaluation of one successfully implemented cut-through switch design, providing constant switching latency regardless of packet size and cross-traffic, without compromising throughput.

1 Introduction

The rise of cloud computing and the ever increasing amounts of data generated by user applications have driven the development of high performance network switches. The performance of these switches has increased by two orders of magnitude in less than a decade [21], matching the market's requirements. This class of high performance switches provides not only superior throughput, but also low latency, in the order of hundreds of nanosecond for a full-fledged layer two packet switch [20, 17].

Packet switches can be crudely divided to two classes of switches: store-and-forward and cut-through. Store-and-forward switches wait for an entire packet to arrive and for a checksum to be validated before a packet starts being processed. In contrast, cut-through switches start processing the packet as soon as the first chunk of data arrives, and do not wait for any error-detection process to be completed. By the time an error can be detected, the packet has already propagated through the chip, and the design needs to attend to dropping or marking the packet as faulty.

Cut-through switches go back over 40 years [15], in the days where networks were slow, memory was fast and writing packets to the memory took "negligible" time. As time went by, networks became significantly faster and memory access time was no longer negligible, invalidating past cut-through designs. Newer designs have avoided memory access to enable higher performance [13], yet these designs are well over a decade old, and don't capture some current design considerations. Despite the importance of cut-through switches, little has

been published in this area in the recent years, and most advancements are covered by patents (e.g., [18, 16, 4]).

High Performance Networking (P51), is a graduate course at the University of Cambridge Department of Computer Science and Technology. In 2018, the students in the course were challenged to develop a packet-switch that has a significantly higher performance than a base reference design, running on the NetFPGA SUME platform [19]. As part of this work, three architectures were proposed for cut-through switches. In this paper, we build upon this experience to inform the reader of cut-through switches design considerations and reflect on the importance of different design trade-offs. In particular, we make the following contributions:

- we introduce three realistic cut-through packet switch architectures,
- we explore design considerations in each architecture and compare-and-contrast the three different designs, and
- we provide an evaluation of one successfully implemented cut-through switch design.

The rest of this document is organized as follows: §2 provides a brief introduction to the NetFPGA platform. §3 describes the three switch architectures, followed by a comparison of the three in §4. We provide an evaluation of one of the architectures in §5, before discussing related work in §6 and concluding in §7.

2 NetFPGA

NetFPGA (`netfpga.org`) is an open platform enabling researchers and instructors to build high-speed, hardware-accelerated networking systems. The platform can be used by researchers to prototype advanced services for next-generation networks. It can also be used in the classroom to teach students how to build Ethernet switches and Internet Protocol (IP) routers using hardware rather than software. The most prominent NetFPGA success is OpenFlow, which in turn has reignited the Software Defined Networking movement. NetFPGA enabled OpenFlow by providing a widely available open-source development platform capable of line-rate operation and was, until its commercial uptake, the reference platform for OpenFlow.

The NetFPGA SUME [19] is the third generation of NetFPGA platforms. The board is a low-cost, PCIe host adapter card able to support 40Gbps and 100Gbps applications, with a Xilinx Virtex-7 690T FPGA at its core. The board uses four 10Gbps SFP+ Ethernet interfaces and a PCIe Gen 3 $\times 8$ interface to the host. Other features include off-chip SRAM and DRAM memories, MicroSD and two SATA interfaces for storage, debug interfaces and more.

NetFPGA supports three reference projects: Reference Network Interface Card (NIC), Reference Ethernet Switch and Reference IPv4 Router. The Reference Ethernet Switch is the basis for the designs described in this paper. In its original form, the Reference Switch is a simple store-and-forward switch, supporting packet sizes of 64B to 1518B. Its data-path architecture is depicted in Figure 1. The control-plane is omitted from this drawing. Packets arrive to the switch through one of four 10G Ethernet ports, implemented using Xilinx's 10G Ethernet subsystem and some wrapping logic (receive and transmit queues, clock domain crossing and data-path width converter) or through the PCIe interface from the host. Packets are admitted into the NetFPGA 256b-wide data-path in the *Input-Arbiter* module, using round-robin

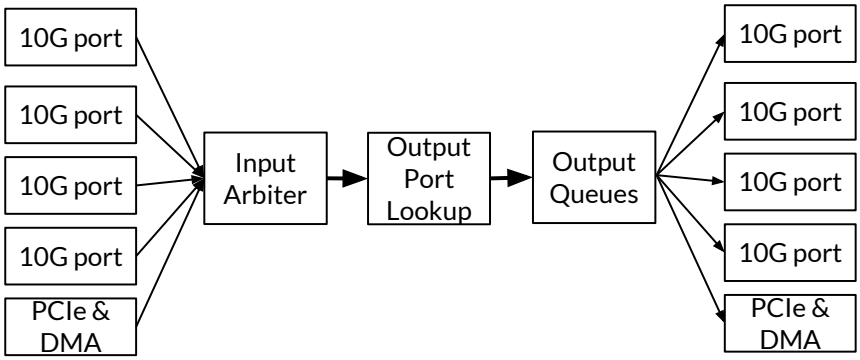


Figure 1: NetFPGA Reference Switch data-path

arbitration. Next, an output port is assigned to every packet in the *Output Port Lookup* module, according to the destination MAC address. The destination address is matched against an output port using a content addressable memory (CAM). If the destination does not exist in the output port lookup table, the packet is broadcasted to all ports, excluding the source port. Once the packet was assigned an output port, it is sent to the *Output Queues* module, where it is queued until the transmit side of the 10G port module is available to process and send the packet.

3 Switch Architectures

In this section we introduce three cut-through switch architectures, proposed by the authors as part of the assessment in the High Performance Networking course. The main goal of all designs was to provide constant, or close to constant, latency to all packets going through the switch, regardless of packet size, and to minimize the effect of cross traffic on latency.

3.1 Architecture A: Slotted Pipeline

The first cut-through switch architecture, proposed by Dudziak and Jadczak, maintains the main components of the original NetFPGA pipeline, and the same order of modules, but using a slotted pipeline, as illustrated in Figure 2.

