IN-NETWORK AGGREGATION FOR SHARED MACHINE LEARNING CLUSTERS

Nadeen Gebara [1] **Paolo Costa** [2] **Manya Ghobadi** [3]

ABSTRACT

We present PANAMA, a novel in-network aggregation framework for distributed machine learning (ML) training on shared clusters serving a variety of jobs. PANAMA comprises two key components: (i) a custom in-network hardware accelerator that can support floating-point gradient aggregation at line rate without compromising accuracy; and (ii) a lightweight load-balancing and congestion control protocol that exploits the unique communication patterns of ML data-parallel jobs to enable fair sharing of network resources across different jobs while ensuring high throughput for long-running jobs and low latency for short jobs and other latency-sensitive traffic. We evaluate the feasibility of PANAMA using an FPGA-based prototype with 10 Gbps transceivers and large-scale simulations. Our simulation results demonstrate that PANAMA decreases the average training time of large jobs by up to a factor of 1.34. More importantly, by drastically decreasing the load placed on the network by large data-parallel jobs, PANAMA provides significant benefits to non-aggregation flows too, especially latency-sensitive short flows, reducing their 99%-tile completion time by up to 4.5×.

1 INTRODUCTION

The ever-growing demand for accurate machine learning (ML) models has resulted in a steady increase in dataset and model sizes of deep neural networks (DNN). As a result, training modern DNN models goes well beyond the capabilities of a single device, and thousands of accelerators are required for training large models today (Huang et al., 2018; Lepikhin et al., 2020; Moritz et al., 2018; Sun et al., 2019).

Several academic institutions and companies have recently advocated the use of *in-network aggregation* (Costa et al., 2012; Mai et al., 2014) to improve the performance of distributed data-parallel ML training workloads (Bloch, 2019; Klenk et al., 2020; Li et al., 2019; Sapio et al., 2021). By aggregating gradients *inside* network switches rather than at end-hosts, the communication bottleneck of data-parallel training could be mitigated to reduce the training time.

Existing proposals, however, focus on relatively simplistic scenarios that limit in-network aggregation to a single switch (Lao et al., 2021; Sapio et al., 2021), a low switch radix (Li et al., 2019), or a single ML job. In practice, though, as the popularity and variety of neural network models grow, to reduce costs and resource wastage, both first-party and third-party training workloads are increasingly transitioning to shared clusters (Abadi et al., 2016; Amazon, 2021; Azure, 2021; Google, 2021; Jeon et al., 2019), spanning hundreds of racks and executing a multitude of different ML jobs. Those jobs include traditional ML jobs such as K-means clustering, as well as more recent incarnations such as reinforcement learning and deep learning. From a network perspective, this leads to a widely heterogeneous set of flows, ranging from a few KBs to tens of GBs in size (Abadi et al., 2016; Azure, 2021; Google, 2021; Jeon et al., 2019), of which only a fraction might require aggregation (§2). Without a proper mechanism to efficiently share network resources at large scale, the practical viability of in-network aggregation under more realistic settings becomes questionable as confirmed by anecdotal evidence in early deployments: e.g., when testing their in-network aggregation at larger scale, NVIDIA observed lower throughput than expected due to congestion (Klenk et al., 2020).

In this paper, we address this shortcoming by presenting PANAMA (*P*rogr*A*mmable Network *A*rchitecture for *M*L *A*pplications), an in-network aggregation framework tailored for shared environments with a heterogeneous set of workloads. PANAMA consists of two main components. The first is a new aggregation hardware accelerator, designed from the ground up to support multiple active training jobs concurrently (§5). While existing programmable switches based on the Reconfigurable Match Table (RMT) architecture (Bosshart et al., 2013), e.g., the Tofino switch (Intel, 2018), can be used for in-network aggregation (Lao et al., 2021; Sapio et al., 2021), the lack of floating point support and the inflexibility of their pipeline architecture (Gebara et al., 2020) forces compromises in accuracy and performance. In contrast, we opt for a *bump-in-the-wire* approach in which our in-network accelerator is decoupled from the switch. This design is inspired by recent cloud deployments

[1]Imperial College London [2]Microsoft Research [3]MIT. Correspondence to: Nadeen Gebara <n.gebara17@imperial.ac.uk>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021.

of programmable network interface cards (NICs) (Firestone et al., 2018) and provides maximum flexibility without sacrificing the training accuracy or requiring any changes to the switch logic. Our design can operate at line rate (10–100 Gbps) and makes efficient use of logic area.

The second component in PANAMA is a load-balancing mechanism in conjunction with a lightweight congestion control protocol that allows efficient and fair sharing of network bandwidth across different jobs (§4). PANAMA distributes the aggregation traffic across multiple *aggregation trees* to reduce network hot spots and enhance network performance. Existing data center congestion control protocols (Alizadeh et al., 2010; Kumar et al., 2020; Mittal et al., 2015; Zhu et al., 2015) are not suitable for in-network operations as they assume point-to-point connections between servers rather than a tree-based configuration. In contrast, our novel congestion control protocol takes advantage of the unique features (and opportunities) of in-network aggregation to improve performance while ensuring the limited buffer space on accelerators is shared fairly by ML jobs without being overflown. Moreover, our congestion control algorithm is compatible with existing ECN-based protocols used in data centers (Alizadeh et al., 2010; Zhu et al., 2015), thus enabling fair sharing of network resources with other, non-in-network aggregation traffic.

Our work sheds new light on the benefits of in-network aggregation that have gone unnoticed in prior work. The research community has expressed skepticism about the practical value of in-network aggregation; notably, its impact on the overall training time is limited, as gradient aggregation constitutes a small fraction of the entire training task (§2). We argue that, perhaps counter-intuitively, the real motivation for in-network aggregation is not so much to improve the performance of the training jobs themselves, as it is to reduce the volume of traffic generated by data-parallel gradient exchange. Aggregating traffic within the network drastically decreases the network usage (a single data-parallel job can generate more than 1 PB of data during its execution), thus freeing up network resources for other flows, including non-data-parallel ML jobs and latency-sensitive flows.

We demonstrate the feasibility of our hardware accelerator using an FPGA-based prototype (§6) and provide an extensive simulation analysis to evaluate the benefits of PANAMA load-balancing and congestion control at scale (§7). Our results show that PANAMA reduces the 99%-tile completion time of latency-sensitive transfers by up to 4.5× (resp. 2× for the mean) while simultaneously reducing the average training time of ML training jobs by up to 1.34×.

2 THE CASE FOR IN-NETWORK AGGREGATION

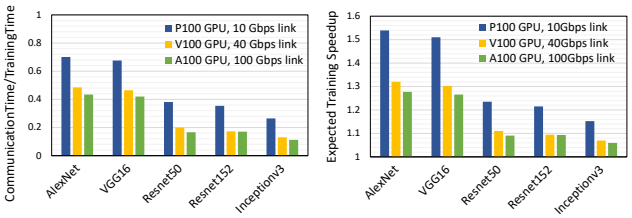
One of the most common distributed training approaches is *data-parallel* training in which the neural network is replicated across N workers (or replicas), with each worker processing a small subset of the training data (mini-batch) to compute its local model gradients. At every iteration, workers must synchronize their models by exchanging and aggregating their gradients to ensure convergence (Narayanan et al., 2019). This step is called *allreduce* and it can be implemented using a parameter server (Li et al., 2014) or Ring-AllReduce (Sergeev & Balso, 2018; Uber Eng., 2017).

The allreduce step places significant pressure on the network fabric, as the entire set of model gradients are exchanged many times throughout the training process. For example, for a training job with 1,000 workers and a 1 GB DNN model size requiring 1,000 iterations required to achieve target accuracy, the total traffic generated would be around 2 PB, as gradient exchange involves sending gradients, as well as receiving their aggregated values. Recent proposals advocate the use of *in-network aggregation* to improve the performance of ML training workloads (Bloch, 2019; Klenk et al., 2020; Lao et al., 2021; Li et al., 2019; Sapio et al., 2021). The intuition is that reducing the amount of data transferred over the network during the allreduce step will shorten the communication time and lead to faster training.

However, analytical results show the overall training time improvement resulting from in-network aggregation is limited to 1.01-1.8× (Table 2 in (Sapio et al., 2021)). There are two reasons for this. First, compute time occupies a significant chunk of the overall training time. Therefore, only a fraction of the overall training time is spent on communication, and this bounds the maximum achievable gain. Second, even if the communication time comprised the entirety of the training time, the maximum theoretical improvement in-network aggregation could achieve compared to the state-of-the-art Ring-AllReduce strategy is a factor of two reduction in communication time for large message sizes (Klenk et al., 2020).

To validate this observation, we train five popular image classifications models, using three generations of NVIDIA GPUs: Pascal (P100), Volta (V100), and the recently introduced Ampere (A100). We choose these DNN models as they cover a wide range of sizes and computation requirements. We train the models using TensorFlow framework and ImageNet dataset with batch sizes specified in the TensorFlow benchmark suite (TensorFlow, 2021).

To avoid any measurement bias caused by inefficiencies in distributed training frameworks or network stacks, we measure the *computation* time per iteration by training the model on a single-GPU node, and estimate the *communica-*



(a) Communication ratio (b) Expected speedup

Figure 1: The expected speed-up for a single in-network aggregation job is limited by the ratio of communication time over total training time.

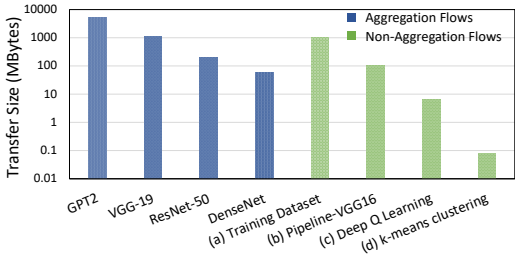


Figure 2: Data transfer sizes in a shared ML cluster.

tion time by dividing the size of the gradients exchanged in a single iteration by the link bandwidth. For consistency in different GPU generations, we assume link bandwidths of 10 Gbps, 40 Gbps, and 100 Gbps for P100, V100, and A100, respectively.

Fig. 1a reports the ratio of communication time to total training time (communication + computation) across different DNNs, GPU generations, and network bandwidths. The figure shows the fraction of training time spent on communication ranges between 0.11 and 0.70, and as the network becomes faster, this fraction is reduced. This suggests the benefits of in-network aggregation for data-parallel jobs are also diminishing with time. Fig. 1b illustrates this takeaway by showing the expected training time speedup of in-network aggregation for A100 GPUs with 100 Gbps links ranges between 1.06 and 1.28 (resp. 1.15 and 1.53 for P100 with 10 Gbps links). In fact, in this analysis, we assume an optimal case for in-network aggregation in which there is no overlap between gradient computation and communication. This is very conservative, as modern distributed ML frameworks exploit overlap to minimize the impact of communication on the training time; hence, we expect the benefits of reducing communication to be less pronounced.

In this paper, instead, we argue that in-network aggregation has a real opportunity to enhance the performance of non-data-parallel jobs in shared clusters by reducing the overall data-parallel traffic injected into the network, thus freeing up network bandwidth for the other traffic. Shared ML clusters comprise a heterogeneous set of flows with transfer sizes that range from a few KBs to hundreds of GBs, as shown in Fig. 2. The left part of the figure (blue bars) shows transfer sizes for data-parallel flows containing model gradients. We refer to these transfers as *aggregation flows* to indicate they are good candidates for in-network aggregation. The right side of the figure (green bars) shows flows that are poor candidates for in-network aggregation, *non-aggregation flows*. These include flows that don't require any aggregation, e.g., (a) dataset transfers and (b) flows generated by pipeline parallelism (Huang et al., 2018; Narayanan et al., 2019) or model parallelism (Shoeybi et al., 2020), as well as short flows that require aggregation but for which in-network aggregation has no pronounced benefits, e.g., (c) flows generated by reinforcement learning training *(RL)* (Li et al., 2019; Moritz et al., 2018) and (d) more traditional ML jobs, such as k-means clustering (Rossi & Durut, 2011).

We show that while non-aggregation flows are not suitable candidates for in-network aggregation, they benefit indirectly from in-network aggregation. Existing in-network aggregation solutions, however, are not designed to operate in shared environments. As we show in §7, today's proposals do not have appropriate load-balancing and congestion control mechanisms to cope with multi-tenant environments where the data center is shared between aggregation and non-aggregation flows. In contrast, with its combination of native hardware support for multiple ML jobs and a congestion control protocol tailored for in-network aggregation, PANAMA closely approximates the ideal performance for non-aggregation flows even in the presence of a large number of aggregation jobs, while at the same time, reducing the training time of the aggregation jobs themselves.

3 PANAMA OVERVIEW

In this section we provide a high-level description of PANAMA and we detail its key components in §4 and §5. We assume a traditional data center multi-tier folded Clos topology (Al-Fares et al., 2008; Singh et al., 2015) similar to the one depicted in Fig. 3. Each switch in a PANAMA cluster (PSwitch) comprises a traditional switch, e.g., a Broadcom Tomahawk (Broadcom, 2020), connected with our bump-in-the-wire aggregation accelerator.

❶ **Worker placement.** When a new ML training job is submitted, the data center scheduler determines the optimal distributed training strategy (Jia et al., 2019; Narayanan et al., 2019; Sergeev & Balso, 2018) and instantiates the job on a set of worker nodes running directly on servers in the cluster or virtual machines (VMs) within them. These workers can be co-located in the same rack or distributed across multiple racks: PANAMA makes no assumptions on the placement of workers. The choice of flows that should use in-network aggregation (aggregation flows) is based on the operator's preference and can be configured in the

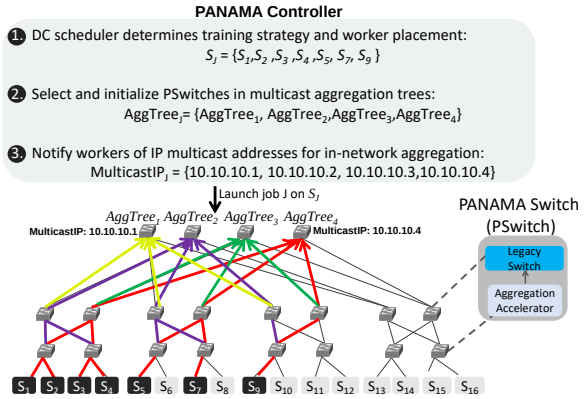


Figure 3: High-level workflow of PANAMA.

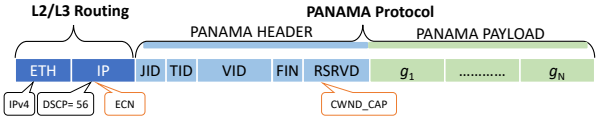


Figure 4: PANAMA aggregation packet format.

controller logic. In our experiments, we mark allreduce data-parallel flows with a size larger than 40 MB as aggregation traffic. We select this flow size threshold empirically as negligible benefits were observed with shorter flows.

❷ PSwitch selection and initialization. When the PANAMA controller selects in-network aggregation for a job, it initializes all the PSwitches belonging to the spanning trees connecting the workers. These trees could include a single Top-of-the-Rack (ToR) switch if all workers are in the same rack or multiple tiers of switching if workers are distributed across racks (see Fig. 3). PANAMA exploits the multi-path connectivity of modern data centers by distributing a job's aggregation traffic across multiple trees, resulting in higher network efficiency and lower congestion (§4.1).

The PANAMA controller generates a unique IP multicast address for each aggregation tree and installs forwarding entries within PSwitches along the tree: in the upstream direction, PSwitches are configured to forward aggregation packets towards aggregation tree roots, while in the downstream direction, aggregation packets are routed back to the worker nodes using native IP multicast support (if available) or by adding individual entries in forwarding tables. The PANAMA controller also initializes our in-network aggregation accelerators on path with the state of the job and the information needed to perform the aggregation (§5).

❸ Worker setup. Finally, the PANAMA controller configures the selected workers to use in-network aggregation and notifies them of the IP multicast address of the selected aggregation trees. PANAMA does not require any specific hardware support on the servers. Our PANAMA communication library can replace communication libraries, e.g., NCCL and MPI (NVIDIA, 2021; The Open MPI Project, 2021), used by mainstream ML frameworks. Our library encapsulates the gradients into a packet format supported by our accelerator (see Fig. 4).

4 NETWORK DESIGN

In this section, we describe how PANAMA addresses the challenge of running in-network aggregation in a shared environment comprising both aggregation and non-aggregation flows (see Fig. 2). This requires resolving two issues. The first is how to load-balance aggregation traffic in a multi-tier data center to efficiently use network resources and minimize congestion with non-aggregation traffic. PANAMA addresses this by exploiting multiple aggregation trees (§4.1). The second is how to fairly share the network bandwidth between aggregation and non-aggregation flows without overflowing the hardware accelerators' buffers. To this end, we propose an ECN-based congestion control algorithm with a congestion window cap for aggregation packets and PAUSE frames for the entire fabric to ensure deterministic lossless operation, even in the presence of congestion (§4.2).

4.1 Routing and Load Balancing

Today's data center networks often rely on the equal-cost multi-path (ECMP) protocol to balance the load in the network and ensure all packets belonging to a given TCP flow are routed through the same network path, thus avoiding out-of-order arrivals at the receiver (Hopps, 2000). Since the majority of data center flows are short (Alizadeh et al., 2010; 2013; Greenberg et al., 2009), ECMP is usually sufficient to ensure traffic is reasonably spread across the network. However, aggregation flows are typically very large, and bounding such flows to a single path (aggregation tree) as dictated by ECMP could create a significant network imbalance. This would be particularly harmful to latency-sensitive short flows competing for bandwidth; they would suffer increased queuing delays. To avoid this, PANAMA utilizes multiple aggregation trees per training job to spread the traffic across multiple paths and avoid congestion hotspots.