The architecture uses a single data-path, that all ports share. This data-path maintains the 256b width of the original Reference Switch. In each clock cycle, only a single port is allowed to write a "fragment" of 256b to the data-path. The modules are modified to work on "fragments", rather than on packet granularity, and multiple packets may be interleaved within the pipeline. The number of interleaved packets is at most four: as the number of 10G ports.

The 10G port module, used in the Reference Switch (Figure 1) remains largely unchanged: the three modules indicated in Figure 3(a) as 10G Ethernet subsystem, Rx Queue and Gearbox are the composing units of the 10G Port. While the Xilinx 10G Ethernet subsystem remains the same, the Rx Queue (crossing clock domains using a BRAM-based FIFO) and the Gearbox (converting from 64b to 256b and adding meta-data to the packet) are modified: while these blocks were previously store-and-forward, each of them now works on a data-unit level without

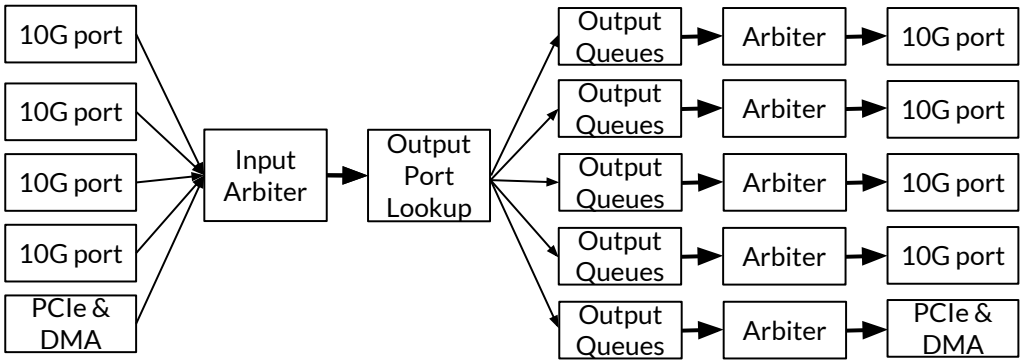
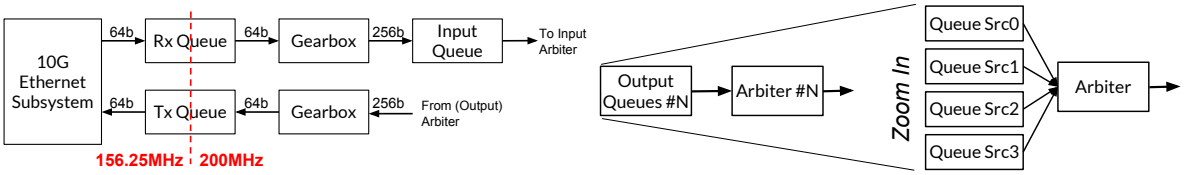


Figure 2: Architecture A: Standard Pipeline



(a) 10G Port Module

(b) Output modules

Figure 3: Architecture A: A zoom in on the 10G Port module (a) and the breakdown of output queues and arbiter modules (b).

waiting for an entire packet to arrive. The gearboxes are followed by inputs queues, with one queue per port, storing chunks of data until they can be admitted into the data-path. Under normal circumstances (i.e., no back pressure) that means until the next free slot allocated to the port.

The Input Arbiter governs the inputs to the internal data-path. It applies a simple round robin between the ports to admit chunks of data. Unlike the Reference Switch's input arbiter, this module selects a new input port every clock, assuming the respective input queue is not empty, rather than arbitrating on packet level. As a result, data within the pipeline after the input arbiter is multiplexed on a fragment-level.

The output port lookup is modified to work on a fragment level rather than on a packet level. This requires a change to the metadata going through the pipeline: while in the reference design only the metadata in the first fragment of a packet is valid, here every fragment carries valid metadata, indicating the source and destination port. Header processing is done on packet level: as a new packet begins, its header is being processed and the output port is identified (using the output port lookup table). Once the output port is identified, the information is latched per-port, and updated in the metadata of all the following fragments from the same port, until an end of packet indication is received[1].

Cut-through switches implement, in a sense, store-and-forward of headers only. In this architecture, as the switch is an Ethernet switch, the first fragment of data (256b, meaning 32B), already carries the entire header. Where larger headers are in use, there may be an impact on latency or resources. For example, had the lookup been done for, e.g., IPv4 address, it would

[1] A *TLAST* signal on AXI-STREAM bus is used in the design

have been required to save the relevant header fields in the first fragment of data, until the entire header has arrived on the second fragment of data.

While packets are going through the data-path in fragments, interleaved with other packets, they are required to be reassembled into packets before being sent out. This is done in the output queues module, shown in Figure 3(b), that is significantly modified to support this function. While in the reference switch there is only a single queue in front of each output port, here there is a queue in front of an output port per each source port. As in the NetFPGA SUME design there are four 10G ports and a DMA port, this means that four queues are required in front of each total port (destination port must always be different to the source port, thus four rather than 5 ports are required). The overall number of queues in the Output Queues module is therefore, including the DMA, twenty.

A fragment arriving from the Output Port Lookup to the Output Queues is placed, using a demultiplexer, into one of the queues based on the combination of source and destination port. By aggregating fragments of source-destination pairs inside queues, the packets are reassembled. At the output of the Output Queues module there is a round-robin arbiter that sends each time a packet from a different queue into the output port. Here already the operation is on packet level, rather than fragments. The arbiter is work conserving, thus empty queues do not delay the transmission of packets from non-empty queues. The Output Queues module does not assert flow control under normal operating conditions: if an output port is over-subscribed, as two queues or more are competing on sending packets to the same destination, the packets (not just fragments) will be dropped rather than admitted to full queues.

As the 10G output port operates on the reassembled packets, there is conceptually no need for it to be changed from the reference 10G port design. The only consideration is avoiding any buffering of complete packets within the module, by handling every fragment as soon as it is available. This is balanced with the requirement to continuously send data out through the port, once a packet starts transmitting, and until the packet ends. The architecture guarantees this property.