As described previously, the PANAMA controller provides the workers with the set of IP multicast addresses representing the selected aggregation trees for a job. Workers distribute the gradient packets to different trees in a round robin fashion. For instance, in the topology in Fig. 3, with four aggregation trees $AggTree_i$, $i = 1, \ldots, 4$ and eight aggregation packets with IDs p_j, $j = 1, \ldots, 8$, assuming workers start by sending a single packet to each of the trees, our protocol balances the aggregation load as follows: $\{p_1, p_5\} \rightarrow AggTree_1$; $\{p_2, p_6\} \rightarrow AggTree_2$; $\{p_3, p_7\} \rightarrow AggTree_3$; $\{p_4, p_8\} \rightarrow AggTree_4$, where $\rightarrow$

indicates the aggregation tree to which each set of packets is destined; e.g., p_1 and p_5 are sent to $AggTree_1$. Note that the packet numbers also reflect the sending order; p_5 is only sent after the preceding packets have been sent to other trees due to the round-robin ordering. This mechanism balances the load across trees, and as the order is deterministic, no coordination among workers is required. Non-aggregation traffic is not impacted by this and it is forwarded using the operator-defined load-balancing protocol.

4.2 Congestion Control

In this section, we describe the PANAMA congestion control protocol. We first outline the requirements of a congestion control protocol for aggregation traffic and then explain how our proposed protocol satisfies these requirements.

4.2.1 Requirements

R_1 Support for multipoint communication. Traditional congestion protocols assume unicast, point-to-point communication between *(source,destination)* pairs. In contrast, in-network aggregation involves many-to-many communication between different entities. Therefore, typical rate-limiting mechanisms, e.g., based on packet loss or congestion notifications, are not directly applicable. One naive solution would be to implement congestion control in the PSwitches, replacing the tree-like multipoint flow with a sequence of hop-by-hop flows. This, however, would make inefficient use of precious chip area.

R_2 Small buffers. Since our hardware accelerator needs to operate at line rate across hundreds of ports, we cannot rely on external memory, e.g., DRAM, and we are restricted to on-chip buffering. On-chip memories consume significant chip area making it critical to limit the size of on-chip memory and use the limited memory efficiently by sharing it fairly across multiple jobs. A key difference from traditional networks is that buffers are not only needed because of congestion but also because of the need to store aggregation packets until aggregation packets from all workers arrive at the switch. This introduces dependencies among flows. As the result can only be computed after receiving all packets, the sending rate of workers must match; otherwise, the packets originating from faster transmitters must be buffered until the packets from the slowest transmitter are received, thus wasting resources that could be used for other ML jobs. This property differentiates in-network aggregation from the apparently similar *co-flows* abstraction (Chowdhury & Stoica, 2012) where individual flows can use different rates.

R_3 Compatibility with legacy protocols. A key requirement for our protocol is the ability to co-exist with mainstream congestion control protocols. In particular, the protocol should be TCP-friendly, as TCP is the de-facto standard congestion control protocol in data centers. Using weighted fair queues (WFQs) at switches to separate aggregation and non-aggregation flows might appear as a simple solution, but this is not the case; it fails to provide fair sharing across aggregation flows belonging to different jobs, and it also involves the complex task of dynamically selecting the optimal weights assigned to each of the two traffic classes.

Parameters: N: number of job aggregation trees, *ssthresh*: initial slow start threshold, g: weighting factor for fraction of ECN marked result packets, α: moving average of ECN marked fraction of packets.

Initialization ▷ Independent aggregation tree congestion control
for $i = 1 : N$ **do**
  $ssthresh_i \leftarrow 64$
  $\alpha_i \leftarrow 1$
  $cwnd_i \leftarrow 2$
end for

Input: Aggregation Result Packet (pkt) ▷ Implicit acknowledgment
  $i \leftarrow pkt.treeid$
  $rcvd\_agg\_packets_i \leftarrow rcvd\_agg\_packets_i + 1$
  $ecn\_count_i \leftarrow ecn\_count_i + pkt.ecn$ ▷ ECN marking
if $rcvd\_agg\_packets_i == cwnd_i$ **then**
  $\alpha_i \leftarrow \alpha_i(1-g) + g \times \frac{ecn\_count_i}{cwnd_i}$
  $rcvd\_agg\_packets_i = 0$
  if $ecn\_count_i == 0$ **then** ø Window size increase
    if $cwnd_i < ssthresh_i$ **then**
      $cwnd_i \leftarrow 2 \times cwnd_i$
    else
      $cwnd_i \leftarrow cwnd_i + 1$
    end if
  else ø Window size decrease
    $cwnd_i \leftarrow cwnd_i \times (1 - \frac{\alpha_i}{2})$
    $ssthresh_i \leftarrow cwnd_i$
  end if
end if
if $cwnd_i > pkt.cwnd\_cap$ **then**
  $cwnd_i \leftarrow pkt.cwnd\_cap$ ▷ Congestion window capping
end if

Figure 5: PANAMA congestion control algorithm.

R_4 Lossless operation. Unlike traditional TCP point-to-point flows, if an aggregation packet is lost in PANAMA, several packets need to be retransmitted, and this could drastically reduce the overall throughput. Therefore, it is critical to ensure buffers are never overflown, even under high network loads.

4.2.2 Design

We now discuss the key features of PANAMA's congestion control protocol and explain how they meet the aforementioned requirements. We provide the protocol's pseudo-code in Fig. 5. The protocol is part of the PANAMA communication library at the end-hosts; its use does not require any changes to the training framework.

Implicit acknowledgments. Existing data center congestion control mechanisms use signals from network switches (such as packet loss or ECN marks) and end-hosts (such as RTT) to detect the onset of congestion and adjust the sending rate of packets at the sender (Alizadeh et al., 2010; Dong et al., 2015; Ha et al., 2008; Handley et al., 2017; Mittal et al., 2015; Zhu et al., 2015). This mechanism cannot

be replicated for in-network aggregation, however, because new packets are constructed inside the network at each switch by aggregating several incoming packets into one. This disrupts the one-to-one mapping between packets and their corresponding acknowledgements. Our design, instead, takes advantage of the unique properties of the in-network aggregation operation. As the number of aggregation results is equal to the number of locally computed gradients, each worker expects a result packet for each aggregation packet sent. Therefore, workers can treat aggregation result packets as *implicit* acknowledgement signals to increase the window size as shown in Fig. 5. This overcomes the need to maintain a per-flow congestion state at PSwitches (R_1). Further, our congestion control operates independently on each aggregation tree. This avoids the need to re-order packets across multiple paths and, when combined with our load-balancing protocol (4.1), it can provide the benefits of multi-path congestion control protocols (Peng et al., 2013) without the additional complexity.

ECN marking. Our congestion control protocol is inspired by DCTCP (Alizadeh et al., 2010) and relies on ECN marks in the IP header to react to the observed network congestion. In PANAMA, we extend this mechanism to enable aggregation job rate synchronization across workers. A distinctive feature of in-network aggregation is that as packets move upwards in an aggregation tree, PSwitches must aggregate the gradients in multiple packets and produce an aggregation result packet. This can result in the loss of information on the state of network congestion. In PANAMA, however, aggregation accelerators within PSwitches retain ECN field information of aggregation packets: each hardware accelerator performs a bitwise OR operation on the ECN field of packets received to mirror the ECN bit into the IP header of generated aggregation packet (see Fig. 4). As a result, the aggregation packet will carry the ECN bit back to all the workers. Unlike traditional ECN-based congestion control schemes, there is no need to echo the ECN back to the senders because results packets are used as implicit acknowledgments. Workers inspect the result packets and if the ECN bit is set, they adjust the sending rate as detailed in Fig. 5. This mechanism ensures the congestion window for each aggregation tree grows and shrinks in a synchronized fashion across workers in the aggregation tree (R_2). Further, since the congestion control mechanism matches that of DCTCP, we can guarantee compatibility with existing legacy protocols as we show in our evaluation in §7 (R_3).