3.2 Architecture B: Parallel Pipelines

The second architecture, proposed by Parks and Rietmann, presents a completely different approach to a cut-through design. Instead of using a single data-path, it uses a dedicated data-path per input port. Furthermore, instead of using 256b data-path, a 64-bit data-path is used. With four pipelines running in parallel, this means that effectively every clock 256b are handled - same as before. Another difference is the use of 156.25MHz clock across the design, the same clock frequency used by the port, saving the need to cross clock domains, but also losing speed up.

The core architectural decisions described above simplify a lot of the design considerations, but complicate others. The 10G port is significantly simplified: first, there is no need to synchronise between clock domains or to aggregate a bus from 64b to 256. This turns redundant the Rx queue and gearbox within the 10G port, allowing the new architecture to connect (almost) directly from the 10G Ethernet subsystem to the data-path. Similar logic is applied also on the transmit side of the 10G Port.

Instead of an Input Arbiter, the 10G port is followed by the Output Port Lookup module. The Output Port Lookup maintains its functionality, except with one significant difference: the output port lookup table, implemented in a CAM, is now shared between four different

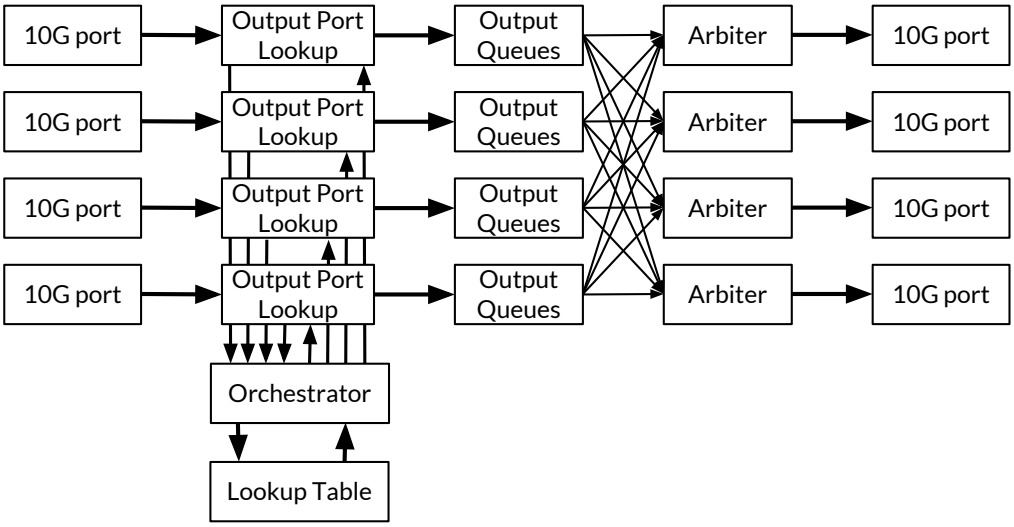


Figure 4: Architecture B: Reordered, Multiple Pipeline

pipelines. The reason for sharing is not simply to save resources: as this is a learning switch, where the lookup table should be updated with every new MAC address learned, it is imperative for the pipelines to share information: either by broadcasting newly learned entries to all pipelines, or, as done here, by sharing a single table.

The property that enables the sharing of lookup resources is the data-path width: as the data-path width is just 64b, and the minimum packet size is 64B, it is guaranteed that at least eight clock cycles will pass between every pair of packets from the same input port. The architecture builds upon this knowledge to orchestrate access to the lookup table. Whenever a new packet starts and its header is parsed, the respective pipeline asserts a request signal to an orchestrator module, with the destination address that requires lookup. The orchestrator is responsible to access the lookup table (CAM) and return the reply (destination port) to the correct pipeline. With eight clock cycles between queries from a given pipeline, and four parallel pipelines, the maximum access rate to the lookup table is once every two clock cycles.

The Output Port Lookup is followed by the Output Queues. This module is unmodified (except for the bus width), but there is one instance of it in every pipeline. As a result, there are now four queues in front of each output port: one from each pipeline. The decision which pipeline will be sending a packet next to the output port is taken by the Arbiter (similar to the Input Arbiter), which operates on a packet level and is work conserving. While the operation here is on packet boundary, the cut-through nature of the device is maintained, as two queues will be competing on the same output port only if it is (momentarily or continuously) over-subscribed.

3.3 Architecture C: Split Pipeline

Safronov and Zuo proposed a different approach to cut-through switch design, separating the header from the data[2]. This architecture, illustrated in Figure 5, takes the "traditional" cut-through approach [15]. Packets arriving at the Input Arbiter from the 10G Port, using the internal 256b-wide bus, are split: the header of the packet is sent to the Output Port Lookup,

[2]The description here slightly amends the original proposal.

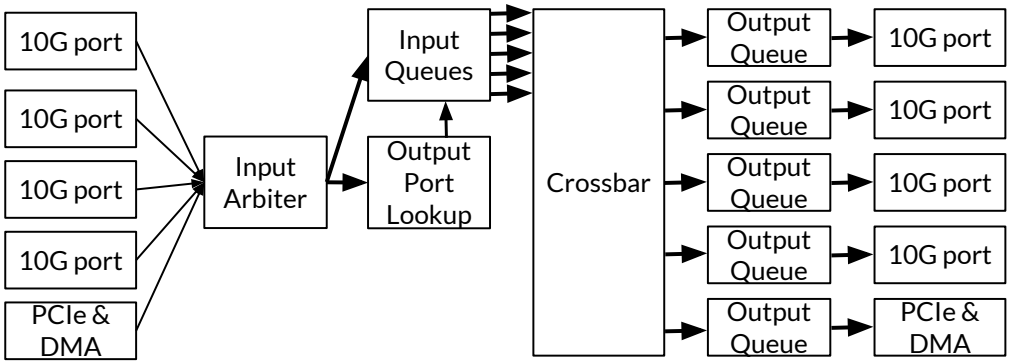


Figure 5: Architecture B: Split Pipeline

while concurrently the entire packet is stored in a queue. An Input Arbiter is responsible to admit a header to the Output Port Lookup, arbitrating between the four 10G ports in a round robin manner. The arbiter can admit a new header only once every two clock cycles, the time required for an access to the output port lookup table. As far as the Output Port Lookup module is concerned, little changes, except that every valid data cycle is a header, and no payload goes through the module. The main difference is the output: the only significant output is the packet's metadata, which is now sent to the queue associated with the source port (noted in the metadata). A queue holds two FIFOs: one with the packet and one with its associated metadata, received from the Output port lookup. This structure is similar to the queue structure in the reference Output Queues.

Once a packet receives the metadata information from the Output Port Lookup, it starts sending the packet to the respective output queue. Sending to an output queue is done through a crossbar, similar in implementation to the four output-arbiters used in the previous architectures. At any given time, only a single input queue can send a packet to a given output queue, but all four input queues can simultaneously send packets to output queues, for a given non-congested permutation. Once a packet arrives at the output queue, it is transmitted through the 10G port to its destination.

While Output Queues may seem at first sight redundant under this scheme, given that packets are already stored in queues, they in fact serve an important role: avoiding head of line (HOL) blocking and incorrect packet drop. Let us assume a scenario where packets from Port 1 are sent to Port 3, and that packets from Port 2 are sent either to Port 1 or to Port 3, with an equal probability. In this scenario, Port 3 is over-subscribed, which means that packets will need to be dropped as queues become full. If only the input queues had been used, packets were dropped from both Input Queue 1 and Input Queue 2, with an equal probability of packets destined to Port 1 and Port 3 being dropped. By using an output queue, only packets sent to Port 3 will be dropped, and no packet that was destined to Port 1.

Note that this design maintains cut-through principles: unless two packets are contending on the same port, at no point is a full packet buffered, nor does it need to wait for a packet from another port to be admitted or dequeued. The maximum delay a packet may need to wait, in a non-subscribed switch, is six clock cycles, as packet headers from $N - 1$ ports are admitted ahead of it to the Output Port Lookup.

Table 1: Cut-Through Switch Architectures: A Comparison

	A: Slotted	B: Parallel	C: Split
Number of pipelines	$1\times$ header+payload	$4\times$ header+payload	$1\times$ header + $1\times$ payload
Data-path width	256b	64b	256b
Data-path frequency	200MHz	156.25MHz	200MHz
Number of lookup tables	1	1	1
Number of output queues (assuming 4 output ports+ DMA)	20	20	5
"Penalty"	Latency	Logic resources	Memory resources

4 Design Decisions

In this section we consider design decisions taken by each of the architectures. When we compare the different architectures we consider aspects such as complexity, resource usage and scalability. While complexity can be considered a one-time effort, it is usually reflected in resource usage, maintainability and, unfortunately, the probability of bugs in the design. Table 1 summarises the main differences between the three architectures.

A cut-through switch obviously means a change to the way incoming packets are handled by the logic compared with a store-and-forward switch, as the logic needs to operate on a header, or a fragment, rather than on a packet level. The important question then becomes the treatment of multiple packets, arriving on multiple ports, at the same time. The reference architecture, which used a single pipeline for the data-path, rotated between the ports while handling one packet at a time. This approach is not applicable here, as we no longer operate at the packet level: every competing packet "ahead" in the pipe can add hundreds of nanosecond latency, leading to latency variability at the scale of traversing the entire switch. Our three architectures demonstrate three different approaches here: (A) interleaving the packets within a single pipeline, (B) keeping four separate pipelines, and (C) the mix of both approaches: keeping packets separate and interleaving the headers.

The disadvantage of the first approach (A) is clear: latency. As the data-path is pipelined, once a packet is admitted into the pipeline, the latency does not depends on the number of slots. Still, the admission into the pipeline depends on the number of slots, meaning that the worst case admission latency will be four times higher than in four parallel pipelines. In a design with N ports, the admission latency will be $\times N$ higher, which won't provide reasonable scalability, as switches already scale today to 260 ports [5]. In practice, however, the latency penalty is not quite so significant: the output port lookup and output queues are not affected by this choice, and the Input Arbiter will only incur $N-1$ clocks penalty. This leads to a second design choice: "hard" slotting or "soft" slotting? "hard" slotting, in this sense, is the constant assignment of a port to a slot, regardless of traffic from other slots (so the pipeline may be idle on certain slots), whereas a "soft" use of slots is work conserving, and a fragment is admitted on the next available slot. "Soft" slotting obviously requires the metadata to indicate for each and every fragment its source port (resource overhead).

The second approach (B) appears at first inspection to require significantly more resources than the first approach, using four pipelines instead of one, but this is not completely accurate. While the design uses four pipelines, each of them is just 64bit wide, rather than 256b. This

is directly reflected in the amount of logic resources required per pipeline. Further, the lookup table is shared and not dedicated, meaning that memory resources are not replicated. There is, of course, still resource overhead, such as the per-pipeline metadata bus. This design also requires a queue for each source-destination queue pair, to prevent head of line blocking, yet this requirement is shared also with architecture A.

The third approach (C) uses resources very conservatively. Its main disadvantage is the use of memory resources for input queues. The solution, however, can be implemented using shared memory resources instead of dedicated ones. Instead of reserving memory per-port using a FIFO, the port can just hold pointers (descriptors) to the packets in the memory, saving overall memory resources. It is not surprising that such a memory sharing approach is also used in high-end network switches [12, 8].

Architecture C is not the only one to require more memory resources: so do both A and B, which use multiple output queues per output port. The value of implementing these queues in a shared memory is, however, questionable: these queues will be underutilized when a port is not congested, and using a shared memory will incur an additional (and unnecessary) latency. On the other hand, when the number of input ports is high, a naive implementation of an arbiter that tries in a single clock cycle to check the status of all ports is unlikely to fit timing requirements. It can instead be implemented using several clock cycles per decision (while the current packet is being transferred), or use a more sophisticated arbitration algorithm. The decision whether to use here shared or dedicated memory resources is therefore a question of scale, meeting timing requirements and overall available resources.