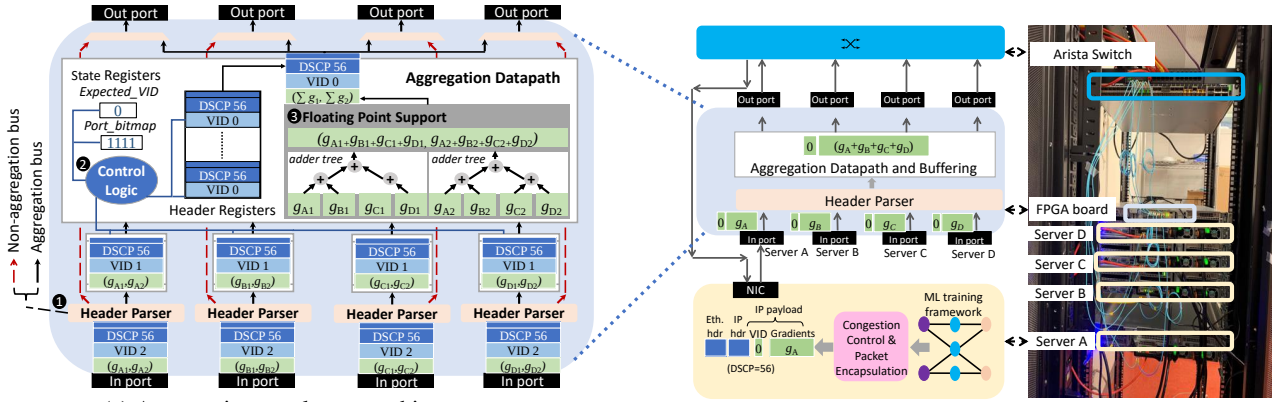
Congestion window capping. To avoid packet loss due to accelerator buffer overflow, PANAMA's congestion protocol caps the congestion window size of the training worker to match the minimum available buffer space in the accelerators of each aggregation tree. This ensures incoming packets are always accommodated and not dropped because of lack of available buffer space (R_4). To maintain an up-to-date view of the available buffer space, the hardware accelerators update each aggregation packet using a field called `cwnd_cap`. We reserve 16 bits for `cwnd_cap` in the packet to capture the minimum available memory to store packets at the accelerators as the aggregation packet makes its way up to the root of the aggregation tree. Each accelerator calculates its available memory based on the number of active training jobs ($packet\_memory/num\_jobs$) and overwrites `cwnd_cap` if its available buffer space is smaller than the `cwnd_cap` of the received aggregation packets; otherwise it retains the minimum `cwnd_cap` value. The final value is then sent to all the workers, along with the gradient aggregation result. As described above, the arrival of an aggregation packet is treated as an acknowledgement signal enabling the workers to send the next set of allowed in-flight packets. The value in `cwnd_cap` is used as a cap on the maximum number of in-flight packets for each worker, similar to the way TCP negotiates window size (packets are assumed to have a fixed size when created by the workers). We rely on standard Ethernet flow control, using PAUSE frames, to ensure the in-network accelerators never overflow the switch buffers, thus resulting in an end-to-end lossless architecture. Packet loss is still possible in cases of packet corruption or failures (Zhuo et al., 2017), but a simple timeout mechanism can be used to handle this. Due to the lossless property of our protocol, the timeout value does not need to be set aggressively, thus preventing spurious re-transmissions.

5 AGGREGATION ACCELERATOR DESIGN

Next, we describe the architecture of our hardware accelerator used to support floating point line-rate aggregation in the PSwitch (see Fig. 6a).

❶ **Packet header parser.** A parser module at each input port inspects the `EtherType` and `DSCP` fields (Fig. 4) of incoming packets to separate aggregation packets from the rest of the traffic. IPv4 packets with a `DSCP` field value equal to 56 are recognized as aggregation packets. These packets are sent to dedicated aggregation buffers, while other packets are forwarded directly to the switching chip. This ensures non-aggregation packets do not suffer from head-of-line blocking due to aggregation packets and only experience minimal delays when traversing the accelerator.

❷ **Control logic.** Accelerators maintain the following states for jobs: *Ports_bitmap* and *Expected_VID*. The *Ports_bitmap* register is set by the PANAMA controller when a job's aggregation tree is configured; it identifies the accelerator input ports included in the aggregation. The *Expected_VID* register is used for correct aggregation and corruption-induced loss detection and is initialized to 0. Accelerators rely on the Job ID (`JID`) and Tree ID (`TID`) fields shown in Fig. 4



(a) Aggregation accelerator architecture.

(b) FPGA-based prototype.

Figure 6: Details of PANAMA hardware design.

to identify the *Ports_bitmap* and *Expected_VID* that must be used when performing an aggregation task. In practice, a *Ports_bitmap* must be maintained for each job, and an *Expected_VID* must be maintained for each aggregation tree within a job. We illustrate the accelerator's aggregation control logic using the example in Fig. 6a for a single job and a single aggregation tree. Therefore, one *Ports_bitmap* register and one *Expected_VID* register are maintained. Since the *Ports_bitmap* register is set to 1111, the controller must wait until at least one packet is available at each of the four input ports. When this requirement is met, the packets' headers are copied into header registers, and their `VID` fields are compared against the value of 0 in the *Expected_VID* register. The `VID` field on packets acts as a gradient identifier, because workers encapsulate the same set of gradients in packets with the same `VIDs`. It also acts as a packet sequence number because workers assign incremental `VIDs` to packets sent in an aggregation tree. Since aggregation trees guarantee in-order delivery, the accelerator uses the *Expected_VID* register to keep track of the expected packets. Therefore, this comparison ensures correct gradient aggregation and allows losses caused by packet corruption to be detected. When all the `VID` fields match the current value of the *Expected_VID* register, as shown in Fig. 6a, the gradients are streamed through the *adder trees* within the aggregation datapath, and the *Expected_VID* register is incremented. Otherwise, gradients with missing values are dropped and aggregation proceeds with the next *Expected_VID*. Workers notify accelerators that they have sent all their aggregation packets by setting the `FIN` bit in the packet header, and this resets the state in the accelerator.

❸ **Floating point (FP) support.** A key challenge in making in-network aggregation practical is supporting floating point aggregation at today's data rates (10 Gbps and 40 Gbps) and future ones (100 Gbps). Our design achieves this goal by *tuning the width of the bus* that transfers data from each input port to each output port to match the desired port rate assuming a certain clock rate (see Fig. 11 in Appendix A). The aggregation packet payload is streamed from each dedicated buffer participating in a job to multiple adder trees. The number of parallel adder trees is proportional to the number of gradients that can be carried in the data bus. As depicted in Fig. 6a, this allows a *SIMD* architecture in which the gradients are partitioned across the two adder trees. The adder trees operate in parallel and their results are concatenated and sent to the output ports.

6 FPGA-BASED PROTOTYPE

We evaluate the feasibility of our aggregation accelerator by implementing it on a NetFPGA-SUME board (Zilberman et al., 2014) equipped with a Xilinx Virtex-7 FPGA and four 10- Gbps transceivers. We incorporate the Xilinx's LogiCORE IP core (Xilinx, 2014) into our design to support floating point addition (fully compatible with the IEEE-754 single-precision standard). The LogiCORE IP core limits our maximum clock rate to 220 MHz. We therefore instantiate two replicas of the adder tree to meet the required line rate (10 Gbps), and opted for a clock rate of 200 MHz. Table 1 summarizes the accelerator's cut-through latency observed by non-aggregation packets and aggregation packets. As shown, the latency introduced by the accelerator is minimal, even to aggregation packets. We also measure the resource utilization targeting the recent VU19P FPGA board (Xilinx, 2021) (see Table 2 in Appendix A). Results show our design has a small resource footprint, using only 1% and 0.26% of available Lookup-Tables (LUTs) of Flip-Flops (FFs) respectively. This is important because it demonstrates our design can easily fit onto a small chip, e.g., opening up the possibility of co-packaging it with the main switching die using a chiplet design. Further, it shows there is room to increase parallelism by instantiating more adder trees to sustain higher data rates. Our preliminary analysis suggests that our design could scale to more than 100 ports

Packet Size (Bytes)	Packet Latency (ns)	
	Non-Aggregation	*Aggregation*
512	340	370
1024	665	695
1500	957.5	987.5

Table 1: Prototype cut-through latency.

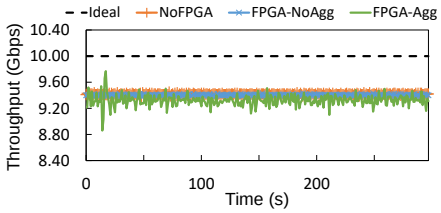


Figure 7: Prototype aggregation throughput.

with 100 Gbps per port using an FPGA and 400 Gbps per port through an ASIC implementation (see Appendix A).

Prototype evaluation setup. We build a PANAMA testbed with four Dell R740 dual-core PowerEdge servers, 10 Gbps optical transceivers, and an Arista switch 7050S. We connect our FPGA board between the servers and the switch using a bump-in-the-wire configuration as illustrated in Fig. 6b.

Functional correctness. We assess the correctness of our architecture by using the *libtins* library (libtins, 2021) to craft PANAMA packets, as shown in Fig. 4. Randomly-generated floating point numbers emulating gradients generated by ML jobs are encapsulated in PANAMA packets and sent to the PSwitch for aggregation. In all experiments, we observe correct aggregation values in all received packets.