This paper does not include a quantitative evaluation of resource usage as architectures A and B were implemented using different design flows (Verilog vs Haskell/CλaSH), and architecture C was not implemented.

Scaling the number of ports has consequences that go beyond memory resources. While it is common to assume that the limiting performance metric of a switch is bandwidth, it is often packet rate. For a given clock frequency f, a single data-path can't process more than f packets every second, and as in our case every lookup takes two clocks, $f/2$ packets per second. This means that all three architectures are bounded by a packet rate of $f/2$, as in all cases the access to the lookup table is shared. Architectures A and C need to scale their data-path width with the number of ports to $N\ ports \times 64b$. This increase is beneficial for small packet sizes only as long as the time it takes to process a single packet is the same or smaller than the time it takes for the packet to pass through a given point in the design. For example, if the data-path width is 64B, it will take one clock to pass a 64B packet. If a new packet can be processed only once every two clock cycles, there is no gain in such a bus width from packet rate perspective, and 32B provide the same packet rate for less resources. Note that bandwidth-wise, a 64B data-path will be able to support twice the bandwidth at the same clock frequency.

Scaling the number of ports will affect architecture B as well. Currently, the use of four ports, with two clock cycles per lookup and eight clocks to propagate a 64B packet, provides an optimal balance. The sharing of the lookup mechanism will not be as efficient when using, e.g., eight ports. With eight ports, each port will be able to access the lookup table only once every 16 clock cycles, therefore supporting the same maximum packet rate as with a four-port switch, but with twice the maximum bandwidth (for packets of 128B or more). One way to scale the packet rate is to use a small cache in each pipeline, keeping the last 2-4 lookup entries, reducing the access rate to the shared table. This solution will improve the average packet rate for small packet sizes, but will not provide a better packet rate guarantee. Furthermore, using multiple

caches will require maintaining coherency between the caches, as entries in the lookup table may be updated during run time.

One aspect the three architectures do not address is error handling. An error can be detected in an incoming frame only once it has fully arrived. In a 64b bus, for a 1518B packet, and using 156.25MHz clock (the 10G port parameters) that would mean 1.2 microseconds: in current day cut-through switches, including the designs discussed in this paper, the switch would have already started transmitting the packet to the destination by the time an error is detected. If the packet is short, forwarding a detected error to a later stage along the path indicating a packet should be dropped is possible, but it won't help once the packet is longer than the number of stages in the data-path. A concern here is not only that a faulty packet is forwarded, but also that the lookup table will become corrupted. The switch is a learning switch, thus a new source MAC address and its matching source port are learned every time a new source MAC address and source port are identified. If an undetected error corrupted the source MAC address, the lookup table may become corrupted. A solution to that may be to wait with updating the lookup table with new entries until the checksum of the packet providing this entry is validated.

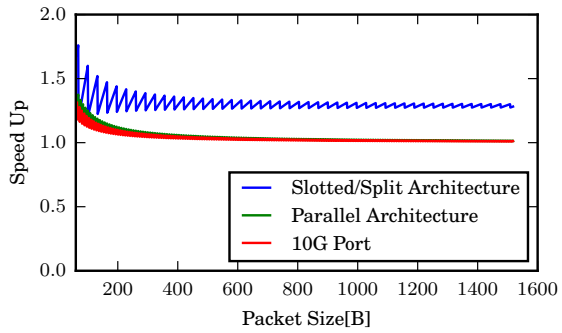
5 Performance

In this sections we cover several performance aspects of the three architectures. First, we discuss the throughput speed-up provided by each design. Next, we discuss the latency within each module. Last, we provide measurement-led evaluation results of architecture A.

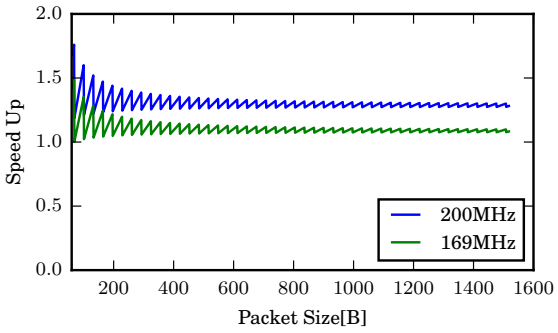
5.1 Data-path speed-up

A common mistake in data-path design is to assume that a data-path will support full line rate for all packet sizes if it can support some packet sizes. However, a fully utilized data-path can provide significantly higher bandwidth than a partly utilized one. Consider a 256b data-path. A data-path full of 64B packets will have 100% utilization, whereas if 65B packets are used, every packet will require 3 clock cycles, the last one using just one byte of data, leading to 67.7% utilization. In this subsection we validate that the proposed designs provide the required throughput for all packet sizes, by examining the speed-up of the data-path. We define the data-path speed-up, as the ratio between the bandwidth required for a given packet size and the bandwidth supported by the data-path. To serve all packet sizes at line rate the minimal speed-up required is 1. A small speed-up benefit of the NetFPGA pipeline is that the Ethernet Frame Check Sequence (FCS) is stripped in the MAC, saving four bytes per packet while passing through the pipeline.

In Tables 2 and 3 we provide some examples of the packet rate and data-path speed-up of the three different architectures. Architectures A and C are both represented by Table 2, as they provide the same packet-rate performance. Figure 6 shows the data-path speed-up of the different architectures. For small packet sizes, the effect of the last fragment is significant, in terms of bus utilisation. This is evident more in architectures A and C, as more bytes are unused in the data-path. For large packet sizes, the effect of the last fragment diminishes, as its relative weight in the packet is smaller. It is important to note that architecture B uses 156.25MHz data-path clock, whereas A and C use 200MHz clock, and thus have a higher



(a) Different Architectures



(b) Different Clock Frequencies

Figure 6: The data-path throughput speed-up of (a) the three architectures and (b) Slotted architecture using different data-path clock frequencies.

Table 2: Throughput Analysis, Slotted (A) and Split (C) architectures. $4 \times 10GE$**, Core clock @200MHz, 10G Port clock @156.25MHz.**

Packet size (Wire)	Max Packet rate (Wire)	Packet size (Internal)	Clocks/Packet (10G Port[3])	Max packet rate (10G Port)	Clocks/Packet (data-path)	Max packet rate (data-path)	Speed-up
64B	14.88Mpps	60B	8	19.53Mpps	2	100Mpps	1.68
69B	14.04Mpps	65B	9	17.36Mpps	3	66.67Mpps	1.19
128B	8.45Mpps	124B	16	9.77Mpps	4	50Mpps	1.48
133B	8.17Mpps	129B	17	9.19Mpps	5	40Mpps	1.22
261B	3.35Mpps	257B	33	4.73Mpps	9	22.22Mpps	1.25
1518B	0.81Mpps	1514B	190	0.82Mpps	42	4.17Mpps	1.28

Table 3: Throughput Analysis, Parallel (B) architecture. $4 \times 10GE$**, Core clock @156.25MHz, 10G Port clock @156.25MHz.**

Packet size (Wire)	Max Packet rate (Wire)	Packet size (Internal)	Clocks/Packet (10G Port)	Max packet rate (10G Port)	Clocks/Packet (data-path)	Max packet rate (data-path)	Speed-up (data-path)
64B	14.88Mpps	60B	8	19.53Mpps	8	78.13Mpps	1.31
69B	14.04Mpps	65B	9	17.36Mpps	9	69.44Mpps	1.24
128B	8.45Mpps	124B	16	9.77Mpps	16	39.06Mpps	1.16
133B	8.17Mpps	129B	17	9.19Mpps	17	36.76Mpps	1.13
261B	4.45Mpps	257B	33	4.73Mpps	33	18.94Mpps	1.06
1518B	0.81Mpps	1514B	190	0.82Mpps	190	3.29Mpps	1.01

speed-up. However, architecture B maintains a speed-up of more than 1 even at 156.25MHz, whereas A and C require a clock frequency of at least 169MHz to support full line rate. Figure 6 shows the speedup gain of architectures A and C using 200MHz data-path frequency rather than 169MHz.

There are benefits to using a lower clock frequency: easier timing closure during the design, lower power consumption and lower chances of errors during operation due to corner timing issues. There are, however, benefits in using a higher clock frequency, including the ability to absorb momentary flow control, and to infrequently send packets from the host to the network.

[3]In Tables 2 and 3, 10G Port excludes the Xilinx 10G Ethernet Subsystem.

Table 4: The latency in clock cycles of going through the data-path in a Slotted Architecture (A).

Module	Input Arbiter		Port Lookup		Output Queues		Total	
	Min	Max	Min	Max	Min	Max	Min	Max
Latency	3	6	2	5	3	147	8	155

Table 5: The latency in clock cycles of going through the data-path in a Parallel Architecture (B).

Module	Port Lookup		Output Queues		Arbiter		Total	
	Min	Max	Min	Max	Min	Max	Min	Max
Latency	5	8	4	574	5	5	13	587

5.2 Data-path latency

We consider the latency of a single data fragment traversing through the data-path. We omit the discussion of the 10G port, but note that in Architecture B the latency will be lower as there is no need to cross clock domains or use a gearbox. Architecture C was not implemented and is therefore excluded.

Table 4 presents the latency, measured in clock cycles, through each of the data-path modules in the Slotted architecture (A). The minimum latency is extracted from simulation and the maximum latency is based on design analysis. The minimum latency reflects the latency through the pipeline when there is only one active source port. As can be seen, the propagation time through the data-path is very short: only 8 clock cycles. The maximum latency shows the variance of the design by exploring the worst case for each module. In the Input Arbiter, the maximum latency is a result of four ports receiving packets at the same time. In the worst case, a port will need to wait for three clock cycles for its slots, while the other ports have the right to send data. The maximum latency in the Output Port Lookup module is five clocks, which in fact are not experienced by the packet: this is the latency for learning a new entry. There is no stall in the pipeline while an entry is being learned, instead a packet is being forwarded according to the currently existing entries in the lookup table. The maximum latency within the Output Queues module reflects the latency of momentary congestion, where all four ports try to send a (single) packet to the same output[4]. As Output Queues send out data as packets, rather than interleaved fragments of packets, in the worst case a packet will need to wait for three other maximum-size packets ahead of it to finish sending, on top of the latency within the module, adding up to 147 clock cycles. This delay is not in violation of cut-through principles, but a result of congestion.

Table 5 presents the latency through each of the data-path modules in the Parallel architecture (B). The minimum latency is extracted from simulation and the maximum latency is based on design analysis. The minimum latency reflects the latency through the pipeline when there is only one active source port. When more than one port is active, the contention point between parallel pipelines is the lookup table. The latency reflects the use of 64b bus within the data-path, compared with 256b in the other architectures. Consequently, the minimum latency through the data-path is slightly higher: 13 clock cycles. The maximum latency in the Output Port Lookup module reflects the access pattern to the CAM: every clock cycles, one pipeline can access the lookup table, meaning that in the worst case a request needs to "wait" three clock cycles for the other ports. Note that the design is pipelined, and thus does not create

[4]We ignore the case of longer congestion, leading to packet drop.