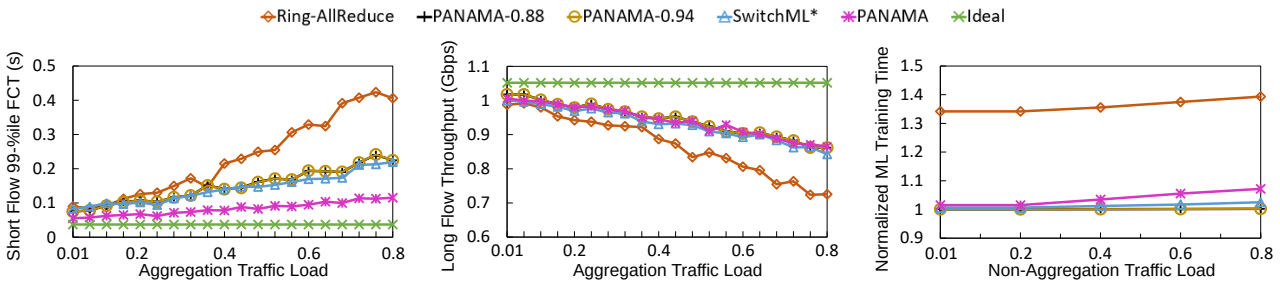
Throughput performance. We first measure the maximum throughput that can be achieved by two servers when they are directly interconnected through the switch without any intermediate FPGA (*NoFPGA*). We use four parallel `iperf` instances to ensure the experiments are not CPU-bound. We then connect the bump-in-the-wire FPGA to the switch emulating our PSwitch architecture (§3) and run two additional experiments using the same workload to measure the throughput of *FPGA-noagg* and *FPGA-agg* paths. For compatibility with iperf, we modify our control logic such that a single input traverses the aggregation path; we initialize the value in the *Port_bitmap* register to `0001`, and we connect the packet-generating server to the first port of the accelerator. We use UDP packets instead of TCP packets, because in the aggregation process of the latter, the packet payload would be modified by the aggregation, and, hence, the TCP checksum would fail. Fig. 7 shows the *FPGA-noagg* throughput closely matches the *NoFPGA*, indicating that the overhead introduced by the FPGA is negligible for non-aggregation packets. However, in the case of *FPGA-agg*, the throughput exhibits a slightly higher variability (oscillating from 8.86 Gbps to 10.48 Gbps). This is a consequence of using UDP rather than TCP, as UDP results in traffic spikes. Nonetheless, the average throughput is consistent with the previous experiments, confirming our prototype's ability to support aggregation at 10 Gbps.

7 LARGE-SCALE SIMULATIONS

In this section, we evaluate the performance of PANAMA at scale using a customized version of the `OMNeT++` packet-level network simulator (Ltd., 2021). The main takeaways of our simulation analysis are: (i) PANAMA reduces the 99-%tile flow completion time (FCT) of short flows (<40 MB) up to a factor of 4.5, improves the throughput of long flows up to a factor of 1.33, and speeds up the training time of aggregation traffic by a factor of 1.34 compared to state-of-the-art Ring-AllReduce. (ii) PANAMA's higher performance is a product of its ability to reduce the volume of data transferred over the network because of in-network aggregation, as well as its ability to control the packet send rate of workers during congestion periods. We show PANAMA outperforms a baseline without our congestion control protocol by a factor of 3.5, reducing the 99-%tile FCT of short flows. (iii) PANAMA's multi-tree aggregation technique can balance the aggregation traffic's load on network paths. (iv) PANAMA's congestion control algorithm provides a fair bandwidth allocation across all flows.

Methodology and setup. In our experiments, we assume a non-blocking folded Clos network topology (Al-Fares et al., 2008) comprising 1,024 servers interconnected with 10 Gbps links. We set the ECN marking threshold to 85 packets and limit the buffering capacity of aggregation accelerators to 64 MB in accordance with the maximum buffering capacity of state-of-the-art FPGAs (Intel, 2016; Xilinx, 2021). We assume our workload consists of a mix of aggregation traffic and non-aggregation flows of varying sizes taken from Websearch and Datamining traffic distributions (Alizadeh et al., 2013; Greenberg et al., 2009). Flow inter-arrival times are drawn using an exponential distribution (Poisson process), and we vary the mean inter-arrival time to model different network loads. For aggregation traffic, the number of workers assigned to each job is chosen randomly, and ranges from 16 to 96, while the DNN models are chosen out of six prominent image classification models (VGG16, AlexNet,Resnet152, Resnet50, Inceptionv3, and GoogleNet) using a weighted random distribution To model the computation time, we use the values measured with P100 GPUs in our experiments in §2.

Scheduling jobs. Our job scheduler places the workers belonging to the same job as close as possible to ensure the best baseline performance. The source and destination of non-aggregation flows are chosen uniformly (Alizadeh et al., 2013). We use DCTCP (Alizadeh et al., 2010) as the



(a) 99%-tile FCT for non-aggregation short flows (<40 MB)

(b) Average throughput for non-aggregation long flows

(c) Average training time for ML jobs for increasing network load

Figure 8: Performance of PANAMA in a shared data center setting compared to other training schemes.

default transport layer protocol for non-aggregation flows and PANAMA congestion control for aggregation flows.

Configurations. We consider five network configurations: (1) *Ideal*: an ideal setting where aggregation flows, short flows, and long flows are completely separated and served in their respective dedicated clusters without any sharing of resources between them. (2) *Ring-AllReduce*: the state-of-the-art technique for distributed training used in Horovod (Sergeev & Balso, 2018). (3) *SwitchML**: an augmented version of a recent Tofino-based in-network aggregation proposal (Sapio et al., 2021). Our augmentation enables SwitchML to support multi-tenancy, job sharing, and load-balancing. As in its original implementation, however, it does not a congestion control mechanism. (4) PANAMA: our proposal. (5) PANAMA-*0.88 and* PANAMA-*0.94*: two partially synchronized versions of PANAMA. The numbers 0.88 and 0.94 represent the fraction of aggregation workers that PANAMA synchronizes at each iteration. These two configurations are useful to assess the impact of ignoring the slowest links and proceeding with the aggregation as soon as 88% (resp. 94%) of the packets from the worker nodes have been received.

PANAMA reduces the impact of aggregation traffic on short flows (<40 MB). We start with a baseline network: 20% load consisting of non-aggregation traffic. We introduce aggregation traffic into the network by slowly increasing the frequency of DNN training job arrivals up to 80% load. Fig. 8a shows that as the aggregation load grows, the 99%-tile FCT for short flows using the conventional Ring-AllReduce approach significantly increases. In contrast, PANAMA mitigates the impact of congestion and follows the Ideal line closely, even at high loads. At the highest load (80%), PANAMA reduces the FCT by a factor of 4.5 compared to Ring-AllReduce. PANAMA's gains come from in-network aggregation combined with congestion control and load-balancing. SwitchML* is able to reduce the FCT, as it performs in-network aggregation and is further coupled with load-balancing. However, it cannot match PANAMA

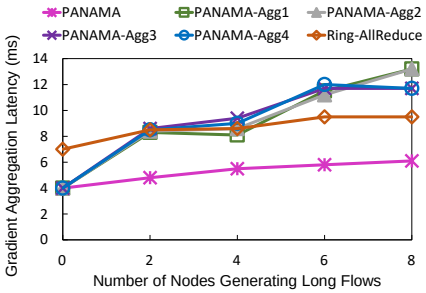
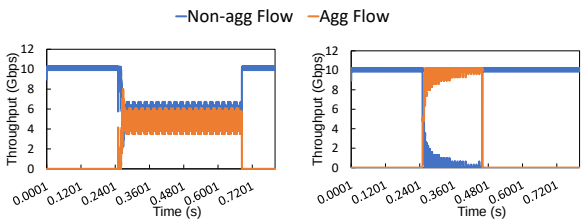


Figure 9: Impact of load balancing.

because its lack of congestion control gives an unfair advantage to aggregation flows at the expense of short flows' FCT. Similarly, PANAMA-0.88 and 0.94 partial aggregations are close to SwitchML*, as they do not slow the workers down enough to match the congested link.

In-network schemes improve throughput of long flows. Next, we investigate the impact on long flows of increasing aggregation load. The results in Fig. 8b indicate that PANAMA is able to improve the throughput of long flows up to a factor of 1.33 compared to Ring-AllReduce. However, the improvement matches that of the other in-network schemes because long flows and aggregation flows are large (>40 MB) and can therefore fully utilize the additional bandwidth available.

In-network schemes improve ML job completion time. PANAMA improves the training time of ML jobs by a factor of 1.34x over the baseline Ring-AllReduce approach and is only slightly outperformed by SwitchML* by a factor of 1.05, as shown in Fig. 8c. SwitchML* and other PANAMA-variants outperform PANAMA as they ignore congestion within the network at the expense of short flows. However, the training time measured for these approaches does not consider potential accuracy loss (or increase in the number of iterations) incurred due to lack of support for floating point operations in SwitchML*, or the impact of ignoring some gradients in PANAMA-0.88 and PANAMA-0.94.



(a) With PANAMA (b) Without PANAMA

Figure 10: PANAMA achieves a fair bandwidth allocation between aggregation and non-aggregation flows.

Impact of load-balancing. To determine the importance of load-balancing in-network traffic, we create the topology illustrated in Fig. 3 and run one aggregation training job with eight workers. We increase the network load by increasing the number of nodes generating long flows. Unlike previous experiments where we consider training job completion time, in this experiment we only consider the time taken to aggregate gradients computed within an iteration (*aggregation latency*). Fig. 9 shows the aggregation latency for six scenarios: PANAMA, PANAMA with only one aggregation tree (i.e., four scenarios labeled PANAMA-Agg1,..., PANAMA-Agg4), and Ring-AllReduce. As shown, PANAMA outperforms all other scenarios since it uses all four aggregation trees. Interestingly, the performance of in-network aggregation with a single aggregation tree can be worse than that of Ring-AllReduce. Unlike end-host paths that can be diversified via routing techniques such as ECMP or packet spraying, paths from each worker to a PSwitch are unique; if the load is not properly balanced, the aggregation time will be severely affected.