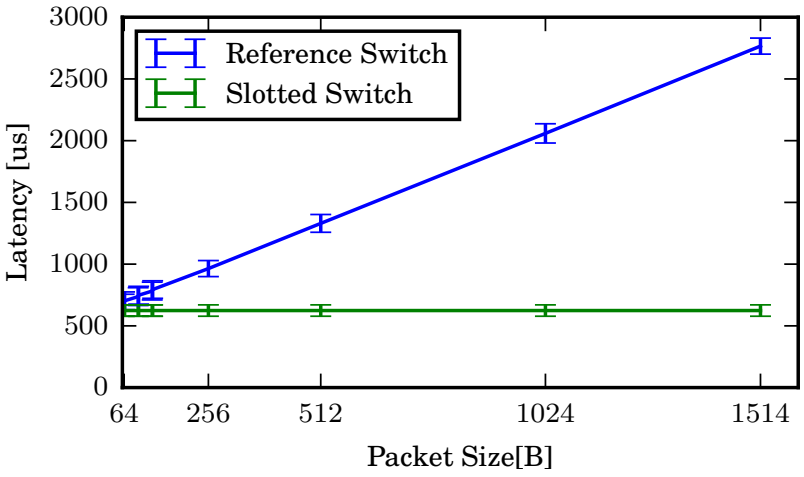


Figure 7: Measured Latency of Slotted Architecture (A) and Reference Switch, using $1 \times 10GE$ stimulus.

stalls. Even though a reply is not immediately returned, a new request can be issued while the previous request is still processed, and pipelined replies are returned to the requesting pipelines when ready. The scenario leading to a maximum latency in the output queues is similar to the first architecture (i.e., momentary congestion), but the latency is almost four times higher, a result of the data path being a quarter of the width. The latency through the arbiter at the end of the data-path is not sensitive to the load, maintaining a constant latency through the module.

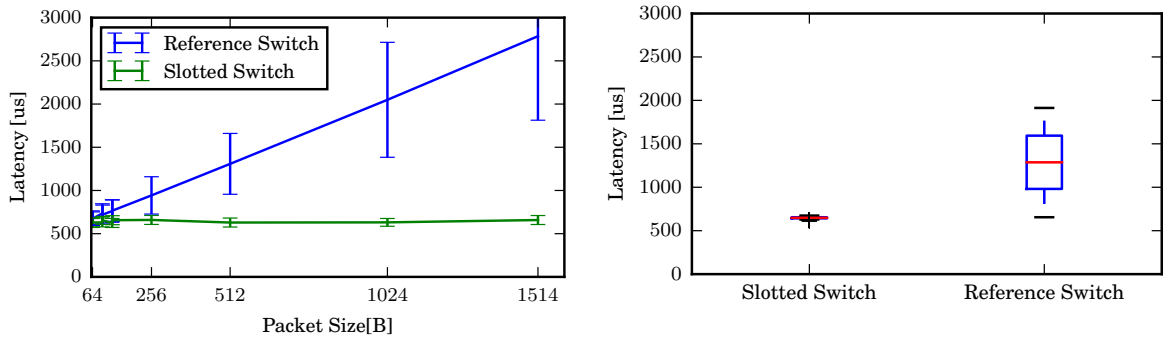
5.3 Measured Performance

The performance of the Slotted architecture (A) is evaluated in the lab. The NetFPGA SUME platform [19] is used for the prototyping of the architecture, using Vivado 2016.4 and running on Ubuntu 2016.4. An open source network tester (OSNT) [3] is used for traffic generation and capture. OSNT is also prototyped on NetFPGA SUME which allows it to send and receive traffic at line rate. Synthetic traffic (pcap files) are used for the testing, using a single packet size at the time, ranging from 64B to 1518B. Special attention is paid to non-aligned packet sizes, such as 65B, 69B, and 97B[5]. OSNT is connected to the switch under test using $4 \times 10GE$ ports. PCIe access is not tested as part of the experiments. The switch supports full line rate on all four ports, regardless of packet size. This is a non-trivial requirement of switches, as it shows not just bandwidth but also packet rate requirements being fulfilled.

To demonstrate the cut-through nature of the switch, Figure 7 shows the latency of different packet sizes traversing the NetFPGA Reference Switch and the cut-through switch, with only one active port. As the figure shows, the latency through the cut-through switch is independent of packet size, while in the reference design, a store-and-forward switch, the latency increases with packet size. Figure 8 repeats the same experiment, but now two ports are sending traffic into the switch: one is the monitored port, and the second serves as cross traffic (sent to a different destination port). As the figure shows, the cut-through switch is not affected by the cross traffic.

Last, we explore the latency variability as a function of cross traffic. In this experiment,

[5]The indicated packet sizes are 'on the wire', including FCS. 12 different packet sizes were evaluated.



(a) 2-Port cross traffic

(b) 4-Port cross traffic, 64B test stream

Figure 8: The effect of cross traffic on the latency through the Slotted Architecture (A) and Reference Switch. (a) using one 10GE cross-traffic port of equal-sized packets. (b) using $3 \times 10GE$ of 1514B packets and one monitored port of 64B packets.

1518B packets are injected at line rate traffic on three ports, each to a different destination port. A fourth port, the one being monitored, injects 64B packets at low rate. The intention of this experiment is to expose internal sensitivities and delays within the pipeline, and to corroborate the results presented in the previous subsections. Figure 8 shows that under these conditions the latency through the cut-through switch remains unchanged, with very little variance. This result is in contrast with the Reference Switch, where the median latency is not only more than twice, but also with variance of hundreds of nanoseconds. This experiment also nicely demonstrates the effect of cross traffic when put in comparison to Figure 8, where the port being monitored encountered cross traffic only from one other port, and using packets of the same size. The effect of cross traffic on the cut-through switch was small: the slot-arbitration delay between four active ports (a handful of cycles). The store-and-forward switch, on the other hand, experiences significant variance caused by the input arbiter: in some cases the monitored 64B packet is immediately admitted to the data-path, whereas in other cases it needs to wait for up to three other 1514B packets. The delay is also linearly dependant of the data-path clock frequency, thus the gain from increasing the data-path bus width and speed up is also reducing the latency variance.

6 Related Work

The concept of the cut-through switch was first introduced in the days where memory access was cheap (fast) and network communication was expensive (slow). The seminal work [15] tried to attend to the communication across multiple nodes, and the need to buffer a packet until it has arrived in its entirety. Instead Kermani and Kleinrock [15] suggested starting to transmit a packet to the next node as soon as the header has arrived and the output port was identified. While their work set a lot of the infrastructure to current day cut-through switches, many of its assumptions (e.g., instantaneous acknowledgements) are no longer valid on today's scales.