Fairness. To demonstrate that PANAMA's congestion control mechanism achieves a fair rate allocation across flows, we set up an experiment in which a bottleneck link is shared between an aggregation flow and a latency-sensitive non-aggregation flow. We begin with the non-aggregation flow; after 0.25 seconds, we start the aggregation flow. Fig. 10a shows PANAMA shares the link bandwidth equally between both flows. In contrast, Fig. 10b shows that without PANAMA's congestion control protocol, the latency-sensitive non-aggregation flow is starved.

8 RELATED WORK

Our work is closely related to SwitchML (Sapio et al., 2021) and ATP (Lao et al., 2021). Both approaches use commercially available programmable switches (Intel, 2018) to perform gradient aggregation. Although using programmable switches simplifies deployment, it presents two significant limitations. First, today's programmable switches only support fixed-point arithmetic. As a result, SwitchML and ATP require careful model-specific conversion of floating point gradients to fixed-point representation, which can affect the training time needed to reach a target accuracy. Second, Tofino switches do not maintain state across different pipelines and have a limited number of stages in each pipeline (Gebara et al., 2020). Hence, SwitchML and ATP are limited in terms of the number of ports that can be used for a single job and also in terms of the maximum packet size. PANAMA's accelerator design, in contrast, is flexible and can support floating-point operations at line rates and can scale to hundreds of ports. Li et al. propose in-network aggregation to accelerate reinforcement learning jobs using a design that implements the required aggregation logic and switching functionality within a single FPGA (Li et al., 2019). Unlike ours, their proposal assumes small model sizes (RL models) that can be stored on-chip, but this prevents their design from being applicable to today's large-scale DNN models with billions of parameters. Further, it requires a fabric speed up that grows proportionally with the aggregate line rate. Mellanox proposes an in-network aggregation solution called Sharp (Bloch, 2019) using a custom hardware specialized for collective reduction operations (allreduce) inside the switch. But because there is no publicly available design, it is hard to speculate how this has been achieved and if/how it can scale to high data rates. Moreover, Sharp is geared towards HPC and requires exclusive network access, making it a poor fit for today's shared ML clusters in cloud data centers. Similarly, Klenk et al. propose a hardware unit that can be incorporated within switches to accelerate allreduce operations (among other collective primitives) (Klenk et al., 2020). However, their solution relies on shared memory primitives, and this introduces additional complexity to ensure jobs have no aliasing pointers. More generally, unlike all prior work, we focus on the impact of using in-network aggregation on the entire data center traffic by considering a realistic shared environment as opposed to a dedicated cluster.

9 CONCLUSION

Recent proposals have advocated the use of in-network aggregation to improve distributed ML training time. However, the practical viability of these approaches is limited by the lack of efficient aggregation hardware, and routing and congestion control protocols, making them unsuitable for shared data center environments. In this paper, we take a first step towards filling this gap by presenting PANAMA, a novel in-network aggregation framework designed to operate in shared clusters. PANAMA leverages the unique properties of in-network aggregation to achieve fast hardware acceleration without sacrificing accuracy, and efficient and fair usage of network resources. Contrary to common wisdom, we demonstrate the benefits of in-network aggregation extend to non-aggregation traffic and are not solely limited to data-parallel ML jobs.

Acknowledgments. We thank our anonymous reviewers. This work was partly supported by the Microsoft Research PhD scholarship program, the UK EPSRC grant EP/L016796/1, NSF grants CNS-2008624, ASCENT-2023468, and SystemsThatLearn@CSAIL Ignite Grant.

REFERENCES

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI'16)*, pp. 265–283, USA, 2016. USENIX Association. ISBN 9781931971331.

Al-Fares, M., Loukissas, A., and Vahdat, A. A scalable, commodity data center network architecture. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (SIGCOMM'08)*, pp. 63–74, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781605581750. doi: 10.1145/1402958.1402967. URL https://doi.org/10.1145/1402958.1402967.

Alizadeh, M., Greenberg, A., Maltz, D. A., Padhye, J., Patel, P., Prabhakar, B., Sengupta, S., and Sridharan, M. Data Center TCP (DCTCP). In *Proceedings of the ACM SIGCOMM 2010 Conference (SIGCOMM'10)*, pp. 63–74, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450302012. doi: 10.1145/1851182.1851192. URL https://doi.org/10.1145/1851182.1851192.

Alizadeh, M., Yang, S., Sharif, M., Katti, S., McKeown, N., Prabhakar, B., and Shenker, S. Pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM'13*, pp. 435–446, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi: 10.1145/2486001.2486031. URL https://doi.org/10.1145/2486001.2486031.

Amazon. Amazon Elastic Graphics. https://aws.amazon.com/ec2/elastic-graphics/, 2021.

Azure, M. GPU-ACCELERATED MICROSOFT AZURE. https://www.nvidia.com/en-us/data-center/gpu-cloud-computing/microsoft-azure/, 2021.

Bloch, G. Accelerating Distributed Deep Learning with In-Network Computing Technology, Aug. 2019. URL https://conferences.sigcomm.org/events/apnet2019/slides/Industrial_1_3.pdf.

Bosshart, P., Gibb, G., Kim, H.-S., Varghese, G., McKeown, N., Izzard, M., Mujica, F., and Horowitz, M. Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN. In *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM (SIGCOMM '13)*, pp. 99–110, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320566. doi: 10.1145/2486001.2486011. URL https://doi.org/10.1145/2486001.2486011.

Broadcom. BCM56990 25.6 Tb/s Tomahawk 4 Ethernet Switch. https://docs.broadcom.com/docs/12398014, 2020.

Chowdhury, M. and Stoica, I. Coflow: A Networking Abstraction for Cluster Applications. In *HotNets*, HotNets-XI, pp. 31–36, 2012.

Costa, P., Donnelly, A., Rowstron, A., and O'Shea, G. Camdoop: Exploiting In-network Aggregation for Big Data Applications. In *Proceedings of the 9^{th} USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, 2012.

Dong, M., Li, Q., Zarchy, D., Godfrey, P. B., and Schapira, M. PCC: Re-Architecting Congestion Control for Consistent High Performance. In *Proceedings of the 12th USENIX Conference on Networked Systems Design and Implementation (NSDI'15*, pp. 395–408, USA, 2015. USENIX Association. ISBN 9781931971218.

Firestone, D., Putnam, A., Mundkur, S., Chiou, D., Dabagh, A., Andrewartha, M., Angepat, H., Bhanu, V., Caulfield, A., Chung, E., Chandrappa, H. K., Chaturmohta, S., Humphrey, M., Lavier, J., Lam, N., Liu, F., Ovtcharov, K., Padhye, J., Popuri, G., Raindel, S., Sapre, T., Shaw, M., Silva, G., Sivakumar, M., Srivastava, N., Verma, A., Zuhair, Q., Bansal, D., Burger, D., Vaid, K., Maltz, D. A., and Greenberg, A. Azure Accelerated Networking: SmartNICs in the Public Cloud. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pp. 51–66, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/conference/nsdi18/presentation/firestone.

Gebara, N., Lerner, A., Yang, M., Yu, M., Costa, P., and Ghobadi, M. Challenging the Stateless Quo of Programmable Switches. In *ACM Workshop on Hot Topics in Networks (HotNets)*. ACM, November 2020. URL https://www.microsoft.com/en-us/research/publication/challenging-the-stateless-quo-of-programmable-swi

Google. Cloud GPUs. https://cloud.google.com/gpu/, 2021.

Greenberg, A., Hamilton, J. R., Jain, N., Kandula, S., Kim, C., Lahiri, P., Maltz, D. A., Patel, P., and Sengupta, S. VL2: A Scalable and Flexible Data Center Network. In *Proceedings of the ACM SIGCOMM 2009 Conference on Data Communication (SIGCOMM'09)*, pp. 51–62, New York, NY, USA, 2009. Association for Computing Machinery. ISBN 9781605585949. doi: 10.1145/1592568.1592576. URL https://doi.org/10.1145/1592568.1592576.

Ha, S., Rhee, I., and Xu, L. CUBIC: A New TCP-Friendly High-Speed TCP Variant. *SIGOPS Oper. Syst. Rev.*, 42(5):64–74, July 2008. ISSN 0163-5980. doi: 10.1145/1400097.1400105. URL https://doi.org/10.1145/1400097.1400105.

Handley, M., Raiciu, C., Agache, A., Voinescu, A., Moore, A. W., Antichi, G., and Wójcik, M. Re-architecting datacenter networks and stacks for low latency and high performance. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'17)*, pp. 29–42, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450346535. doi: 10.1145/3098822.3098825. URL https://doi.org/10.1145/3098822.3098825.

Hopps, C. Analysis of an Equal-Cost Multi-Path Algorithm. RFC 2992, 2000. URL https://rfc-editor.org/rfc/rfc2992.txt.

Huang, Y., Cheng, Y., Chen, D., Lee, H., Ngiam, J., Le, Q. V., and Chen, Z. GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism. *CoRR*, abs/1811.06965, 2018. URL http://arxiv.org/abs/1811.06965.

Intel. Intel stratix 10 tx device overview. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_tx_overview.pdf, 2016.

Intel. P4-programmable Ethernet Tofino 2. https://www.intel.com/content/www/us/en/products/network-io/programmable-ethernet-switch/tofino-2-series/tofino-2.html/, 2018.

Jeon, M., Venkataraman, S., Phanishayee, A., Qian, u., Xiao, W., and Yang, F. Analysis of Large-Scale Multi-Tenant GPU Clusters for DNN Training Workloads. In *Proceedings of the 2019 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '19)*, pp. 947–960, USA, 2019. USENIX Association. ISBN 9781939133038.

Jia, Z., Zaharia, M., and Aiken, A. Beyond Data and Model Parallelism for Deep Neural Networks. In Talwalkar, A., Smith, V., and Zaharia, M. (eds.), *Proceedings of Machine Learning and Systems*, volume 1, pp. 1–13, 2019. URL https://proceedings.mlsys.org/paper/2019/file/c74d97b01eae257e44aa9d5bade97baf-Paper.pdf.

Klenk, B., Jiang, N., Thorson, G., and Dennison, L. An In-Network Architecture for Accelerating Shared-Memory Multiprocessor Collectives. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA'20)*, pp. 996–1009, 2020.

Kumar, G., Dukkipati, N., Jang, K., Wassel, H. M. G., Wu, X., Montazeri, B., Wang, Y., Springborn, K., Alfeld, C., Ryan, M., Wetherall, D., and Vahdat, A. Swift: Delay is Simple and Effective for Congestion Control in the Datacenter. In *Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication on the Applications, Technologies, Architectures, and Protocols for Computer Communication (SIGCOMM '20)*, pp. 514–528, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450379557. doi: 10.1145/3387514.3406591. URL https://doi.org/10.1145/3387514.3406591.

Lao, C., Le, Y., Mahajan, K., Chen, Y., Wu, W., Akella, A., and Swift, M. ATP: In-network Aggregation for Multi-tenant Learning. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI 21)*. USENIX Association, April 2021. URL https://www.usenix.org/conference/nsdi21/presentation/lao.

Lepikhin, D., Lee, H., Xu, Y., Chen, D., Firat, O., Huang, Y., Krikun, M., Shazeer, N., and Chen, Z. GShard: Scaling Giant Models with Conditional Computation and Automatic Sharding, 2020.

Li, M., Andersen, D. G., Park, J. W., Smola, A. J., Ahmed, A., Josifovski, V., Long, J., Shekita, E. J., and Su, B.-Y. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI'14)*, pp. 583–598, USA, 2014. USENIX Association. ISBN 9781931971164.

Li, Y., Liu, I.-J., Yuan, Y., Chen, D., Schwing, A., and Huang, J. Accelerating Distributed Reinforcement Learning with In-Switch Computing. In *Proceedings of the 46th International Symposium on Computer Architecture (ISCA'19*, ISCA '19, pp. 279–291, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450366694.

libtins. libtins packet crafting and sniffing library. https://libtins.github.io/, 2021.

Ltd., O. OMNeT++ Discrete Event Simulator, 2021. https://omnetpp.org/.

Mai, L., Rupprecht, L., Alim, A., Costa, P., Migliavacca, M., Pietzuch, P. R., and Wolf, A. L. NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres. In Seneviratne, A., Diot, C., Kurose, J., Chaintreau, A., and Rizzo, L. (eds.), *Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies (CoNEXT '14)*, pp. 249–262. ACM, 2014. doi: 10.1145/2674005.2674996. URL https://doi.org/10.1145/2674005.2674996.

Mittal, R., Lam, V. T., Dukkipati, N., Blem, E., Wassel, H., Ghobadi, M., Vahdat, A., Wang, Y., Wetherall, D., and Zats, D. TIMELY: RTT-Based Congestion Control for the Datacenter. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM'15)*, pp. 537–550, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335423. doi: 10.1145/2785956.2787510. URL https://doi.org/10.1145/2785956.2787510.

Moritz, P., Nishihara, R., Wang, S., Tumanov, A., Liaw, R., Liang, E., Elibol, M., Yang, Z., Paul, W., Jordan, M. I., and Stoica, I. Ray: A Distributed Framework for Emerging AI Applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018.

Narayanan, D., Harlap, A., Phanishayee, A., Seshadri, V., Devanur, N. R., Ganger, G. R., Gibbons, P. B., and Zaharia, M. Pipedream: Generalized pipeline parallelism for dnn training. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles (SOSP'19)*, pp. 1–15, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368735. doi: 10.1145/3341301.3359646. URL https://doi.org/10.1145/3341301.3359646.

NVIDIA. NVIDIA Collective Communications Library (NCCL). https://docs.nvidia.com/deeplearning/nccl/, 2021.

Omkar R., D. and M., A. Asic implementation of 32 and 64 bit floating point alu using pipelining. *International Journal of Computer Applications*, 94:27–35, 05 2014. doi: 10.5120/16452-6184.

Peng, Q., Walid, A., and Low, S. H. Multipath TCP: analysis and design. *CoRR*, abs/1308.3119, 2013. URL http://arxiv.org/abs/1308.3119.

Rossi, F. and Durut, M. Communication challenges in cloud k-means. In *ESANN*, 2011.

Sapio, A., Canini, M., Ho, C., Nelson, J., Kalnis, P., Kim, C., Krishnamurthy, A., Moshref, M., Ports, D. R. K., and Richtárik, P. Scaling Distributed Machine Learning with In-Network Aggregation. In *18th USENIX Symposium on Networked Systems Design and Implementation (NSDI'21)*. USENIX Association, April 2021. URL https://www.usenix.org/conference/nsdi21/presentation/sapio.

Sergeev, A. and Balso, M. D. Horovod: fast and easy distributed deep learning in TensorFlow, 2018.

Shoeybi, M., Patwary, M., Puri, R., LeGresley, P., Casper, J., and Catanzaro, B. Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism, 2020.

Singh, A., Ong, J., Agarwal, A., Anderson, G., Armistead, A., Bannon, R., Boving, S., Desai, G., Felderman, B., Germano, P., Kanagala, A., Provost, J., Simmons, J., Tanda, E., Wanderer, J., Hölzle, U., Stuart, S., and Vahdat, A. Jupiter rising: A decade of clos topologies and centralized control in google's datacenter network. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 183–197, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335423. doi: 10.1145/2785956.2787508. URL https://doi.org/10.1145/2785956.2787508.

Stillmaker, A. and Baas, B. Scaling equations for the accurate prediction of cmos device performance from 180nm to 7nm. *Integration*, 58:74–81, 2017. ISSN 0167-9260. doi: https://doi.org/10.1016/j.vlsi.2017.02.002. URL https://www.sciencedirect.com/science/article/pii/S0167926017300755.

Sun, P., Feng, W., Han, R., Yan, S., and Wen, Y. Optimizing Network Performance for Distributed DNN Training on GPU Clusters: ImageNet/AlexNet Training in 1.5 Minutes. *CoRR*, abs/1902.06855, 2019. URL http://arxiv.org/abs/1902.06855.

TensorFlow. TensorFlow Benchmarks. https://github.com/tensorflow/benchmarks, 2021.

The Open MPI Project. Open MPI:Open Source High Performance Computing. https://www.open-mpi.org/, 2021.

Uber Eng. Meet Horovod: Uber's Open Source Distributed Deep Learning Framework for TensorFlow. https://eng.uber.com/horovod, 2017.

Xilinx. Logicore ip floating-point operator v7.0. https://www.xilinx.com/support/documentation/ip_documentation/floating_point/v7_0/pg060-floating-point.pdf, 2014.

Xilinx. Ultrascale+ FPGAs Product Tables and Product Selection Guide. https://www.xilinx.com/support/documentation/selection-guides/ultrascale-plus-fpga-product-selection-guide.pdf, 2021.

Zhu, Y., Eran, H., Firestone, D., Guo, C., Lipshteyn, M., Liron, Y., Padhye, J., Raindel, S., Yahia, M. H., and Zhang, M. Congestion Control for Large-Scale RDMA Deployments. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication (SIGCOMM '15)*, pp. 523–536, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 9781450335423. doi: 10.1145/2785956.2787484. URL https://doi.org/10.1145/2785956.2787484.

Zhuo, D., Ghobadi, M., Mahajan, R., Förster, K.-T., Krishnamurthy, A., and Anderson, T. Understanding and mitigating packet corruption in data center networks. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication (SIGCOMM '17)*, pp. 362–375, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4653-5. doi: 10.1145/3098822.3098849. URL http://doi.acm.org/10.1145/3098822.3098849.

Zilberman, N., Audzevich, Y., Covington, G., and Moore, A. W. Netfpga sume: Toward 100 gbps as research commodity. *IEEE Micro*, 34(05):32–41, sep 2014. ISSN 1937-4143. doi: 10.1109/MM.2014.61.

A HARDWARE ACCELERATOR SCALABILITY ANALYSIS

Here we provide a more detailed analysis of the scalability of the PANAMA hardware accelerator and illustrate how it can scale to higher data rates and to a large number of ports.