While in the past research describing the internals of cut-through switches ([9, 2, 10, 1] to name a few) was common, the last decade has seen little work published in this field. Iyer

et al. [14] work from 2008, discussing cut-through architecture within the switch as memory became slower than the network, is in fact a revisit of their 2002 Technical Report [13]. As cut-through switches became a commodity, innovation in cut-through switch architecture moved to companies such as Mellanox, Intel and Broadcom, who chose to cover their knowledge by patents [18, 6, 7, 16, 4, 11] rather than share it. With switch bandwidth increasing, time scales rapidly decreasing and physical limitations pushing the boundaries of switch design, it is important to keep exploring the considerations for cutting-edge switch designs.

7 Conclusion

Cut-through switches are commonly used in networks requiring high performance, such as data-centre networks. While many works assume cut-through switch operation, little is known of the architectures used in today's commodity switches. This paper presented three different approaches to the design of cut-through switches, adapted to today's technology, proposed by students as part of the High Performance Networking (P51) graduate course at the University of Cambridge. The architectures, analysed and evaluated, support full line rate and low latency regardless of packet size. Our work shows that while the architectures fit the requirements of a $4 \times 10G$ Ethernet switch, they are unlikely to scale to switches of 128 or 256 ports without penalty: packet rate, latency or significant resource allocation.

8 Acknowledgements

We would like to thank Xilinx, Cypress and Micron for their continuing support of the NetFPGA project, and Andrew W Moore for his feedback on this paper. We acknowledge support by the Leverhulme Trust (ECF-2016-289) and the Isaac Newton Trust.

References

[1] N. R. Adiga, G. Almási, G. S. Almasi, Y. Aridor, R. Barik, D. Beece, R. Bellofatto, G. Bhanot, R. Bickford, M. Blumrich, et al. An overview of the bluegene/l supercomputer. In *Supercomputing, ACM/IEEE 2002 Conference*, pages 60–60. IEEE, 2002.

[2] H. Ahmadi and W. E. Denzel. A survey of modern high-performance switching techniques. *IEEE Journal on Selected Areas in Communications*, 7(7):1091–1103, 1989.

[3] G. Antichi, M. Shahbaz, Y. Geng, N. Zilberman, A. Covington, M. Bruyere, N. McKeown, N. Feamster, B. Felderman, M. Blott, A. W. Moore, and P. Owezarski. OSNT: Open source network tester. *IEEE Network*, 28(5):6–12, 2014.

[4] S. Anubolu and M. V. Kalkunte. Scalable low-latency mesh interconnect for switch chips, Aug. 10 2017. US Patent App. 15/063,387.

[5] Barefoot Tofino. `https://www.barefootnetworks.com/products/brief-tofino/`, 2018.

[6] G. Bloch, D. Crupnicoff, M. Kagan, I. Bukspan, I. Rabenstein, A. Webman, and A. Marelli. High-performance adaptive routing, Nov. 5 2013. US Patent 8,576,715.

[7] G. Bloch, D. Crupnicoff, M. Kagan, I. Bukspan, A. Webman, and I. Rabenstein. Data switch with shared port buffers, Feb. 4 2014. US Patent 8,644,140.

[8] Broadcom. *BCM56980 series, 12.8 Tbps StrataXGS Tomahawk 3 Ethernet switch series*, dec 2017. `https://www.broadcom.com/products/ethernet-connectivity/switching/strataxgs/bcm56980-series`.

[9] W. J. Dally and C. L. Seitz. The torus routing chip. *Distributed computing*, 1(4):187–196, 1986.

[10] J. Duato, A. Robles, F. Silla, and R. Beivide. A comparison of router architectures for virtual cut-through and wormhole switching in a now environment. *Journal of Parallel and Distributed Computing*, 61(2):224–253, 2001.

[11] F. Gabbay, A. Marelli, A. Webman, and Z. Haramaty. Credit based low-latency arbitration with data transfer, Feb. 28 2017. US Patent 9,582,440.

[12] V. Gurevich. Barefoot networks, programmable data plane at terabit speeds. In *DXDD*. Open-NFP, 2016.

[13] S. Iyer, R. R. Kompella, and N. McKeown. Designing packet buffers for router line cards. *HPNG Technical Report-TR02-HPNG-031001, Stanford University*, 2002.

[14] S. Iyer, R. R. Kompella, and N. McKeown. Designing packet buffers for router linecards. *IEEE/ACM Transactions on Networking (ToN)*, 16(3):705–717, 2008.

[15] P. Kermani and L. Kleinrock. Virtual cut-through: A new computer communication switching technique. *Computer Networks (1976)*, 3(4):267–286, 1979.

[16] B. H. Kwan, P. Agarwal, M. Kalkunte, and N. Kucharewski III. Scalable, low latency, deep buffered switch architecture, Aug. 28 2014. US Patent App. 14/045,199.

[17] Mellanox. *Mellanox Spectrum Ethernet Switch*. `http://www.mellanox.com/page/products_dyn?product_family=218&mtag=spectrum_ic`.

[18] D. Olson, G. B. Lindahl, and J. B. Rubin. Cut-through decode and reliability, Feb. 15 2011. US Patent 7,889,749.

[19] N. Zilberman, Y. Audzevich, G. A. Covington, and A. W. Moore. NetFPGA SUME: Toward 100 Gbps as research commodity. *IEEE MICRO*, 34(5):32–41, Sept. 2014.

[20] N. Zilberman, M. Grosvenor, D. A. Popescu, N. Manihatty-Bojan, G. Antichi, M. Wójcik, and A. W. Moore. Where has my time gone? In *International Conference on Passive and Active Network Measurement*, pages 201–214. Springer, 2017.

[21] N. Zilberman, A. W. Moore, and J. A. Crowcroft. From photons to big-data applications: terminating terabits. *Phil. Trans. R. Soc. A*, 374(2062):20140445, 2016.