Increasing port rates. All the modules in our design are pipelined and inter-connected via a streaming bus interface with a specific bus width (W). Therefore, the maximum data rate (R) per port that can be supported by our accelerator can be easily computed as the product of the bus width and the clock rate used (f):

$$R = W \times f \tag{1}$$

In our FPGA-based prototype targeting a 28-nm Virtex-7 FPGA, we had to set the clock rate to 200 MHz to meet timing. Therefore, supporting 100 Gbps would require increasing the bus width to 512 bits (we use W=64 bits in our prototype because we use 10-Gbps ports). In our design, the limiting bottleneck for the clock rate is the floating-point adder unit. Scaling to higher port rates thus requires moving to an ASIC implementation. Prior work has shown that a 180-nm implementation of a floating-point ALU can support a clock rate of up to 1 GHz using six pipeline stages (Omkar R. & M., 2014). This would allow to scale port rates up to 512 Gbps assuming W=512 bits (see Fig. 11).

Increasing the number of ports. Scaling to higher port count requires a larger degree of parallelism, and this results in larger logic resource utilization. To understand the scaling implications, we first consider the breakdown of the FPGA look-up tables (LUTs) and flip-flops (FFs) for our 4-port 10 Gbps accelerator prototype targeting the VU19P FPGA (Xilinx, 2021) (see Table 2). Increasing the number of ports primarily impacts the number of header parsers and floating-point adders. The number of parsers grows proportionally with the number of ports, as in our design, we assign a different parser to each port. The number of adders is a function of the *total* accelerator bandwidth, i.e., the product of the port rate (R) and the total number of ports (N). Higher port rates require more parallel adder trees while higher port counts require deeper trees with more adders. Assuming 32-bit floating-point values, the total number of adders N_a can be expressed as follows:

$$N_a = \frac{R}{f} \times \frac{N-1}{32} = W \times \frac{N-1}{32} \tag{2}$$

Using this formula and the values in Table 2, we can estimate the LUT and FF resource utilization for our reference FPGA as we scale the number of ports (we assume 100 Gbps per port). The chart in Fig. 12 shows that even for a high number of ports (128) the LUT utilization is still below 35% (resp. below 5% for FF utilization). While these results are only indicative because they only focus on the core logic components, excluding other elements of the design (e.g., IO), they are very promising and suggest it could be possible to use FPGAs to implement our accelerators, even at 100 Gbps with a relatively high number of ports.

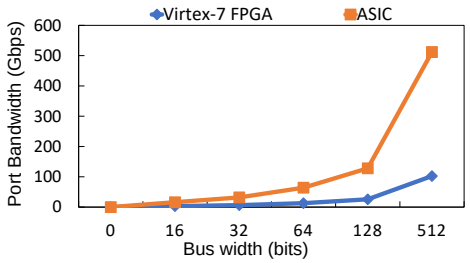


Figure 11: Port bandwidth scaling. Increasing the bus width allows supporting higher port bandwidth.

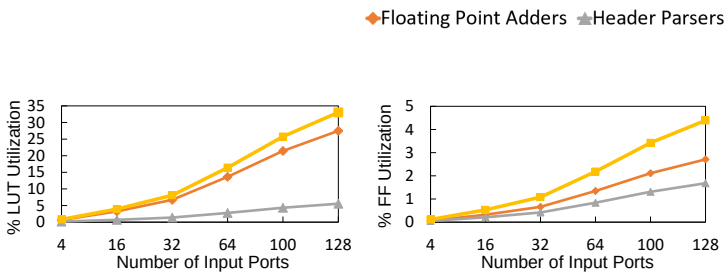
Component	Instances	Logic Utilization per Instance	
		LUTs	FFs
Header Parser	4	188	118
Buffering Control	4	127	118
Aggregation Datapath Adder	6	352	72
Aggregation Datapath Control Logic	1	1089	963
Total Utilization (% VU19P FPGA Resources) (Xilinx, 2021)		4461 (1.03%)	2339 (0.26%)

Table 2: Prototype FPGA Logic Utilization

As we mentioned, scaling the port rates beyond 100 Gbps requires an ASIC implementation. To estimate the chip area needed for the core logic, we consider the area size reported in literature for a single floating-point ALU unit implemented using 180-nm technology node (Omkar R. & M., 2014). We extrapolate this value (0.936860 mm^2) to today's 7 nm technology using published conversion tables (Stillmaker & Baas, 2017). In this preliminary analysis, we only focus on adders and ignore the contribution of parsers because the former dominates: as we assume W=512 bits, we have approximately $\frac{512}{32} = 16$ times more adders than parsers (see Eq. (2)). As shown in Fig. 13, the core logic occupies only a modest area (5.94 mm^2 at 128 ports). Clearly this analysis is very preliminary and much more work is needed to provide an accurate estimate of the chip area but these early estimates are encouraging.

Impact on latency. We conclude our analysis by focusing on the scaling impact on the cut-through latency (L), i.e., the time taken for a packet to traverse the accelerators. This is a function of the packet size in bits (S), the bus width (W), the clock rate (f), the latency of the individual adders (L_a), and the number of ports (N):

$$L = \left(\frac{S}{W} - 1\right) \times \frac{1}{f} + L_a \times \log_2 N \tag{3}$$



(a) LUT utilization (b) FF utilization

Figure 12: Logic utilization scaling on the FPGA with number of ports (100 Gbps per port).

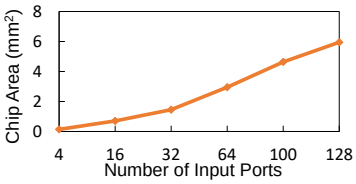
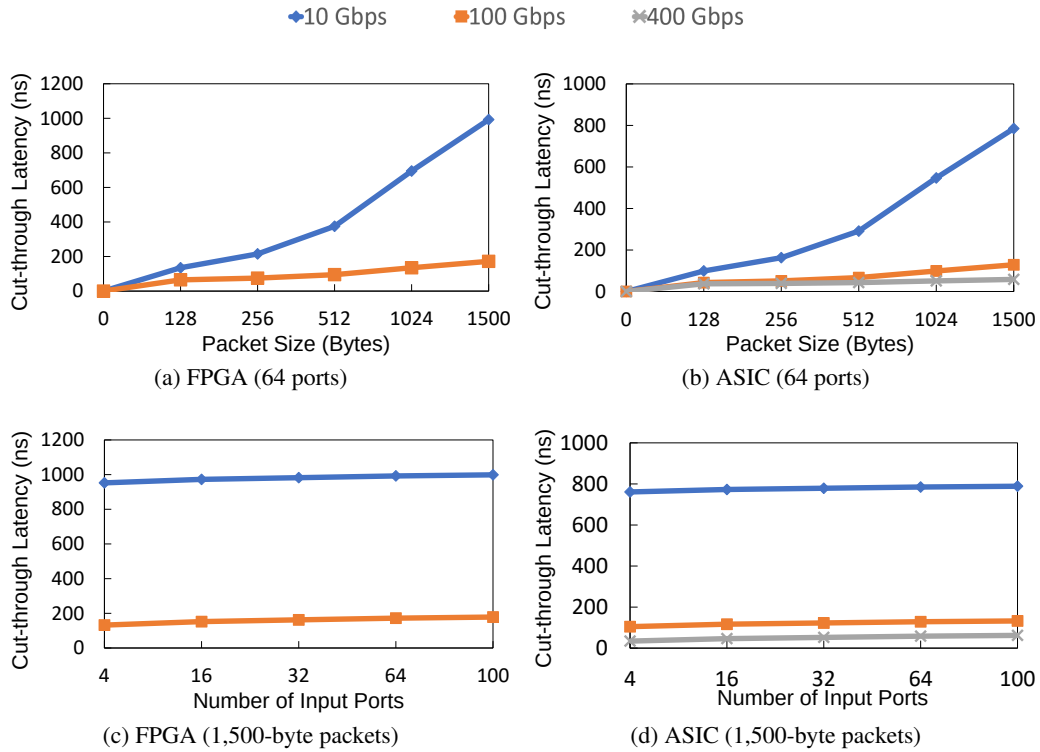


Figure 13: Chip area scaling with number of ports at a *port bandwidth* of 400 Gbps.



(a) FPGA (64 ports) (b) ASIC (64 ports)

(c) FPGA (1,500-byte packets) (d) ASIC (1,500-byte packets)

Figure 14: Cut-through latency for different packet sizes assuming 64 ports (a and b) and for increasing number of ports with 1,500-byte packets (c and d).

To estimate the FPGA latency, we use the value L_a=10 ns as measured in our prototype while for ASIC estimates, we use L_a=6 ns as reported in literature (Omkar R. & M., 2014). Fig. 14 plots the cut-through latency for both an FPGA and an ASIC implementation for different port rates as a function of the packet size assuming 64 ports (Fig. 14a and Fig. 14b), and as we scale the number of ports with 1,500-byte packets (Fig. 14c and Fig. 14d). In all cases, for rates of 100 Gbps or higher, the latency is equal to or lower than 133 ns, which is comparable to the cut-through latency of Ethernet packet switches(Broadcom, 2020).